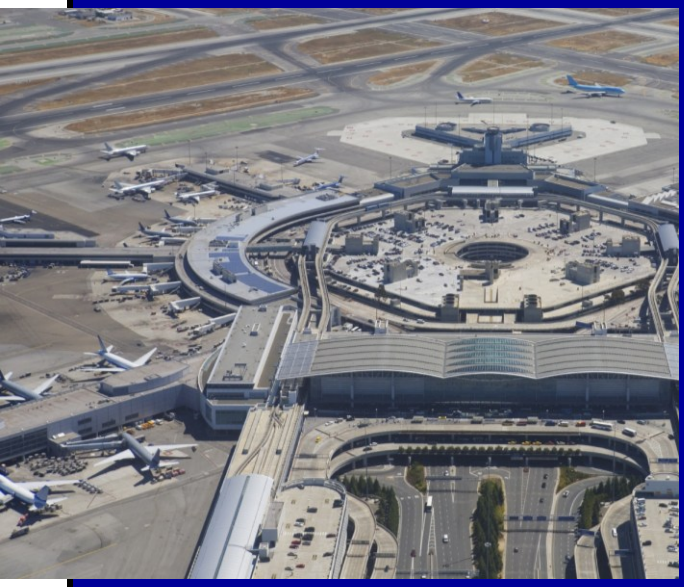
Simio と シミュレーション

—モデリング・解析・応用—

第 7 版



原著者

Jeffrey S. Smith

David. T. Sturrock

訳

野村　淳一

三輪　冠奈

譚　奕飛

岳　理恵

楊　文賀

監修

高桑　宗右ヱ門

Japanese Translation of:

Simio and Simulation:
Modeling, Analysis, Applications

7th Edition

Simio

Forward Thinking

Simio とシミュレーション
―モデリング・解析・応用―

第 7 版

原著者
Jeffrey S. Smith
David. T. Sturrock

訳
野村　淳一
三輪　冠奈
譚　奕飛
岳　理恵
楊　文賀

監修
高桑　宗右ヱ門

第７版　まえがき

この第７版は、ヘルスケア、マイニング、重工業、サプライチェーンなど、あらゆる応用の場面において、より良いビジネスの意思決定を行うため、シミュレーションの用法について説明したものです。そして、理系・技術系ならびに文系・非技術系のいずれの読者にとっても、シミュレーションの概念や有用性をよりよく理解できるように書かれています。さらに、教科書としてだけでなく、独習用の教材としても用いることができるように工夫されています。近年のソフトウェアは、従前にもましていっそう利用しやすくなっており、本書はシミュレーションソフトウェア業界のリーダーたる Simio®を用いてシミュレーションの概念を解説したものです。なお、第 6 版のコンテンツは、https://textbook.simio.com/SASMAA6/にありますので、併せて参考にしてください。

第７版は、Simio Version 17 以降を用いて書かれており、第６版の刊行以降の３年間で行われた変更事項に合わせて、操作手順、図、例題が更新されています。特に、第 6 版からは、オンラインにて無料で入手できるように頒布方法を変更しました。また、多くの機能が追加されています。インダストリー4.0 下におけるシミュレーションによるスケジューリングシステムに関する章を更新し、実際のケーススタディを加えました。また、デジタルツイン、深層学習、ニューラルネットワークによるインテリジェントな配送計画における AI の役割に関するセクションを含む AI 対応シミュレーションモデリングに関する包括的な章を追加しました。章末問題は改善され拡張され、多くの読者の提案が取り入れられています。資料の流れを改善するために再編成し、最近追加された多くの新しい Simio 機能に関して全編に更新を加えました。

本書は、学部ならびに大学院初学年を対象とした内容となっています。そして本書は、一般的な概念よりは、むしろ特定の事例を中心としたアプローチを理解しやすいようにチュートリアル形式で書かれており、広範囲な応用を扱っています。これまでの経験により、このような特徴をもたせることで、いろいろな分野の学生に関心をもってもらうことができ、かつテキストの内容を読みやすくさらに理解しやすいように工夫しました。

授業のシミュレーションコース（前期）においては、第 1 章から第 8 章まで、さらに学部上級学年および大学院初学年では第 9 章から第 11 章を取り上げるとよいでしょう。また、シミュレーションコース（後期）では、第 1 から 3、6 章を復習し、第 12 章までの内容をじゅうぶんに学習したうえで、付録 A および B の内容を宿題やプロジェクトの課題として学習するのもよいでしょう。

シミュレーションとしての単独の授業がない場合（たとえば MBA）には、数週間の講義で本書の一部分を学習することで、シミュレーションの単元を理解することができます。そこでは、 第 1、4、5 章と、そして第 7、8 章を概観するとよいでしょう。さらに、付録 B の実社会に関するシミュレーション応用事例も講義内容に応じて紹介されるとよいでしょう。

第 11 章で特に取り上げた内容は、プログラミングの学習経験のある学生にと

っては、他の研究テーマに発展させることができるので、興味深いプロジェクト研究のテーマを提供することができます。同様に、第 12 章は、シミュレーションベースの計画とスケジューリング、およびスマートファクトリーやインダストリー4.0 におけるシミュレーションの応用に関する発展学習あるいは最新の研究への糸口として用いることもできます。また、付録 A の内容は、応用研究や個別の研究課題について、学生への課題として用いることもできます。本書は、Microsoft®、Windows®の OS や Microsoft Excel®や Microsoft Word®のようなアプリケーションに習熟されていることを前提として執筆しました。また、確率や統計学についても履修済みであることを前提としており、確率や統計学について、読者は基本事項については理解されていることを前提としています。なお、第 2 章と第 6 章の最初の部分に概要について説明しました。

Jeffrey S. Smith
Vice President, Simio LLC

David. T. Sturrock
Technical Fellow, Simio LLC

[Guest Authors]
Mohammad Dehghani
Rylan Carney

日本の読者へ：Pegden 博士からのメッセージ

Simio はパフォーマンスを予測したり変更に対する影響を理解したりするために、幅広く用いられるシミュレーションモデリングのツールとなりました。これは、生産・物流・輸送・ビジネスプロセス・通信などの多様なシステムに応用され、多大な成果を収めてきました。

シミュレーションモデリングは、ユーザに多くの便益をもたらすとてもわくわくする技術です。シミュレーションにより、システムを構築するまえに、その設計について検討することができるため、新規のシステムを実装したり、あるいは既存のシステムを変更したりすることに伴うリスクや時間を軽減することができます。この技術に習熟された方であれば、シミュレーションを用いることなく、新規のシステムを設計したり、あるいは既存のシステムを改善したりすることは考えられないでしょう。Simio は、複雑なモデリングを容易に行うための Processes、データ駆動型（データドリブン）・データ生成モデリング、シミュレーションによるスケジューリング、シミュレーションによるデジタルツイン、そしてシミュレーションによる意思決定のあり方をさらに高めるための AI やニューラルネットワーク技術の付与など、これまで技術的なリーダーシップを発揮してきました。

シミュレーションを広く応用することの阻害要因のひとつに、この技術を使いこなせる熟練者の不足が挙げられます。シミュレーションは確かに多大な便益をもたらしますが、モデリングや解析における基本的な概念を理解する必要があります。ユーザはシミュレーションの方法論を理解すると同時に、シミュレーションのモデリングツールを用いて、その方法論を首尾よく実施できなければなりません。本書の原著者はシミュレーションの基本概念ならびに Simio シミュレーションツールを用いたシミュレーションの応用について、卓越した実績をもたれた方がたです。したがって、本書は Simio を用いたシミュレーションを首尾よく適用する際の重要な事柄を完全に網羅しています。

本書『Simio とシミュレーション－モデリング・解析・応用－』が刊行されて以来、世界中の多数の大学において、シミュレーションの概念を教育するために重要な役割を果たしてきました。そして今日では、本書は最も多く採用されているシミュレーションの教科書のひとつとなっています。さらに、本書は、この技術について洞察力をもち熟練したユーザとなるための手助けとなるべく、シミュレーションを首尾よく適用する際の、要となる役割を果たしています。

私は、髙桑宗右ヱ門教授がこの重要な本書を日本の読者のために翻訳されたことに大変感謝しています。高桑教授はシミュレーションモデリングならびに Simio において、国際的に評価されているエキスパートであり、氏のこれら両分野における知識と経験によって、本書の日本語版がすばらしいものとなることでしょう。

私と同様に、この日本語版をどうか楽しんでお読みいただきたいと思います。本書はシミュレーションの初学者向けに執筆されていますが、Simio を用いたシミュレーションを首尾よく応用することにご関心のある皆さまにも必ずやお役に立つものと思います。

C. Dennis Pegden, CEO, Simio LLC

第 7 版（日本語版）　監修者まえがき

科学技術の分野においては、イノベーションにより当該技術が日々進歩しています。システムシミュレーション言語においても、これまで主流であったプロセス指向シミュレーションからオブジェクト指向シミュレーションへと、いわば次世代のシミュレーション言語が登場し新たな段階を迎えました。本書で取り上げられている Simio はこのオブジェクト指向シミュレーション言語であり、Arena（前身は SIMAN）を開発された Dennis Pegden 博士と Simio LLC のチームが、シミュレーション言語の長年にわたる豊富な開発経験に基づいて、新たに開発されたものです。本書の原著者である、Jeffrey S. Smith および David. T. Sturrock の両氏は、いずれもシステムシミュレーションの分野における世界的な権威であり、これまで多数の著作を発表され、また世界最大規模のシミュレーションの国際コンファレンス（Winter Simulation Conference）において指導的な役割を果たされた方がたです。

日本語版第 7 版の原稿準備には、第 6 版に引き続き、野村淳一（第 1、2、3、6、10、13 章、付録 C）、三輪冠奈（第 5、8、9、付録 A、B）、譚奕飛（第 7 章）、岳理恵（第 4、11 章）、楊文賀（第 12 章）の各博士がそれぞれ担当されました。また、全体の構成内容の統一には、野村淳一、三輪冠奈両氏が担当され、ここに完成することができました。いずれもシステムシミュレーションに対する深い理解と研究実績を備えた気鋭の研究者であり、前版までと同様、メンバーの力を結集して、Simio という画期的なシミュレーション言語に関する日本語版テキストを刊行することができました。本書を通して、読者のみなさんとともに、すばらしい Simio シミュレーション言語を一緒に勉強していきたいと思います。また、日本語版の刊行準備にあたり、Simio LLC の David T. Sturrock 氏には特段の配慮をいただき、本書の内容に関して、随時打合せや確認をしながらようやく日本語版を完成することができました。ここに謝意を表したいと思います。

本書がわが国におけるシステムシミュレーションの教育や研究、ならびに実務への応用に対して、いささかでも貢献ができれば望外の幸せです。

2025 年 3 月
髙桑 宗右ヱ門

技術士事務所 シミュレーションリサーチ

Simio® を日本語で学ぶ

▶教材　▶実践的トレーニング　▶応用事例

目次

第 1 章　シミュレーションとは 1
1.1　本書について 2
1.2　システムとモデル 4
1.3　ランダム性とシミュレーションプロセス 6
1.4　シミュレーションを適用する場合（適用しない場合） 10
1.5　シミュレーションの成功に必要なスキル 11

第 2 章　待ち行列理論の基礎 21
2.1　待ち行列システムの構造と用語 22
2.2　リトルの法則とその他の関係式 25
2.3　複数窓口待ち行列ステーションの特性 26
2.4　待ち行列ネットワーク 30
2.5　待ち行列理論とシミュレーションの比較 33
2.6　問題 34

第 3 章　シミュレーションの種類 37
3.1　シミュレーションの分類 37
3.2　静的かつ確率的なモンテカルロシミュレーション 41
3.3　専用ソフトウェアを用いない動的シミュレーション 54
3.4　動的シミュレーションに対するソフトウェアの選択 64
3.5　問題 66

第 4 章　初めての Simio モデル 71
4.1　基本的な Simio ユーザインタフェース 72
4.2　Model 4-1：Standard ライブラリオブジェクトを用いた最初のプロジェクト 77
4.3　Model 4-2：Processes を用いたモデル 94
4.4　Model 4-3：現金自動預払機(ATM) 100
4.5　平均値を超えて：Simio MORE(SMORE)プロット 104
4.6　追加的統計分析に対する出力データのエクスポート 111
4.7　インタラクティブ・ログとダッシュボードレポート 112
4.8　モデルアニメーションの基礎 115
4.9　モデルのデバッグ 119
4.10　まとめ 123
4.11　問題 124

第 5 章　Simio の中級モデリング 127
5.1　Simio のフレームワーク 127
5.2　Model 5-1：PCB 組立 146
5.3　Model 5-2：PCB 組立の拡張 150
5.4　Model 5-3：工程選択のある PCB モデル 165
5.5　Model 5-4：複数の代替シナリオの比較 171
5.6　まとめ 176
5.7　問題 176

第 6 章　入力解析 179
6.1　単一変量の入力確率分布の特定 180
6.2　入力のタイプ 203
6.3　乱数生成法 206
6.4　確率変量および確率過程の生成 209
6.5　Simio Input Parameters の利用 212
6.6　問題 221

第 7 章　モデルのデータ操作 223
7.1　データテーブル 223
7.2　スケジュール 242
7.3　レートテーブルと Model 7-4 245
7.4　Lookup テーブルと Model 7-5 247
7.5　リストとチェンジオーバー 249
7.6　状態変数配列 251
7.7　データ駆動型モデル 252
7.8　データ生成モデル 255
7.9　まとめ 257
7.10　問題 258

第 8 章　アニメーションとエンティティ移動 261
8.1　アニメーション 261
8.2　エンティティ移動 276
8.3　まとめ 295
8.4　問題 295

第 9 章　Simio の高度なモデリング 297
9.1　Model 9-1：ED モデルの再考 297
9.2　Model 9-2：ピザのテイクアウトモデル 320
9.3　Model 9-3：固定容量バッファ 333
9.4　まとめ 338
9.5　問題 339

第 10 章　その他のモデリングトピック 343
10.1　Search ステップ 343
10.2　Model 10-3：退去と放棄 346
10.3　マテリアル要素とインベントリ要素 348
10.4　タスクシーケンス 349
10.5　イベントに基づく決定ロジック 353
10.6　その他のライブラリとリソース 355
10.7　実験 359
10.8　まとめ 360
10.9　問題 360

第 11 章　カスタマイズと Simio の拡張 363
11.1　オブジェクト定義に関する基本的なコンセプト 363
11.2　Model 11-1：階層的オブジェクトの構築 369

11.3　Model 11-2：ベースオブジェクトの作成.....373
11.4　Model 11-3：オブジェクトのサブクラス化.....377
11.5　ユーザ拡張機能の利用.....380
11.6　まとめ.....383
11.7　問題.....383

第 12 章　インダストリー4.0 時代のシミュレーションスケジューリング....385
12.1　産業革命の進化と変遷.....385
12.2　第四次産業革命：スマートファクトリー.....386
12.3　デジタルツインの必要性.....388
12.4　インダストリー4.0 におけるデザインシミュレーションの役割.....390
12.5　シミュレーションによるスケジューリングの役割.....392
12.6　計画とスケジューリングの課題.....395
12.7　シミュレーションを用いたスケジューリング.....398
12.8　リスクに基づく計画とスケジューリング.....400
12.9　Simio RPS を用いた計画とスケジューリング.....402
12.10　スケジューリングのインタフェース.....404
12.11　Model 12-1：モデルファーストアプローチによるスケジューリング....406
12.12　Model 12-2：データファーストアプローチによるスケジューリング....412
12.13　Uniko スケジューリング事例.....415
12.14　追加情報と例.....429
12.15　まとめ.....431
12.16　問題.....432

第 13 章　AI を活用したシミュレーションモデリング.....433
13.1　デジタルツインのレベル.....433
13.2　ディープラーニングの基礎.....435
13.3　例題 13-1：ニューラルネットワークによるインテリジェント作業手配...439
13.4　テストと性能評価.....453
13.5　まとめ.....454
13.6　問題.....455

付録 A　Simio を活用したケーススタディ.....459
A.1　機械加工・検査工程.....459
A.2　遊園地.....469
A.3　小さなレストラン.....474
A.4　小さな銀行支店.....476
A.5　Vacation 市の空港.....480
A.6　シンプルな最良の病院.....483

付録 B　Simio アカデミックケーススタディ・学生コンペティション課題...487
B.1　Simio Fresh Mex.....487
B.2　Simio Warehouse.....487
B.3　革新的なレンタカーサービス.....487
B.4　Simio 掘削ロジスティクス社.....488
B.5　Simio 救急治療センター.....488
B.6　航空機の生産システム.....489

B.7　ラテンアメリカのサプライチェーン 489
B.8　製紙・パルプ産業における原材料供給 490
B.9　グローバル通貨取引 490
B.10　Sunrun のソーラーパネル設置 490
B.11　種子生産 491
B.12　メガストア物流センター 491
B.13　地方空港の改善計画 491
B.14　ミーバック配送センター 492
B.15　小規模レストランの運営 493
B.16　DDMRP によるサプライチェーン計画 493
B.17　診療所と救急医療センター 494
B.18　洋上補給物流 494
B.19　サッカースタジアム運営 494
B.20　外国為替取引 495

付録 C　本書の使い方 497
C.1　フォーマットガイドライン 497
C.2　Simio ソフトウェア 497
C.3　本書のファイルとリソース 498
C.4　オンラインブックツールバーの使い方 499

参考文献 501

第 1 章　シミュレーションとは

過去 50 年にわたってシミュレーションが利用され続けてきたが、その重要性は衰えていない。むしろ、これから最盛期を迎えようとしている。業界最大規模の ICT アドバイザリ企業のガートナー（www.gartner.com）は、10 年以上前に戦略的テクノロジートップ 10 を発表した際、シミュレーションを含む**高度な分析**を第 2 位に位置づけた（Gartner 2009）。さらに、2012 年（Robb 2012）および 2013 年（Gartner 2013）においても、ガートナーは分析とシミュレーションの価値を改めて強調している：

> 分析は「ビジネスのエンジン」であるため、組織は時間的制約下でも、ビジネス・インテリジェンスへの投資を続けている。ガートナーは、より情報に基づいた意思決定を実現するため、ビジネス・インテリジェンスが次の段階として、シミュレーションや予測分析へと発展していくと予測している。
>
> 性能向上とコスト低下により、IT リーダーはビジネスのあらゆる活動について、分析とシミュレーションを実行できるようになった。クラウド・ベースのアナリティクス・エンジン、ビッグ・データ・リポジトリ、モバイル・クライアントの連携により、場所や時間を問わず最適化とシミュレーションが可能になっている。この進展により、ビジネス・プロセスのあらゆる場面で、シミュレーション、予測、最適化などのアナリティクスを活用し、より柔軟な意思決定を強力に支援することができる。

シミュレーション関連のハードウェアとソフトウェア技術は、この 10 年で飛躍的な進歩を遂げた。PC の処理能力は大幅に向上し、数年前では想像もできなかった高速な計算が可能になっている。ユーザインタフェースと製品設計の改善により、シミュレーションを効果的に活用するために必要な専門知識のハードルが下がり、より使いやすいソフトウェアが開発されている。オブジェクト指向技術の革新により、モデリングの柔軟性が向上し、非常に複雑なシステムも正確にモデル化できるようになった。さらに、直感的なシンボルと最新のハードウェアおよびソフトウェアを組み合わせることで、初心者でも魅力的な 3D アニメーションを用いたシミュレーションを作成でき、さまざまな背景を持つ関係者との円滑なコミュニケーションが可能になっている。これらの技術革新により、シミュレーションは重要な技術として確固たる地位を築いている。

2023 年、ガートナーは 2028 年に向けて、データとアナリティクスに関する 100 を超える予測を公表した（James and Duncan 2023）：

> （省略）データとアナリティクスは、民間・公共部門を問わず、あらゆる業界、ビジネス機能、IT 分野において不可欠な要素となっている。もっとも重要な点は、データとアナリティクスがデジタル・ビジネスの成功を左右する鍵だということである。

また、ガートナーは AI がシミュレーションに与える影響についても言及した：

> 人工知能（AI）が企業活動の中心となりつつある現在、IT リーダーは差別化を図るため、高度なアナリティクスを統合したシミュレーションプラットフォームを導入し、認知科学を軸にチームを再編成する必要がある。AI の導入を積極的に進める IT リーダーは、AI 投資からより大きな価値を引き出すことができる。（省略）シミュレーションと高度な AI 技術の組み合わせが、今後の AI 投資をさらに促進していくだろう。

これは、シミュレーションにおける AI の活用に関する最近の進展を的確に表現している。AI を活用したシミュレーションモデリングについての新しい章では、最新の実装事例を詳しく解説する。

本書では、シミュレーションの基礎技術と実践的なスキルに加え、最新のシミュレーションパッケージの実用的な解説を提供する。これを通じて、読者がシミュレーションの世界への第一歩を踏み出すことを願っている。

1.1　本書について

本書では、まず一般的なシミュレーション概念を紹介し、特定のソフトウェアに依存しない基本技術の理解から始める。1 章「**シミュレーションとは**」では、典型的な応用例、シミュレーションの適用を適切に判断する方法、およびプロジェクトの実施方法を扱う。2 章「**待ち行列理論の基礎**」では、待ち行列理論の概念とその長所・短所を紹介し、シミュレーションモデリングの検証における活用法を解説する。3 章「**シミュレーションの種類**」では、技術的側面と用語を紹介し、さまざまな観点からシミュレーションを分類して具体例とともに解説する。

続いて、Simio を用いた豊富な例を通じて、より詳細なシミュレーションの概念を説明する。これらの概念は、検証や出力分析といった技術要素ごとに章立てすることも可能だが、本書では各例を小規模プロジェクトとして扱い、それぞれで複数の概念を紹介する方法を選んだ。このアプローチにより、初期段階から実践的なスキルを総合的に習得でき、各プロジェクトを通じてそれらのスキルを強化できる。

4 章「**初めての Simio モデル**」では、Simio の基本的な解説から始め、単一サーバ待ち行列モデルの構築へと進む。本章では、Simio を用いたシミュレーションモデル構築プロセスを詳しく説明する。このプロセスは、モデリングのスキルに加え、シミュレーション出力の統計解析、実験、モデルの検証を含む。基本的なモデル構築と分析のプロセス自体は Simio 固有のものではないが、実装手段として Simio を活用する。同じモデルを低レベルのツールで再現することで、別のモデリング手法を説明し、「舞台裏」で行われている処理への深い理解を提供する。続いて、ATM モデルを用いて Simio の革新的な SMORE プロットによる出力分析を紹介する。最後に、モデルに潜む厄介な「バグ」の発見と追跡方法を解説して、本章を締めくくる。

5 章「**Simio の中級モデリング**」では、前章で学んだ Simio の基本的なモデリングと分析の概念をさらに深める。より現実的なシステムのモデル構築と実験から始め、Simio の動作と一般的なフレームワークについて詳しく解説する。次に、エ

レクトロニクス組立モデルを構築し、複数プロセスのモデリング、条件分岐、合流などの機能を追加していく。モデル開発の過程で、新しい Simio の機能を段階的に紹介する。また、複数の代替シナリオを比較する一般的な状況を想定し、統計的なシミュレーション実験の計画と分析方法を再検討する。本章を終える頃には、Simio を使用して、ある程度複雑なシステムのモデル化と分析が確実にできるようになるだろう。

本書では、ここまでいくつかの興味深いシミュレーションの応用例を紹介してきた。これを踏まえ、次はモデルを動作させるための入力分布と入力過程について説明する。6 章「**入力解析**」では、シミュレーションにおける入力の種類、実世界のデータをシミュレーションプロジェクトで活用できる形に変換する方法、そして多くのシミュレーションに不可欠な適切な乱数の生成方法を解説する。

7 章「**モデルのデータ操作**」では、現実のシステムを表現するために必要なさまざまデータについて幅広く検討する。まず、救命救急（ED）の基本モデルを構築し、Simio のデータテーブル構造を用いてモデルの入力データ要件を満たす方法を学ぶ。次に、モデルをより詳細に発展させながら、シーケンステーブル、リレーショナルデータテーブル、到着テーブルの概念とデータテーブルの入出力方法を説明する。さらに、ED モデルを拡張して、ワークスケジュール、レートテーブル、関数テーブルについて解説する。最後に、リスト、配列、転換行列を簡潔に紹介する。本章を通じて、モデルで一般的に使用されるデータ型の扱い方を習得できる。

8 章「**アニメーションとエンティティ移動**」では、シミュレーションプロジェクトへの 2D および 3D アニメーションの追加が、検証とコミュニケーションを深め、モデルの信頼性を高める点について述べる。背景アニメーション、カスタムシンボル、ステータスオブジェクトなどの多様なアニメーションツールを解説し、既存のエレクトロニクス組立モデルを用いて新しいアニメーションスキルを実践する。さらに、さまざまな種類のリンクの解説と、作業の流れを処理するコンベヤの追加を行う。最後に、Simio の Vehicle および Worker オブジェクトを導入してエンティティの移動機能を拡張し、ED モデルを用いて要員配置とアニメーションの改善を検討する。

9 章「**Simio の高度なモデリング**」では、ED モデルの初期バージョンを用いて、意思決定のためのモデル活用法、特にシミュレーションによる最適化について解説する。続いて、ピザショップの例題を通じて新しいモデル構成要素を紹介し、これまでの概念を体系的にまとめる。さらに、組立ラインのモデルを用いて、スループットを最大化するためのバッファ容量の最適な割当を検討する。

10 章「**その他のモデリングトピック**」では、Search ステップ、待ち行列における退去と放棄、タスクシーケンス、イベントに基づく決定ロジックなどの高度なモデリング概念を解説する。また、フロー処理のための Flow ライブラリ、クレーンやエレベータなどの設備を扱う Extras ライブラリ、そして追加ツールを提供する Shared Items フォーラムについても紹介する。最後に、実験、反復実行、シナリオを効率的に実行するためのオプションについて説明する。

11 章「**カスタマイズと Simio の拡張**」では高度な内容を扱う。アドオンプロセ

スを活用して、独自のカスタムオブジェクトやライブラリを構築する方法を解説する。ベースオブジェクトからの階層的なオブジェクト構築例や、標準ライブラリのオブジェクトのサブクラス化例を示す。さらに、Simio に独自のルール、コンポーネント、アドオンを実装して機能を拡張する方法を説明する。

12 章「**インダストリー4.0 時代のシミュレーションスケジューリング**」では、計画とスケジューリングのツールとしてのシミュレーションの活用法を探る。この分野は長年研究されてきたが、近年のシミュレーションソフトウェアの進歩により、実装と活用が格段に容易になっている。インダストリー4.0 の実装における**デジタルツイン**としてのシミュレーションモデルの活用についても論じ、Simio のリスクベースの計画・スケジューリング技術（RPS）の解説で締めくくる。

前述のとおり、13 章「**AI を活用したシミュレーションモデリング**」は、デジタルツインにおける AI の役割、ディープラーニングおよびニューラルネットワークによるインテリジェントな配置など、AI を活用したシミュレーションモデリングについて包括的に解説する。AI の活用により、組織はデジタルツインの可能性を最大限に引き出し、業務を効率化し、ダウンタイムを最小限に抑え、全体的な生産性を向上させることができる。

付録 A「**Simio を活用したケーススタディ**」では、Simio モデルを用いたシステム分析について、4 つの入門的なケーススタディと 2 つの発展的なケーススタディを紹介する。これらは比較的大規模な事例で、各章の演習問題とは異なり、より実践的な問題解決スキルを養う機会を提供する。

付録 B「**Simio アカデミックケーススタディ・学生コンペティション課題**」では、規模を拡大している学生シミュレーションコンペティションの最近の課題を概説する。これは、挑戦的なプロジェクトを探したり、独自のプロジェクトを始める際のヒントを得るよい機会となる。

付録 C では、本書で参照される各種ファイルや補足資料の構成を説明する。

1.2　システムとモデル

システムは、ある目的に向かって協調して動作する構成要素の集合を指す、広範な意味を持つ用語である。システムには、現金自動預け払い機（ATM）前の待ち行列のような単純なものから、空港全体や世界的な流通ネットワークのような複雑なものまで含まれる。これらのシステムが（既存か計画中かを問わず）、さまざまな構成や状況下でどのように動作するかを理解しようとすることは自然であり、ときには不可欠である。

既存のシステムの場合、注意深い観察によって洞察を得ることができる。しかし、このアプローチには欠点がある。特定の条件を観察するために長時間の観測が必要となる可能性があり、信頼できる結論を得るにはじゅうぶんなサンプル数が必要となる。また、世界的な流通ネットワークのように、適切な詳細さでシステム全体を観測できる視点を得ることが困難なシステムも存在する。

システムの変更を検討する場合には、また別の課題が生じる。シフトに一時的に 2 人目の要員を追加するなど、実際のシステムを容易に変更できる場合もある。

しかし、多くの場合、この方法は現実的ではない。たとえば、30 万ドルの標準的な機械と 40 万ドルの高性能な機械のどちらを選ぶべきかを評価する場合、投資額を考えれば実験的な変更は難しい。さらに、システムがまだ存在しない場合、観察自体が不可能である。

こうした理由から、システムの理解を深めるために**モデル**が用いられる。モデルにはさまざまな種類があり、それぞれに長所と制約がある。自動車や飛行機の**物理モデル**は、風洞試験に見られるように、現実感と物理環境との相互作用の両面を検討できる。**解析モデル**は数学的な表現を用い、特定の問題領域では非常に有用だが、その適用範囲は限定的である。これらに比べ、シミュレーションははるかに広い適用性を持つモデリングアプローチである。

コンピュータシミュレーションは、システムとその内部プロセスの挙動を模倣し、時間経過に伴うシステムの動作結果を適切な詳細度で描き出すものである。シミュレーションモデルは通常、システムの構成要素を表現し、その経時的な動作を記録するように設計されたソフトウェアを用いて作成される。シミュレーションは、既存システムの変更による影響や新規システムのパフォーマンスを予測するために活用され、さらにシステムの設計、エミュレーション、運用にも広く用いられている。

シミュレーションは、確率的モデルと確定的モデルに分類される。一般的な**確率的**シミュレーションでは、システムに内在する変動を表現するためにランダム性が導入されている。たとえば、作業完了時間や品質水準などの人的活動の結果は常に変動し、顧客や材料などの外部入力もさまざまで、故障などの例外も発生する。一方、**確定的**モデルには変動が存在しない。これは設計での使用は少ないものの、スケジューリングやエミュレーションなど、モデルベースの意思決定支援では一般的に用いられる。これらの概念については、3.1.3 項「確定的モデルと確率的モデル」で詳しく説明する。

シミュレーションの主要な分類として、**離散**型と**連続**型がある。この区分は、システムの状態変化の性質に基づいている。待ち行列の長さや作業員の状態のように、特定の時点（**イベント時刻**）でのみ変化する離散的な状態がある一方で、タンク内の圧力やオーブン内の温度のように、時間とともに連続的に変化する状態もある。純粋な離散型や連続型のシステムもあるが、両方の特性を併せ持つシステムも存在する。3.1.2 項「連続型動的モデルと離散型動的モデル」では、これらをより詳しく検討し、連続型シミュレーションの例を示す。

連続型のシステムは、変化率を規定する**微分方程式**によって定義される。シミュレーションソフトウェアは、時間経過に伴って微分方程式の解を生成するために数値積分を使用する。**システムダイナミクス**は、連続型と同じ基本概念を用いて簡潔なモデルを構築するためのグラフィカルなアプローチである。これは主に、人口動態や市場の成長・減衰、その他の数式に基づいた相互関係のモデル化に活用されている。

離散型モデリングには、主に 3 つのパラダイムがある。**イベント**は、顧客の到着や出発などのシステム状態が変化する時点をモデル化する。**プロセス**は、時間

経過に伴って発生する一連の活動をモデル化する。たとえば、生産システムでは、作業者の占有、サービス時間の経過、作業者の解放などの活動が含まれる。**オブジェクト**は、設備の観点からモデルを記述する。**エージェントベースモデリング**（ABM）は、オブジェクトの特殊なケースで、兵士、市場での企業、伝染病の感染者など、多数の自律的な意思決定を行うオブジェクトの相互作用からシステムの挙動を考察する。

最近のシミュレーションパッケージは複数のパラダイムを組み込むようになり、これらのパラダイム間の境界は次第にあいまいになっている。Simio はマルチパラダイムのモデリングツールで、1 つのフレームワークですべてのパラダイムを扱うことができる。単一のパラダイムを使用することも、同じモデル内で複数のパラダイムを組み合わせることも可能である。Simio は、オブジェクトの使いやすさとプロセスの柔軟性を両立させている。

シミュレーションは幅広い分野で応用されている。以下に、システムの有効性の理解と改善のためにシミュレーションが活用された主な分野の例を示す：

- 空港：駐車場のシャトルバス、チケットカウンタ、セキュリティ検査、ターミナル間輸送、フードコート、手荷物取り扱い、ゲート割当、航空機の防氷処理。
- 病院：救急部門、災害対策、救急車配備、地域医療戦略、資源配分。
- 港湾：入出港管理、港湾管理、コンテナ保管、設備投資、クレーン作業。
- 採掘：原材料や作業員の輸送、機器配置、バルク材混合。
- 遊園地：来場者輸送、アトラクション設計・運営、待ち行列管理、従業員配置、混雑対策。
- コールセンタ：人員配置、スキル評価、サービス改善、教育計画、スケジューリング。
- サプライチェーン：リスク管理、発注点設定、生産・在庫配置、輸送、拡張計画、危機管理。
- 製造業：設備投資分析、ライン最適化、製品構成の変更、生産性向上、物流、労働力計画。
- 軍事：後方支援、整備、戦闘、治安維持、偵察、人道支援。
- 通信：メッセージ伝送、経路制御、信頼性確保、障害・攻撃への耐性評価。
- 刑事司法制度：保護観察・仮釈放の運用、刑務所の収容率と収容能力の管理。
- 緊急対応システム：応答時間、拠点配置、設備水準、人員配置。
- 公共部門：投票区への投票機配置。
- 顧客サービス：ダイレクトサービスの改善、後方業務、資源配分、能力計画。

シミュレーションは製造業向けツールという枠を超え、その適用分野は幅広く、実質的に無限の可能性を持っている。

1.3　ランダム性とシミュレーションプロセス

本節では、シミュレーションプロセスの典型的な手順について説明する。また、シミュレーションモデルの入力と出力における不確実性とランダム性が果たす重

要な役割についても解説する。

1.3.1　シミュレーションにおけるランダム性と確率変数

シミュレーションモデリングには確定的な値のみを用いる例もあるが、モデル化の対象となるシステムの多くはランダム性を含んでいるため、シミュレーションモデルにもそれが組み込まれている。典型的なランダム性を持つ要素には、処理時間、サービス時間、顧客やエンティティの到着時刻、移動時間、機械やリソースの故障・修理などがある。

たとえば、深夜にファストフードのドライブスルーに向かう場合を考えてみよう。到着までの所要時間、待っている顧客の数、注文から食事を受け取るまでの時間など、これらの値を正確に予測することはできない。過去の経験や知識から推測は可能でも、確実な予測は不可能である。このような確率的な値に対して確定的な推定値を用いると、妥当性を欠いた（多くの場合、過度に楽観的な）性能予測となってしまう。

標準的な解析モデルでは、こうしたランダムな要素の組み込みが困難または不可能な場合もあるが、シミュレーションを用いれば容易に実現できる。実際、シミュレーションがモデリングや解析のツールとして広く支持されている大きな理由は、確率的なふるまいを簡単に組み込める点にある。これは本書全体を通じての基本的なテーマとなっている。

シミュレーションモデルにおけるランダム性は、**確率変数**を用いて表現される。そのため、確率変数の理解と活用は、シミュレーションモデリングと解析の基本となる（Ross（2010）、Nelson（1995）を参照）。確率変数のもっとも基本的な特徴は、実験の結果によって決定される値を返す関数であるということだ。つまり、実験を実施するまで、その値を知ることはできない。シミュレーションにおいて、実験とは任意の入力群によるシミュレーションモデルの実行を指す。

確率変数Xの確率的なふるまいは、確率分布関数（**累積分布関数**（CDF）とも呼ばれる）$F(x) = Pr(X \leq x)$で表される。この関数は、確率変数Xが値x以下となる確率を示す。離散確率変数の場合は確率質量関数$p(x_i)$を、連続確率変数の場合は確率密度関数$f(x)$を評価する（表 1.1）。

表 1.1　確率変数に対する確率質量関数（PMF）と確率密度関数（PDF）

離散確率変数	連続確率変数
$p(x_i) = Pr(X = x_i)$ $F(x) = \sum_{\substack{\forall i \ni \\ x_i \leq x}} p(x_i)$	$f(x)$は以下の性質を持つ： 1.　$f(x) \geq 0 \quad \forall \quad$ 実数x 2.　$\int_{-\infty}^{\infty} f(x)dx = 1$ 3.　$Pr(a \leq x \leq b) = \int_a^b f(x)dx$

確率変数Xを分類すると、期待値（$E[X]$）や分散（$Var[X]$）、分位数、対称性／歪度といった確率分布の特徴を測定できる。ただし、多くの場合、母集団のパラメータを直接得ることは難しいため、標本平均$\overline{X}$や標本分散$S^2(X)$などの標本統計量

に頼ることになる。これらの推定には適切なサンプルサイズの決定が重要である。シミュレーション分析の利点は、他の実験手法と異なり、必要に応じてサンプルサイズを調整できることである。

シミュレーションは、システムを評価するために入力と出力を必要とする。シミュレーションの**入力側**では、確率変数を特定し、対応する分布からサンプルを生成する。**出力側**では、生成された観測値に基づいて、分布の特性（平均、分散、パーセンタイルなど）を分析する。例として、小規模な医療クリニックのモデルを考えてみよう。このシステムへの入力は、患者の到着時刻、介護士の診察および治療時間であり、これらはすべて確率変数である（図 1.1）。

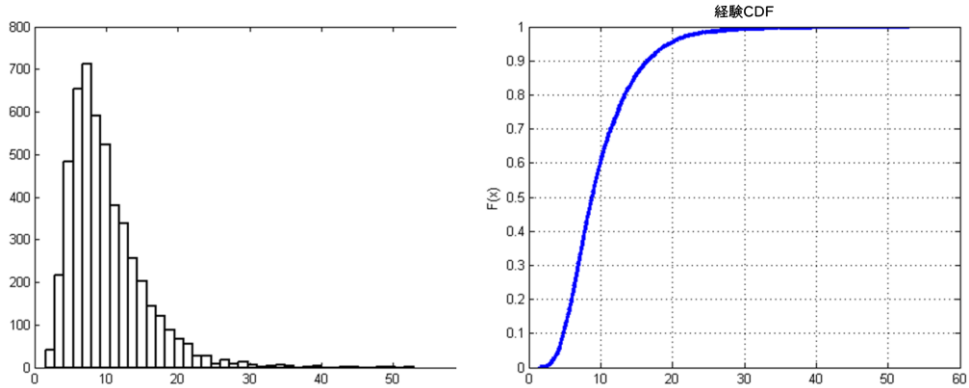


図 1.1　患者の治療時間のサンプルと対応する経験 CDF

システムをシミュレートするには、これらの確率変数を理解し、その観測値を入力として生成する必要がある。多くの場合、入力確率変数の特定には**実際の**システムから得られたデータを活用できる。典型的な出力には、患者の待ち時間、システム内時間、介護士および施設の利用率などがある。シミュレーションモデルはこれらの確率変数の観測値を生成し、モデルの実行を通じて、関心のある出力を特定する。以下の節では、一般的なシミュレーションプロセスにおける入力および出力の分析について説明する。

1.3.2　シミュレーションプロセス

基本的なシミュレーションプロセスを図 1.2 に示す。このプロセスは厳密な順序で進むわけではなく、通常は反復的に行われることに注意が必要である。以降の項で、これらの概念について概説し、本書を通じてより詳しく説明する。

概念設計では、モデル化対象のシステムと、シミュレーションモデル構築のための基本的なモデリング手法を詳細に理解する必要がある。概念設計は、紙とペン、ホワイトボード、あるいは自由な発想を促進する類似のコラボレーションスペースで行われる。これにより、シミュレーションソフトウェアの制約に縛られることを避けられる。

概念設計には明確に定義されたプロセスや方法論が理想的だが、実際にはそう単純ではない。むしろ、プロジェクトの計画は、問題とそのモデリング手法について詳細を検討し議論する非公式なプロセスとなる。その後、モデル開発者がモデ

リング手法の詳細を体系的に整理し、ソフトウェアの特性に応じた実装方法を決定する。

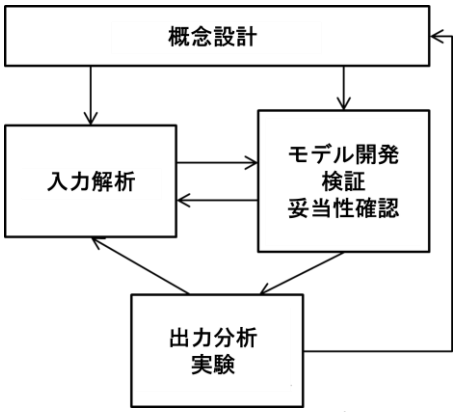


図 1.2　シミュレーションプロセス

シミュレーションモデルは特定の目的のために開発されるため、概念設計の重要な側面として、モデルが必要な疑問に確実に答えられることを保証する必要がある。しかし、シミュレーションの初心者や新人モデル開発者は、概念設計にじゅうぶんな時間を費やさない傾向がある。多くの場合、モデル開発に性急に取り掛かってしまう。概念設計への投資時間が不足すると、プロジェクト全体の完了までに要する時間は増加することになる。

入力解析（6 章で詳述）は、システムへの入力の特定と、入力確率変数および確率過程から観測値を生成するためのアルゴリズムとコンピュータコードの開発を含む。事実上すべての商用シミュレーションソフトウェア（Simio を含む）は、入力観測値を生成する機能を備えている。そのため、入力解析の主な作業は、入力確率変数の特定と、シミュレーションソフトウェアにおける適切な確率分布および確率過程の指定となる。

多くの場合、現実世界のサンプル観測値が存在するため、実データを標準確率分布または経験確率分布に**適合させる**ことが一般的なアプローチとなる(図 1.1)。また、実際の観測データから直接ランダムサンプリングを行う方法もある。入力に関する実データが存在しない場合は、一般的な経験則と感度分析を活用して入力解析を行う。いずれの方法でも、選択した入力がモデル出力にどの程度影響を与えるかを分析することが重要である。6 章では、入力パラメータの活用方法と入力解析への応用について詳しく解説する。

モデル開発は、概念モデルを実行可能なシミュレーションモデルへと変換する**コーディング**プロセスである。「コーディング」という言葉に尻込みする必要はない。現代のシミュレーションパッケージの多くは、洗練されたグラフィカルユーザインタフェースを備えており、モデルの開発や修正を容易にサポートする。実際、コーディングの大半は、モデルの構成要素をドラッグアンドドロップし、ダイアログボックスやプロパティウィンドウに値を入力するだけである。ただし、効果的なモデル開発には、シミュレーションの一般的な方法論と、使用するソフト

ウェアの具体的な操作方法を深く理解することが不可欠だ。

検証および妥当性の確認段階では、モデルの正確性を確認する。**検証**とは、モデルが開発者の意図どおりに動作することを確認するプロセスである。一方、**妥当性の確認**は、モデルが実際のシステムを正確に表現していることを確認するプロセスである。ただし、単純なモデルを除いて、完全な正確性を証明することは不可能だ。そのため、モデル開発者（または顧客）が納得できる程度の証拠を収集することに重点を置く。この現実的なアプローチは、シミュレーションの初心者には戸惑いの種となるかもしれないが、これが実務の実態だ。モデル開発、検証、妥当性の確認については、4章以降で詳しく説明する。

モデルの検証と妥当性が確認されたら、元のシステムに関する情報を収集するためにモデルを実行する。この段階には**出力分析**と**実験**が含まれる。たとえば、患者が介護士の診察を受けるまでの平均待ち時間、待合室の患者数の 90 パーセンタイル値、ドライブスルーの平均待機車両数といったパフォーマンス指標の評価が重要となる。また、患者の平均待ち時間を 30 分以下に抑えるために必要な介護士の人数や、注文から 5 分以内に対応するために必要なキッチンスタッフの人数といった意思決定も関心事となる。出力分析と実験は、これらのパフォーマンス指標の評価と意思決定を可能にする。出力分析では、シミュレーションで生成された個々の観測値を用いて、統計的に妥当な方法で潜在的な確率変数を特定し、モデル化されたシステムを推定する。実験では、モデルの入力と構造を体系的に変更して、代替的なシステム構成を検討する。出力分析については、モデリングに関する4章、5章、9章で詳しく扱う。

1.4　シミュレーションを適用する場合（適用しない場合）

複雑なシステムのシミュレーションが近年注目を集めている。その主な理由の1つは「複雑」という言葉そのものが示している。対象となるシステムが解析モデルで正確に表現できるほどじゅうぶんに単純な場合、シミュレーションは不要であり、使用すべきではない。代わりに、待ち行列理論、確率論、代数学、微積分などの正確な分析手法を用いるべきである。解析的な解を正確に求められる単純なシステムをシミュレートすることは、結果に不確実性というノイズを加えるだけである。

しかし現実には、世界は複雑さに満ちており、前述の「簡単な」モデルの領域を超えることになる。複雑なシステムについて**妥当な**モデルを適切に構築しようとすれば、そのモデル自体も必然的に複雑となり、単純な解析的分析には適さなくなる。正確な解析解を得ることを優先して複雑なシステムを単純化したモデルで表現すると、そのモデルは**過度に**単純化され、システムを適切に表現できているか疑問が生じる。単純なモデルは洗練された正確な解析解を導くが、解析的に扱いやすいモデルを得るために（現実的には疑わしい）多くの簡略化の仮定を置かざるを得ない。それは確かに**モデル**としての解にはなるが、現実をじゅうぶんに表現できていなければ、実際の**問題**の解決にはならない。

モデルが**どの程度**非現実的かを測定するのは困難だ。そもそもその問いの意味

自体が明確ではない。ただし、最終的に解析解を得る必要がなければ、システムを適切に模擬するために必要な複雑さを持つモデルを構築できる。解析的に扱いやすい簡潔なモデルが使えない場合は、複雑なシステムをシミュレーションで模擬する必要がある。その際は、複雑だが現実的なモデルをコンピュータで数値的にシミュレーションし、結果を観察する。より現実的なモデルとするため、一般に入力値は固定値ではなく**確率的**なものとし、確率分布から「抽出」できるようにする。これにより、シミュレーションモデルの結果も確率的となり、不確実性を含むものとなる。

シミュレーションの出力における不確実性や精度の低さは確かに欠点である。しかし、この不正確さの程度を測定することは容易であり、結果が許容できないほど不正確な場合は、単にコンピュータシミュレーションの実行時間を増やすことで対処できる。ほとんどの統計的サンプリングの実験と異なり、シミュレーションでは**ランダム性**と反復実行回数を完全に制御でき、必要な精度を得るためにこの利点を活用できる。シミュレーションの実用性において、計算時間は確かに制約となる。ただし、現在では高速なマルチプロセッサコンピュータやクラウドコンピューティングを容易に利用でき、不正確さを測定可能かつ許容可能な水準まで低減し、妥当な結果を得るための反復実行が可能である。

これまでシミュレーションは、「他のすべての手段が失敗したとき」にのみ選択される「最後の手段」として扱われることがあった（Wagner 1969、pp. 887-890）。確かに、**妥当で**解析的に扱いやすいモデルが利用可能な場合、シミュレーションを用いるべきではない。しかし、実際のシステムの多く（おそらく大半）は非常に複雑で、規則的なふるまいを示さないため、信頼できる妥当性を持つ解析的モデルの構築と分析は困難である。本書は、誤ったモデルに対して正確な解析を行い、その誤差さえ定量化できない解を得るよりも、適切なモデルをシミュレートし、不正確さを客観的に測定・削減できる近似解を得るほうが望ましいと考える。前者の分析は、単なる不正確さ以上に問題がある。

さて、本書の例および図は、Simio のバージョン 17 で作成されている（他のバージョンの Simio を利用している場合は、本書 Web サイトの補足情報を参照してほしい）。Simio の各バージョンでは、低レベルの挙動（同時発生イベントの処理順序など）に影響を及ぼす可能性のある変更が含まれることがある。そのため、異なるバージョンでシミュレーションを実行すると、本書と異なる数値結果が得られる場合がある。「どちらの結果が正しいのか」と疑問に思われるかもしれない。本書では、**統計的に**妥当な結果を得る方法と、その結果の解釈方法について解説する。バージョン間のまれなバグを除き、すべてのバージョンは、個々の実行で数値が異なっても、同じモデルに対して**統計的に**等価な（妥当な）結果を生成する。

1.5　シミュレーションの成功に必要なスキル

シミュレーションツールの使用法と基礎技術を理解するだけでは、成功は保証されない。シミュレーションプロジェクトを成功に導くには、それ以外にもさまざまな能力が必要となる。シミュレーションに初めて取り組む方々からよく「成

功するにはどうすればよいか」という質問を受ける。単純にいえば「努力を惜しまず、すべてを丁寧に行う」ということだが、この答えだけではじゅうぶんな説明とはならないだろう。以下で、考慮すべき重要な事項を見ていこう。

1.5.1　プロジェクトの目的

多くのプロジェクトは納期の確定から始まるが、必要な成果や遂行方法については漠然としたアイディアしかないことがよくある。シミュレーションプロジェクトで最初に問われるのは「目的は何か？」である。一見、簡単に答えられる自明な質問のように思えるかもしれないが、実際には関係者が答えられないという事態が頻繁に起こる。

あなたの最初の役割は目的を明確にすることである。ただし、目的を決定する前に、利害関係者についてじゅうぶん理解しておく必要がある。**利害関係者**とは、プロジェクトの依頼者、資金提供者、ユーザ、およびプロジェクトから影響を受けるすべての人々を指す。上司は明らかな利害関係者である（学生の場合は、指導教員が確実に利害関係者の1人となる）。ただし、主要な利害関係者を特定するのに時間がかかることもある。利害関係者は多くの場合、異なる（ときには矛盾する）目的を持っているため、慎重な配慮が必要である。

たとえば、ある大企業の生産施設をモデル化し、400万ドルの新しいクレーンが期待する成果（製品処理量の増加、待ち時間の短縮、メンテナンス費用の削減など）を達成できるか評価を依頼されたとする。この場合、想定される利害関係者と、典型的な状況における彼ら／彼女らの目的は以下のとおりである：

- 生産技術部（IE）課長（上司）：彼女はIEが会社に付加価値をもたらしていることを示したいため、劇的なコスト削減や生産性向上を実証することを望んでいる。また、社内に作成したモデルを展開できるよう、魅力的な3Dアニメーションを期待している。
- 生産管理部課長：彼は新しいクレーンの購入が生産目標達成への唯一の道だと確信しており、モデルでそれを証明し、キーパーソンを説得する材料にしたいと考えている。
- 生産管理部長：彼は長年の生産担当経験から、「シミュレーション」が実際の利益につながるという確信が持てない。社内の力学から、この取り組みへの期待は低く、プロジェクトの失敗を予想している（密かにそれを望んでいる）。
- 財務部長：彼女はクレーン購入への投資と導入後の生産性に強い懸念を抱いている。前回の幹部会議では、客観的な分析のため、シミュレーションではなく実地での試運転を主張した。
- 生産ラインリーダー：彼女は15年間材料運搬を担当しており、より安価で効果的な生産性向上の方法を知っている。この情報を共有したいと考えている。
- 材料運搬作業員：勤務時間の大半を材料運搬に費やしており、新しいクレーン導入による人員削減を懸念している。そのため、クレーン導入が現場を悪化させるとの見解をあなたに伝え、納得させようと努めている。
- 技術部課長：彼はスタッフの負担増を懸念し、できるだけ関与を避けたいと考

えている。ただし、クレーン導入が決まった場合の具体的な構成と運用方法について、明確なアイディアを持っている。

このシナリオは、複数の実例を組み合わせて作成したものである。小規模なプロジェクトや中小企業では利害関係者の数は少なくなるが、基本原理は変わらない。利害関係者の目的や動機が相反することは一般的だ。各利害関係者はプロジェクトに関する貴重な情報を持っているが、その情報を評価する際には、プロジェクトに対する彼らのバイアスや参加動機を考慮することが重要である。

利害関係者の全体像が見えてきたところで、各関係者がプロジェクトの目的をどのように捉え、どのように貢献しようとしているか、そして自身の目的をどのように優先しようとしているのかを考察する。プロジェクトの主要な目的を明確にするために、以下の質問を投げかける必要がある：

・何を評価したいのか。あるいは、何を証明したいのか。
・モデルの範囲は何か。システムの各構成要素にどの程度の詳細さを求めるのか。
・もっとも重視する構成要素は何か。重要でない構成要素のうち、近似で済むものはどれか。
・利用可能な入力データは何か。データの質はどの程度か。誰が提供してくれるのか。また、提供時期はいつか。
・何回の実験が必要か。最適解を求める必要があるか。
・アニメーションをどのように活用するのか（妥当性確認用のアニメーションと、取締役会向けプレゼンテーション用のアニメーションでは、まったく異なる目的を持つ）。
・どのような形式の報告を求めるのか（口頭発表、数値、要約、グラフ、文書）。

明確な目的を立てるための効果的な方法の１つは、最終報告書のモックアップを設計することである。これにより、「**この形式でこのような情報を盛り込んだレポートは、あなたのニーズを満たしているか**」と確認できる。最終報告書の形式と内容について利害関係者の合意が得られれば、内容の詳細度は後回しにして、他のモデリング上の課題に集中できる。このプロセスを通じて、潜在的なモデリングの目的も明らかになる。

ときには、プロジェクトに必要な明確さが欠如している場合がある。そのような状況で、成果物やリソース、日時などプロジェクト全体の計画を性急に進めることは、失敗への近道となる。明確さの欠如は、プロジェクトを段階的に進めるべきというシグナルだ。小規模なプロトタイプから始めることで、多くの場合、重要な課題が浮き彫りになる。このプロトタイプ作成の経験を基に、後続フェーズの詳細な計画を立てることができる。この点については、次項で詳しく説明する。

1.5.2　仕様書

「目的地がわからなければ、到着時間も予測できない」
ある大工の格言：「二度測って、一度に切れ（慎重に準備を整えよ）」

これらの教訓に従えば、少なくともプロジェクトの基本的な目的は共有できる

はずだ。では、モデル構築の準備は整ったのだろうか？実はまだ早計である。多くの場合、利害関係者は以下のような具体的な約束を求めるだろう。

・プロジェクトの完了はいつか？
・コストはどのくらいかかるのか？（どの程度のリソースが必要なのか？）
・モデルはどのように簡略化されるのか？（どのシステムの側面が含められるのか？）
・品質はどの程度なのか？（どのように検証し、妥当性を確認するのか？）

これらの質問に対して、信頼できる回答を用意できているだろうか？おそらくまだ難しいはずだ。

もちろん、最悪の（しかし一般的な）ケースは、**利害関係者**がこれらの質問に対する答えをすべて用意し、あなたがそれを実現するために働くような状況である。「5 日後に◯◯について徹底的に検証した分析を行えば、5,000 ドル支払う」という提案を想像してほしい。この提案を受け入れた場合、残業を厭わず取り組んだとしても、たいてい 1、2 週間後には一部しか完成せず、未検証のモデルが残るだけである。約束した報酬は、依頼された成果物を提供できないため、支払われることはない。

顧客にとっては、これらの質問のうち **2 つ**、あるいは **3 つ**程度に答えるだけでじゅうぶんだろう。ただし、それらの回答のうち、少なくとも 1 つか 2 つを調整できる権利を確保する必要がある。納期を守るためにモデル化の範囲を狭めたり、逆にモデル化対象はそのままで納期の延長を求めたりすることもある。あるいは、モデル化範囲の基準を達成し、かつ納期を守るために、リソースとコストの両方を倍増することも考えられる（品質の調整は通常、望ましくない）。

幸運なら、利害関係者は上記 4 つの問に対する答えをすべてあなたに委ねるだろう（もちろん、あなたの提案を拒否する権利を留保しながらだが）。しかし、どのようによい回答を導き出すのだろうか。それには、いつ、どのように、誰が回答するのかを正確に記した文書である**仕様書**を作成するとよい。仕様書に求められる詳細度は、応用対象とプロジェクトのサイズにより異なるが、一般に以下の項目が含まれる：

幸運なら、利害関係者は上記 4 つの問いに対する答えをすべてあなたに委ねるだろう（もちろん、あなたの提案を拒否する権利は留保しつつ）。では、どのように適切な回答を導き出せばよいのだろうか。それには、いつ、どのように、誰が回答するのかを正確に記した文書である**仕様書**を作成するとよい。仕様書に求められる詳細度は、応用対象とプロジェクトの規模により異なるが、一般に以下の項目が含まれる：

1.　はじめに
  a)　シミュレーションの目的：このプロジェクトで期待される成果は何か？高次の目標について検討する。
  b)　利害関係者の特定：モデルの結果に関与する主要な関係者は誰か？他の影響を受ける関係者は誰か？モデルは誰が、どのように使用するのか？使用

者はどのようにモデルを習得するのか？

2. システム概要とモデリング手法：システムの構成要素とそのモデル化手法の概要。以下の要素を含む（ただし、これらに限定されない）：
   a) 設備：各設備について、その動作、段取り、スケジュール、信頼性、その他モデルに影響を与える側面を詳細に記述する。必要に応じてデータ一覧表や図を含める。データが不足する場合は、その対応策を別途検討する。
   b) 製品種：対象製品は何か？製品間の違いと相互関連性は何か？各製品または製品群に必要な詳細度はどの程度か？
   c) 運用：各作業について、設備と同様に、その動作、段取り、スケジュール、信頼性、その他モデルに影響を与える側面を詳細に記述する。必要に応じてデータ一覧表や図を含める。データが不足する場合は、その対応策を別途検討する。
   d) 運搬：内部および外部の運搬について、じゅうぶんな詳細を記述する。
3. 入力データ：モデルに必要な入力データは何か？誰が、いつ、どのような形式でその情報を提供するのか？
4. 出力データ：モデルから生成すべきデータは何か？最終報告書のモックアップを作成し、全関係者の期待を明確化する。
5. プロジェクトの成果物：合意済みのすべての成果について検討する。このリストの完成をもってプロジェクト完了とする。
   a) 文書：提供するモデルの資料、手引き、ユーザマニュアルの種類と詳細度を定める。
   b) ソフトウェアとトレーニング：ユーザによる直接操作を想定する場合、必要なソフトウェア、見積価格への含有、必要なカスタムインタフェースを検討する。また、推奨または提供するプロジェクトや製品のトレーニング内容を明記する。
   c) アニメーション：成果物としてのアニメーションの内容と目的（モデルの妥当性確認、関係者の承認、マーケティング）を定める。2D か 3D か、既存のレイアウトやシンボルの利用可否、利用可能な場合の形式、提供元、提供時期を明確にする。
6. プロジェクトのフェーズ：各フェーズの工数、納期、費用の概算を記載する。
7. 署名：主要な利害関係者の署名欄を設ける。

プロジェクトの開始時には、すぐにモデリングを始めたくなるものだ。時間的な余裕があり、アイディアが湧き出る興奮の瞬間だからである。その衝動を抑えて仕様書を作成するのは確かに難しい。しかし、**仕様書の作成には努力する価値がある**ということを理解してほしい。本節冒頭の引用文を思い出してほしい。目的地とそこに至る道筋を決めるために立ち止まることで、的外れな努力と無駄な時間を避けることができる。

プロジェクト全体の期間のうち、最初の約 10%をプロトタイプと仕様書の作成に充てることを推奨する。これは決して無駄な時間ではない。報告書のデザインなど、計画段階で決定すべき重要なタスクを早期に完了できるからだ。たとえば、

プロジェクトの完了までに 20 日間を見込む場合、約 2 日をこの準備作業に充てるべきである。仮に後になってプロジェクトが 40 日間必要だと判明しても、その時点で慌てて代替案（優先順位の変更、モデル範囲の縮小、リソースの追加など）を検討するよりも、事前に検討しておくほうが賢明である。

1.5.3　プロジェクトの反復

シミュレーションプロジェクトは反復プロセスとして実施するのが最適である。ときには最初のステップから繰り返すこともある。初期段階では、目的を決め、仕様書を作成し、プロトタイプを開発するだけだと考えるかもしれない。しかし、仕様書の作成中に新たな目的が見えてくることがあり、またプロトタイプの開発中には仕様書に追加すべき重要な要素が見つかることもある。

プロジェクトが進むにつれ、この反復アプローチの重要性は増していく。シミュレーションの初心者は、アイディアを思いつくとすぐにモデリングを始め、完成までひたすら機能を追加し続け、最後に**初めて**モデルを実行する傾向がある。しかし、熟練したモデル作成者であっても、最高のツールを使用していても誤りは発生する。「モデルのどこかに誤りがある」という漠然とした状況では、その発見と修正は非常に困難だ。シミュレーション教育の経験から、これは学生にとって特に大きな課題となっている。

経験豊富なモデル作成者は、モデルの小さな部分を構築し、実行・テスト・デバッグして、意図したとおりに動作するかを検証する。その後、この手順を別の小さな部分に対して繰り返す。現実のシステムと比較可能なモデルが完成したら、すぐにその部分が意図したシステムと合致するかを確認する。このプロセスをモデルが完成するまで繰り返す。各ステップで問題が見つかった場合、それは直近に追加した部分に存在する可能性が高いため、発見と修正が容易になる。また、各ステップのモデルを異なる名前（例：MyModelV1、MyModelV2、または日付・時間を付加）で保存すれば、必要に応じて以前のバージョンに戻ることができる。

この反復アプローチの大きな利点は、特に初心者にとって、潜在的な問題を早期に発見できることにある。たとえば、エンティティのグループ化について誤った前提でモデル全体を構築し、最終段階でその誤りに気づいた場合を考えてみよう。この時点では、モデルを根本から変更するための膨大な修正が必要となる。一方、反復的なアプローチであれば、グループ化要素を最初に導入した時点で誤りを発見でき、より適切な方法に容易に修正できただろう。

最後に、反復アプローチのもっとも重要な利点である優先順位づけについて述べる。**各反復では、モデル化に残された部分のうち、もっとも重要な箇所から取り組む。**あらゆるソフトウェア開発では、予定よりも工期が延びる傾向がある。シミュレーションモデルの構築でも同様だ。反復アプローチを取らずに期限を迎え、モデルが未完成の場合、検証も妥当性確認もできず、投じた努力を示すこともできない。しかし反復アプローチでは、期限が来ても、検証と妥当性確認を済ませた、使用可能な部分的モデルが存在する。各反復で優先度の高いタスクに取り組んでいれば、完成部分はプロジェクトの目的の大半を満たすことになる（80 対 20 の法則、またはパレートの法則として知られている）。

プロジェクトや適用対象によって異なる場合もあるが、シミュレーション研究の一般的な手順は以下のとおりである：

1. 高次の目的を定義し、利害関係者を特定する。
2. 仕様書を作成する（詳細な目標、モデル範囲、詳細度、モデリング手法、出力尺度を含む）。併せて最終報告書の設計を行う。
3. プロトタイプを構築する。必要に応じて、手順１および２に戻る。
4. システムの優先度の高い部分をモデル化する、または機能を強化する。その部分を文書化し、検証する。この手順を繰り返す。
5. モデルの入力データを収集し、実装する。
6. モデルの検証と妥当性確認を行う。利害関係者と協働する。必要に応じて、手順４に戻る。
7. 実験を計画し、実施する。利害関係者と協働する。必要に応じて、手順４に戻る。
8. 結果とモデルを文書化する。
9. 結果を報告し、成果を共有する。

反復プロセスの進行中は、**利害関係者と定期的にコミュニケートする**機会を持つことが重要だ。利害関係者は予期せぬ展開を好まない。プロジェクトが当初の想定と異なる結果を示す場合は、その理由をともに検討する。また、スケジュールに遅れが生じた場合は、深刻な問題を防ぐため、早めに利害関係者に報告すべきである。利害関係者は単なるクライアントでも敵対者でもなく、パートナーだ。プロジェクトの最良の成果を得るには、互いに協力が不可欠である。多くの場合、最良の成果は、モデル化される実際のプロセスを解明するためのシステムの詳細な調査から得られる。実際、多くのプロジェクトでは、シミュレーションの「結果」が正式に出る前に、その大半の内容が把握されている。これは、モデル作成者による早期の説明と、利害関係者との密接な協働によって得られる知見のおかげだ。

1.5.4　プロジェクト管理とアジリティ

プロジェクトの成功要因は多岐にわたるが、もっとも重要なものの１つが期限の遵守である。意思決定がすでに行われた後に結果が出るプロジェクトは、ほとんど価値がない。他の重要な成功要因には、コスト、リソース、工数がある。予算超過のプロジェクトは途中で中止される可能性があるため、納期とコストにはじゅうぶんな注意を払う必要がある。これらは、日々のプロジェクト管理の綿密さに大きく依存する。

効果的なプロジェクト管理は、意思決定の指針となる明確な目標と詳細な仕様書の策定から始まる。プロジェクトの進行中は、以下のような重要な判断を行う必要がある：

1. モデルの各部分をどの程度詳細に作り込むべきか？
2. どの程度の入力データを収集する必要があるか？
3. どの出力データを重点的に分析すべきか？
4. モデルの妥当性をいつ確認すべきか？

5.　アニメーションと分析にそれぞれどの程度の時間を配分すべきか？
6.　次のステップは何か？

ほとんどの場合、仕様書に答えが直接的または間接的に示されている。主要な利害関係者の目的とその優先順位をすでに理解しているため、この情報が意思決定の基盤となる。

最優先すべきは仕様書の更新、つまり利害関係者からの新たな要求による**仕様変更**への対応である。「仕様書に含まれていない内容は実装できない」という強硬な姿勢を取ることも可能だが、そのような対応が適切なケースは珍しい。シミュレーションは本質的に探索と学習のプロセスであり、新しい領域の探求や対象システムへの理解が深まるにつれて、研究の課題やアプローチ、範囲が変化するのは自然なことである。これらの変化への対応を拒否することは、シミュレーションの潜在的価値（そしてソリューションプロバイダとしてのあなたの価値）を著しく制限することになる。

もう1つの極端な対応は、「利害関係者は常に正しい」というアプローチを取ることだ。つまり、利害関係者から新しい作業の依頼があれば、無条件に**正しいこと**として受け入れる態度である。この対応を取れば、利害関係者は一時的には満足するかもしれないが、長期的には納期の遅延や、プロジェクトの未完了という結果を招く可能性が高くなる。そうなれば、利害関係者は**非常に**不満を抱くことになるだろう。新しいアイディアばかりを追求していては、優先度の高い基本的な作業を完了させる時間さえ確保できなくなる恐れがある。

重要なのは、これらの変更要求を適切に管理することである。まず利害関係者との率直なコミュニケーションを図り、仕様書の項目と優先順位を見直す。新しい要素を追加する際は、他の部分の調整が必要だ。場合によっては、プロジェクトの期限延長を検討する必要がある。期限延長が不可能な場合は、新規項目と既存の作業の優先順位を慎重に比較検討する。既存の作業は、進捗に余裕がある場合の追加開発項目として「ウィッシュリスト」に移す。あるいは、新規項目自体を「ウィッシュリスト」に移すことも検討すべきである。

本書でいう**アジリティ**とは、変化に対して迅速かつ適切に対応する能力のことである。このアジャイルな能力は、シミュレーションの成功に大きく貢献する。

1.5.5　利害関係者およびシミュレーション担当者の権利章典

本章の締めくくりとして、利害関係者があなたに対して持つ合理的な期待について確認しよう（図1.3）。プロジェクトの有効性と成功を高めるために、これらの期待にじゅうぶんな配慮が必要である。一方で、利害関係者もあなたに対していくつかの責任を負っている（図1.4）。プロジェクト開始前に双方の期待について話し合うことで、コミュニケーションが円滑になり、プロジェクトの成功に向けた基盤を築くことができる。これにより、全員のニーズを満たすWin-Winの関係を構築できる。これらの「権利」は、ブログ『Success in Simulation』（Sturrock 2012b）（www.simio.com/blog）から許可を得て引用している。このブログには成功のためのヒントや興味深いトピックが多数掲載されている。

シミュレーションに対する利害関係者の権利章典

プロジェクトの依頼者、資金提供者、ユーザ、およびシミュレーションプロジェクトとその結果から影響を受けるすべての人々のことを、利害関係者と呼ぶ。あらゆるシミュレーションプロジェクトにおいて、利害関係者は、実際に仕事を担当している人たちからの、合理的な期待を抱いている。ここで保証されるべき、基本的な利害関係者の権利を挙げる。

1. パートナーシップ：モデル作成者は、要求以上の情報を提供できるように努力する。またモデル作成者は、利害関係者が権利関係の問題を決定し、提示されたソリューションを確認し評価する権利を有することを前提に取り組む。
2. 仕様書：プロジェクトの開始時に仕様書が作成され、プロジェクトの目的、納期、データ、責任、報告内容、その他の事柄について、明確に決定する。この仕様書は、プロジェクト全体の羅針盤として活用され、特にトレードオフを考慮しなければならない場合に用いられる。
3. プロトタイプ：極めてシンプルなものを除き、すべてのプロジェクトにおいて、利害関係者とモデル作成者が、プロジェクトの範囲やアプローチ、成果について、意見交換し可視化するために、プロトタイプを作成する。多くの場合、プロトタイプは、仕様書の一部として作成される。
4. 詳細度：決められた目的に対処するために、適切なレベルの詳細さでモデルを作成する。詳細さが過大／過小であると、不完全で誤解を招く、あるいは役に立たないモデルにつながる可能性がある。
5. 段階的アプローチ：プロジェクトはフェーズに分けられ、暫定的な結果を利害関係者と共有する必要がある。これにより、アプローチや詳細度、データ、適時性、その他の分野における問題を、早期に発見し、対処することができる。また、プロジェクトの終了時に残念な驚きを与える機会を減らすことができる。
6. 適時性：意思決定日が明確に判明している場合は、そこで利用可能な結果がその日までに提供される。プロジェクトの完成が遅れている場合は、理由や過失の有無にかかわらず、既存の作業が価値を持ち、効果的な意思決定に貢献するために、モデルの範囲を再定義する。
7. 俊敏性：モデリングは発見のプロセスであり、プロジェクトの過程で新しい方向性が生まれる。詳細度や適時性、および仕様書の他の項目に関する制約を考慮しながら、モデル作成者は、変化するニーズを満たすために適切にプロジェクトの方向を調整する。
8. 妥当性の確認および検証：モデル作成者は、モデルが仕様書の設計に準拠し、実際の運営を適切に表現していることを証明する。精度を確認するための時間が不十分なことが見込まれる場合は、モデリング作業の時間を減らして対応する。
9. アニメーション：あらゆるモデルには、利害関係者との検証やコミュニケーションを支援するため、少なくともシンプルなアニメーションが有用である。
10. 明確で正確な結果：プロジェクトの結果は、利害関係者にとって有用な形式および用語で要約され、表現される。シミュレーションの結果は推定値であるが、利害関係者が結果の精度を理解できるように、適切な分析が加えられる。
11. 文書化：モデルは、当面の目的と長期にわたるモデルの実行可能性の双方をサポートするために、内部・外部に向けて十分に文書化される。
12. 品位：結果と推奨事項は、事実と分析に基づいており、政治、労力、その他の不適切な要因によって影響されない。

図 1.3　シミュレーションに対する利害関係者の権利章典

シミュレーション担当者の権利章典

姉妹編の「シミュレーションに対する利害関係者の権利章典」では、シミュレーションプロジェクトの顧客が抱いている合理的な期待を提示した。しかし、これは一方通行ではない。モデル作成、つまりシミュレーション担当者も、合理的な期待を保持する必要がある。

1. 明確な目的：シミュレーション担当者は、利害関係者が目的を発見し、絞り込むことを助ける。しかし、利害関係者はプロジェクトの目的に明確に合意する必要がある。主要な目的は、プロジェクト全体で一致していなければならない。
2. 利害関係者の参画：プロジェクトの初期段階および全体を通して、システムを熟知している人々が、適切に関与し、協力しなければならない。利害関係者は、進捗状況を評価し、未解決の問題を解決するために、定期的に関与する必要がある。
3. タイムリーなデータ：仕様書には、必要なデータが何であり、それがいつ、誰から提供されるのかを明記しなければならない。遅延、消失、あるいは質の劣るデータは、プロジェクトに悪い影響を与える可能性がある。
4. 経営陣のサポート：シミュレーション担当者の上司は、必要に応じてプロジェクトをサポートするべきである。それは、以下で説明するツールやトレーニングに留まらず、政治や官僚的な組織への対応でシミュレーション担当者のエネルギーが搾り取られないように守らなければならない。
5. 俊敏性のコスト：利害関係者がプロジェクトの変更を要請する場合、納期や詳細度、範囲、プロジェクトのコストのような他の面で、柔軟に対応しなければならない。
6. タイムリーなレビューとフィードバック：中間報告は、適切な人々により、迅速かつ思慮深く検討される必要がある。これにより、有意義なフィードバックを得ることができ、必要な方向性の修正を即座に行うことができる。
7. 合理的期待：利害関係者は、技術やプロジェクトの制約に対する限界を認識し、非現実的な期待を持ってはならない。長時間労働の前提に基づいたプロジェクトは、管理が不十分なプロジェクトである。
8. 「代弁者を追いつめるな」：結果が想定外の、または望ましくない結論を勧めるものであっても、モデル作成者を批判すべきではない。
9. 適切なツール：シミュレーション担当者には、プロジェクトに適したハードウェアおよびソフトウェアを提供するべきである。「最新で最高性能のもの」が常に必要になる訳ではないが、シミュレーション担当者に、最新でない不適切なソフトウェアや非効率なハードウェアで時間を浪費させるべきではない。
10. トレーニングとサポート：シミュレーション担当者は、トレーニングをしないで、慣れていないソフトウェアやアプリケーションを使いこなせると期待してはならない。適切なトレーニングとサポートを提供するべきである。
11. 品位：シミュレーション担当者は、強制されてはならない。プロジェクトを開始する前に、利害関係者が適切な答えを「持っている」場合は、プロジェクトを行う意味がない。それがない場合、分析の客観性を保つために、所望の結果が得られるようにモデルを変更するような強制をしないことを尊重すべきである。
12. 尊敬：優秀なシミュレーション担当者は、簡単な仕事だと思われることでも、それを当然のものとしては受け止めない。シミュレーション担当者がすべてを正しく行った時のみ、プロジェクトは簡単に「見える」だけであり、その偉業を達成するのは極めて困難である。時には、同僚が深夜や週末に一緒に仕事をしていない状況だけで、プロジェクトが簡単に見えることもある。

図 1.4　シミュレーション担当者の権利章典

第 2 章　待ち行列理論の基礎

多くのシステム（すべてではない）は、**待ち行列**システムとして表現できる。たとえば、患者が病院に到着すると（予約なしでランダムに訪れる場合）、通常は列（**待ち行列**）に並んだ後、受付で診察申込書に記入する必要がある（図 2.1）

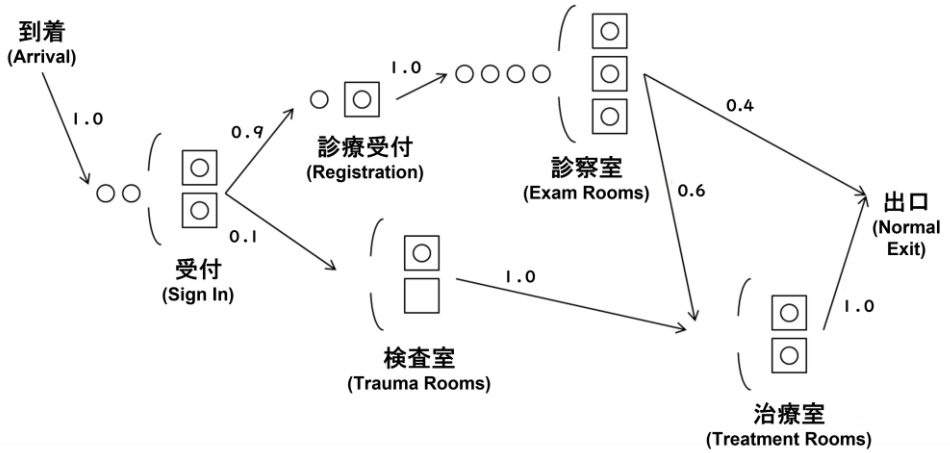


図 2.1　救急病院を表す待ち行列システム

受付後、患者は診療受付か、重症の場合は検査室に向かう。どちらの場所でも待ち行列で待機する可能性がある。診察室に向かった患者は、そのままシステムを離れるか、治療室へ移動してから（おそらく待ち行列に並んで）システムを離れる。検査室へ向かった患者は全員、治療室に移動し（おそらく待ち行列に並び）、その後システムを離れる。このような施設の設計・運営に関する主な課題として、以下が挙げられる：各診療科に必要なスタッフ数とその配置時期、適切な待合室の規模、医師・看護師の診察時間の増減が待合患者に与える影響、患者数が 10% 増加した場合の影響、先着順から重症度順への診察順序の変更による影響。

本章では、待ち行列**理論**（待ち行列**シミュレーション**ではない）の基礎を紹介するにとどめるが、この内容と用語に精通することは、多くのシミュレーションモデルの開発において重要である。初歩的な待ち行列理論から導かれる比較的シンプルな数式は、待ち行列システムのシミュレーションモデルの**検証**に有用である。待ち行列理論のモデルは、シミュレーションの結果と比較するためのベンチマークとしても活用できる。ただし、これはシミュレーションモデルが待ち行列理論の厳格な仮定を満たすほど（おそらく非現実的な）シンプルな場合に**限られる**（たとえば、実際のシステムでは適切でない指数確率分布を仮定する場合など）。簡略化したシミュレーションモデルが待ち行列理論の結果とおおよそ一致した場合、少なくともシミュレーションモデルのロジックが正しいという確信が得られる。後に Simio のシミュレーションモデルを構築する際、シミュレーション「コード」が極めて複雑で検証が困難な場合には、待ち行列理論の考え方を援用する。

待ち行列理論には膨大な研究テーマがある。1909 年に A. K. Erlang がデンマークのコペンハーゲンにおける電話システムの設計・運営に関する研究を始めて

以来、数学的な研究が続けられている（Erlang 1909、Erlang 1917）。待ち行列理論に関する文献は、専門書（たとえば Gross et al. 2008、Kleinrock 1975）だけでなく、確率統計の教科書の一章として（たとえば Ross 2010）、あるいは製造業などの応用事例として（たとえば Askin and Standridge 1993）、幅広く出版されている。「待ち行列理論（queueing theory）」でウェブ検索すると、140 万件以上のページがヒットする。本章は、待ち行列理論の完全な解説を目指すもの**ではない**。むしろ、用語を紹介し、基本的な結果を示すことで、待ち行列理論とシミュレーションの相対的な強みと弱みを明らかにする。そして、後の章で Simio のシミュレーションモデルの検証に用いる特定の数式について理解してほしい。

本章では、読者が以下の確率の基礎知識を習得していることを前提としている：

・確率的な**実験**、**標本空間**、**事象**に関する基本概念
・離散型および連続型の確率変数（RV）
・RV の**分布**：離散型の**確率質量関数**（PMF）、連続型の**確率密度関数**（PDF）、および両方に共通する**累積分布関数**（CDF）
・RV の**期待値**（平均）と**分散**、および PMF、PDF、CDF を用いて RV が特定区間に含まれる確率を計算する方法
・RV 間の**独立性**（またはその欠如）

これらの基礎知識を習得していなければ、本書を読み進める前に、まず基礎的な確率論を勉強してほしい。

本章および本書全体を通じて、指数分布、一様分布、三角分布などの、特定の確率分布を頻繁に用いる。従来の確率論の教科書では、各分布の PMF や PDF の定義、CDF、期待値、分散の導出など、その理論的根拠に多くのページを割いている。しかし、現在ではそうした情報はさまざまな情報源から容易に入手できるため、本書では割愛している（たとえば、Simio のマニュアル（6.1.3 項参照）や、約 100 種の一変量確率分布を網羅したオンラインリソース `https://en.wikipedia.org/wiki/List_of_probability_distributions` などがある）。また、包括的な確率事典も刊行されている（たとえば Johnson et al. 1994、1995、2005 や Evans et al. 2000）。確率分布についての詳細は 6.1.3 項を参照してほしい。

2.1 節では待ち行列システムの一般的な構造、用語、表記法を解説する。2.2 節では待ち行列システムにおける主要な評価指標間の関係性を説明し、2.3 節では具体的な待ち行列システムの出力結果を示す。2.4 節では待ち行列ネットワークの扱いについて概説し、2.5 節では分析ツールとしての待ち行列理論とシミュレーションを比較検討する。両者には相互補完的な長所と短所がある（両方にメリットがあるが、本書ではシミュレーションのほうが有用と考えている）。

2.1　待ち行列システムの構造と用語

待ち行列システムは、エンティティ（顧客、患者、仕事、メッセージなど）が到着し、1 つあるいは複数のステーションで順番にサービスを受け、必要に応じて

待機した後、システムを離れる仕組みである（システムを離れるタイプを**オープン型**、システム内を巡回し続けるタイプを**クローズド型**と呼ぶ）。

図 2.1 の病院は、オープン型待ち行列システムとしてモデル化されている。5 つのサービスステーション（受付、診療受付、検査室、診察室、治療室）があり、各ステーションは**複数窓口の待ち行列ネットワーク**の**ノード**として扱われる。診療受付は単一窓口だが、複数窓口の特殊な場合として考えられる。ノードに複数の並列窓口がある場合（例：診察室の 3 窓口）、各窓口が個別の待ち行列を持つのではなく、1 つの待ち行列からすべての窓口にエンティティが振り分けられる。各窓口は、能力とサービス率について独立していると仮定する。図 2.1 では、弧（矢印）横の数値が患者の移動確率を示している。受付と診察室のように、次の移動先を選択するステーションでは、これらの確率値が必要となる。一方、到着、診療受付、検査室、治療室のように、すべての患者が同じ場所へ移動する場合は、確率は 1.0 となるが、表記の一貫性のために図示している。エンティティが待ち行列内にいるがサービスを受けていない状態を**待機中**と呼ぶ。たとえば、図 2.1 の診察室では、4 人の患者が待機中で、**システム内**に 7 人の患者がおり、そのうち 3 人が**サービス中**（サービスを受けている状態）である。

窓口がエンティティへのサービスを完了したとき、その待ち行列に他のエンティティが存在する場合、次にサービスを受けるエンティティを選択する必要がある。これを**サービス規律**という。もっとも一般的で「公平」とされる規律は、**先着順**（**先入先出**、**FIFO**）である。一方、**後入先出**（**LIFO**）のような異なる規律も存在する。これは、たとえばサラダバーの冷たい皿のように、新しい皿が上に積まれ、顧客が上から取っていく方式である。

また、待ち行列内のエンティティの特性に基づいて**優先順位**をつける規律もある。その一つが**最短ジョブ時間順**（**SJF**）または**最短処理時間順**（**SPT**）である。SJF では、待ち行列内でもっとも処理時間の短いエンティティが次に選ばれる（ただし、処理時間は事前に把握し、各エンティティに割り当てておく必要がある）。この方式により、短時間のジョブを長時間のジョブの後ろに回すことなく処理できるため、全体の平均待ち時間を削減できる。ただし、長時間のジョブが待ち行列の後方に押しやられ続ける可能性があるため、SJF と FIFO のどちらが「よりよい」かは、平均滞在時間と最大滞在時間のどちらを重視するかによる。

SJF の対極にある考え方として、各エンティティに特定の値（例：サービス後の利益）を割り当て、その値が最大のものを次のジョブとして選ぶ方式がある。図 2.1 の病院システムでは、患者を症状の重症度でトリアージし、もっとも深刻な症状の患者から診察する規律となる。同じ重症度の患者間では、FIFO などのタイブレイキングルールを適用する。

待ち行列システムにおける主要な**評価指標**（**出力指標**）は、以下のとおりである：

・**待ち時間**：エンティティが待ち行列で費やす時間（サービス時間は除く）。図 2.1 のような待ち行列ネットワークでは、各ステーションの個別の待ち時間、あるいはシステム全体での累積待ち時間を患者ごとに測定できる。

・**滞在時間**（システム内時間）：待ち時間とサービス時間の合計。待ち行列ネットワークでは、各ステーション単位または入口から出口までの総合的な時間として計測可能である。
・**待ち行列内数**（**待ち行列長**）：サービス中のエンティティを除いた、待ち行列内のエンティティ数。各ステーションまたはシステム全体で測定する。図 2.1 では、受付に２人、診療受付に１人、診察室に４人の患者が待機しており、検査室と治療室には待機患者がいないため、システム全体の待ち行列長は７となる。
・**システム内人数**：待ち行列内数とサービス中のエンティティ数の合計。各ステーションまたはシステム全体で測定する。図 2.1 では、受付４人、診療受付２人、診察室７人、検査室１人、治療室２人で、システム全体では 16 人の患者がいる。
・窓口（または併存する窓口群）の**利用率**：稼働中（サービス提供中）の窓口（サーバ）数の時間平均を、総窓口数で割った値。たとえば、3 窓口ある診察室で、時刻tにおける稼働中（占有中）の窓口数を$B_E(t)$とすると、利用率は、

$$\frac{\int_0^h B_E(t)dt}{3h} \qquad (2.1)$$

となる。ここで、hは観測時間の長さ（あるいは**範囲**）である。

待ち行列理論から得られる結果は、極めてまれな例外を除き、**定常状態**（**長時間実行**または**無限の範囲**）の条件下で導かれる。つまり、実時間またはシミュレーション時間が無限に続くという条件において、**定常状態の平均値**としての結果が得られる。以下に、一般的な（ただし普遍的ではない）評価指標の表記を示す：

・W_q =エンティティの定常状態における平均待ち時間（サービス時間を除く。各ステーションで個別に、またはシステム全体で計測）
・W =エンティティの定常状態における平均滞在時間（サービス時間を含む。各ステーションで個別に、またはシステム全体で計測）
・L_q =定常状態における平均待ち行列長（各ステーションで個別に、またはシステム全体で計測）。これは**時間平均**であり、通常の離散的な数値の平均とは異なる性質を持つ。$L_q(t)$を時刻tにおける待ち行列長とすると、連続する時刻tのすべての値について$L_q(t) \in \{0, 1, 2, \cdots\}$となる。時刻$t$に対して$L_q(t)$をプロットすると、待ち行列へのエンティティの出入りに応じて上下する、0,1,2,…の水準で区分的に一定の曲線となる。有限の時間軸$[0, h]$上において、待ち行列長の時間平均は$\overline{L_q}(h) = \int_0^h L_q(t)\,dt/h$となる。これは、$L_q(t)$の各水準が費やす時間の割合で重みづけられた**加重平均**である。したがって、$L_q = \lim_{h\to\infty} \overline{L_q}(h)$となる。
・L =定常状態における平均システム内人数（各ステーションで個別に、またはシステム全体で計測）。L_qと同様に**時間平均**であり、同じ方法で定義される。$L(t)$は時刻tにおける**システム内**人数（待ち行列内数とサービス中のエンティティ数の合計）を表す。

・ρ =単一窓口または併存する窓口群の定常状態における利用率。通常、ステーションごとに定義される。前述の診察室の例では、式 2.1 を$h \to \infty$とすることで求められる。

数十年にわたる待ち行列理論に関する研究は、主にこれら 5 つの定常状態における平均評価指標の値を導出することに焦点を当ててきたが、これは本書の範囲を超えている。待ち行列理論において特に重要な特性である指数分布は、多くの応用や証明で中心的な役割を果たしている。もっとも基本的な待ち行列モデルでは、到着時間間隔（連続する 2 つのエンティティの到着間隔）とサービス時間の両方が指数分布に従うと仮定される。より発展的なモデルでは、これらの確率分布は指数分布の派生型（アーラン分布や、**互いに独立で同一の分布に従う**（IID）指数確率分布の和からなる RV）に一般化されるか、より一般的な確率分布が採用される。ただ、このような一般化によって、結果の導出と適用はより複雑になる。

最後に、複数窓口の待ち行列ステーションを表す標準的な記号を紹介する。これは、**ケンドールの記号**と呼ばれ、次のように簡潔に表現できる：

$A/B/c/k$

Aは到着プロセス（到着時間間隔の分布）を示し、Bはサービス時間の RV を表す。cは並列同一の窓口数である（図 2.1 では、診察室は$c = 3$、診療受付は$c = 1$）。kはシステムの容量（待ち行列と窓口の容量の合計の上限）を表す。容量に制約がない場合（$k = \infty$）は、$/\infty$の記号を省略するのが一般的である。ケンドールの記号は、入力エンティティの母集団（**呼の母集団**）の性質やサービス規律を示すように拡張できる。

AとBには次のような記号がある。M（マルコフ過程の頭文字）は、到着時間間隔とサービス時間が指数分布に従うことを示す。たとえば、待ち行列モデル$M/M/1$は、到着時間間隔（互いに独立と仮定）とサービス時間が指数分布に従う単一窓口モデルを表す。図 2.1 の診察室は、到着時間間隔とサービス時間が指数分布に従う場合、$M/M/3$と表現できる。アーラン分布（m個の IID 指数分布の和で構成）は、E_mと表記される。たとえば、待ち行列モデル$M/E_3/2/10$は、指数分布の到着時間間隔、フェーズ 3 のアーラン分布のサービス時間、２つの窓口、システム容量 10（待ち行列容量 8）のシステムを表す。一般分布を用いる場合は、到着時間間隔またはサービス時間のいずれかに対してGの記号を使用する。

2.2　リトルの法則とその他の関係式

2.1 節で定義した定常状態における平均評価指標W_q、W、L_qおよびLの間には重要な関係式がいくつか存在する。これらの指標のうち 1 つが判明すれば（あるいは推定できれば）、残りの指標を簡単に計算（あるいは推定）できる。本節では、主に単一の複数窓口待ち行列ステーション（たとえば、図 2.1 の診察室のみ）を考察する。ここで、新たな記号を導入する：λ =**到着率**（到着時間間隔分布の期待値の逆数）、μ =**サービス率**（個別窓口のサービス率であり、複数窓口群全体のものではない。したがって、$\mu = 1/E(S)$となる。ここでSは個別窓口におけるエンテ

ィティのサービス時間を表す確率分布）。

これらの関係式の中でもっとも重要なのが、**リトルの法則**である。この法則は、Little（1961）で初めて導かれ、その後徐々に一般化された（たとえば Stidham 1974）。最近では 50 周年を迎えている（Little 2011）。リトルの法則は、上述の記号を用いて、次のように簡潔に表される：

$$L = \lambda W$$

ここでは、単一の複数窓口待ち行列ステーションを主に扱うが、この法則はより一般的な状況にも適用可能である。リトルの法則の重要な特徴は、**時間平均**（左辺のL）と**エンティティ単位で観測された平均**（右辺のW）を関係づけている点にある。

リトルの法則と同様に、待ち行列ステーションの待ち行列部分（窓口を除く）について、次の関係が成り立つ：

$$L_q = \lambda W_q$$

リトルの法則よりもさらに直感的な関係として、次の式が導かれる：

$$W = W_q + E(S)$$

この式では、サービス時間分布の期待値$E(S)$が既知であると仮定している。これは、システム内の平均滞在時間が、平均待ち時間と平均サービス時間の和として表現できることを示している。この関係式を、$L = \lambda W$および$L_q = \lambda W_q$と組み合わせることで、W_q、W、L_q、Lのいずれか１つの値から他のすべての値を算出できる。たとえば、W_qが既知（または推定可能）な場合、$L = \lambda(W_q + E(S))$が得られる。同様に、Lが既知の場合は、$W_q = L/\lambda - E(S)$と変形できる。

2.3　複数窓口待ち行列ステーションの特性

本節では、複数窓口の待ち行列ステーション（すなわち、図 2.1 の待ち行列ネットワーク全体ではなく、診察室のような個別のステーション）について、文献から得られた公式を紹介する。定常状態における平均評価指標を求めるには、2.2 節で説明したリトルの法則とその他の関係式を活用できる。窓口群の利用率$\rho = \lambda/(c\mu)$は混雑度の指標となる（ここでcは待ち行列ステーションの個別窓口数）。これらの結果はすべて、$\rho < 1$（$\rho \leq 1$ではない）という条件下で成り立つ（待ち行列理論の一般的な前提）。また、定常状態を前提とするため、長期間の運用においてエンティティ数が無限に増加する「爆発的状態」を避ける必要がある。すなわち、窓口群全体のサービス提供速度は、エンティティの到着速度を上回っていなければならない。

本節では 4 つの待ち行列モデルを解説する。他の待ち行列システムにも同様の式や結果が存在するが、詳細は待ち行列理論に関する豊富な文献を参照されたい。これらの式は非常に複雑になることがあり、クローズドな形で解が得られない場合もある。必要な評価指標を含まない式の場合は（以下の 4 番目の例で示すように）、数値計算法や近似手法による解法が必要となる。

2.3.1　$M/M/1$

待ち行列システムの中でもっともシンプルなモデルは、到着時間間隔とサービス時間が指数分布に従う単一窓口システムである。

定常状態の平均システム内人数は$L = \rho/(1-\rho)$で表される。ここで、単一窓口の場合、ρは平均到着率λを平均サービス率μで除したものである。Lは次のように別の形式でも表現できる：

$$L = \frac{\lambda}{1/E(S) - \lambda}$$

Lの値が得られれば、2.2 節の関係式を用いて他の評価指標W_q、W、L_qを算出できる。

2.3.2　$M/M/c$

このモデルも到着時間間隔およびサービス時間は指数確率分布に従うが、単一の待ち行列に対して並列に配置された同一のc個の窓口を持つモデルである。

まず、$p(n)$を定常状態においてシステム内人数がnである確率とする。ここで実際に必要となる唯一の値は、システムが空である場合の定常状態における確率であり、これは次式で表される：

$$p(0) = \frac{1}{\frac{(c\rho)^c}{c!\,(1-\rho)} + \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!}}$$

この式は一見複雑だが、cは有限であるため、スプレッドシートでも計算可能な数式であり、一般に極めて小さな値となる（なお、任意の正の整数jについて、$j! = j \times (j-1) \times (j-2) \times \cdots \times 2 \times 1$を$j$の階乗という。便宜的に 0!=1 と定義される）。2.2 節の関係式から得られる他の評価指標と、$L = L_q + \lambda/\mu$を用いることで、次式が導かれる：

$$L_q = \frac{\rho(c\rho)^c p(0)}{c!\,(1-\rho)^2}$$

$M/M/c$についての上記の式はすべてクローズドな形（すなわち、原則として具体的な数値を代入できる形式）であるが、やや複雑である。特に$p(0)$については、cが非常に小さい場合を除いて、分母の総和は煩雑になる。そこで、本書ではコマンドラインプログラム mmc.exe を提供する（付録 C 参照）。このプログラムを使用すれば、到着率λ、サービス率μ、窓口数cを指定することで、L_qの値を計算できる。mmc.exe を起動するには、まず Windows のコマンドプロンプト（ターミナル）を開く必要がある（Windows の[スタート]-[すべて]、または[スタート]ボタンを右クリックしたメニュー内にある）。次に、システムのルートドライブ（通常は C:）に mmc.exe ファイルを移動し、コマンドプロンプトで cd C:¥と入力する。すると、コマンドプロンプトに C:¥>と表示される。ここで mmc と入力すると、プログラムに与えるパラメータの構文が表示される（図 2.2）。構文に従って mmc の後にλ、μ、cの値を入力すると、$\rho < 1$の安定した待ち行列のパラメータである

ことを前提とした定常状態の待ち行列の評価指標が返される。これらの評価指標はすべて定常状態におけるものであり、入力パラメータの時間単位は任意である（ただし、λおよびμの時間単位は同一でなければならないが、mmc.exe に対してその時間単位を明示する必要はない）。

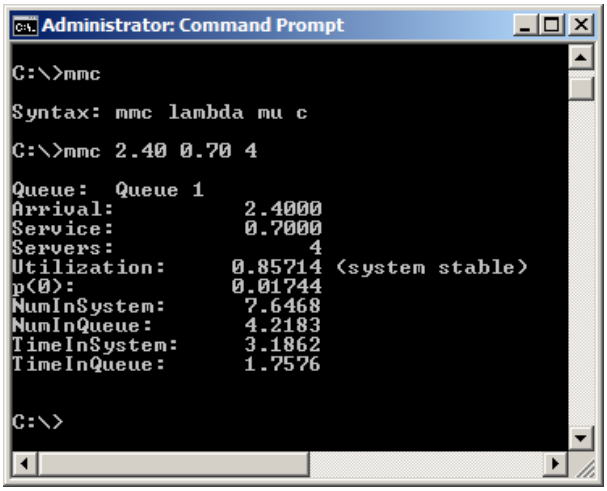


図 2.2　mmc.exe の利用例

Python ユーザなら、https://github.com/auie86/mmc に Python の mmc モジュールとサンプルが公開されている。このモジュールは、単一窓口システムの解析に加え、2.4 節で説明するジャクソンネットワークモデルの解析にも利用できる。

2.3.3　$M/G/1$

このモデルは、窓口が１つ、到着時間間隔が指数分布、サービス時間が一般分布に従うモデルである。$M/M/1$モデルと比べてより現実的なモデルといえる。なぜなら、$M/M/1$モデルではモード（最頻値）が０の指数サービス時間を仮定しているが、これは現実には極めてまれな状況だからである。

ここで、σをサービス時間 RV Sの標準偏差とすると、σの 2 乗はその分散となる。$E(S) = 1/\mu$であるため、次式が導かれる。

$$W_q = \frac{\lambda(\sigma^2 + 1/\mu^2)}{2(1 - \lambda/\mu)}$$

この式は**ポラチェック・ヒンチンの公式**として知られており、リトルの公式や 2.2 節の各関係式と組み合わせることで、定常状態における平均待ち行列長（L）などの評価指標を算出できる。

注目すべき点として、W_qはサービス時間分布の**分散**σ^2に依存するが、分布の平均$1/\mu$には直接依存しない。つまり、σ^2が大きくなるとW_qも大きくなる。これは、まれに発生する極めて長いサービス時間が、単一窓口システム全体を長時間占有することを意味する。このような状況で新規エンティティが到着すると混雑が発生し、それがW_qなどの評価指標に反映される。サービス時間の分散が大きいほど分布の右裾が長くなるため、このような長時間サービスの発生頻度も高くなる。

2.3.4　$G/M/1$

最後の例は、前のモデルとは逆に、到着時間間隔が一般分布に従い（ただし、正の値のみ）、サービス時間が指数分布に従うモデルである。本書では、指数関数的なサービス時間は極端な状況を生むという立場を取る。しかし、この前提条件は、特にその無記憶性により必要とされている。このモデルは単一窓口であり、これまでの 3 つのモデルよりも複雑で分析が難しい。システム内の人数だけでは、将来を確率的に予測するのに不十分である。指数関数以外（つまり無記憶性を持たない）の到着時間間隔では、直近の到着から経過時間も把握する必要があるためである。その結果、式はクローズドで具体的な値を与えた場合にのみ意味を持つものとなる。証明については、Gross et al.（2008）や Ross（2010）を参照してほしい。

ここで、$g(t)$を到着時間間隔分布の密度関数とする。簡単のため、連続分布の正の値のみを考える。これまで同様、μをサービス率（ここでは指数関数）とすると、サービス時間の平均は$1/\mu$となる。zは、次の**積分方程式**を満たす 0 から 1 の間の数とする。

$$z = \int_0^\infty e^{-ut(1-z)} g(t)dt \tag{2.2}$$

以上より、次式を得る。

$$W = \frac{1}{\mu(1-z)}$$

前述のモデルと同様に、この式から 2.2 節の関係式を用いて他の評価指標を算出できる。

式 2.2 を満たすzの値について疑問が生じるかもしれない。残念ながら、式 2.2 をzについて解くには、一般にニュートン・ラフソン法などの反復的な数値計算法が必要となる。さらに、式 2.2 の右辺に求めるべき変数zを含む積分があるため計算過程は複雑になる。

したがって、これは一般的にWの厳密な公式とはいえないが、特定の到着時間分布については解を得ることができる。たとえば、到着時間間隔が$[a, b]$（$0 \le a \le b$）の連続一様分布の場合、確率密度関数は次のようになる。

$$g(t) = \begin{cases} 1/(b-a) & a \le t \le b\text{の場合} \\ 0 & \text{その他の場合} \end{cases}$$

これを式 2.2 に代入すると下式を得る。

$$z = \int_a^b e^{-ut(1-z)} \frac{1}{b-a} dt$$

計算を進めると、式 2.3 が導出される。

$$z = -\frac{1}{\mu(b-a)(1-z)}\left[e^{-u(1-z)b} - e^{-u(1-z)a}\right] \tag{2.3}$$

式 2.3 は数値計算法によって解くことができる。

Microsoft Excel のゴールシーク機能は数値的な求解アルゴリズムを採用しているため、この問題の解法に適している。Excel 2007/2010 では、この機能は[データ]タブの[What-If 分析]内にある。他のバージョンでは異なる場所に配置されている可能性がある。図 2.3 に、Uniform/M/1待ち行列の解析にゴールシーク機能を使用した例を示す。この例は、`Uniform_M_1_queue_numerical_solution.xls` ファイルを用いており、付録 C を参照してダウンロードできる。

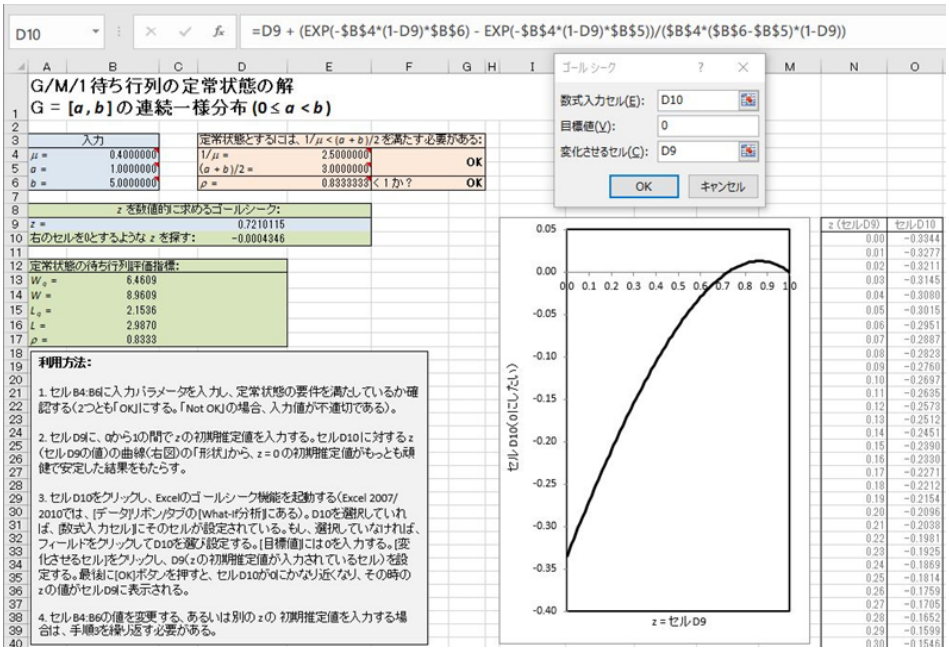


図 2.3　待ち行列評価指標を得るためにゴールシーク機能を用いる Excel シート

セル D10 には式 2.3 が入力されており（図上部の数式バーを参照）、ゴールシーク機能を使用して、セル D10 の値とセル D9 の値（zの値）の差が 0 にもっとも近くなるzの値を求める。zの初期値（推定値）を 0 から 1 の間で設定し、セル D9 に入力する必要がある。セル D9 のzの関数としてセル D10 に曲線が表示されるため、セル D9 を$z = 0$で初期化するともっとも安定した結果が得られる。図 2.3 には、ゴールシークのダイアログ設定と探索完了後の結果が示されている。セル D10 が 0 にじゅうぶん近い近似解zを得ると、定常状態における標準的な待ち行列評価指標が計算される。図 2.3 の下部には、このスプレッドシートモデルの詳細な使用方法が記載されている。

2.4　待ち行列ネットワーク

待ち行列ネットワークは、複数窓口の待ち行列ステーションを表す**ノード**（図

2.1 の救急診療病院における診療受付や治療室など）と、ノード間の経路（システム外からノードへ、またはノードからシステム外への経路）を表す**弧**で構成される。エンティティはシステム外から任意のノードへ入ることができるが、この病院の例では受付ノードに限られる。同様に、エンティティは任意のノードからシステム外へ出ることができるが、この例では診察室と治療室からのみ可能である。システム内では、エンティティは現在のノードを離れる際、図 2.1 に示された分岐確率に従って次のノードへ移動する。たとえば、受付からは 90%の確率で診療受付へ、10%の確率で検査室へ移動する。また、すべてのエンティティが同一のノードへ向かうケースもあり、この例では診療受付と検査室がこれに該当する。

システム外からのすべての到着過程は、互いに独立した指数分布に従うと仮定する（これは**ポアソン過程**と呼ばれ、一定時間内に到着するエンティティ数が離散ポアソン分布に従う）。この例では、システム外からの到着は 1 箇所のみで、すべてのエンティティはそこから受付ノードへ向かう。また、すべてのサービス時間は互いに独立した指数分布に従い、入力到着過程からも独立している。各待ち行列の容量は無限とし、各ノードの利用率（**トラフィック密度**）ρ は 1 未満とする。これにより、システムは定常状態を保つ。これらの条件下で構成されるネットワークは**ジャクソン網**と呼ばれ、多くの理論的知見が得られている（Jackson 1957）。

病院の例では、受付ノードは$M/M/2$待ち行列であり、到着率を$\lambda_{\mathrm{SignIn}}$と表す（つまり到着時間間隔は平均$1/\lambda_{\mathrm{SignIn}}$の IID の指数分布に従う）。安定した$M/M/c$待ち行列ステーションでは、出力も入力率（ここでは$\lambda_{\mathrm{SignIn}}$）と同率のポアソン過程となる点が重要である。つまり、窓口ステーションの出口での確率的なふるまいは、入口で観察されるものと同一となる。この例では、独立な（偏りのある）コイントスにより出力ストリームが分割され、各患者は確率 0.9 で診療受付に、確率 0.1 で検査室に向かう。その結果、2 つの出力ストリームが生じる（診療受付に$0.9\lambda_{\mathrm{SignIn}}$、検査室に$0.1\lambda_{\mathrm{SignIn}}$）。これらは互いに独立したポアソン過程となり、これはポアソン過程の**分流**として知られている。そのため、診療受付は到着率$0.9\lambda_{\mathrm{SignIn}}$の$M/M/1$待ち行列として、検査室は到着率$0.1\lambda_{\mathrm{SignIn}}$の$M/M/2$待ち行列として分析できる。また、ここまで述べた 3 つのノード（受付、診療受付、検査室）は、互いに確率的に独立である。

同様にネットワーク全体を考えていくと、すべてのノードを次のように分析できる：

・受付：到着率$\lambda_{\mathrm{SignIn}}$の$M/M/2$
・診療受付：到着率$0.9\lambda_{\mathrm{SignIn}}$の$M/M/1$
・検査室：到着率$0.1\lambda_{\mathrm{SignIn}}$の$M/M/2$
・診察室：到着率$0.9\lambda_{\mathrm{SignIn}}$の$M/M/3$
・治療室：到着率$(0.9)(0.6)\lambda_{\mathrm{SignIn}} + 0.1\lambda_{\mathrm{SignIn}} = 0.64\lambda_{\mathrm{SignIn}}$の$M/M/2$
　治療室への入力過程は、2 つの独立ポアソン過程の**合流**であり、単純に到着率を合算することで全体の到着率を計算できる（合流した過程もポアソン過程となる）。

このため、2.3 節の$M/M/c$の公式（あるいは mmc.exe）を用いることで、定常状態における平均待ち行列長や平均システム内人数などの評価指標を、各ノードで個別に求めることができる。

同様に、各ノードの利用率（トラフィック密度）も計算できる。上述のリストから各ノードの到着率は既知であり、各ノードの窓口のサービス率がわかれば、その利用率を計算できる。たとえば、診察室の 3 つの窓口のサービス率をμ_{Exam}とすると、診察室の「ローカル」トラフィック密度は、$\rho_{\mathrm{Exam}} = 0.9\lambda_{\mathrm{SignIn}}/(3\mu_{\mathrm{Exam}})$となる。これらの利用率の計算は、指数分布に限らず、任意の到着間隔分布およびサービス時間分布においても妥当である。ただし、到着率とサービス**率**が、それぞれ到着間隔とサービス時間 RV の期待値の逆数として定義されている場合に限る。

前述のとおり、病院の例には存在しなかったが、システム外から直接任意のノードに到着する、より多くのポアソン入力流が存在する可能性もある。この場合、あるノードへの全体的な到着率は、そこに流入する個々の到着率の単純な和となる。また、自己ループが存在する可能性もある。たとえば、治療室からの出力では、80%の患者がシステムを離れる一方、20%の患者が（再治療のために）治療室に戻ることもある。このような場合、$\lambda_{\mathrm{TreatmentRooms}}$について一次方程式を解く必要がある。ただし、これらの分析を妥当なものとするには、各ノードの「ローカル」トラフィック密度が厳密に 1 未満であることを確認する必要がある。これらの特徴を持つ一般的なネットワークについて、次に例を示す。

図 2.4 に示す待ち行列ネットワークを考える。ここでは、定常状態における平均システム内人数（L）、エンティティの平均滞在時間（W）、窓口のトラフィック密度（ρ_{i}）の決定を目的とする。まず、各窓口への到着率を求める必要がある。窓口のトラフィック密度が厳密に 1 未満の場合、退去率は到着率と等しくなる。この条件がすべての窓口で成り立つと仮定し、以下で検証を行う。

到着率は次のように計算される：

$$\lambda_1 = 100$$
$$\lambda_2 = 50$$
$$\lambda_3 = \left(\frac{1}{0.9}\right)(0.3\lambda_1 + 0.2\lambda_2)$$
$$\lambda_4 = 0.7\lambda_1 + 0.4\lambda_5$$
$$\lambda_5 = 0.6\lambda_2$$
$$\lambda_6 = \lambda_4 + 0.9\lambda_3 + 0.6\lambda_5$$

これらの到着率と図 2.4 に示すサービス率、および各窓口で c =1 という仮定について、2.3 節の式を適用し、各窓口jのL_j、W_jおよびρ_{j}を求める（表 2.1）。

ローカルトラフィック密度は厳密に 1 未満であり、これにより、ネットワーク全体の平均システム内人数Lと平均滞在時間Wを以下のように計算できる：

$$L = \sum_{j=1}^{6} L_i$$
$$W = \frac{L}{\lambda_1 + \lambda_2}$$

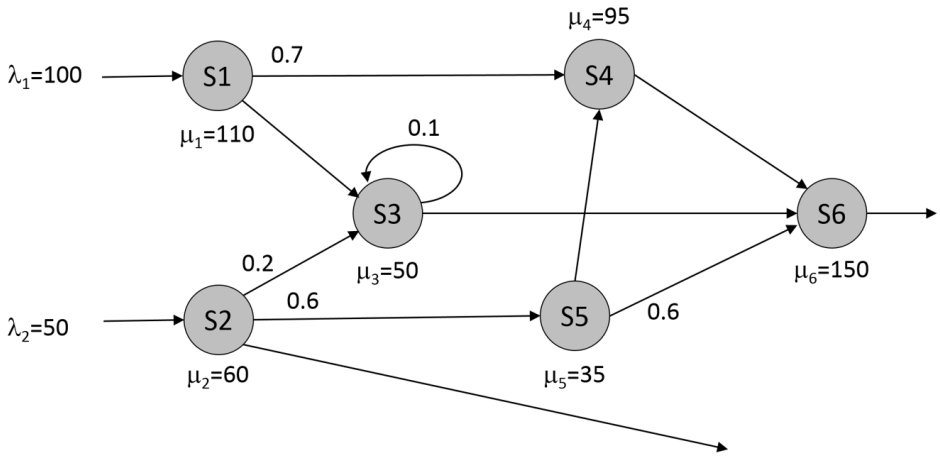


図 2.4　待ち行列ネットワークの例。すべての窓口で c=1 を仮定

表 2.1　図 2.4 のネットワークの各窓口における解

	1	2	3	4	5	6
λ_j	100.00	50.00	44.44	82.00	30.00	140.00
μ_j	110.00	60.00	50.00	95.00	35.00	150.00
ρ_j	0.91	0.83	0.89	0.86	0.86	0.93
L_j	10.00	5.00	8.00	6.31	6.00	14.00
W_j	0.10	0.10	0.18	0.08	0.20	0.10

本書で後述するように、この種の解析は、シミュレート対象のシステムが標準的な待ち行列理論の仮定を満たさない場合でも、モデルの検証において非常に有用である。

2.5　待ち行列理論とシミュレーションの比較

待ち行列の理論的な結果の最大の利点は、統計的な変動に左右されない厳密な解が得られることである（ただし、2.3.4 項の$G/M/1$待ち行列のように数値計算が必要な場合は、丸め誤差が生じる可能性がある）。一方、シミュレーションの結果には統計的な不確実性が**存在する**ため、その不確実性を適切に認識し、測定し、対処する必要がある。

しかし、待ち行列理論には重要な短所がある。それは主に 2.3 節の数式を導出する際に必要な前提条件に関係している。多くの現実的な状況では、これらの前提は当てはまらず、その結果、モデルの正確性や妥当性にどの程度影響するかを判断することが困難である。特に、**指数分布に従うサービス時間**を仮定することは、そのモードが 0 である性質から、多くの場面で非現実的となる。たとえば、空港の保安検査場で、サービス時間が 0 に近い値で頻繁に発生するとは考えにくい。また、これらの理論的結果は定常状態の評価指標にのみ適用可能であり、**短期**の（または**有期**の、もしくは**過渡的**な）評価には適していない。さらに、近似手法

がじゅうぶんな精度を持たない限り、待ち行列理論の結果を任意の到着時間間隔やサービス時間分布に適用することはできない。

これに対し、シミュレーションは短期（または有限）の時間枠を容易に扱うことができる。ただし、定常状態の分析には課題がある。長時間の実行が必要となり、不適切な初期条件による偏りにも注意を払う必要があるためである。シミュレーションでは、研究対象のシステムに最適な到着間隔とサービス時間の分布を自由に選択できる（6.1 および 6.2 節参照）。実データから指数分布が適切と判断される場合を除き、**すべての分布**を指数分布と仮定する必要はない。このため、本書ではシミュレーションモデルを、より現実的で妥当なモデルを構築する手段として位置づけている。特に、モデル構造が複雑で厳密な数理解析が困難な場合や、標準的な待ち行列モデルでは表現できない場合に有効である。唯一の課題は、シミュレーション結果が統計的な推定値であることだ。そのため、正確な結論を導くには、適切な統計手法による分析が不可欠となる。この点については本書で繰り返し取り上げる。図 2.1 の病院例を含む本章の議論は、適切な統計解析手法とともに Simio を用いて実際にシミュレートする。さらに、9 章でより現実的な枠組みへと発展させる。

2.6　問題

1．$M/M/1$待ち行列において、平均到着時間間隔が 1.25 分、平均サービス時間が 1 分の場合、W_q、W、L_q、L、ρを求めなさい。それぞれの指標について、単位と時間を明記して説明しなさい。

2．問題 1 のモデルについて、サービス時間を指数分布から連続一様分布（$a = 0.1$、$b = 1.9$）に変更する。この一様分布の期待値は$(a + b)/2 = 1$で、問題 1 のサービス時間の期待値と同一である。5 つの評価指標を問題 1 と比較し、サービス時間分布の変更による影響を説明しなさい。**ヒント**：区間$[a, b]$の連続一様分布の標準偏差は$\sqrt{(b - a)^2/12}$である。

3．問題 1 のモデルについて、サービス時間を三角分布（最小値$a = 0.1$、最大値$b = 1.9$、最頻値$m = 1.0$）に変更した場合の 5 つの評価指標を、問題 1 および問題 2 の結果と比較しなさい。**ヒント**：三角分布（$a < m < b$）の期待値は$(a + m + b)/3$、標準偏差は$\sqrt{(a^2 + m^2 + b^2 - am - ab - bm)/18}$である。

4．問題 1、問題 2、問題 3 のそれぞれについて、到着率を段階的に増加させた場合の影響を分析しなさい（例：広告やクーポンで集客を増やす窓口 1 の理髪店）。スプレッドシートやプログラムを使用し、元の到着率から 1%ずつ増加させ、システムが非定常（$\rho \geq 1$）になるまで 5 つの評価指標を計算しなさい。到着率の増加率を横軸として評価指標をプロットし、結果を考察しなさい。

5．問題 1 のモデルを$M/M/3$待ち行列に拡張し、平均到着時間間隔 1.25 分、平均サービス時間 3 分の場合の 5 つの評価指標を求めなさい。**ヒント**：スプレッドシートやプログラムを作成する。または `mmc.exe` を使用する。

6．問題 5 について、問題 4 と同様の分析を行いなさい。到着率を 1%ずつ増加させ、元の到着率の 200%（つまり 2 倍）まで分析する。システムの定常性を維持するために必要最小限の窓口を追加し、各シナリオでW_q、W、L_q、L、ρを計算しなさい。

7．2.3 節の$M/M/c$待ち行列に関する式L_qについて、特殊な場合として$M/M/1$を含むことを説明しなさい。

8．図 2.1 の病院において、患者の到着時間間隔が平均 6 分の指数分布に従うとする。各ステーションの窓口数と分岐確率は図 2.1 のとおりである。各ノードのサービス時間は指数分布に従い、平均値（分）は次のとおりである：受付 3、診療受付 5、検査室 90、診察室 16、治療室 15。5 つのステーションそれぞれについて、「ローカル」トラフィック密度 ρ_{Station} を計算しなさい。この病院は運営可能か。つまり、外来患者の来院に適切に対応できるか。運営の可否について理由を述べなさい。また、窓口を 1 つ増設できる場合、5 つのステーションの内、どこに追加すべきか。理由とともに説明しなさい。**ヒント**：スプレッドシートやプログラム、または mmc.exe を使用する。

9．問題 8 について、5 つの各ステーションの W_q、W、L_q、L、ρ を計算し、説明しなさい。問題 8 での窓口追加の判断は、この問題の分析結果でも同じか。理由を述べなさい（なお、来院者は重篤な症状の患者を含む病気の方々であり、工場の部品とは異なる配慮が必要である点に留意すること）。**ヒント**：スプレッドシートやプログラムの使用を推奨する。代替手段として mmc.exe も利用可能。

10．問題 8 および問題 9（窓口追加なしの場合）では、流入率は毎時 10（平均到着間隔 6 分と同値）であった。病院の運営を維持しながら、この流入率をどこまで上げられるか（つまり、どの程度まで外来患者数を増やせるか）。この問題では、必要最小限の計算でじゅうぶんである。厳密な解を求めるには、求解アルゴリズムを用いたプログラムの作成か、Excel のゴールシーク機能の使用を推奨する。

11．問題 8 および問題 9（窓口追加なしの場合）において、予算削減により現在の 10 窓口から 1 つを閉鎖する必要がある。ただし、各ステーションには最低 1 つは窓口が必要である。病院運営への影響が最小となるのは、どの窓口の閉鎖か。さらに閉鎖する窓口を 2 つに増やしても運営は可能か。それ以上の窓口削減については、どのような影響があるか。

第 3 章　シミュレーションの種類

シミュレーションには幅広く多様なトピックと種類があり、その実行方法も多岐にわたる。本章では、その代表的なものを紹介し、関連する用語を解説する。

3.1 節では、複数の観点からシミュレーションを分類する。3.2 節では、静的かつ確率的（別名、**モンテカルロ**）シミュレーションの 3 つの例を紹介する。3.3 節では、専用ソフトウェアを使用しない動的シミュレーションの 2 つの方法（手作業とスプレッドシート）を説明する。3.4 節では、動的シミュレーションのソフトウェア選択肢を解説する。3.4.1 項では、C++、Java、Matlab などの汎用プログラミング言語によるシミュレーションロジックの実装方法を概説し、3.4.2 項では、Simio などの専用シミュレーションソフトウェアについて説明する。

3.1　シミュレーションの分類

シミュレーションは、多様な目的、活動、手法を包含する優れた概念である。しかし、この多様さゆえに「シミュレーション」という言葉が漠然とした、ほとんど意味をなさないものになってしまう可能性がある。本節では、モデル構造に基づいてシミュレーションを分類し、それぞれの例を紹介する。シミュレーションの分類方法は複数存在するが、ここでは**静的/動的**、**連続変化/離散変化**（動的のみ）、**確定的/確率的**という 3 つの分類法を用いる。本書では、主として動的で離散的に変化し、確率的なシミュレーションモデルを扱う。

3.1.1　静的モデルと動的モデル

静的シミュレーションモデルとは、時間の経過がモデルの動作や実行に重要な役割を果たさないモデルである。例として、乱数ジェネレータを用いたギャンブルやロトのシミュレーション、積分値や行列の逆行列の推定、会計上の損益計算書の評価などが挙げられる。モデル内に時間の概念は存在しても、時間経過がモデルの構造や動作に影響を及ぼさない場合、静的である。3.2 節で扱うスプレッドシートを用いた単一期間の在庫シミュレーションがその一例である。

動的シミュレーションモデルでは、（シミュレーション）時間の経過がモデルの構造と動作において本質的かつ明示的な役割を果たす。このため、時間経過の表現なしではモデルの構築も実行も不可能である。待ち行列システムのシミュレーションは、到着やサービス完了などの表現に時間が必要なため、ほぼ必ず動的となる。在庫シミュレーションも、在庫水準や在庫補充方策などと時間との間に論理的な関連がある場合は動的となる。また、輸送やロジスティクスを含むサプライチェーンのモデルは、出発、移動、到着を時間軸上で表現する必要があるため、典型的な動的モデルである。

すべての動的シミュレーションの中核は**シミュレーションクロック**である。これは、シミュレーション上の現在時刻を表す変数で、シミュレーションの進行とともに増加し、モデル全体からアクセス可能なグローバル変数である。シミュレーションクロックの時間単位は、モデル内で一貫性を保つ必要がある。現行のシ

ミュレーションソフトウェアでは、クロックは**実数型**変数（コンピュータ用語では**浮動小数点型**変数）を使用しており、事象の発生時刻を正確に表現できる。少なくとも浮動小数点演算と同程度の精度があり、完璧ではないものの**極めて**正確である。

ただし、古いシミュレーションソフトウェアや特殊な戦闘シミュレータの一部では、モデル作成者が１分などの決められた小さな**時間ステップ**（時間スライスまたは時間増分）を選択しなければならない場合がある。この場合、シミュレーションクロックは時間ステップ単位（例：１分ごと）でしか進まず、各時点でモデル変数の更新が必要な事象が発生したかを評価する。これには２つの問題がある。第一に、毎分ごとの事象発生チェックに多くの計算時間が費やされ、何も起こらなかった場合はその確認が無駄になる。第二に、現実では事象は連続した時間の**あらゆる**時点で発生しうるが、この方法では１分単位に強制されるため、モデルに時間誤差が生じる。この誤差は、時間ステップを秒やミリ秒、ナノ秒にすることで軽減できるが、シミュレーションの実行時間が増大する。そのため、時間ステップ型のモデル（整数型クロック）には重大な欠点があり、ほとんどの場合において利点は少ないといえる。

静的シミュレーションは通常、スプレッドシートで実行され、@RISK や Crystal Ball などの静的シミュレーションアドイン（3.2 節参照）や、C++、Java、Matlab などの汎用プログラミング言語と組み合わせて使用される。一方、動的シミュレーションには、専用に開発されたより強力なソフトウェアが必要となる。3.3.1 項では、本書がもっとも重要と考える動的離散型シミュレーションのロジックについて、スプレッドシートでデータを追跡する方法で説明する。3.3.2 項で述べるように、スプレッドシートでは**極めて**単純な待ち行列モデルを動的シミュレーションとして扱うことはできるが、それ以上の複雑なモデルは実現できない。これまで長年、動的シミュレーションモデルの構築には汎用プログラミング言語が使用されてきたが、このアプローチは退屈で面倒な上、時間がかかり、特に大規模で複雑なモデルではエラーが発生しやすい（ただし、非常に複雑な特殊ロジックを含むモデルの一部分の構築には、現在でも汎用プログラミング言語が使用される）。このような大規模で複雑なモデルは、Simio のような動的シミュレーション専用ソフトウェアを使用するのが最適である。

3.1.2　連続型動的モデルと離散型動的モデル

動的モデルでは、（シミュレーション）時刻の任意の時点で、シミュレートされたシステムの状態を記述する**状態変数**が存在する。待ち行列システムでは、状態変数には待ち行列長、顧客の到着回数、窓口の状態（遊休、稼働、使用不能）などがある。在庫シミュレーションでは、在庫水準や発注量が状態変数となる。

シミュレーションモデル内の状態変数が時間に対して連続的に変化する場合、そのモデルは**連続型**の変化特性を持つ。たとえば、タンク内の水量は、上部から注水され下部から排出されるとき、時間の任意の時点で連続的に変化する（つまり、微小に増減する）。このようなモデルは多くの場合、状態の時間変化率と、その状態変数の水準や他の状態変数との関係を定める微分方程式で記述される。ときに

は、個々の車両による交通流のような本来は離散的な現象が、連続変化型モデルとして近似される場合もある（この例では、交通を液体として扱う）。厳密には離散的な状態を連続的にモデル化した例として、**ランチェスター方程式**がある。ランチェスター方程式では、両軍の戦闘力（兵員数）を時刻tにおける$x(t)$および$y(t)$で表す。ランチェスター方程式にはさまざまな形式があるが、もっともも基本的なものは**近代戦モデル**である。このモデルでは、両軍が相互に射撃を行い、ある時点での両軍の損耗**率**（時間に対する微分）がその時点の両軍の兵員数（水準）に比例する。これは以下の式で表される：

$$\frac{dx(t)}{dt} = -ay(t)$$
$$\frac{dy(t)}{dt} = -bx(t) \qquad (3.1)$$

ここで、aおよびbは正の定数である。式 3.1 の 1 つ目の方程式は、x軍の変化率がy軍の兵員数に比例することを示している。aはy軍の効率性（武器性能）を表す。つまり、定数aが大きくなれば、y軍の戦闘力がより高くなり、結果としてx軍の損耗が激しくなる。同様に、式 3.1 の 2 つ目の方程式はy軍の損耗率を表しており、定数bはx軍の戦闘力を示している。この基本的なランチェスター方程式は解析的に解くことができるため、シミュレーションは不要である。ただし、より複雑なランチェスター方程式では解析的な解法が不可能なため、シミュレーションが必要となる。

状態変数が時間軸上の瞬間的に区分された離散点でのみ変化する場合、その動的モデルは**離散型**の変化特性を持つ。大部分の待ち行列シミュレーションモデルがこれに該当する。待ち行列長や窓口の状態を記述する状態変数は、顧客の到着、サービスの完了、窓口の休憩、故障といった離散的なイベントが発生する**時点でのみ**変化する。離散変化型モデルでは、シミュレートされた時間はこれらの離散的なイベントの時点でのみ進行する。つまり、連続するイベント間では何も発生しないため、その間の状態を観察することはない。そのため、シミュレーションクロック変数は、あるイベントの時点から次のイベントの時点まで直接進む。各時点において、シミュレーションモデルは発生する事象を評価し、状態変数（および 3.3.1 項で説明するイベントカレンダや統計的なトラッキング変数など）に必要な更新を行う。

3.1.3　確定的モデルと確率的モデル

シミュレーションモデルで使われるすべての入力値が固定値で非ランダムな定数の場合、そのモデルは**確定的**である。たとえば、待ち行列システムで表現できる簡単な生産ラインにおいて、各部品のサービス時間と部品間の到着時間間隔が一定（かつ、故障や他のランダムなイベントがない）であれば、確定的である。確定的シミュレーションでは、入力定数を変更しない限り、モデルを何度実行しても同じ結果が得られる。

このようなモデルは現実的ではないように思われるだろう。実際、ほとんどの

シミュレーションモデルは**確率的**である。少なくとも一部の入力値は定数ではなく、確率分布からの無作為**抽選**（無作為**標本**あるいは**実現値**）である。たとえば、生産ラインのサービス時間や到着時間間隔の変動を特徴づける確率分布がこれにあたる。これらの入力確率分布や確率過程を特定することはモデル構築の重要な部分であり、６章で詳しく説明する。

確率モデルは乱数ジェネレータにより駆動するため、１回の実行では**たまたま**生成された１つのサンプルしか得られない。これは、実際の生産ラインの１日の稼働で得られる**当日**の生産量を観測することや、サイコロを１回振って４の目が出ることに似ている。１日の観察結果だけでは、その日の出来事が毎日発生すると結論づけることはできないし、サイコロを１回振って４が出たからといって他の面も４だとはいえない。したがって、結果の傾向について有意義な洞察（生産量が目標に達しているか、サイコロに不正がないかなど）を得るには、生産ラインを複数回稼働させる（あるいはサイコロを何度も振る）必要がある。同様に、確率的シミュレーションモデルから意味のある知見（重要な評価指標の平均、標準偏差、範囲など）を得るには、モデルを複数回実行する必要がある。この点については、本書の４章および５章で詳しく説明する。

確率的シミュレーションの出力に含まれるランダム性や不確実性を避けるため、入力確率分布を単純な平均値で置き換えたくなるかもしれない。この手法は**平均値分析**と呼ばれ、一見合理的に思える。確かにこの方法によってシミュレーション出力は非ランダムで確定的になるが、多くの場合、誤った解釈を導く。不確実性は現実の本質的な要素であり、出力パフォーマンス指標にも当然含まれるべきものだ。さらに重要なことに、入力における変動性は出力指標の**平均値**そのものに大きく影響する。

たとえば、平均１分の指数分布に従う到着時間間隔と、平均 0.9 分の指数分布に従うサービス時間を持つ単一窓口の待ち行列を考えてみよう。待ち行列理論（2 章参照）によれば、定常状態での待ち時間の期待値は 8.1 分となる。しかし、これらの到着時間間隔とサービス時間を単純に平均値（1 分と 0.9 分）で置き換えると、待ち行列はまったく発生せず、待ち時間の平均は 0 分となってしまう。これは正しい値である 8.1 分から大きく外れている。この理由は直感的に理解できる。到着時間間隔とサービス時間のばらつきにより、短い到着間隔が連続して発生したり、長いサービス時間が続いたり（あるいは極端に長いサービス時間が１回発生したり）する可能性がある。これらはいずれも混雑を引き起こし、待ち時間を増加させる。この現象は、Sam Savage が著書『The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty』で指摘している（Savage 2009）。

シミュレーションソフトウェアで確率的モデルを実行すると、毎回同じ数値結果が得られることに初心者は戸惑うことがある。乱数ジェネレータを使用しているのに、なぜ結果が同じになるのかと疑問に思うだろう。しかし、6 章で説明するように、多くの乱数ジェネレータは真にランダムではなく、同じ「無作為な」数列を繰り返し生成するように設計されている。そのため、**同一**のシミュレーション

実験では、モデルを複数回**反復実行**する必要がある。この方法では、ある反復実行から次の反復へと一定の乱数系列を継続して使用する。その結果、各反復実行で異なる（無作為な）出力が得られ、不確実性に関する重要な情報を収集できる。このような乱数の再現性は、デバッグや**分散減少法**による出力結果の精度向上に不可欠である。詳しくは、Banks et al.（2005）や Law（2015）を参照されたい。

3.2　静的かつ確率的なモンテカルロシミュレーション

確率的なコンピュータシミュレーションのもっとも初期の用途は、時間の概念を持たないシステムのモデル化であった。これらのモデルは、地中海のギャンブル天国にちなんで、**モンテカルロ**あるいは**確率的モンテカルロ**と呼ばれている。本節では、4 つの基本的なモデルを解説する。1 つ目（Model 3-1）は 2 つのサイコロの目の合計、2 つ目（Model 3-2）は解析的に解くことが難しい積分値のモンテカルロ推定、3 つ目（Model 3-3）は単一期間の生鮮品在庫問題、4 つ目（Model 3-4）は新製品の生産・販売プロジェクトの期待収益のモデルである。

なお、本書での「Model x-y」の表記は、x 章の y 番目のモデルを示している。付録 C に記載した方法で入手可能な関連ファイルは、「`Model_x_y.`**拡張子**」という命名規則に従っている。拡張子は、Excel ファイルの場合は xls、Simio プロジェクトファイルの場合は spfx である。したがって、これから説明するファイルは `Model_03_01.xls` となる。

本節の例題では、広く普及していることから Microsoft Excel®を使用するが、この種のシミュレーションは C++、Java、Matlab などの汎用プログラミング言語でも容易に実装できる。

3.2.1　Model 3-1：2 つのサイコロ

2 つの偏りのないサイコロを投げ、出た目の和を結果とする。各サイコロは、それぞれ 1/6 の確率で整数 1 から 6 の目を出すため、合計は整数 2 から 12 のいずれかになる。和の確率分布は簡単に求められ、初級の確率論の教科書でも扱われている一般的な例題である。本来ならシミュレーションは不要だが、静的モンテカルロの概念を説明するために、あえてシミュレートしてみることにする。

`Model_03_01.xls`（図 3.1）では、Excel を使用してこのシミュレーションを構築した。2 つの偏りのないサイコロを 50 回投げ、その和を記録する。可能な目の値はセル C4:C9 に、各目の確率はセル A4:A9 に、サイコロがその目の値より小さくなる確率はセル B4:B9 に入力されている（B4:B9 の確率は各サイコロのシミュレーションに役立つ）。列 E は試行回数、列 F と G は 2 つのサイコロの出目、列 H はその合計を示す。セル J4:L4 には 50 回分の和の基本的な要約統計量が表示される。なお、`Model_03_01.xls` を開くと図 3.1 とは異なる結果が表示されるが、これは Excel の乱数ジェネレータ（後述）が**揮発性**を持ち、スプレッドシートの再計算時（ファイルを開く際も含む）に「新しい」乱数を生成するためである。

2つのサイコロの和の静的モンテカルロシミュレーション

P(目 = i)	P(目 < i)	i
0.1667	0.0000	1
0.1667	0.1667	2
0.1667	0.3333	3
0.1667	0.5000	4
0.1667	0.6667	5
0.1667	0.8333	6
1.0000	←検算	

和の平均	和の最小	和の最大
6.7400	3	12

回数	サイコロ1	サイコロ2	和
1	1	6	7
2	4	2	6
3	3	4	7
4	2	5	7
5	1	3	4
6	5	5	10
7	6	3	9
8	6	4	10
9	4	2	6
10	6	2	8
11	3	6	9
12	5	1	6
13	2	5	7
14	2	1	3
15	2	1	3
16	2	1	3
17	2	5	7
18	1	3	4
19	2	2	4
20	5	2	7
21	1	6	7
22	5	6	11
23	3	4	7
24	3	5	8
25	5	1	6
26	1	4	5
27	3	5	8
28	6	6	12
29	2	4	6
30	5	6	11
31	4	6	10
32	4	3	7
33	5	1	6
34	2	5	7
35	2	2	4
36	1	3	4
37	3	2	5
38	1	2	3
39	4	1	5
40	1	2	3
41	2	4	6
42	6	5	11
43	3	2	5
44	3	2	5
45	6	5	11
46	3	4	7
47	1	2	3
48	3	4	7
49	6	6	12
50	3	5	8

図 3.1　２つのサイコロの和に対する静的モンテカルロシミュレーション

このシミュレーションの核心は、セル F4:G53 で各サイコロの目を生成（つまり、シミュレート）することにある。これを２段階で説明しよう。第１段階では、Excel の**乱数ジェネレータ**である RAND()を使用して、0 から 1 の連続一様分布に従う観測値を生成する。「観測値を生成する」と表現したのは、RAND()のアルゴリズムが真の無作為性ではなく、0 と 1 の間で一様に分布するように見える値を生成する特別な数学的手順に基づいているためである（ただし、公式な統計的検定はクリアしている）。これは**擬似乱数**と呼ばれるが、以下では単に**乱数**と呼ぶ。乱数ジェネレータの詳細は 6 章で説明するが、ここでは確立された数学研究に基づく複雑なアルゴリズムだと理解しておけばじゅうぶんである。Excel の RAND()には批判もあるため、ここでは例示目的でのみ使用する。より信頼性の高い選択肢として、Palisade Corporation の@RISK®（www.palisade.com）などの Excel シミュレーションアドインや、Simio のような動的シミュレーションソフトウェアの乱数ジェネレータがある。

第 2 段階では、RAND()から得られる連続一様分布の観測値Uを、偏りのないサイコロの目を表す整数値に変換する。確率変数からの観測値（確率変量とも呼ば

れる）の生成については 6 章で詳しく説明するが、ここでの目的には直感的な方法でじゅうぶんである。連続区間$[0,1]$を、等幅の 6 つの連続部分区間（$[0,1/6),[1/6,2/6,\ldots,[5/6,1]]$に分割する。$U$は連続確率変量であるため、これらの部分区間の端点が閉じているか開いているかは重要ではない。Uは確率 0 でしか境界上に位置しないためである。連続一様分布の性質により、各部分区間に分類される確率は区間の幅$1/6$に等しくなる。そのため、値は第 1 区間なら 1、第 2 区間なら 2 というように、6 つの区間それぞれに対応した値が生成される（**返される**）。これを実装するため、セル F4:G53 に次の式を入力する（すべてのセルで同じ式を使用）：

```
= VLOOKUP(RAND(), $B$4:$C$9, 2, TRUE)
```

Excel 関数 VLOOKUP の第 4 引数に TRUE を設定すると、RAND()（第 1 引数）の値が B4:C9（第 2 引数）の最初の列のいずれかの行に分類され、その行に対応する B4:C9 の 2 番目の列（第 3 引数の 2）から値が返される（$B$4:$C$9 の$記号は、列 F と列 G でこの式をコピーする際に参照元を固定する）。たとえば、RAND() が B4 と B5 の間（0 から 0.1667 の間）にある場合、これは第 1 の部分区間に該当し、C4（値 1）が返される。RAND()が第 1 区間にない場合、VLOOKUP は第 2 の部分区間（B5 と B6 の間、つまり 0.1667 から 0.3333）を確認する。RAND() がこの範囲内にあれば、C5（値 2）が返され、それはおおよそ確率$0.3333-0.1667=1/6$で起こる。この処理は必要に応じて最後の区間（0.8333 から 1 の間）まで続き、その場合は C9（値 6）が返される。VLOOKUP 関数の詳細については、インターネットや Excel のオンラインヘルプで確認できる。

和の範囲は 2 から 12 で、真の期待値が 7 であることから、J4:L4 の要約統計量は妥当な結果を示している。Excel の RAND()関数は揮発性があるため、F9 キーを押すなどしてスプレッドシートが再計算されるたびに、新しい乱数が生成される。試行を繰り返して、サイコロの目の変化や、J4 の和の平均（および K4、L4 の最小値と最大値）の変動を観察してほしい。

このアプローチは、有限個の値（この例では 6 つ）を持つあらゆる離散確率分布に応用できる。また、不等確率分布（たとえば、6 つの目の出る確率が異なるイカサマサイコロ）にも適用可能である。このサイコロ投げシミュレーションの応用例については、問題 1 を参照してほしい。

3.2.2　Model 3-2：モンテカルロ積分

定積分$\int_a^b h(x)dx$（$a<b$）の数値、すなわちaないしb間の$h(x)$下の面積を評価したいとしよう。ただし、$h(x)$の形状からその不定積分を導くのが難しく（あるいは不可能で）、解析的に計算する方法がないものとする。説明を簡単にするため、すべてのbについて$h(x)\geq 0$とするが、この条件が満たされない場合でも本手法は機能する。この問題に対する通常のアプローチは**数値解析**である。すなわち、区間$[a,b]$を垂直に狭い区間に分割し、$h(x)$下の各区間について（長方形や台形、あるいは高度な式によって）近似解を求めることで面積を計算し、最後にすべての面積を合計する。区間を狭めるほど、この数値解析による近似値は真の積分値に近

づく。しかし、この積分あるいは面積の**統計的推定値**を得るために、静的モンテカルロシミュレーションを利用することも可能である。

この方法では、まずaないしbの間で連続一様に分布する確率変量Xを生成する。変量生成の一般的な方法は 6 章で説明するが、ここでは直感的に式を導く。U =RAND()は 0 から 1 の間で連続一様に分布するため、$b-a$をかけてUの範囲を「拡張」すれば、分布の一様性を保ったまま、0 から$b-a$の間に一様に分布するように調整できる（「拡張」という表現は、$b-a>1$を想定しているが、$b-a<1$の場合でも同様に適用できる。後者の場合は「拡張」ではなく「圧縮」が適切な表現となる）。その後、範囲にaを加えれば、幅$b-a$を保ったまま、目的の範囲$[a,b]$へと右側に移動させることができる。この変換は線形であるため、移動後も一様分布は保たれる（ここでは$a>0$を前提に「右側への移動」と表現したが、$a<0$の場合は、aを加えることで「左側へ移動」させる）。

次に、前段落で説明したように、範囲$[a,b]$の連続一様分布から生成された値Xを用いて、その値Xにおいて関数$h(x)$を評価し、最後に$b-a$をかけて確率変量$Y=(b-a)h(X)$を求める。$h(x)$が平坦な（あるいは定数）関数の場合、いずれのXも$[a,b]$の間に落ちるため、Yはaないしbの間で$h(x)$下の面積と等しくなる（これは、推定しようとしている積分$\int_a^b h(x)dx$である）。したがって、長方形の面積を求めることと同等になる。しかし、仮に$h(x)$が平坦な関数であったとしても、はじめからわからないため、$h(x)$は任意の曲線であると仮定したほうが安全だろう。この仮定の下では、長方形の幅$b-a$および高さ$h(X)$を$[a,b]$間の$h(x)$の面積と等しくするXを極めて偶然に生成できない限り、Yは$\int_a^b h(x)dx$と等しくなることはない。ただし、aないしbのどこかに上述の「幸運な」Xが（少なくとも 1 つ）存在することは、**積分の第一平均値定理**として知られている。しかしながら、ここでは極めて稀有な幸運を願うよりも、Xの独立した実現値を大量に生成（あるいは**シミュレート**）して、それぞれのXについて$Y=(b-a)h(X)$、つまり幅$b-a$と高さ$h(X)$の長方形の面積を計算し、最後にこれらのYの面積すべてを平均する。いくつかのYは希望する$\int_a^b h(x)dx$よりも大きくなり、逆にいくつかは小さくなるが、平均すればおおよそ正しくなることは想像できるだろう。実際、基本的な確率論で証明されているとおり、確率変量の**期待値**、つまりYの期待値$E(Y)$は$\int_a^b h(x)dx$に等しい。厳密な証明については問題 3 を参照してほしい。

例として、$h(x)$を平均μ、標準偏差$\sigma>0$の正規確率分布の確率密度関数としよう。これは通常$\varnothing_{\mu,\sigma}(x)$と表記され、次式で与えられる：

$$\varnothing_{\mu,\sigma}(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

ここでxは実数である。$\varnothing_{\mu,\sigma}(x)$に関して、単純な閉じた形の不定積分は存在しない（伝統的な統計書で正規分布表が付録に掲載されていたのはこの理由である）。したがって、これはモンテカルロ積分の適切なテストケースとなり、正規分布表（あるいは Excel の NORMDIST 関数）から非常に正確なベンチマーク値を得るこ

とができる。それでは、$\mu = 5.8$、$\sigma = 2.3$、$a = 4.5$、$b = 6.7$の場合について見てみよう。

Model_03_02.xls（図 3.2）では、Excel を用いてこのシミュレーションを実装した。

正規密度関数の静的モンテカルロ積分

μ:	5.80
σ:	2.30
a:	4.50
b:	6.70

モンテカルロの推定値	正解
0.3682	0.3663

i	X_i	$(b-a)\,\phi_{\mu,\sigma}(X_i)$
1	4.9933	0.3588
2	4.7946	0.3468
3	5.7652	0.3816
4	4.5976	0.3329
5	4.7312	0.3425
6	5.4067	0.3761
7	5.4722	0.3777
8	5.7685	0.3816
9	5.2387	0.3704
10	4.8844	0.3525
11	6.2944	0.3729
12	5.4732	0.3778
13	5.4156	0.3763
14	6.3380	0.3713
15	4.7908	0.3466
16	4.8701	0.3516
17	5.6356	0.3806
18	6.3572	0.3706
19	6.0168	0.3799
20	6.5484	0.3619
21	5.1100	0.3548
22	5.8168	0.3816
23	6.3061	0.3725
24	4.5683	0.3306
25	4.9984	0.3591
26	5.9045	0.3812
27	5.7270	0.3814
28	5.5467	0.3793
29	5.0294	0.3608
30	6.0850	0.3787
31	6.1714	0.3767
32	5.6010	0.3802
33	6.3453	0.3710
34	5.2457	0.3707
35	6.4784	0.3654
36	6.6217	0.3580
37	5.6991	0.3812
38	5.8957	0.3813
39	5.6860	0.3811
40	5.2290	0.3700
41	6.2646	0.3739
42	6.1071	0.3782
43	4.9679	0.3574
44	4.5349	0.3280
45	6.5756	0.3605
46	6.6373	0.3571
47	5.6842	0.3811
48	5.9676	0.3806
49	5.4583	0.3774
50	5.8865	0.3813

図 3.2　正規密度関数の静的モンテカルロ積分

$a = 4.5$ないし$b = 6.7$の間で連続一様に分布するX（$i = 1, 2, \ldots, 50$についてX_iと表記する）の 50 個のサンプルをシミュレートし（列 E）、列 F に$(6.7 - 4.5)\varnothing_{5.8,2.3}(X_i)$の値を 50 個得る。F 列では、密度関数$\varnothing_{5.8,2.3}(X_i)$を評価するために Excel 関数 NORMDIST を使用している。列 E および列 F の式の詳細は、読者自身で確認してほしい。列 F の 50 個の値の平均値はセル H4 に計算され、セル I4 は NORMDIST を用いた正確なベンチマーク値が示されている。F9 キーを押すことで、シミュレーション結果のランダム性を観察できる。

前述の非確率的な数値的領域推定法のほうがより効率的で極めて正確であるため、この例のような 1 次元の積分では、モンテカルロ積分はめったに使用されない。モンテカルロ積分は、より高次元の積分や不規則な形状の積分の推定におい

て多く活用される。この例題のモンテカルロ積分に関する分析および拡張については、問題３および問題４を参照してほしい。

3.2.3　Model 3-3：単一期間在庫の収益

スポーツ用品店を経営していると想定し、地元のスポーツチームがリーグ優勝した場合、チームのロゴ入り帽子の最適な発注数を検討する。過去の経験から、来月（ピーク期）の需要Dは、$a = 1000$と$b = 5000$の間で一様分布すると推定される。帽子は離散的な品目であるため、$a = 1000, 1001, 1002, \cdots, 5000 = b$の帽子がそれぞれ売れる事象の確率は、$1/(b + 1 - a) = 1/(5000 + 1 - 1000) = 1/4001 \approx 0.000249938$となる。帽子はサプライヤから卸売単価$w = \$11.00$で仕入れ、ピーク期には小売価格$r = \$16.95$で販売する。ピーク期に完売した場合は再発注ができないため、追加の売上（および利益）機会を失う。一方、ピーク期後に在庫が残った場合は、在庫処分価格$c = \$6.95$で販売することになり損失が発生する。なお、残った帽子はすべてこの処分価格で売り切れるものとする。これらの条件下で、利益を最大化する最適な発注数を求めたい。

ピーク期の帽子１つ当たりの利益は$r - w = \$16.95 - \$11.00 = \$5.95$であり、需要をすべて満たせるように発注したいところだが、売れ残りが出ると１つ当たり$w - c = \$11.00 - \$6.95 = \$4.05$の損失が生じるため、過不足のない発注が望ましい。実需要が判明してから発注できれば理想的だが、生産リードタイムの制約があり顧客も待ってくれないため、需要実現前の発注が必要となる。これは在庫の陳腐化に関する古典的な問題で、一般に**新聞売り子問題**として知られている。日刊**新聞**が発刊日を過ぎると価値をほぼ失うことから名づけられ、生鮮食品、園芸品、血液製剤などにも応用される。従来の研究の多くは特定ケースの解析解を追求してきたが、本節では汎用的な簡易スプレッドシートモデルを構築し、さまざまな問題設定に対応できるようにする。

はじめに、サプライヤに発注する帽子の数hの関数として、利益の式をたてる必要がある。ここでDをピーク期の需要とすると、Dは整数$\{a = 1000, 1001, 1002, \cdots, b = 5000\}$で一様に分布する離散確率変数となる。ピーク期の小売価格$r = \$16.95$で販売される帽子の数は$\min(D, h)$であり、ピーク後に在庫処分価格$c = \$6.95$で販売される帽子の数は$\max(h - D, 0)$である。これらの式を理解するには、$h$および$D$に具体的な数値を代入するとよい。その際には、売り切れを表す$D \geq h$のケース、およびピーク期に売れ残る$D < h$のケースの双方を検討してほしい。したがって、利益$Z(h)$は(総収入)－(総費用)として、次のように表現される：

$$Z(h) = \underbrace{\underbrace{r\min(D,h)}_{\text{ピーク期収入}} + \underbrace{c\max(h-D,0)}_{\text{在庫処分収入}}}_{\text{総収入}} - \underbrace{wh}_{\text{総費用}} \qquad (3.2)$$

利益$Z(h)$は、式の中に確率変数Dを含むため自身も確率変数となること、また帽子の発注数hにも依存することに注意が必要である。

スプレッドシートに移る前に、需要の確率変数Dの観測値を生成する方法を確認

しよう。ここでは、3.2.1 項で用いた Excel の乱数生成関数 RAND()から始める。この関数により、0 と 1 の間の連続一様分布から独立した観測値が生成される。それでは、$a < b$の整数aおよびbについて（たとえば、$a = 1000$、$b = 5000$）、0 と 1 の間で連続かつ一様に分布する乱数Uを、整数$\{a, a+1, \cdots, b\}$の離散一様乱数Dの観測値へと、どのように変換するのだろうか。6 章で詳述するため、ここでは 3.2.1 項および 3.2.2 項のスプレッドシートシミュレーションと同様に、直感的に理解することとしよう。まず、$a + (b + 1 - a)U$は、aと$b + 1$の間で連続かつ一様に分布する。$U = 0$あるいは$U = 1$とすることで範囲を確認できる。また、これは単なる線形変換であるため、分布の一様性は保持される。次に、これを直近の整数に切り捨てる。この演算は**整数床関数**と呼ばれ、$\lfloor \cdot \rfloor$と表記される。この記号を使うと、先の演算は次の式となる：

$$D = \lfloor a + (b + 1 - a)U \rfloor \qquad (3.3)$$

この式によりDは、aとbの間の両端を含む整数となる（$U = 1$の場合のみ$D = b + 1$となるが、確率 0 の事象である）。また、Dがこれらの整数iのうち任意の 1 つと等しくなる確率は、$i \leqq a + (b + 1 - a) < i + 1$であり、移項すると次式を得る：

$$\frac{i - a}{b + 1 - a} \leq U < \frac{i + 1 - a}{b + 1 - a} \qquad (3.4)$$

ここで、両端が区間$[0, 1]$であることに注意してほしい。Uは 0 と 1 の間に連続一様に分布しているため、式 3.4 における事象の確率は、ちょうどこの区間の幅となり、次のように望む結果が得られる：

$$\frac{i + 1 - a}{b + 1} - \frac{i - a}{b + 1 - a} = \frac{1}{b + 1 - a} = \frac{1}{5000 + 1 - 1000} = \frac{1}{4001} \approx 0.000249938$$

Excel には床関数 INT があるため、式 3.3 におけるピーク期の需要Dの観測値を生成する Excel の数式は次のようになる：

```
= INT(a + (b + 1 - a) * RAND())
```
（3.5）

上式のaおよびbは、数値が入力されているセルへの絶対参照に置き換えられる。

ファイル Model_03_03.xls の最初のシートは、Excel に備わっているツールだけを用いて作成したシミュレーションである。図 3.3 にはシートの大部分が示されているが（紙面の節約のため、行範囲 27:54 および度数分布表は省略）、セルに入力された数式を確認するために、Excel ファイルを開いてみてほしい。

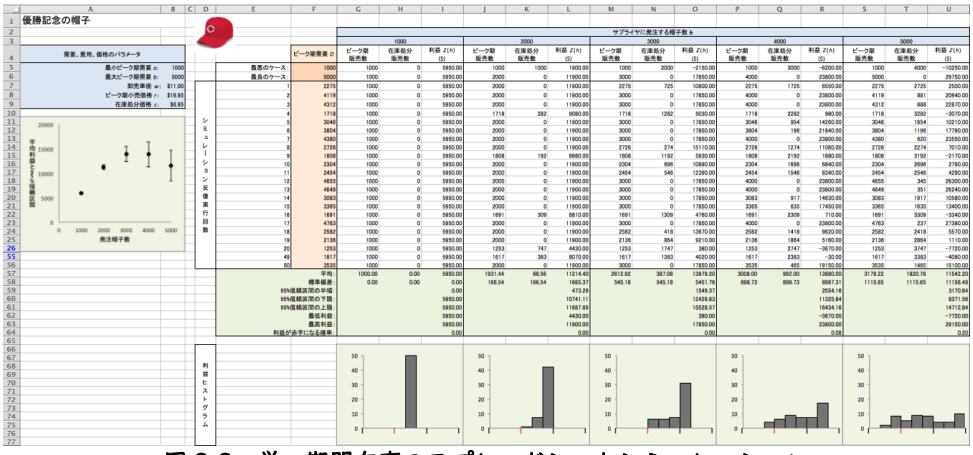


図 3.3　単一期間在庫のスプレッドシートシミュレーション

セル範囲 A4:B9 の青色のセルには、モデルの入力パラメータが配置されている。入力パラメータをこのように独立して設定し、数式で参照することで、「What-if」実験が容易になる。つまり、青色のセルの値を変更するだけで済み、シート全体の数式を修正する必要がない。本シミュレーションでは、50 回の IID 反復実行を行う（各行が 1 回の反復実行に相当）。列 E には反復実行回数を、列 F のオレンジのセルには 50 回分のピーク期需要Dを示している。これらの需要Dは、式 3.5 に基づく以下の数式で生成される：

```
= INT($B$5 + ($B$6 + 1 - $B$5) * RAND())
```

なお、aはセル B5 に、bは B6 に設定されている。F7 に上記の数式を入力し、$記号を使って絶対参照とすることで、残りの反復にコピーできるようにした。これらの式の上（行範囲 5:6）には、ピーク期需要Dの最小値（$a = 1000$）と最大値（$b = 5000$）における、発注帽子数hの各試行値での収益を示している。

このワークシートを開くと、図 3.3 の数値結果と異なる値が表示される。これは、Excel 関数 RAND()がシートの再計算のたびに乱数を**再生成する**ためである。F9 キーを押すと強制的に再計算され、数値とグラフが変化する。50 回の反復実行を行っているが、F9 キーを押すたびに結果は大きくばらつく。これは、適切な実験計画とシミュレーション実験の分析、およびじゅうぶんなサンプルサイズの重要性を示している。

前提条件の下、ピーク期需要Dの範囲から複数のh（1000, 2000, 3000, 4000, 5000）についてシミュレーション実験を行った。上記のhの各値において、50 回の反復実行による収益を計算し、数値および図で統計値を要約した。

$h = 1000$の帽子を発注する場合（列範囲 G:I）、確実に$D \geq 1000$であるため、すべての帽子がピーク期の価格で売り切れる。したがって、収益は$(r - w)D = (\$16.95 - \$11.00) \times 1000 = \$5{,}950$で毎回一定となる。在庫処分価格で販売する帽子は生じず(列 H には 0 が並ぶ)、この収益を下回るリスクもない。しかし、$5,950 という収益を**上回る**機会も失われる。このケースは、$(1 - 1/(5000 + 1 - 1000) \approx 0.999750062$と非常に高い確率で起こる事象であり、$h = 1000$という控えめで保守

的な選択は、潜在的な利益を逃していることになる。

スプレッドシートの右側では、より大きい発注量hのケースを示している。$h=3000$から4000の辺りで平均収益（行 57）は最大となり、$h=5000$に至ると、ピーク期後の売れ残り帽子が増えるため収益は減少する。この傾向は列 A のプロットにも表れており、点が標本平均を、垂直のひげが母平均の 95%信頼区間を示している。hが増加するにつれて、ピーク期の販売量と在庫処分の販売量の両方の変動が大きくなる（ただし両者の合計は常にhである）。その結果、収益の変動も大きくなり（行 58 の標準偏差参照）、期待収益の 95%信頼区間も広がる（行範囲 59:61 および列 A のプロット参照）。また収益のヒストグラムも裾野が広がっていく（ヒストグラムのスケールは統一されている）。これらの結果は、非常に大きな収益を得られる可能性がある一方で、収益が減少するリスクも示している（行 64 およびヒストグラムの赤色の記号で示された損益分岐点より左の領域参照）。以上の分析から、収益の増加と減少の両面のリスクを考察してほしい。

Model_03_03.xls の 2 番目のシートは、同じシミュレーションモデルを Palisade Corporation の@RISK シミュレーションアドインを使用して実装したものである。図 3.4 には@RISK のリボンが表示されているが、読者自身でファイルを開いて確認することを推奨する。

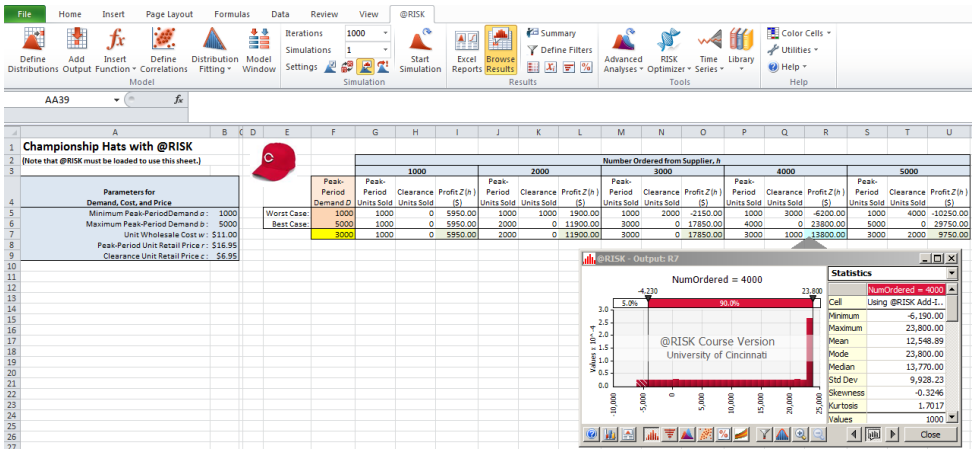


図 3.4　@RISK を用いた単一期間在庫のスプレッドシートシミュレーション

このシートを実行（または正しく表示）するには、まず@RISK を Excel に読み込む必要がある。本書では@RISK の詳細の説明は割愛するため、別の書籍（たとえば Albright et al.（2009））を参照してほしい（@Risk の入手方法と入門動画は付録 C を参照）。ここでは@RISK を利用するシミュレーションの主要なポイントと、Excel の基本機能だけで作成したものとの違いを説明する：

1. 基本的なモデルの作成は必要で、特に入力領域（青色のセル）と出力（このモデルでは収益）に対応する行 7 の数式がもっとも重要である。
2. シミュレーションの反復実行を表す多数の行を個別にコピーする必要はない。
3. @RISK には、このモデルで必要なピーク期需要の離散一様分布を含む、多くの入力確率分布が用意されている。入力分布を設定するには、セル F7 に

@RISK 関数=RiskIntUniform(1000, 5000)を入力する。具体的には、セルをクリックし、@RISK リボンの Define Distributions ボタンから必要な分布を選択して入力パラメータを設定する（このモデルでは 1 つの確率分布でよい）。

4. **反復実行回数**（@RISK では Iterations と呼ばれる）は、@RISK リボンの中央付近のフィールドで入力または選択できる。今回は 1000 回を選択したが、これは Excel の基本機能だけで行う場合の 1000 行分のシミュレーションに相当する。
5. 分析したい出力を表すセル（今回は収益を示す I7、L7、O7、R7、U7）を指定する必要がある。設定は各セルをクリックして@RISK リボンの Add Output ボタンを押すだけである。必要に応じて、出力に Name でラベルをつけることもできる。シミュレーションファイルでは、これらのセルを薄緑色で表示している。
6. シミュレーションを開始するには、@RISK リボンの Iterations フィールドの横にある Start Simulation ボタンをクリックする。
7. @RISK は自動的に出力（平均、信頼区間、ヒストグラムなど）を収集・要約するため、手動での準備は不要である。
8. @RISK は多様な出力表示に対応しており、その中でもっとも有用なのは注釈つきヒストグラムである。これを表示するには、出力セル（図では$h = 4000$の収益を表すセル R7）をクリックし、@RISK の Browse Results ボタンを選択する。垂直の分割バーを左右にドラッグすると、観測値の割合を確認できる（図では損失の割合が 15.6%）。ヒストグラムの右側には、平均値、標準偏差、最小値、最大値などの基本統計量も表示される。

@RISK のようなアドインを活用すれば、スプレッドシートでの静的シミュレーション作業を大幅に効率化できる。さらに、これらのアドインは Excel 関数 RAND() よりも優れた乱数ジェネレータを提供する利点もある。Excel の静的シミュレーションアドインには他にも、Oracle の Crystal Ball™（www.oracle.com/crystalball/）や Frontline Systems の Risk Solver™（www.solver.com/risk-solver-pro）なども選択肢として存在する。

3.2.4 Model 3-4：新製品決定モデル

本節では、新製品の生産・販売プロジェクトの期待収益を分析するモンテカルロモデルを解説する。このモデルには 2 つの実装方法がある。1 つは前述のモデルに類似した Excel モデルで、もう 1 つは Python によるプログラミングアプローチである。この例題は、Anthony Sun による類似の問題を、許可を得て利用している（オリジナルの問題へのリンクは、http://www.geocities.com/WallStreet/9245/vba12.htm であり、現在はアクセス不可だが、参考として記載する）。元の問題は、新製品の製造とマーケティングを分析し、その潜在的な収益を予測するものである。

プロジェクトの総利益は、総売上高から製造コストを差し引いて算出され、以

下の式で表される：

$$TP = (Q \times P) - (Q \times V + F) \qquad (3.6)$$

ここで、TP は総利益、Q は販売量、P は販売価格、V は変動費（限界費用）、F は生産設備を整えるための固定費を表す。このモデルでは簡略化のため、貨幣の時間的価値などの要因は考慮していないが、必要に応じてこれらを追加してモデルの現実性を高めることができる。企業は製品生産における将来の状況を**予測**したいと考えるが、利益を構成する要素には不確実性が伴う。特に、Q、P、V については、以下の確率分布に従う**確率変数**として定義される：

・$Q \sim \mathrm{Uniform}(10000, 15000)$
・$P \sim 1 + \mathrm{Lognormal}(3.75, 0.65)$
・$V \sim 12 + \mathrm{Weibull}(10)$

なお、F は$150,000 で一定である。このプロジェクトの核心的な問いは、新製品の生産・販売を開始すべきかどうかである。Q、P、V が確率変数であるため、総利益 *TP* も確率変数となる。企業の意思決定を支援するために、この確率変数の確率分布を推定し、その分布から利益の平均、標準偏差、パーセンタイルなどのパラメータを算出する。

図 3.5 では、各確率分布から得られた 250,000 個のサンプルのヒストグラムを用いて、Q、P、V の分布を示している。ここで「これらの分布はどのように導き出されたのか」という疑問が生じるだろう。企業が確率分布の適合に使える時系列データを保有している場合もあれば（6.1 節参照）、分布を推定できる専門家や専門チームがいる場合もある。本モデルでは、販売量にじゅうぶんな不確実性を持たせるため、一様分布を採用した。これは、「販売量は 10,000 から 15,000 の間のどこかだろう」という程度の漠然とした推測に基づいている。販売価格は主に市場で決定されるため、これにも不確実性を含める必要がある。ただし、ある程度の中心傾向を持たせ、製品が大成功を収める場合（高価格が実現可能）と完全な失敗に終わる場合（対数正規分布の最小値である 1 に近い価格設定を余儀なくされる）の両方の可能性を考慮した。変動費については、企業は適切な管理が可能と考えているものの、原材料費や労務費による若干の不確実性は避けられない。このケースで使用する対数正規分布とワイブル分布は、一般的な分布形状を示す特徴がある。

前述のモンテカルロの例題と同様に、各確率変数の観測値をサンプリングし、式 3.6 を用いて総利益の確率変数のサンプルを計算した後、このプロセスを繰り返す。このモンテカルロモデルの疑似コードを、アルゴリズム 1 に示す（なお、計算を簡略化するために式 3.6 を並べ替えている）。

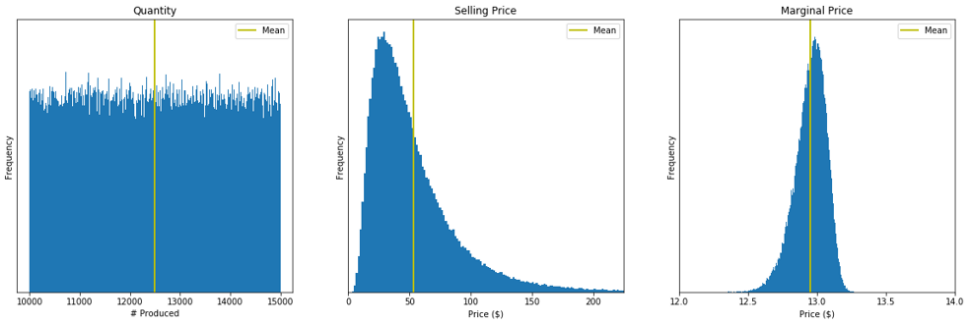


図 3.5　Model 3-4 の Python バージョンで生成された Q、P、V の分布からのサンプル

アルゴリズム I　Sample TP

```
n = 250,000
F = 150,000
tp = []
For{i = 1 to n}
sample P
sample V
sample Q
profit = Q(P－V)－F
tp.append(profit)
ENDFOR
```

（モデル実行時、*TP* はサンプリングされた総利益の観測値のリスト/配列となり、この値を用いて確率分布のパラメータを推定する）

Q、P、V の観測値のサンプリングはアルゴリズム I の重要な要素である。確率変数から観測値を生成できる任意のツールや言語を使用して、このアルゴリズムを実装できる（6.4 節で説明するが、これらの観測値を**確率変量**と呼ぶ）。確率変量の生成に関する一般的な内容は 6.4 節で扱うが、Model 3-4 の Excel と Python の両バージョンでは、表 3.1 に示す数式/関数を使用する（これらの数式の妥当性については、6.3 節および 6.4 節を参照）。

表 3.1　Model 3-4 における Q、P、V の確率変量生成のための数式/関数

	Excel 数式	Python（NumPy）関数
Q	RANDBETWEEN(L, U)	np.random.randint(L, U, n)
P	LOGNORM.INV(RAND(),μ,σ)	np.random.lognormal(μ,σ,n)+1
V	((-(LN(RAND())))^(1/α))+12	np.random.weibull(α,n)+12

L　：　一様分布の下限
U　：　一様分布の上限
μ　：　対数正規分布のパラメータ（正規分布の平均）
σ　：　対数正規分布のパラメータ（正規分布の標準偏差）
α　：　ワイブル分布の形状パラメータ
n　：　生成する観測値の数

図 3.6 は、Model 3-4 の Python バージョンの計算部分と、250,000 回の反復で生成された総利益（TP）のヒストグラムを示している。図 3.7 には、Model 3-4 の Excel バージョンの一部（100,000 回の反復結果）を示している。両バージョンとも、アルゴリズム 1 に基づいて総利益の観測値の配列を生成し、その結果から要約統計量を計算して確率分布を推定するためのヒストグラムを作成している。

```
import numpy as np
import matplotlib.pyplot as plt

#
# Generate solution
# number of replications
observations = 250000
# sample Q values
Q = np.random.randint(10000,15000, observations)
# sample P values
P = np.random.lognormal(3.75, 0.65, observations) + 1
# sample V values
V = np.random.weibull(10, observations) + 12
# fixed cost
F = 150000
# calculate total profit values
TP1 = Q * (P-V) - F
```

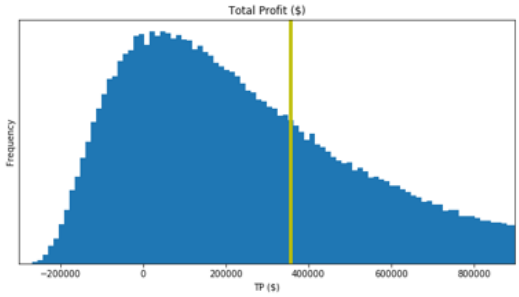
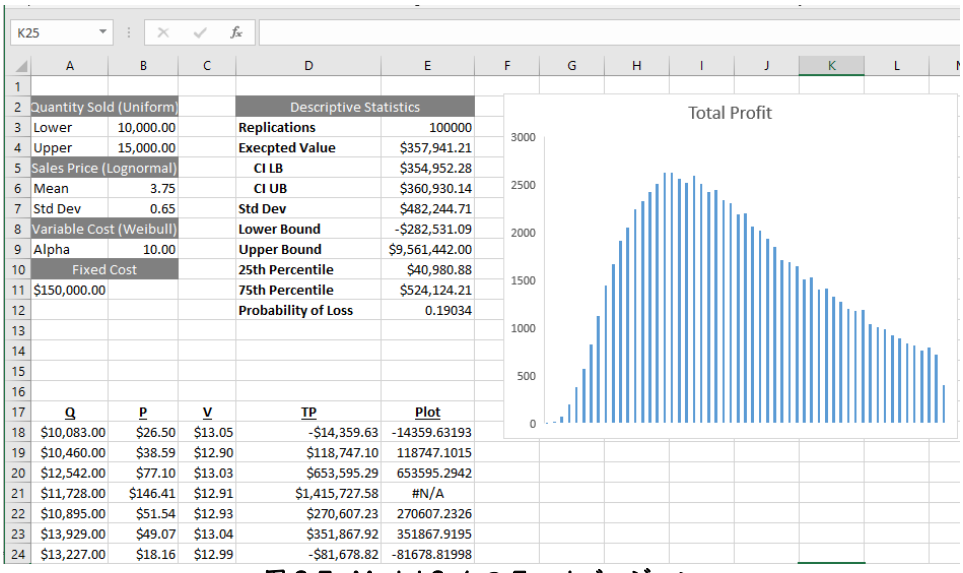


図 3.6　Model 3-4 の Python バージョンの計算部分と総利益のヒストグラム



	A	B	C	D	E
2	Quantity Sold (Uniform)			Descriptive Statistics	
3	Lower	10,000.00		Replications	100000
4	Upper	15,000.00		Execpted Value	$357,941.21
5	Sales Price (Lognormal)			CI LB	$354,952.28
6	Mean	3.75		CI UB	$360,930.14
7	Std Dev	0.65		Std Dev	$482,244.71
8	Variable Cost (Weibull)			Lower Bound	-$282,531.09
9	Alpha	10.00		Upper Bound	$9,561,442.00
10	Fixed Cost			25th Percentile	$40,980.88
11	$150,000.00			75th Percentile	$524,124.21
12				Probability of Loss	0.19034
17	Q	P	V	TP	Plot
18	$10,083.00	$26.50	$13.05	-$14,359.63	-14359.63193
19	$10,460.00	$38.59	$12.90	$118,747.10	118747.1015
20	$12,542.00	$77.10	$13.03	$653,595.29	653595.2942
21	$11,728.00	$146.41	$12.91	$1,415,727.58	#N/A
22	$10,895.00	$51.54	$12.93	$270,607.23	270607.2326
23	$13,929.00	$49.07	$13.04	$351,867.92	351867.9195
24	$13,227.00	$18.16	$12.99	-$81,678.82	-81678.81998

図 3.7　Model 3-4 の Excel バージョン

Model 3-4 の実行結果から、プロジェクトを開始すべきかという当初の問いに答えることができる。記述統計量とヒストグラム（図 3.7 と図 3.6）は重要な洞察を提供している。期待利益は約$360,000 だが、19%の確率で損失が発生する可能性がある。また、標準偏差（$482,000）と第 1・第 3 四分位点（$41,000、$524,000）から、プロジェクト開始に伴う**リスク**の程度も把握できる。4 章では、この状況における**リスク**と**誤差**の概念をより詳しく検討する。最終的な意思決定は、企業のリスク許容度、期待資本利益率、他の投資機会などの要因に左右されるが、このモデルは意思決定者に対して明確な判断材料を提供している。

3.3　専用ソフトウェアを用いない動的シミュレーション

本節では、Simio などの動的シミュレーション専用ソフトウェアを使用せずに、動的シミュレーションを実行する方法を検討する。3.3.1 項では、単一窓口待ち行列の基本的なシミュレーションを手作業で実施する。記録と計算に Excel スプレッドシートを使用するものの、これはマニュアルシミュレーションの「モデル」自体には影響しない。3.3.2 項では、単一窓口待ち行列の**一部**をスプレッドシートシミュレーションとして構築する。この方法では特殊な関係性を活用するが、より複雑な動的待ち行列シミュレーションには適用できないことに留意が必要である。

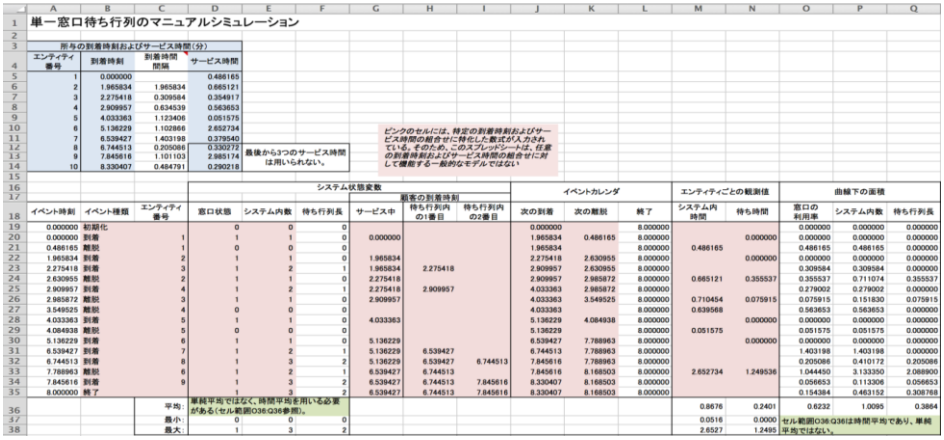
3.3.1　Model 3-5：マニュアル動的シミュレーション

すべての動的離散型シミュレーションは、基本的に同じ仕組みで動作している。本項では、シミュレーションの動作原理を解説する。Simio などの高水準ソフトウェアを使用する場合、このような詳細な手順を実行する必要はないが、その基本概念を理解することは重要である。なお、ここで説明するロジックは、C++、Java、Matlab などの汎用プログラミング言語で動的離散型シミュレーションを実装する際に必要となる。

例として、単一窓口待ち行列システムを考える。**エンティティ**は到着時刻にシステムに入り（最初の到着は時刻 0）、特定のサービス時間を必要とする。時間の単位はすべて分で、待ち行列は先入先出（FIFO）方式である。エンティティは、サービスシステムでは顧客、生産システムでは部品、医療システムでは患者などを表す。シミュレーションは時刻 8 分で終了し、その時点で待ち行列やサービス中のエンティティがあっても終了する。出力として、窓口の利用率（稼働時間の割合）と、エンティティの(a)滞在時間、(b)待ち時間（サービス時間を除く）、(c)システム内数、(d)待ち行列長について、それぞれ平均、最小、最大を算出する。(a)と(b)の平均は単純平均だが、(c)と(d)の平均は**時間平均**である。これは、変数の時系列的な推移を示す曲線下の総面積を、最終シミュレーション時刻 8 で割ったものである。たとえば、時刻tにおける待ち行列長を$L_q(t)$とすると、待ち行列長の時間平均$\int_0^8 L_q(t)dt/8$は、曲線の値が$0,1,2,\cdots$であった時間の割合を重みとする待ち行列長（$0,1,2,\cdots$）の加重平均となる。同様に、時刻tにおける窓口の状態を表す関数$B(t)$（遊休時 0、稼働時 1）と、時刻tにおけるシステム内数$L(t)$（待ち行列内とサービス中のエンティティの合計）から、窓口の利用率$\int_0^8 B(t)dt/8$とシステム内数の時間平均$\int_0^8 L(t)dt/8$が算出される。

図 3.8 に示す `Model_03_05.xls` を使用して、システムの挙動を追跡・計算する（ファイルは付録 C の手順でダウンロード可能）。スプレッドシートには、データと計算式が入力されている。ピンクのセルには、このモデル特有の到着時刻とサービス時刻の数式が含まれている。そのため、このスプレッドシートは任意の到着時刻とサービス時刻に対応する汎用モデルではない。水色のセル範囲 B5:B14 および D5:D14 には、所与の到着時刻とサービス時間が入力されている。セル範囲 C6:C14 には、所与の到着時刻を用いた数式により**到着時間間隔**が計算されている。各行の到着時間間隔は、その行のエンティティの到着時刻から、直前

の行のエンティティの到着時刻を引いた値である（ただし、到着時刻 0 のエンティティ 1 については、到着時間間隔は存在しない）。



単一窓口待ち行列のマニュアルシミュレーション

所与の到着時刻およびサービス時間（分）

エンティティ番号	到着時刻	到着時間間隔	サービス時間
1	0.000000		0.486165
2	1.965834	1.965834	0.665121
3	2.275418	0.309584	0.354917
4	2.909957	0.634539	0.563653
5	4.033363	1.123406	0.051575
6	5.136229	1.102866	2.652734
7	6.539427	1.403198	0.379540
8	6.744513	0.205086	0.330272
9	7.845616	1.101103	2.985174
10	8.330407	0.484791	0.290218

最後から3つのサービス時間は用いられない。

ピンクのセルには、特定の到着時刻およびサービス時間の組合せに特化した数式が入っている。そのため、このスプレッドシートは、任意の到着時刻およびサービス時間の組合せに対して機能する一般的なモデルではない

			システム状態変数			顧客の到着時刻			イベントカレンダ			エンティティごとの観測値		曲線下の面積		
イベント時刻	イベント種類	エンティティ番号	窓口状態	システム内数	待ち行列長	サービス中	待ち行列内の1番目	待ち行列内の2番目	次の到着	次の離脱	終了	システム内時間	待ち時間	窓口の利用率	システム内数	待ち行列長
0.000000	初期化		0	0	0				0.000000		8.000000			0.000000	0.000000	0.000000
0.000000	到着	1	1	1	0	0.000000			1.965834	0.486165	8.000000		0.000000	0.000000	0.000000	0.000000
0.486165	離脱	1	0	0	0				1.965834		8.000000	0.486165		0.486165	0.486165	0.000000
1.965834	到着	2	1	1	0	1.965834			2.275418	2.630955	8.000000		0.000000	0.000000	0.000000	0.000000
2.275418	到着	3	1	2	1	1.965834	2.275418		2.909957	2.630955	8.000000			0.309584	0.309584	0.000000
2.630955	離脱	2	1	1	0	2.275418			2.909957	2.985872	8.000000	0.665121	0.355537	0.355537	0.711074	0.355537
2.909957	到着	4	1	2	1	2.275418	2.909957		4.033363	2.985872	8.000000			0.279002	0.279002	0.000000
2.985872	離脱	3	1	1	0	2.909957			4.033363	3.549525	8.000000	0.710454	0.075915	0.075915	0.151830	0.075915
3.549525	離脱	4	0	0	0				4.033363		8.000000	0.639568		0.563653	0.563653	0.000000
4.033363	到着	5	1	1	0	4.033363			5.136229	4.084938	8.000000		0.000000	0.000000	0.000000	0.000000
4.084938	離脱	5	0	0	0				5.136229		8.000000	0.051575		0.051575	0.051575	0.000000
5.136229	到着	6	1	1	0	5.136229			6.539427	7.788963	8.000000		0.000000	0.000000	0.000000	0.000000
6.539427	到着	7	1	2	1	5.136229	6.539427		6.744513	7.788963	8.000000			1.403198	1.403198	0.000000
6.744513	到着	8	1	3	2	5.136229	6.539427	6.744513	7.845616	7.788963	8.000000			0.205086	0.410172	0.205086
7.788963	離脱	6	1	2	1	6.539427	6.744513		7.845616	8.168503	8.000000	2.652734	1.249536	1.044450	3.133350	2.088900
7.845616	到着	9	1	3	2	6.539427	6.744513	7.845616	8.330407	8.168503	8.000000			0.056653	0.113306	0.056653
8.000000	終了		1	3	2	6.539427	6.744513	7.845616	8.330407	8.168503	8.000000			0.154384	0.463152	0.308768
		平均:	単純平均ではなく、時間平均を用いる必要がある（セル範囲O36:Q36参照）。									0.8676	0.2401	0.6232	1.0095	0.3864
		最小:	0	0	0							0.0516	0.0000	セル範囲O36:Q36は時間平均であり、単純平均ではない。		
		最大:	1	3	2							2.6527	1.2495			

図 3.8　単一窓口待ち行列システムのマニュアルシミュレーション

動的離散型シミュレーションでは、まずシステム状態を変化させる**イベント**を特定する必要がある。例題の単一窓口待ち行列における主要なイベントは、エンティティの**到着**、エンティティのサービス終了と**離脱**、そして**シミュレーションの終了**である（シミュレーションの終了は「人工的な」イベントに見えるかもしれないが、これは一定時間（ここでは 8 分）のシミュレーションを終了させるための手段である）。より複雑なシミュレーションでは、機械の故障と修理完了、患者の症状回復、従業員の休憩、車両の出発と到着などのイベントが発生する。複雑なモデルではイベントの特定が困難な場合があり、別のアプローチが必要になることもある。たとえば、単一窓口待ち行列では、エンティティのサービス開始時点を**考慮することもできる**。ただし、この「サービス開始」イベントは、あるエンティティのサービス終了時に待ち行列に次のエンティティが存在する場合か、新規到着したエンティティが遊休中の窓口で直ちにサービスを受けられる場合にのみ発生する。つまり、「サービス開始」イベントは到着と離脱イベントに含まれているため、独立したイベントとして扱う必要はない。

3.1.2 項で述べたとおり、動的離散型モデルのシミュレーションクロックは、シミュレーションを進行させるため、あるイベント時刻から次の時刻へとジャンプする変数である。**イベントカレンダ**（あるいは**イベントリスト**）は、これから発生するイベントの情報を格納するデータ構造である。データ構造の形式と型は多様だが、イベントカレンダには、将来のイベントの発生時刻、種類、および関連するエンティティの情報が含まれる。シミュレーションは次の手順で進められる：イベントカレンダから直近のイベントを特定する。その時刻までシミュレーションクロックを進める。システム状態変数と統計的変数を更新するイベントロジックを実行する。必要に応じてイベントカレンダを更新する。

イベントごとに実行すべきロジックを定める必要がある。このロジックは、その時点のシステム状態に依存する場合がある。この例題モデルにおいて、各イベント発生時に実行すべき事項は以下のとおりである：

・**到着**：まず、次の到着イベントを予定する。これは現在時刻（列 A のシミュレーションクロック値）に次の到着時間間隔を加え、イベントカレンダの到着スポット（図 8 の Model_03_05.xls の列 J）を更新することで行う。次に、直前のイベントから現在時刻までの期間について、曲線$B(t)$、$L(t)$、$L_q(t)$の連続時間出力結果の面積を計算し、列範囲 O:Q に記録する。この計算には、直前のイベントから現イベント発生直前までの値を使用する。その後の処理は、窓口の状態によって異なる：
  - 窓口が遊休中の場合：到着したエンティティを直ちにサービス開始とする。具体的には、列 G の対応行にエンティティの到着時刻を記録し、現在時刻に次のサービス時間を加えてサービス完了時刻を予定する。また、列 N の対応行に待ち時間 0 を記録する。
  - 窓口が稼働中の場合：到着したエンティティの到着時刻を、待ち行列リスト（列範囲 H:I）の対応行に記録する。この例題では待ち行列長が 2 を超えないため 2 列でじゅうぶんだが、より長い待ち行列が予想される場合は追加の列が必要となる。この到着時刻は、後の待ち時間とシステム内時間の計算に使用する。また、エンティティが実際にサービスを開始する時刻は現時点では不明なため、列 N には待ち時間をまだ記録できない。

・**離脱**（サービス完了）：到着イベントと同様に、直前のイベント以来の連続時間出力の面積を計算し、列 O:Q の対応行に記録する。離脱するエンティティのシステム内時間（シミュレーションクロックの現在時刻から列 G の到着時刻を引いた値）を列 M に記録する。その後の処理は、待ち行列の状況によって異なる：
  - 待ち行列にエンティティが待機している場合：待ち行列の先頭のエンティティをサービス中の状態にする（列 H の到着時刻を列 G の次の行にコピー）。現在時刻に次のサービス時間を加えて、このエンティティの離脱時刻を列 K に設定する。このエンティティの待ち時間（現在のシミュレーションクロック値から到着時刻を引いた値）を列 N に記録する。
  - 待ち行列が空の場合：窓口の状態変数（列 D の 18 行目以降）を遊休中（0）に設定し、イベントカレンダの次の離脱時刻（列 K）を空白にする。

・**シミュレーションの終了**（シミュレーションクロック時刻 8 分）：直前のイベントからシミュレーションの終了時までの、連続時間出力の面積を列 O:Q に計算し、最終評価尺度を行 36:38 に算出する。

イベントを処理するには**コンピュータ**の計算時間が必要だが、イベントの実行中には**シミュレーション**時間は進まない。つまり、シミュレーションクロックは停止したままである。このマニュアルシミュレーションを実行する際、到着時間間隔やサービス時間を注意深く確認することで、これらの時間値が適切に使用され、重複して使用されないことを確認できるだろう。

図 3.8 の Model_03_05.xls では、行 19 から 35 にかけて、時刻 0 での初期化から時刻 8 分での終了まで、各イベントを 1 行ずつ追跡している。列 A にはイベントのシミュレーション時刻t、列 B にはイベントの種類、列 C にはエンティテ

ィ番号が記録されている。各行のデータは、そのイベントが実行された**後**の状態を表している。列 D:F には、窓口の状態$B(t)$（0 が遊休中、1 が稼働中）、システム内エンティティ数、待ち行列長（この例では常にシステム内数から窓口状態を引いた値）が記録されている。列 G にはサービス中のエンティティの到着時刻が、列 H:I には待ち行列の先頭 2 つのエンティティの到着時刻が記録されている（この例では待ち行列長は最大で 2）。

列範囲 J:L はイベントカレンダで、その 3 つの値から最小値を選んで次のイベントを決定する（次の行の列 A には関数 MIN が入力されている）。イベントカレンダには別のデータ構造も考えられる。たとえば、[エンティティ番号, イベント時刻, イベント種類]という形式のレコード（行）をリストとして管理し（ツリーやヒープなどの高度なデータ構造も可能）、新しいイベントを予定する際には時刻順でリストに追加して、次のイベントが常にリストの先頭になるようにする方法である。この方法は実際のシミュレーションソフトウェアの動作に近く、同種の複数イベントを予定できる上、特にヒープ構造を使用すると処理が高速になるという利点がある。これは、多様なイベントが存在し、長いイベントリストを扱う大規模シミュレーションの実行速度に大きく影響する。ただし、この例題では列範囲 J:L の単純な構造でじゅうぶんである。

システム内時間や待ち時間を観測したとき（それぞれ離脱時とサービス開始時）、その値を計算し（列 A のシミュレーションクロック現在値から到着時刻を引いて）、列 M または列 N に記録する。また、列範囲 O:Q には、直前のイベント時刻から現在のイベント時刻までの面積を記録し、これらの曲線の時間平均を算出する。

シミュレーションは時刻 0 の行 19 で初期化から始まる。L19 のシミュレーション終了時刻を 8 に設定し、次の離脱時刻は空白のままとする（Excel は空白セルを大きな数値として扱うため、A20 の関数 MIN は正しく動作する）。

A20 の数式=MIN(J19:L19)は、1 つ上の行のイベントカレンダからもっとも早い時刻である 0 を返す。これは、時刻 0 でエンティティ 1 が到着することを示す。D20 では窓口の状態を稼働中（1）に設定し、E20 のシステム内数を 1 とする。F20 の待ち行列長は 0 のままとする。G20 には到着時刻 0 を記録する。このエンティティは直接サービスを受けられるため、待ち行列は空となり、H20:I20 には記録がない。J20:L20 のイベントカレンダでは、次の到着時刻を現在時刻（0）に到着時間間隔（1.965834）を加えた値に、次の離脱時刻を現在時刻（0）にサービス時間（0.486165）を加えた値に更新する。シミュレーション終了時刻は変更せず、上の行の値をそのまま使用する。到着イベントのため、システム内時間は観測されず M20 は空白となるが、エンティティは直接サービスを受けられるため N20 の待ち時間は 0 となる。セル範囲 O20:Q20 は曲線$B(t)$、$L(t)$、$L_q(t)$下の面積を計算するが、時刻が 0 のため、すべて 0 となる。

行 21 では次のイベント、すなわち時刻 0.486165 におけるエンティティ 1 の離脱を遂行する。離脱イベントのため、M21 にシステム内時間を計算し入力する。次に、セル範囲 O21:Q21 に曲線下の面積を計算する。たとえば、P21 には数式=E20*($A21-$A20)が入力されており、E20 にある直前のシステム内数 1 と、直

前のイベント時刻（A20）と現在時刻（A21）の時間間隔を乗算している。なお、J21 の次の到着時刻は未来に起こることが予定されているイベントであるため、J20 から変化しない。また、現時点では待ち行列から出てサービスを受けるエンティティも存在しないため、K21 の次の離脱時刻も空白のままとなる。

次のイベントは、行 22 の時刻 1.965834 におけるエンティティ 2 の到着である。これは空のシステムへのエンティティの到着で、時刻 0 のエンティティ 1 の到着と同様であるため詳細は省略する。読者自身でスプレッドシート、特に数式が入力されたセルを確認してほしい。

続いて、行 23 で時刻 2.275418 におけるエンティティ 3 の到着イベントが発生する。このエンティティは稼働中のシステムに到着するため、待ち行列長が 1 となり、H23 にはこのエンティティの到着時刻が記録される（A23 からコピー）。この記録は、後でエンティティの待ち時間とシステム内時間を計算するために使用される。この時点では待ち時間もシステム内時間も完了していないため、M23 と N23 は空欄のままとなる。セル範囲 O23:Q23 には、曲線下の新しい面積が計算される。

次のイベントは、行 24 の時刻 2.630955 におけるエンティティ 2 の離脱である。このとき、エンティティ 3 は待ち行列の先頭から窓口に移動するため、このエンティティの到着時刻が H23 から G24 にコピーされ、N24 でエンティティ 3 の待ち時間が計算される（=A24-G24= $2.630955 - 2.275418 = 0.355537$）。

このようにしてシミュレーションは進行し、時刻 8 のシミュレーション終了イベントが次のイベントとなるまで続く（予定されている時刻 8.330407 のエンティティ 9 の到着と時刻 8.168503 のエンティティ 7 の離脱は発生しない）。時刻 8 では、セル範囲 O35:Q35 に終了時の曲線下の面積を計算するのみである。図 3.9 には、シミュレーションにおける曲線$B(t)$、$L(t)$、$L_q(t)$のプロットが示されている。

行 36:38 では、Excel 関数の AVERAGE、MIN、MAX、SUM を用いて評価尺度を算出している。セル範囲 O36:Q36 の時間平均は、曲線下の面積の総和を最終シミュレーションクロック時刻（A35 の 8）で除したものである。本例題の評価尺度は以下のとおりである：窓口利用率は 62.32%（O36）、平均システム内時間は 0.8676 分（M36）、平均待ち時間は 0.2401 分（N36）、そして平均待ち行列長は 0.3864（Q36）となった。4 章では Simio で同一のモデルを構築するため、これらの評価尺度が一致することを確認できるだろう。

ここで、Model_03_03.xls の平均収益で行ったように、列 M および N のシステム内時間や待ち行列内時間の平均、最小、最大を計算する際に、Excel 関数 STDEV を用いて**標準偏差**も計算すべきではないかという疑問が生じるだろう（列範囲 O:Q のエンティティは時間経過による重みづけが不要なため、同様の計算は意味がない）。実際のところ、出力のばらつきは列 M および N のシステム内時間や待ち時間の観測値の変動を確認するだけでじゅうぶんである。標準偏差を計算するために行を追加することは技術的には可能だが、それは**正しいか**どうかを検討せずに、単に**可能だから**実行するという望ましくない例となるだろう。

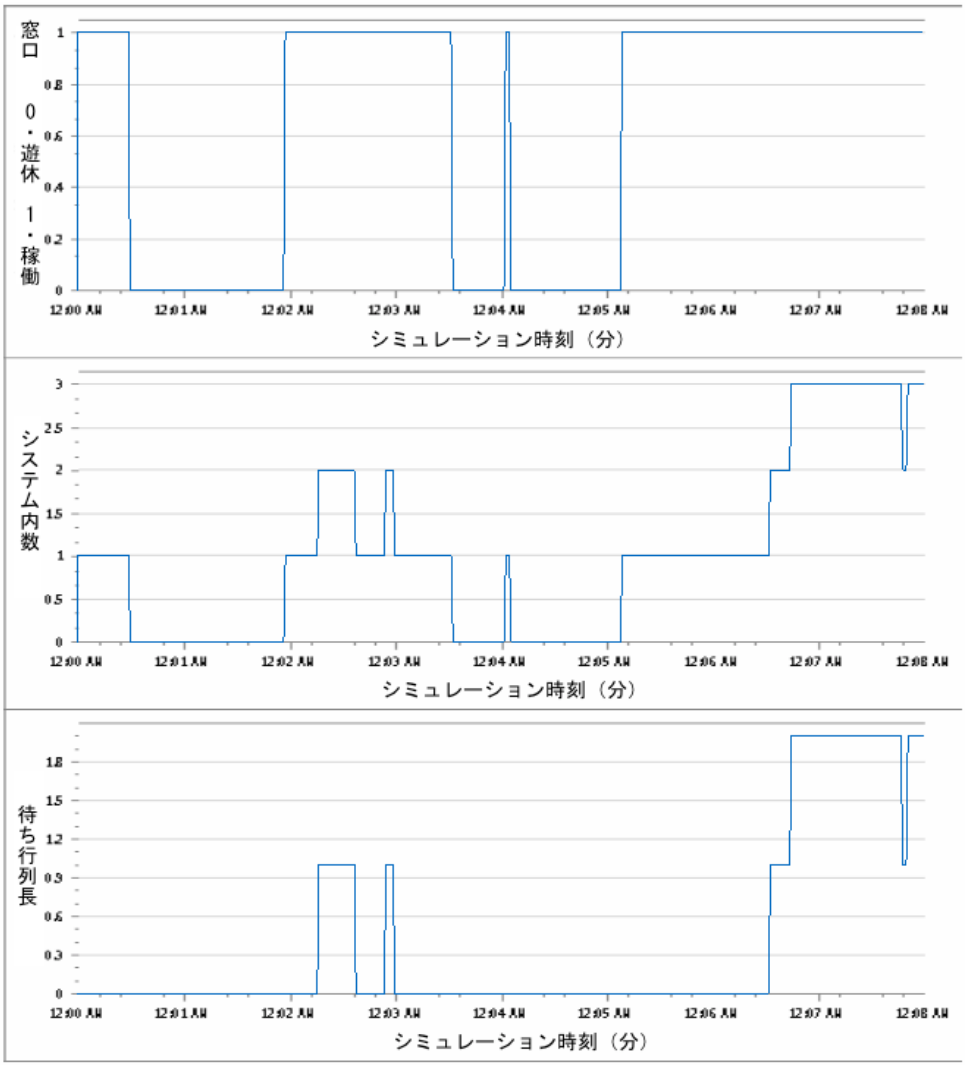


図 3.9　マニュアルシミュレーションにおける曲線$B(t)$、$L(t)$、$L_q(t)$のプロット

では、この例で標準偏差を計算することの問題点は何か。統計学の基礎で学ぶように、標本分散（標準偏差は標本分散の平方根）は、個々の観測値が統計的に**互いに独立の場合にのみ**、母分散の**不偏**推定量となる。しかし、この例題のシステム内時間（および待ち時間）では、連続するエンティティが互いの状況に依存するため、**相関関係**（おそらく**正の相関**）が存在する（この種の順序に基づく相関を**自己相関**または**系列相関**と呼ぶ）。たとえば、単一窓口の銀行で、あなたが不運にもバケツ一杯の小銭を預けようとする顧客の後ろに並んでしまった場合、その顧客の長いシステム内時間が**原因となってあなたの待ち時間も非常に長くなる**。これが正の相関の具体例である。このような場合、標本標準偏差は簡単に計算できるが、独立したサンプルから得られる値と比べると、母集団の標準偏差を過小評価してしまう。つまり、標本標準偏差が**低く**偏り、結果の真の変動性も低く見積もられることになる。この問題は観測値間の相関が強いほど深刻になり、標本分散の推定

値の下方バイアスも大きくなる。

この自己相関の問題があるため、システム内時間や待ち行列内時間のヒストグラムも作成を控えてきた（ただし、Model_03_03.xls の月間収益は独立同一分布のため作成した）。正の相関を持つ観測時間では、実行期間が確率分布の全範囲を適切に「サンプリング」できるほどじゅうぶんに長くない限り、ヒストグラムが本来の分布から左右にずれてしまうリスクがあるためである。

ここで、スプレッドシートの列範囲 M:Q の各イベント行において、システム内時間と待ち時間の**個別**の観測値、および連続するイベント間の曲線$B(t)$、$L(t)$、$L_q(t)$下の**個別**の面積を計算・**記録**した方法について、実用的な注意点を述べる。この小規模な例題では、行範囲 36:38 の Excel 関数だけで要約出力の評価尺度を計算できたため、この方法は有効だった。しかし、例題の 17 行をはるかに超える数千行にわたる長期シミュレーションや、複数回の反復実行が必要なシミュレーションでは、**個々**の値をすべて保存して計算に利用する必要があるため、極めて非効率となる。シミュレーションソフトウェア（または汎用プログラミング言語を用いたシミュレーション）で実際に採用されている方法は、統計**アキュムレータ**変数を設定することだ。これにより、観測値を順次加算していくだけで、個々の値を保存して後で呼び出す必要がなくなる。さらに、各評価尺度の最大値と最小値を保存し、新しい観測値と比較して極値を判断するロジックを追加することもある。ストレージの有効活用の観点から見ても、この方法が計算上はるかに効率的なのは明らかだろう。

最後に、図 3.8 のスプレッドシート Model_03_05.xls は、一般化されている**わけではない**点に注意が必要だ。到着時刻とサービス時間が入力されている青色のセルを変更すると、スプレッドシート全体が正確な結果を示さなくなる可能性がある。これは、多くのセル（ピンクのセル）の数式が、現在入力されているイベントと状況でのみ有効だからである（たとえば、離脱イベントのロジックは待ち行列が空かどうかに依存する）。実際のところ、スプレッドシートは動的離散型シミュレーションには適しておらず、実務でもほとんど使用されない。今回スプレッドシートを使用したのは、データの保存方法や計算方法を説明するためであり、信頼性の高い柔軟なモデルを構築するためではない。3.4.1 項では動的離散型シミュレーションにおける汎用プログラミング言語の利用について簡単に解説するが、複雑なモデルには Simio などのシミュレーションソフトウェアが実質的に不可欠であり、本書の大部分はこれらの解説に充てられる。

3.3.2　Model 3-6：単一窓口待ち行列の待ち時間分析

スプレッドシート（表計算ソフトウェア）は広く普及しており、その基本的な機能は多くの人が理解している。そのため、スプレッドシートで完全な汎用性の高い**動的**シミュレーションモデルが構築できるのかという疑問が生じるだろう。3.2 節では静的かつ確率的なモンテカルロシミュレーションにスプレッドシートを使用したが、プログラミング言語を使用するほうがより効率的だったはずだ。3.3.1 項では、マニュアル動的シミュレーションの数値記録と計算にスプレッドシートを**使用**した。しかし、そのスプレッドシートの数式は特定の入力データ（到着時間

間隔とサービス時間）に依存していたため、完全性、汎用性、柔軟性を備えた**実モデル**とはいえなかった。一般的に、非常に単純な動的シミュレーションでさえ、スプレッドシートでの実装は困難である。ただし、3.2 節で示したような静的シミュレーションには広く活用されている（静的・動的シミュレーションの違いについては 3.1.1 項参照）。

3.2.3 項の単一期間在庫問題では時間の経過について説明した（ピーク期は 1 か月で、その後在庫処分期となる）が、このモデルでは時間は明示的な役割を果たしていなかった。ピーク期を瞬間的な期間として捉え、即座に在庫処分期に移行すると考えることもできる。つまり、時間の経過という観点から見ると、そのモデルは本質的に**静的**なものだった。本項では、時間の経過がモデルの中核となる**非常に**単純な**動的単一窓口待ち行列モデル**をシミュレートする。ただし、スプレッドシートでこれが実現可能なのは、このケースではエンティティの待ち時間（サービス時間を除く）の出力処理のみに焦点を当て、単純な漸化式で表現しているからである。3.3.1 項のマニュアルシミュレーションとは異なり、ここでは他の出力を直接計算することはできない。

時刻 0 において、システムは空かつ遊休状態で始まり、最初のエンティティが到着する単一窓口待ち行列を考える。ここで、エンティティ$i-1$とエンティティi（$i=2,3,\cdots$）間の到着時間間隔をA_iとし、エンティティi（$i=1,2,\cdots$）に必要なサービス時間をS_iとする。A_iおよびS_iは一般に、ある到着時間間隔あるいはサービス時間の確率分布から得られる確率変数である。2 章とは異なり、ここでは解析解を導出するのではなくシミュレーションを行うため、これらの分布は任意でよい。

出力として注目するのは、i番目に到着したエンティティの待ち行列での**待ち時間（遅延）**$W_{q,i}$である。この待ち時間は窓口でのサービス時間を含まず、行列で待機する時間のみを対象とする。最初のエンティティは、システムが空で窓口が遊休中の時刻 0 に到着するため、待ち時間は明らかに 0 となる（$W_{q,1}=0$）。後続の待ち時間$W_{q,2},W_{q,3},\cdots$は、A_iおよびS_iの値に依存するため不確定である。ただし、到着時間間隔とサービス時間が確率分布から得られた後は、連続する$W_{q,i}$間には相互関係が存在することが、次の**リンドレーの漸化式**として知られている（Lindley 1952）：

$$W_{q,i}=\max\left(W_{q,i-1}+S_{i-1}-A_i,0\right)\quad(i=2,3,\cdots)\qquad(3.7)$$

ここで、$W_{q,1}=0$である。本項では、各エンティティに 1 行を割り当て、到着時間間隔とサービス時間の確率分布から観測値を生成するスプレッドシートモデルを作成する。

両方の分布を指数分布とし、到着時間間隔の平均を 1.25 分、サービス時間の平均を 1 分とする。すべての観測値と分布は互いに独立しているものとする。次に必要なのは、所与の平均$\beta>0$の**指数分布**から観測値を生成する方法である。6.4 節で説明するように、連続確率分布からの観測値生成には、$U=F(X)$とする関数を用いる一般的な方法がある。ここでUは 0 と 1 の間の一様乱数であり、Fは確率変数Xの累積分布関数である。この式をXについて解くと、生成された乱数Uに対

応するXの観測値を得る式が導かれる。平均βの指数分布の場合、累積分布関数は次式となる：

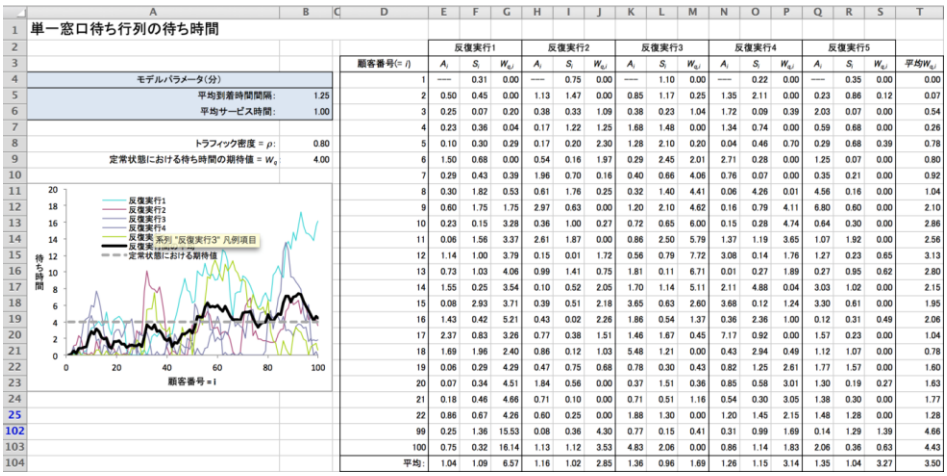
$$F(x) = \begin{cases} 1 - e^{-x/\beta} & x \geq 0 \text{ の場合} \\ 0 & \text{その他の場合} \end{cases} \qquad (3.8)$$

$U = F(X)$を式 3.8 に代入し、Xについて解くと、下式を得る：

$$X = -\beta \ln(1 - U) \qquad (3.9)$$

ここで、lnはeを底とする自然対数関数である。

以上により、式 3.7 のリンドレーの漸化式と式 3.9 の指数分布から観測値を生成する式、そして Excel の乱数ジェネレータ RAND()を用いて、到着時間間隔とサービス時間が指数分布に従う単一窓口待ち行列の待ち時間のプロセスを Excel でシミュレートできる。図 3.10 に Model_03_06.xls を示す（行範囲 26:101 は非表示。すべての行を確認するにはファイルを開く必要がある。もっとも重要なのはセル内の数式である）。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	単一窓口待ち行列の待ち時間																			
2					反復実行1			反復実行2			反復実行3			反復実行4			反復実行5			
3				顧客番号(= i)	A_i	S_i	$W_{q,i}$	A_i	S_i	$W_{q,i}$	A_i	S_i	$W_{q,i}$	A_i	S_i	$W_{q,i}$	A_i	S_i	$W_{q,i}$	平均$W_{q,i}$
4	モデルパラメータ(分)			1	----	0.31	0.00	----	0.75	0.00	----	1.10	0.00	----	0.22	0.00	----	0.35	0.00	0.00
5	平均到着時間間隔:	1.25		2	0.50	0.45	0.00	1.13	1.47	0.00	0.85	1.17	0.25	1.35	2.11	0.00	0.23	0.86	0.12	0.07
6	平均サービス時間:	1.00		3	0.25	0.07	0.20	0.38	0.33	1.09	0.38	0.23	1.04	1.72	0.09	0.39	2.03	0.07	0.00	0.54
7				4	0.23	0.36	0.04	0.17	1.22	1.25	1.68	1.48	0.00	1.34	0.74	0.00	0.59	0.68	0.00	0.26
8	トラフィック密度 = ρ:	0.80		5	0.10	0.30	0.29	0.17	0.20	2.30	1.28	2.10	0.20	0.04	0.46	0.70	0.29	0.68	0.39	0.78
9	定常状態における待ち時間の期待値 = W_q	4.00		6	1.50	0.68	0.00	0.54	0.16	1.97	0.29	2.45	2.01	2.71	0.28	0.00	1.25	0.07	0.00	0.80
10				7	0.29	0.43	0.39	1.96	0.70	0.16	0.40	0.66	4.06	0.76	0.07	0.00	0.35	0.21	0.00	0.92
11				8	0.30	1.82	0.53	0.61	1.76	0.25	0.32	1.40	4.41	0.06	4.26	0.01	4.56	0.16	0.00	1.04
12				9	0.60	1.75	1.75	2.97	0.63	0.00	1.20	2.10	4.62	0.16	0.79	4.11	6.80	0.60	0.00	2.10
13				10	0.23	0.15	3.28	0.36	1.00	0.27	0.72	0.65	6.00	0.15	0.28	4.74	0.64	0.30	0.00	2.86
14				11	0.06	1.56	3.37	2.61	1.87	0.00	0.86	2.50	5.79	1.37	1.19	3.65	1.07	1.92	0.00	2.56
15				12	1.14	1.00	3.79	0.15	0.01	1.72	0.56	0.79	7.72	3.08	0.14	1.76	1.27	0.23	0.65	3.13
16				13	0.73	1.03	4.06	0.99	1.41	0.75	1.81	0.11	6.71	0.01	0.27	1.89	0.27	0.95	0.62	2.80
17				14	1.55	0.25	3.54	0.10	0.52	2.05	1.70	1.14	5.11	2.11	4.88	0.04	3.03	1.02	0.00	2.15
18				15	0.08	2.93	3.71	0.39	0.51	2.18	3.65	0.63	2.60	3.68	0.12	1.24	3.30	0.61	0.00	1.95
19				16	1.43	0.42	5.21	0.43	0.02	2.26	1.86	0.54	1.37	0.36	2.36	1.00	0.12	0.31	0.49	2.06
20				17	2.37	0.83	3.26	0.77	0.38	1.51	1.46	1.67	0.45	7.17	0.92	0.00	1.57	0.23	0.00	1.04
21				18	1.69	1.96	2.40	0.86	0.12	1.03	5.48	1.21	0.00	0.43	2.94	0.49	1.12	1.07	0.00	0.78
22				19	0.06	0.29	4.29	0.47	0.75	0.68	0.78	0.30	0.43	0.82	1.25	2.61	1.77	1.57	0.00	1.60
23				20	0.07	0.34	4.51	1.84	0.56	0.00	0.37	1.51	0.36	0.85	0.58	3.01	1.30	0.19	0.27	1.63
24				21	0.18	0.46	4.66	0.71	0.10	0.00	0.71	0.51	1.16	0.54	0.30	3.05	1.38	0.30	0.00	1.77
25				22	0.86	0.67	4.26	0.60	0.25	0.00	1.88	1.30	0.00	1.20	1.45	2.15	1.48	1.28	0.00	1.28
102				99	0.25	1.36	15.53	0.08	0.36	4.30	0.77	0.15	0.41	0.31	0.99	1.69	0.14	1.29	1.39	4.66
103				100	0.75	0.32	16.14	1.13	1.12	3.53	4.83	2.06	0.00	0.86	1.14	1.83	2.06	0.36	0.63	4.43
104				平均:	1.04	1.09	6.57	1.16	1.02	2.85	1.36	0.96	1.69	1.26	1.15	3.14	1.35	1.04	3.27	3.50

図 3.10　Excel を用いた単一窓口待ち行列システムの待ち時間シミュレーション

青色のセル範囲 A4:B6 にモデルの入力パラメータが入力され、トラフィック密度ρおよび定常状態における待ち時間の期待値W_qがセル範囲 A8:B9 で計算される（$M/M/1$待ち行列の式については 2 章を参照）。D 列にはエンティティ番号$i = 1, 2, \cdots, 100$が示され（エンティティの待ち時間を 100 回観測することとした）、各行に対応するエンティティの待ち時間が記録される。E 列では、セル B5 の平均到着時間間隔に基づく数式=-$B$5+LN(1-RAND())により、到着時間間隔A_iが生成される（LN は自然対数を返す Excel 関数）。列 F も同様の方法でサービス時間を生成する。G 列では、リンドレーの漸化式（式 3.7）を実装している。たとえば、セル G20（エンティティ$i = 17$）には、$W_{q,17}$の計算のために数式=MAX(G19+F19-E20,0)が使用され、$W_{q,16}$はセル G19、S_{16}はセル F19、A_{17}はセル E20 にそれぞれ対応する。この数式をコピーする際に行ごとに参照セルを**変化させたい**ため、セル参照には$を**使っていない**。これは待ち時間の相互依存関係（相関関係）を示

しており、シミュレーション出力には適切な統計解析手法が必要となる（つまり、古典的な統計分析とは異なり、シミュレーション出力の連続する観測値を独立したものとして扱えない。この点については 3.3.1 項の最後で説明した）。

列範囲 E:G は右側に 4 回コピーされ、100 個のエンティティの待ち時間シミュレーションに対する 5 回の独立した**反復実行**を行う準備が整う。これら 3 つの列の各ブロックは、乱数ジェネレータ RAND()からの独立したサンプルを使用する点のみが異なり、数式の内容は同一である。列 T には、$i = 1, 2, \cdots, 100$の各エンティティについて、5 回の反復実行から得られた 5 つの$W_{q,i}$の平均値が計算されている。つまり、この列には$1, 2, \cdots, 100$番目に到着したエンティティの**反復実行間**の待ち行列平均が示されている。行 104 には、A_iおよびS_iの列平均が計算されており、これらはセル B5 および B6 に入力された到着時間間隔とサービス時間の平均値の確認に用いる。ただし、$W_{q,i}$の列平均については、5 回の反復実行それぞれの待ち時間の**反復実行内**平均値となる（3.3.1 項で述べたように、自己相関のバイアス効果があるため、反復実行内の待ち時間の標準偏差計算やヒストグラム作成は行わない）。左側のグラフには、各反復実行の待ち時間（薄い曲線）、反復実行間の待ち時間平均（太く濃い曲線）、および定常状態における待ち時間の期待値（水平の破線）を示している。

RAND()の変動性を確認するため、F9 キーを数回押して、到着時間間隔、サービス時間、待ち時間の変化の様子とグラフの推移を観察してほしい（F9 を押すと乱数が再生成され、最大待ち時間に応じてグラフの縦軸目盛りが自動更新される）。以下に注意点を示す：

- すべてのシミュレーションによる曲線は左端が 0 から始まり、これは最初のエンティティが待ち時間なしで処理されることを示している。時間経過に伴い（横軸のエンティティ番号の増加とともに）、待ち時間は概して増加傾向を示すが、到着時間間隔とサービス時間のランダム性により、単調な増加とはならない。
- 太く濃い曲線は薄い曲線と比べて変動が小さい。これは、太い曲線が薄い曲線の平均値を表しており、各エンティティの待ち時間の増減が相殺されるためである。
- すべての曲線は特徴的な蛇行を示す。曲線はランダムな動きを見せるものの、右側に進むにつれて新しい値は前の値から極端には離れず、ゆるやかな**つながり**を保持している。この特徴は、反復実行内での待ち時間に（正の）自己相関があることを示しており、式 3.7 のリンドレーの漸化式からも確認できる（3.3.1 項の最後でも説明したとおりである）。
- F9 キーを押すと、待ち時間は定常状態の期待値（水平の破線）に収束する場合としない場合がある。これは次の 2 点を示唆している：(a)システムには相当量のノイズが存在する、(b)定常状態の条件に近づくためには、100 を超えるエンティティ数が必要である。

B5 および B6 の平均到着時間間隔と平均サービス時間を変化させる際は、平均到着時間を必ず平均サービス時間よりも大きく設定する必要がある。これを守ら

ないと、定常状態における待ち時間の期待値W_qの式 B9 が妥当性を失う（ただし、シミュレーション自体の妥当性は保たれる）。また、平均到着時間間隔と平均サービス時間の値が近い場合も試してみてほしい（B8 のトラフィック密度ρが 1 に近づき、W_qが著しく大きくなる）。

最後に、このスプレッドシートによる動的シミュレーションが実現できたのは、式 3.7 のリンドレーの漸化式を利用できたことが要因であることを強調したい。さらに、このモデルは待ち時間の出力のみに特化しており、この単純なシステムですら、3.3.1 項のマニュアルシミュレーションで扱った待ち行列長や窓口の利用率などの指標は計算できない。一般的に、複雑な動的システムではリンドレーの漸化式を有効活用できる場合はまれであり、スプレッドシートで実用的な動的システムをシミュレートすることは現実的ではない。これこそが、Simio などの動的シミュレーションソフトウェアが必要とされる本質的な理由である。

3.4　動的シミュレーションに対するソフトウェアの選択

3.3 節では、動的シミュレーションを実行する 2 つの方法（マニュアルとスプレッドシート）について説明したが、いずれも実用的とはいえなかった。本節では、動的シミュレーションを実行するためのより実践的な 2 つの方法について概説する。

3.4.1　汎用プログラミング言語

3.3.1 項で解説したマニュアルシミュレーションは、C++、Java、Matlab などの汎用目的の手続き型**プログラミング言語**で記述することで、より効率的かつ一般的に表現できる。これは、シミュレーション専用ソフトウェアが登場する以前の標準的な手法だった。現在でも、モデルが比較的単純な場合や計算速度が重要な場合など、特定の状況で採用されている。3.3.1 項のスプレッドシートによるマニュアルシミュレーションほどではないものの、この方法は開発に時間と労力を要し、エラーも起きやすい。一方で、高水準の専用シミュレーションソフトウェアでは実現できないような複雑な挙動もモデル化できる柔軟性を備えているという大きな利点がある。

汎用シミュレーション言語を用いて動的離散型シミュレーションを実装する場合も、まずイベントの特定とそのロジックの定義が必要である。その後、各イベントの活動を実行するサブプログラム（または関数）を作成する（3.3.1 項で説明した到着・離脱イベントの箇条書きが、サブプログラムの構造化の参考となる）。これらのイベントサブプログラムは、メインプログラムまたは関数と連携して動作する。メインプログラムは 3.3.1 項のイベントカレンダの更新のような処理を担当し、次のイベントが決定されると対応するイベントサブプログラムを実行する。また、シミュレーション終了時には要約統計量を計算・出力するプログラムも必要となる。さらに、確率的シミュレーションでは、乱数ジェネレータを使用して入力確率分布から観測値を生成する方法や、データ構造を用いてモデルの状態変数を追跡する方法も考慮しなければならない。このように、モデルが複雑化すると必要な作業と分析の時間が膨大となり、エラーの発見も困難になる。

しかし、大規模で特殊なケースや標準的なシミュレーションの実現を目指して、汎用シミュレーション言語による手法を追求する研究も続けられている。詳しくは、Banks et al.（2005）、Law（2015）、Seila et al.（2003）などを参照されたい。

3.4.2　専用シミュレーションソフトウェア

本章では、シミュレーションを実行するための多様なアプローチを述べてきた。これらには、スプレッドシートを用いてシミュレーションの進行を追跡・演算するマニュアル方式、汎用プログラミング言語でシミュレーションロジックをプログラムする方法、そしてスプレッドシートにシミュレーション専用アドインを組み込んでモデルを構築する方法が含まれる。

これらの各アプローチには、それぞれ異なる制約がある。マニュアルシミュレーションは実用的な応用には明らかに不向きである。汎用プログラミング言語は、完全なモデリングの柔軟性と高速な実行を提供するものの、複雑な動的シミュレーションを一から構築するには膨大な時間がかかり、プログラミングエラーのリスクも高い。スプレッドシートは、適切なシミュレーションアドインを使用すれば静的シミュレーションには今後も有効だろうが、大規模な動的シミュレーションのロジックやデータ構造を扱うには機能が不十分である。

実際、高水準な動的シミュレーションソフトウェア、特に離散確率型動的シミュレーションに対するニーズは、スプレッドシートが登場する以前から満たされていた。1960 年代には、GPSS（Schriber 1974、1991）、SIMSCRIPT（Kiviat et al. 1969）、SIMULA（Birtwistle et al. 1973）などの専用シミュレーション言語が開発された。これらの一部は、Simio と同様に**オブジェクト指向型**の先駆けとなった（当時はまだその概念自体が存在しなかった）。また、FORTRAN などの汎用プログラミング言語への拡張も開発され、その代表例が GASP（Pritsker 1974）である。これらの取り組みは、SLAM（Pritsker 1995）や SIMAN（Pegden et al. 1995）といった、より現代的で自己完結的なシミュレーション言語の開発へとつながった。ただし、これらのシミュレーション言語は、テキスト入力と構文を用いる本来の意味での**言語**であったため、使用や結果の解釈が直感的ではなかった。その後、グラフィカルユーザインタフェースとアニメーション機能を備えたソフトウェアが登場し、より多くのユーザがシミュレーションを活用できるようになった。現在では、Arena（Kelton et al. 2015）、AnyLogic（`www.anylogic.com`）、ExtendSim（`www.extendsim.com`）、Promodel（`www.pmcorp.com`）、Flexsim（`www.flexsim.com`）、Simul8（`www.simul8.com`）などが利用可能である。シミュレーションソフトウェアの詳細については、Banks et al.（2005）や Law（2015）を、より包括的なレビューについては Nance and Sargent（2002）を参照されたい。また、Swain（2021）は隔年実施されるシミュレーションソフトウェアの調査報告書の最新版である。

Simio は比較的新しい製品で、最新のソフトウェア開発とモデリング能力の知見に基づいて開発されている。新しいとはいえ、Simio は SIMAN と Arena の開発者が手がけているため、豊富な経験が活かされている。**知的オブジェクト**を核

として、オブジェクトの構築と利用に画期的な変革をもたらす新しいオブジェクト指向のパラダイムを提供する。Simio オブジェクトは、プログラミングを必要とせず、シンプルなグラフィカルプロセスフローで作成できる。これにより、独自のモデリングオブジェクトや目的特化型ライブラリの構築が容易になる。さらに、Simio の柔軟性と広範なリアルタイムデータ統合により、**インダストリー4.0** に不可欠な**デジタルツイン**の作成にも適している。本書の後半では、Simio を用いて実践的な問題を効果的に解決するシミュレーションの方法を解説する。

3.5　問題

1．3.2.1 項のサイコロシミュレーションを、以下の手順で発展させなさい（1 つずつ順に実施すること）。

・スプレッドシートを拡張して 50 回から 500 回の試行に増やし、結果を比較しなさい。F9 キーを押すと新しい乱数の組が生成され、結果が更新される。

・サイコロの各目の出る確率を$1/6$で一定から、異なる確率に変えなさい。ただし、確率の合計が 1 となるように設定すること。

・@RISK（3.2.3 項参照）または他の静的モンテカルロシミュレーション用の Excel アドインを使用して、偏りのないサイコロ 2 個を 10,000 回投げたときの結果を分析しなさい。得られた各目の和の分布を、真の確率分布および期待値 7 と比較しなさい。

2．3.2.1 項のサイコロシミュレーションについて、2 つのサイコロの目の和がそれぞれ$1,2,\cdots,12$となる確率を初級確率論から導きなさい。これらの確率を用いて、2 つのサイコロの目の和の真の期待値と標準偏差を計算し、セル J4 の標本平均と比較しなさい。さらに、スプレッドシートに標本標準偏差の計算を追加し、理論値と比較しなさい。サイコロを投げる回数は 50 回のままとする。

3．確率論と確率変数の期待値の定義（3.2.2 項参照）を用いて、$E(Y)=\int_a^b h(x)dx$を厳密に証明しなさい。まず$E(Y)=E\left[(b-a)h(X)\right]=(b-a)E\left[h(X)\right]$と変形し、次に確率変数関数の期待値の定義を適用する。最後に、Xは$[a,b]$で連続一様に分布することから、$a\le x\le b$について密度関数$f(x)=1/(b-a)$となることを利用して証明を完成させなさい。

4．@RISK または静的スプレッドシートシミュレーション用の別の Excel アドインを使用して、3.2.2 項の例題を拡張しなさい。図 3.2 の `Model_03_02.xls` で使用した 50 個のX_iを 10,000 個に増やし、`Model_03_02.xls` および（ほぼ）正確な数値積分の結果と比較しなさい。

5．3.2.2 項のモンテカルロ積分において、スプレッドシートに 50 個の値に対する標準偏差の計算を追加しなさい。セル H4 の平均値とセル I4 の正確な積分値を用いて、95%信頼区間を計算しなさい。この信頼区間は正確な積分値を含むだろうか？F9 キーを繰り返し押して結果をメモし、包含される頻度を調べなさい。同様の分析を 90%信頼区間と 99%信頼区間についても行いなさい。

6．3.3.1 項のマニュアルシミュレーションと Excel ファイル`Model_03_04.xls`において、サービス時間を 20%短縮してサービス提供速度を向上させなさい。3.2 節の評価指標にどのような影響が生じるか分析し、考察しなさい。

7．問題 6 と同様に、今度はサービス時間を 20%増加させることによりサービス提供速度を低下させなさい。同様の指標への影響を述べ、考察しなさい。

8．3.3.1 項のマニュアルシミュレーションと Excel ファイル`Model_03_04.xls`において、待ち行列規律を FIFO から**最短ジョブ時間順**（SJF）に変更しなさい。つまり、窓口が遊休

となった際に待ち行列ができている場合、もっとも短いサービス時間のエンティティを次に窓口に進ませる規律である。これは、**最短処理時間順**（SPT）とも呼ばれる（詳細は 2.1 節参照）。この実装には、エンティティが到着した際に（将来の）サービス時間をエンティティに割り当て、待ち行列規律の変更に利用する必要がある。3.2 節と同じ評価指標にどのような影響があるか述べなさい。実装方法として、次の 2 つが考えられる：(a)新しいエンティティが待ち行列に加わる際に、サービス時間の昇順に並べる方法（最短のサービス時間のエンティティが常に待ち行列の先頭になる）。(b)待ち行列の並び順は変えず、新しく到着するエンティティを、サービス時間を考慮せずに待ち行列の最後尾に加える。窓口が遊休になるたびに待ち行列を検索し、次にサービスを受けるエンティティを選ぶ方法。この 2 つの方法は同じ結果をもたらすが、長い待ち行列ではどちらの処理速度が速くなるだろうか。

9. 3.2.2.3 項の在庫スプレッドシートシミュレーションと Excel ファイル Model_03_03.xls では、50 回の反復実行（50 行）を任意に設定したため、平均収益の期待値には、サンプルサイズ 50 に対する信頼区間の半幅（行 59）が伴っていた。Model_03_03.xls を 200 回の反復実行（4 倍の実行回数）に変更すると、予想通り半幅は縮小するが、その程度を評価尺度から測定できる（ただし、Excel の乱数ジェネレータの変動性により、正確な測定は難しい）。標準的な統計学の書籍で信頼区間の半幅に関する計算式（平均に対する正規分布およびt分布を用いる方法）を参照し、実証結果を解釈しなさい。半幅を元の 50 回の反復実行時と比べて 10 分の 1 に削減する（つまり平均の推定値の統計的精度を一桁向上させる）には、およそ何回の反復実行が必要か。

10. 3.2.3 項の在庫シミュレーションと Excel ファイル Model_03_03.xls では、各反復実行（行）において F 列の**同じ**ピーク期需要の実現値を、5 つすべての発注量hの試行値に対する収益計算に使用していた。これに代えて、各hの値に対して独立した需要の列を設定することも可能である。Model_03_03.xls をこのように変更し、その妥当性を確認しなさい。**本質的な**関心事は、異なるhの値に対する収益の**差異**にあるが、この新しい方法はより適切といえるだろうか。平均収益の推定値には「ノイズ」（統計的変動性）が存在することに注意が必要である。ノイズの減少は推定値の正確性を高める。これは、所与のhに対する収益の値自体と、異なるh間の収益の差異の両方に当てはまる。

11. 3.2.3 項の在庫シミュレーションと Excel ファイル Model_03_03.xls において、次の 2 つの選択肢を検討しなさい：(1)交渉力の強い人物を雇い、サプライヤからの仕入れ値を$11.00 から$9.00 に削減する、(2)広告代理店に依頼し、宣伝広告によりピーク期の小売価格を$16.95 から$18.95 に引き上げる（いずれの場合も在庫処分価格は$6.95 のまま）。最高収益を記録する$h$の値に基づいて統計分析を行い、2 つの選択肢を比較しなさい。

12. 問題 11 において、広告代理店への依頼内容を変更し、価格引き上げの代わりにピーク期需要の最大値を 5,000 から 6,000 に増加させる宣伝を行うとしよう。仕入れ値$11.00、ピーク期の小売価格$16.95、在庫処分価格$6.95 は変更しない。広告代理店に関する 2 つの選択肢について、最高収益を記録するhの値に基づいて比較しなさい。

13. 3.2.3 項の在庫シミュレーションと Excel ファイル Model_03_03.xls において、ピーク期後の残存在庫を損失価格で**売る**代わりに、慈善団体への**寄付**を検討する。寄付した商品の仕入れ額は法人税課税対象額から控除され、法人税率は 34%とする。この方策は収益にプラスとなるか。2 つの選択肢について、最高収益を記録するhの値に基づいて比較しなさい。

14. 3.2.4 項の Model 3-4 のオリジナルバージョンでは、入力確率変数に次の確率分布を使用していた：$Q \sim \mathrm{Uniform}(8000, 12000)$；$P \sim \mathrm{Normal}(10, 3)$、2.0 で左側を切り捨て；

V~Normal(7,2)、3.5 で左側を、10.0 で右側を切り捨て。切り捨てにより、Pと Vの値に範囲が設定される。これらのオリジナルの分布を用いてモデルを構築し、3.2.4 節と同様の分析を実施しなさい。

15. 3.3.2 項の待ち行列シミュレーションと Excel ファイル Model_03_06.xls において、1,000 エンティティの待ち時間を分析できるようにモデルを実行しなさい。定常状態の待ち時間の期待値W_qについて、100 回の反復実行と 1,000 回のケースを比較し、考察しなさい。

16. 3.3.2 項の待ち行列シミュレーションと Excel ファイル Model_03_06.xls（100 回の反復実行のケース）において、平均到着時間間隔を 1.25 分から 1.1 分に減少させて到着率を増加させ、元のモデルと結果を比較しなさい。さらに、平均到着時間間隔を 1.05 分、その後 1.01 分に減少させ、定常状態の待ち時間の期待値W_qおよびシミュレーション結果の両方について、その影響を確認しなさい。

17. 問題 16 に続けて、さらに平均到着時間間隔を 1.00 分、0.99 分、0.95 分、0.88 分、最後に 0.80 分へと変化させなさい。どのような結果が得られるか。2 章で説明した「安定性」の観点から、これらの結果は「有意義」といえるだろうか。定常状態の待ち時間の期待値W_qと 100 回の反復実行の結果を個別に検討し、「有意義性」について考察しなさい。

18. 露天商のワルサーは、オート麦、えんどう豆、豆、大麦を販売している。彼は各商品をポンド単位で仕入れており、卸売価格はそれぞれ$1.05、$3.17、$1.99、$0.95 である。小売価格はポンド当たりそれぞれ$1.29、$3.76、$2.23、$1.65 で販売する。各商品の 1 日当たりの需要量は、最小が 0 ポンドで、最大はそれぞれ 10、8、14、11 ポンドである。ワルサーはポンド単位でのみ販売する。各商品の需要量は、上記の範囲内で離散一様分布に従う。ワルサーは常にすべての需要を満たせるじゅうぶんな在庫を保有している。夏季の販売期間は 90 日間で、各日の需要量は他の日の需要量とは独立している。各日および季節全体の総費用、総収入、総利益を計算するスプレッドシートシミュレーションを作成しなさい。

19. ある新車の店頭価格（希望小売価格）は$22,000 である。この車には選択可能なオプション（AT、オーディオ、フォグランプ、防眩バックミラー、ムーンルーフ）があり、選択したオプションに応じて$0 から$2,000 の価格が追加される。オプションの総費用はこの範囲内で連続一様分布に従う。また、多くの顧客は値引き交渉を行い、値引き額は$0 から$3,000 の間で連続一様分布に従う（オプション選択とは無関係）。最終的な販売価格は、基本価格$22,000 にオプション費用を加え、値引き額を差し引いた金額となる。

a) Excel の標準機能のみを使用して、100 台分の販売価格を計算するスプレッドシートシミュレーションを作成しなさい（@RISK や他のサードパーティ製アドインは使用不可）。100 台分の販売価格の標本平均、標準偏差、最小値、最大値を求めなさい。将来の販売価格に対する 95%信頼区間を Excel の機能で計算しなさい（**ヒント**：Excel 関数 TINV を使用し、入力パラメータを検討する）。また、100 台分の販売価格のヒストグラムを作成するために、Model 3-5 の手法を参考にするか、Excel のデータ分析ツールの「ヒストグラム」機能を使用しなさい（データ分析ツールは以下の手順でインストールできる：[ファイル]-[オプション]-[アドイン]から、画面下部の[管理]で「Excel アドイン」を選択し[設定]をクリックする。そして「分析ツール」にチェックを入れ[OK]をクリックする）。すべての計算は Excel で行い、電卓や統計表は使用しないこと。

b) 同じ Excel ファイルの 2 番目のシートで、a)を再度実行しなさい。ただし、今回は @RISK または他の静的スプレッドシートシミュレーション用 Excel アドインを使用

し、一様確率変量の生成には独自の数式ではなく、確率変数生成機能を使用すること。販売価格の期待値に対する信頼区間の計算は不要である。100 台ではなく 10,000 台の車について、販売価格の期待値を計算し、なめらかなヒストグラムを作成しなさい（@RISK のヒストグラムを使用）。@RISK を除き、すべての作業は Excel で行うこと。

20. 問題 19 をより詳細に検討するために、表 3.2 に 5 つのオプションの価格と選択確率を示す（たとえば、顧客が AT を選択する確率は 0.6）。顧客は各オプションを独立して選択すると仮定する。つまり、他のオプションの選択状況は影響しない。問題 19 の a)を再度実行しなさい。ただし表 3.2 の価格構造に基づき、顧客が各オプションを選択する。なお、値引き交渉は問題 19 と同様に扱う（顧客のオプションの選択とは独立）。すべての計算は Excel で行い、@RISK は使用しない。

ヒント 1：[0,1]の連続する乱数が 0.6 より小さくなる確率は 0.6 である。

ヒント 2：Excel 関数 IF(条件式，x，y)は、条件式が真のとき x、偽のとき y を返す。

表 3.2　問題 20 の新車オプション

オプション	価格	選択確率（%）
AT	$1,000	60
オーディオ	$120	10
フォグランプ	$200	30
防眩バックミラー	$180	20
ムーンルーフ	$500	40

第 4 章　初めての Simio モデル

本章の主な目標は、Simio を使用したシミュレーションモデルの構築プロセスを紹介することである。モデルの構築とともに、シミュレーション出力結果の統計分析を行い、モデル化されたシステムからどのような有効な推論が導き出せるかを理解する。本章では、1 つの完結した Simio モデルを通じて、シミュレーション出力データのモデル検証、実験、および統計分析の概念を説明する。なお、基本的なモデル構築と分析の手順自体は Simio に固有のものではないが、ここでは Simio を実行手段として使用する。

本章で扱うモデルは、実行期間を除けば、3.3.1 節の手動モデル Model 3-4 や 3.3.2 項のスプレッドシートモデル Model 3-5 と基本的に同じ初歩的なものである。このモデルは理解しやすいため、モデル自体よりも構築プロセスと基本的な Simio 概念の習得に焦点を当てることができる。また、このモデルに簡単な変更を加えることで、Simio の追加的な概念も解説する。次章では、より包括的なシステムに対応するため、追加的な Simio の機能とシミュレーションモデリング技法を組み込んだモデルへと発展させる。

このモデルは、到着率μ=60 エンティティ/時間、サービス率λ=48 エンティティ/時間の単一サーバ待ち行列システムである（図 4.1）。このシステムは、生産システムの機械、銀行の出納係、ファストフード店のレジ係、救急センターのトリアージナースなど、さまざまな場面を表現できる。本章の目的上、具体的な対象は重要ではない。

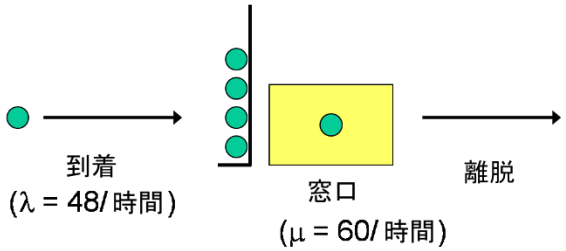


図 4.1　例題の単一サーバ待ち行列システム

モデルの前提条件は以下のとおりである：

- 到着過程は**ポアソン分布**に従う（つまり、到着時間間隔は指数分布に従い互いに独立）
- サービス時間は指数分布に従い互いに独立（サービス開始時間間隔とも独立）
- 待ち行列は無限容量で、待ち行列規律は先入先出（FIFO）

分析の対象は、待ち行列内のエンティティ数（平均と最大）、待ち行列での待機時間（平均と最大）、サーバの稼働率などの一般的な待ち行列の測定基準である。長期実行または定常状態の挙動に関心がある場合、このシステムは標準的な**待ち行列解析法**で分析できる（2 章参照）。ただし本章では、Simio を使用したシステムのモデル化に焦点を当てる。

本章では、Simio を使用して待ち行列システムをモデル化する 2 つの方法を説明する。1 つ目は、Standard ライブラリの Facility ウィンドウと Simio オブジェクトを使用する方法（4.2 節）である。2 つ目は、Simio Processes（4.3 節）を用いた低レベルでのモデル構築方法であり、より詳細なモデル化が必要な場合に適している。これらの 2 つの手法は相互に関連している。実際、Standard ライブラリオブジェクトは Processes を用いて構築されており、あらかじめ用意された高レベルのインタフェースと組み込みアニメーションにより、直感的なモデル構築を可能にしている。一方、Processes によるカスタマイズは、特殊な機能や高速な実行が必要なモデルに適した低レベルのインタフェースを提供している。なお、Standard ライブラリオブジェクトを構成する Processes へのアクセスについては、後の章で説明する。

本章の構成は以下のとおりである。4.1 節で Simio ウィンドウとユーザインタフェースを解説し、4.2 節では Standard ライブラリオブジェクトを使用した Facility ウィンドウでのモデル構築方法を説明する。ここで、統計的に独立した反復、ウォームアップ期間、定常状態と終結シミュレーションの違い、モデル検証といった重要な概念を学ぶ。4.3 節では最初のモデルを Processes で再構築し、4.4 節では初期モデルの時間分布を変更する。4.5 節と 4.6 節では Simio の革新的な手法を用いた出力データの統計分析を紹介し、4.8 節ではアニメーション機能の実装を説明する。モデルの実行により、当初は気づかなかった挙動を発見し、理解を深めることができる。最後に 4.9 節で、モデルの問題点の特定と修正手順を解説する。本章で扱うシステムは比較的単純であるが、この例題を通じて、Simio モデルの構築方法とその効果的な活用方法を学ぶことができる。

4.1　基本的な Simio ユーザインタフェース

この節では、Simio モデルの構築に入る前に、利用可能なモデリングの構成要素とその操作方法を紹介し、Simio のユーザインタフェースの基本を解説する。

Simio を起動すると、デフォルトで新規モデルが表示される。File ページで Load most recent project at startup にチェックを入れている場合は、最後に開いていたモデルが表示される。図 4.2 は新規 Simio モデルの画面である。モデル構築をはじめたい気持ちはあるが、まずはインタフェースと Support リボン（後述）に含まれる重要な情報を確認する必要がある。一見遠回りに見えても、これらを理解しておくことで後々の時間を節約できる。

4.1.1　リボン

リボン（Ribbon）は、Microsoft™ Office 2007 で導入された革新的なインタフェースで、従来のメニューとツールバーを置き換えるものである。リボンは、コンテンツを直感的に整理し自動調整することで、タスクを効率的に実行できる。コマンドは論理的なグループに分類され、それらのグループはタブの下にまとめられている。各タブは、モデルの実行やシンボルの描画など、特定の作業に対応している。タブは自動的に表示され、作業の状況に応じて前面に出てくる。たとえば、シンボルを作成中の場合は Symbols タブが強調表示される。なお、表示され

るリボンは、プロジェクトの状況（インタフェースで選択している要素）によって変わることに注意が必要である。

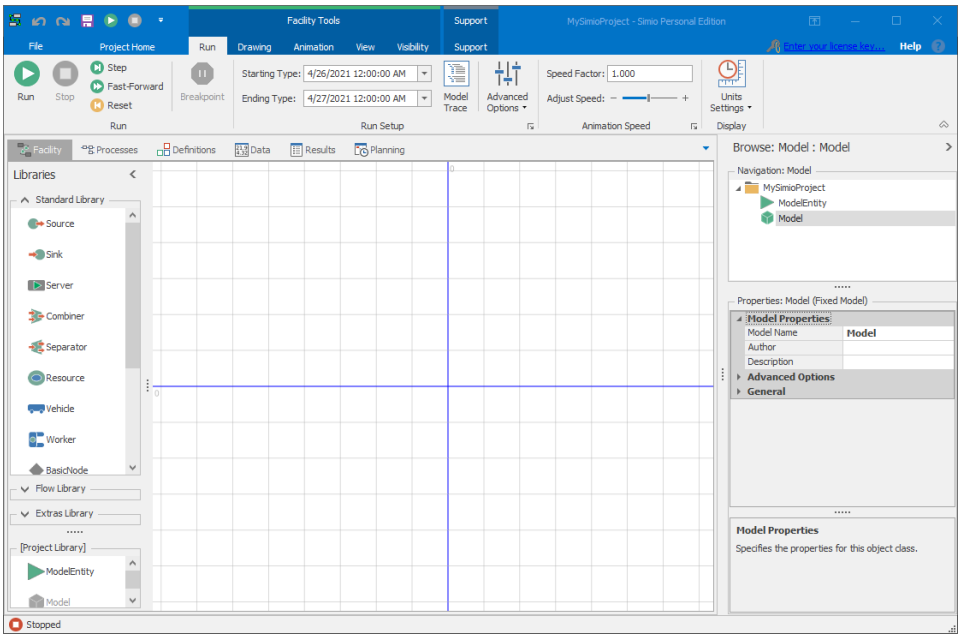


図 4.2　新規モデルの Facility ウィンドウ

4.1.2　Suooprt リボン

図 4.3 に示す Simio の Support リボンには、Simio を最大限に活用するための情報が収められている。モデルに関するアイディアや質問、問題点について、Simio のサポート技術者への連絡方法を確認できる。詳細な情報は、Simio Technical Support（http://www.simio.com/resources/technical-support/）で確認できる。このリンクから、技術支援のポリシーの説明や、Simio User Forum および Simio 関連の各種グループへアクセスできる。また、Support リボンではバージョンやライセンス情報も確認できる。サポートが必要な際には、このリボンの情報が特に重要である。

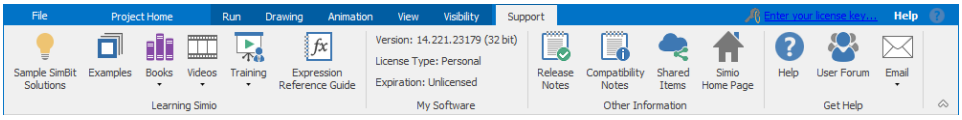


図 4.3　Simio の Support リボン

Simio ウィンドウの右上にある?アイコンまたは F1 キーを押すと、包括的なオンラインヘルプが表示される。印刷版が必要な場合は、Simio Reference Guide（PDF ファイル）を参照されたい。ヘルプとレファレンスガイドには、基本機能から応用機能まで詳しく解説されており、索引と検索機能を備えている。Support パネルでは、トレーニングビデオやオンラインリソースへのリンクを提供しているほか、サンプルプロジェクトや Simbits（後述）、Simio 関連書籍、リリース情報、ユーザフォーラムへのリンクも用意されている。

4.1.3　Project Model タブ

Simio プロジェクトを開くと、ウィンドウの上部のリボンタブに加えて、その直下に 2 つ目のタブセットが表示される。これらの Project Model タブを使用して、アクティブなモデルや実験に関連する複数のウィンドウを切り替えることができる。利用可能なウィンドウは選択したモデルのオブジェクトクラスによって異なるが、通常は Facility、Processes、Definitions、Data および Results が含まれる。RPS ライセンスがある場合は、Planning タブも利用できる。これらのウィンドウについては後ほど詳しく説明するが、まずはモデルの構築、実験、実行の中心となる Facility ウィンドウに焦点を当てて解説する。

4.1.4　オブジェクトライブラリ

Simio の**オブジェクトライブラリ**は、一般的なモデリング領域とテーマに焦点を当てたオブジェクト定義の集まりである。ここではライブラリの基本的な説明にとどめ、5.1.1.1 でオブジェクト、ライブラリ、モデルの関係について詳しく解説する。

ライブラリは Facility ウィンドウの左側に表示され、Standard ライブラリや Flow ライブラリ、Extras ライブラリがデフォルトで配置されている。Project ライブラリもプロジェクトに不可欠な要素として含まれている。Standard、Flow、Extras の各ライブラリを利用するには、ライブラリウィンドウ下部の該当する名称をクリックする（同時に開くことはできない）。Project ライブラリを常時表示するには、「...」のセパレータをドラッグして表示領域を調整する。その他のライブラリは、Project Home リボンの Load Library ボタンから追加できる。

Facility ウィンドウ左側の **Standard Object ライブラリ**は、Simio の標準オブジェクトを集めた汎用セットである。各オブジェクトは、モデル化する対象物、装置、またはアイテムを表現する。モデルの大部分は、Standard ライブラリからオブジェクトをドラッグし、Facility ウィンドウにドロップすることで構築できる。表 4.1 に Simio Standard ライブラリのオブジェクトを示す。

Project ライブラリには、現プロジェクトで定義されたすべてのオブジェクトが含まれている。プロジェクトで作成した新しいオブジェクト定義は、自動的に Project ライブラリに表示される。これらのオブジェクトは Navigation ウィンドウ（後述）で定義・更新し、Project ライブラリを通じて Facility ウィンドウに配置して利用する。モデリングを効率化するため、Project ライブラリには ModelEntity オブジェクトがあらかじめ組み込まれている。

Flow ライブラリは、フロー型の処理システムをモデリングするためのオブジェクト群である。**Extras ライブラリ**には、マテリアルハンドリングや倉庫関連のオブジェクトが収められている。詳細は Simio ヘルプを参照されたい。その他の専門分野向けライブラリは、Support リボンの Shared Items ボタンか、Simio User Forum を利用されたい。独自オブジェクトの作成方法は 11 章で解説する。

表 4.1　Simio Standard ライブラリのオブジェクト

オブジェクト	説明
Source	指定されたタイプと到着パターンに従ってエンティティを生成する。
Sink	プロセスを完了したエンティティをモデルから破棄する。
Server	機械やサービス作業などの処理能力を持つプロセスを表現する。
Combiner	複数のエンティティを親エンティティと結合する（例：パレタイズ）。
Separator	エンティティのバッチを分割する、または単一エンティティを複製する。
Resource	他のオブジェクトから占有され、解放される汎用的なオブジェクト。
Vehicle	固定経路または需要に応じて、運搬作業を行うトランスポータ。
Worker	人的作業をモデル化する。移動可能なリソースまたはトランスポータとして機能し、シフトスケジュールに従う。
BasicNode	複数のリンクが交差する単純な接続点。
TransferNode	目的地と移動手段の変更が可能な高機能な接続点。
Connector	2 つのノード間を瞬時に接続するリンク。
Path	エンティティが独自の速度で移動できるリンク。
TimePath	すべてのエンティティが指定時間で移動するリンク。
Conveyor	蓄積型・非蓄積型のコンベヤシステムを表現するリンク。

4.1.5　プロパティウィンドウ

右下の **Properties ウィンドウ**では、選択中のオブジェクトやアイテムのプロパティ（特性）を表示・編集できる。たとえば、Facility ウィンドウに Server を配置して選択すると、そのプロパティを確認・変更できる（図 4.4）。グレーのバーは、プロパティのカテゴリやグループを示している。よく使用するカテゴリは初期状態で展開され、すべてのプロパティが表示される。使用頻度の低いカテゴリは折りたたまれているが、「+」印をクリックして展開できる。プロパティ値を変更すると、その値は太字で表示され、カテゴリが自動的に展開されてデフォルト値との違いが分かりやすくなる。プロパティをデフォルト値に戻すには、プロパティ名を右クリックして Reset を選択する。

4.1.6　ナビゲーションウィンドウ

Simio **プロジェクト**は、複数のモデル、オブジェクト、シンボル、実験などのコンポーネントで構成される。これらのコンポーネントは右上の Navigation ウィンドウで管理する。新しいコンポーネントを選択すると、関連するタブとリボンが自動的に更新される。たとえば、Navigation ウィンドウで ModelEntity を選

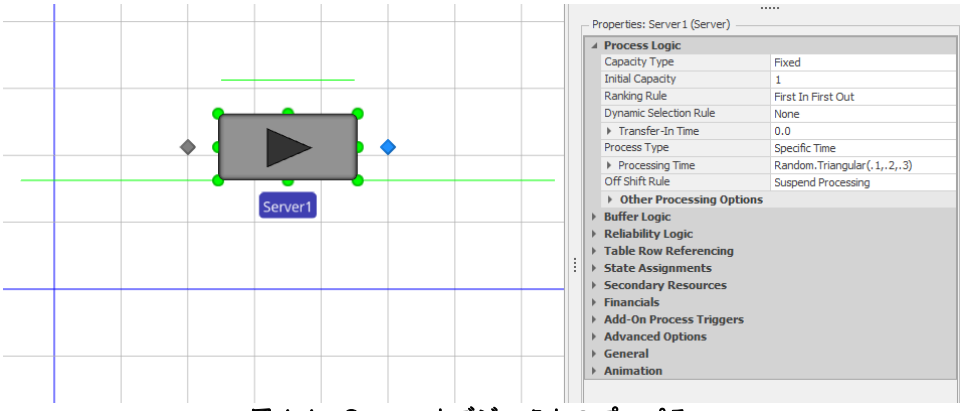


図 4.4　Server オブジェクトのプロパティ

択すると、プロジェクトモデルの新しいタブセットが表示される。このとき、Definitions タブでエンティティオブジェクトの状態を追加したり、Model オブジェクトを選択してメインモデルを編集したりできる。

現在作業中のオブジェクトは、Navigation ウィンドウの上部にあるタイトルバーまたはハイライトされたバーで確認できる。初心者ユーザは、同じオブジェクトがライブラリと Navigation ウィンドウの両方に表示されることで混乱することがある。ここで重要なのは、オブジェクトはライブラリに**置かれ**、Navigation ウィンドウで**定義し修正される**という役割の違いである。詳細は 5.1.1.1 で説明する。

4.1.7　SimBits

SimBits は、もっとも役立つ機能の 1 つである。これは、コンパクトで分かりやすいモデル集で、モデリングの概念や一般的な問題の解決方法を示している。各モデルには、PDF ファイルやオンラインヘルプによる詳細な解説が付属している。メニューの Open から直接読み込むことができるが（現在のモデルは置き換えられる）、より便利な方法は、Support リボンの SimBit ボタンを使用することである。このボタンからターゲットページに進むと、フィルタリング機能付きの SimBits 分類リストが表示され、必要なモデルを素早く検索して読み込める（新しい Simio ウィンドウが起動し、作業中のワークスペースは保持される）。SimBit を使えば、モデリング技術、オブジェクト、構成要素に関する知識を効率的に習得できる。

4.1.8　ウィンドウとタブの移動・構成

これまで初期設定のウィンドウ配置について説明してきたが、各ウィンドウの位置は簡単に変更できる。設計・実験時のウィンドウとタブ（Processes ウィンドウやデータテーブルタブなど）は、右クリックやドラッグ操作で別の位置に移動できる。ドラッグ中には、**レイアウトターゲット**と呼ばれる 2 組の矢印が表示される。1 組はウィンドウの中心に、もう 1 組は外側に表示される。図 4.5 は、テーブルタブをドラッグした際のレイアウトターゲットを示している。タブをいずれかの矢印にドロップすると、その位置に新しいウィンドウとして配置される。

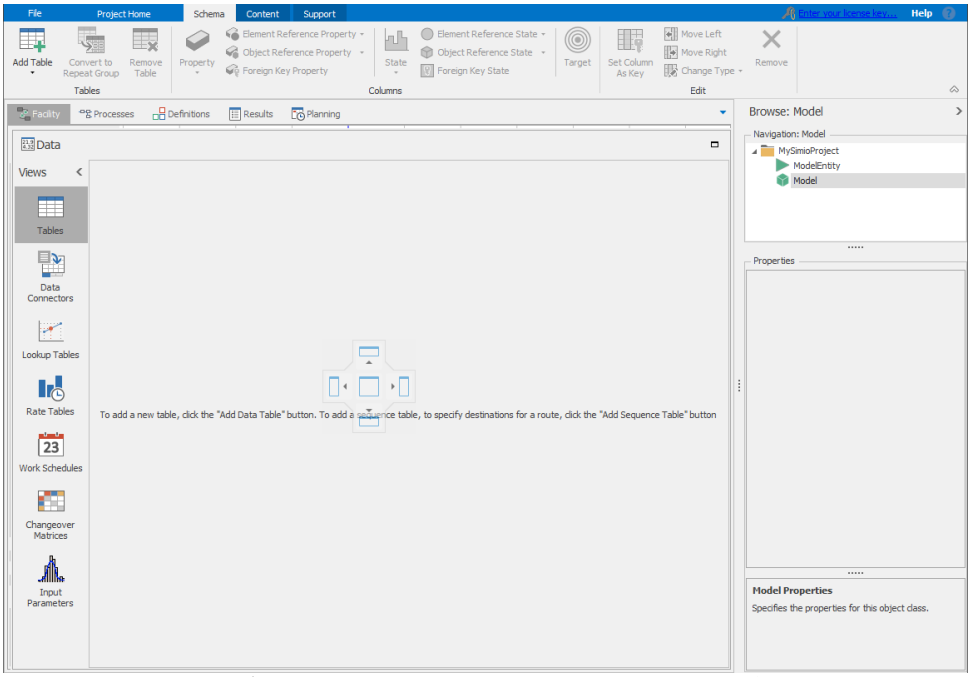


図 4.5　タブ化したウィンドウを新しい表示位置へドラッグする様子

タブを右クリックして適切なオプションを選択すると、ウィンドウを水平・垂直のタブグループに配置できる。また、画面を効率的に使用するために、Search、Watch、Trace、Errors、オブジェクトの Consoles などのウィンドウを Simio アプリケーションの外部や別のモニタにドラッグすることも可能である。ウィンドウ配置をカスタマイズした後、元に戻したい場合や、ウィンドウの位置がわからなくなった場合は、Project Home リボンの Reset ボタンをクリックすれば初期設定に戻る。

4.2　Model 4-1：Standard ライブラリオブジェクトを用いた最初のプロジェクト

本節では、Simio で基本的なモデルを構築し、複数の実験と分析を行う。4.2.1 項では、Simio を用いたモデル構築の方法を解説する。Standard ライブラリを使用して Facility ビューでモデルを作成し、1 回実行して結果を確認する。4.2.2 項では、標準的な待ち行列理論による期待値と比較するため、システムに関する初期の非公式な実験を行う。4.2.3 項では、シミュレーション出力の分析方法、統計的な繰り返しの概念、および Simio ヘルプの活用方法を紹介する。4.2.4 項では、長時間実行と短時間実行について考察する。長期的なモデルの挙動に関心がある場合、モデルの**ウォームアップ**が必要となることを説明する。4.2.5 項では、4.2.2 項で検討した問題を再度取り上げ、特にモデルの妥当性を検証する。ここでは、シミュレーション出力データのウォームアップや統計分析といった高度なツールを使用して、興味のある単一の**シナリオ**（システム構成）を詳しく分析する。さらに、5.5 節および 9.1.1 項では、代替シナリオの比較という一般的な目標につい

て議論し、追加の統計的手法を紹介する。

4.2.1　モデルの構築

Simio でモデルを作成する際は、Standard ライブラリオブジェクトを使用するのが一般的である。これらの既定のオブジェクトは、幅広いモデリングニーズに対応できる。図 4.6 は、Simio の Facility ビューで作成した待ち行列システムの完成モデルである（Facility タブが、Project Model タブエリアでハイライトされている）。以下、このモデルの構築手順を説明する。

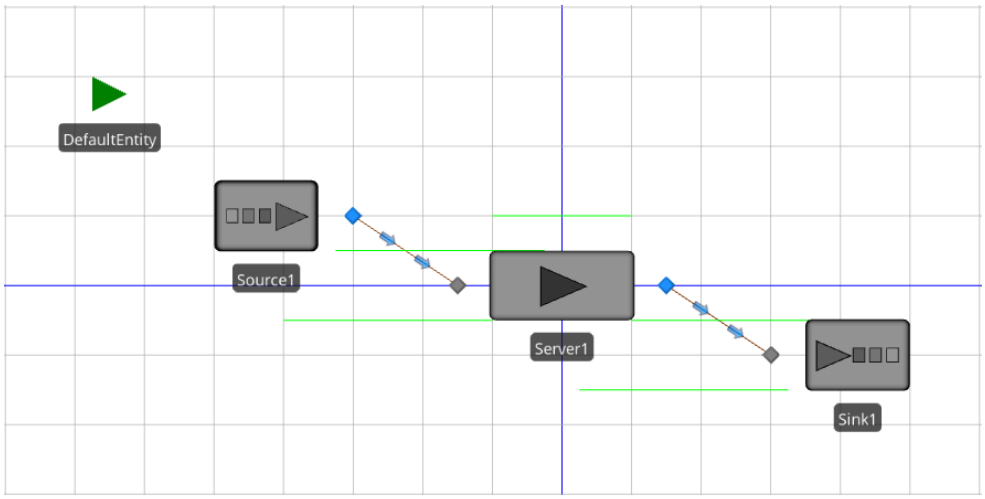


図 4.6　単一サーバ待ち行列システム（Model 4-1）のモデル（Facility ウィンドウ）

待ち行列モデルは、エンティティ、エンティティ到着プロセス、サービスプロセス、離脱プロセスで構成される。Facility ビューでは、**Source**、**Server**、**Sink** オブジェクトを使用してこれらのプロセスをモデル化する。

モデル作成の最初のステップは、Simio アプリケーションの起動である。File ページから必要に応じて New をクリックする（File リボンから利用可能）。新規モデルが開いたら、Facility タブをクリックして Facility ビューを表示し、左の Libraries バーの Standard Library セクションをクリックしてライブラリを開く。図 4.2 でこの画面を確認できる。

まず、Project Library パネルから ModelEntity オブジェクトを選択し、Facility ウィンドウにドラッグ＆ドロップする（Project Library パネルで定義されているため、実際にはその**インスタンス**をドラッグ＆ドロップすることになる）。次に、Standard Library から Source オブジェクトを選択し、同様に Facility ウィンドウにドラッグ＆ドロップする。続いて、Server オブジェクトと Sink オブジェクトのインスタンスも配置する。

次に、Source、Server、および Sink オブジェクトを接続する。ここでは標準の **Connector** オブジェクトを使用する。これは**ノード**間のエンティティの移動が瞬時（ゼロシミュレーション時間）で行われることを表す。Standard Library の Connector オブジェクトをクリックすると、カーソルが十字に変わる。Source オブジェクトの Output Node（右側）をクリックし、次に Server オブジェクトの

Input　Node（左側）をクリックして接続する。これにより Source から Server へのエンティティの即時移動が設定される。同様に、Server の Output　Node と Sink の Input　Node の間にも Connector を追加する。図 4.7 は、Source と Server オブジェクトが接続されたモデルを示している。

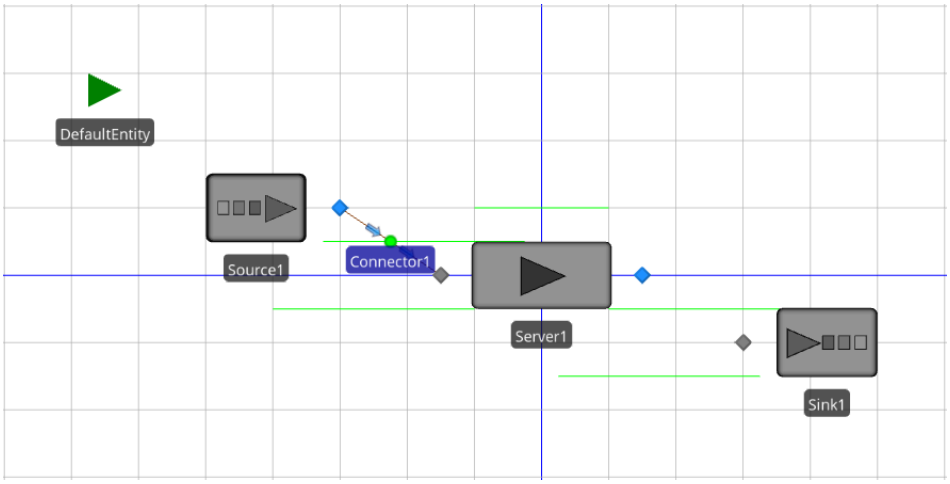


図 4.7　Connector オブジェクトで Source と Server がリンクされた Model 4-1

ModelEntity インスタンスの配置は技術的には必須ではない。これは、手動でモデルに追加しなくても Simio が自動的にエンティティインスタンスを追加するためである（ModelEntity オブジェクトが、ここで使用している他のオブジェクトのように Standard Library ではなく、Project Library にある理由）。ただし、この単純なモデルに ModelEntity、Source、Server、Sink、および Connector のインスタンスが含まれているという考え方を明確にするために、ここでは明示的にインスタンスを追加している。

ここで、モデルを保存するのに適したタイミングである（「早めに保存して、こまめに保存する」という基本原則に従う）。3.2 節で示した例題のファイル名規則に従い、名前を Model_04_01.spfx とした（spfx は Simio プロジェクトファイルの拡張子である）。すべての完成した例題のファイルは、本書のウェブサイトから入手できる（付録 C 参照）。

モデルの構築を進める前に、Standard ライブラリの一部のオブジェクトにデフォルトで含まれている**待ち行列**について説明する必要がある。これらの待ち行列は、図 4.7 に示すように水平の緑の線で表示される。Simio は、エンティティが**待つ**可能性のある場所（つまり、シミュレーション時間の一定期間、モデル上の同じ論理的なところに留まる場所）に待ち行列を設ける。なお、技術的には、エンティティではなく**トークン**が Simio の待ち行列で待機しているが、この詳細については 5 章で説明する。ここではアニメーション表示のため、待ち行列で待機しているものをエンティティとして考えてほしい。Model 4-1 には、以下の待ち行列が含まれる：

・Source1 OutputBuffer.Contents：Source オブジェクトからの出力待ちのエ

ンティティを保持する。
・Server1 InputBuffer.Contents：Server オブジェクトへの入力待ちのエンティティを保持する。
・Server1 Processing.Contents：現在 Server オブジェクトで処理中のエンティティを保持する。
・Server1 OutputBuffer.Contents：Server オブジェクトからの出力待ちのエンティティを保持する。
・Sink1 InputBuffer.Contents：Sink オブジェクトへの入力待ちのエンティティを保持する。

図 4.1 の単一サーバ待ち行列システムでは、1 つの待ち行列のみを示している。この待ち行列は、Server1 オブジェクトの InputBuffer.Contents 待ち行列である。Server1 オブジェクトの Processing.InProcess 待ち行列は、各シミュレーション時刻において処理中のエンティティを保持するものである。他の待ち行列は、このモデルでは使用されない（エンティティはゼロシミュレーション時間で即座に移動するためである）。

モデルの基本構造が完成したので、次にオブジェクトのパラメータを設定する。このモデルでは、エンティティの到着時間間隔とサービス時間を決定する**確率分布**を指定する必要がある。Source オブジェクトは、指定された到着過程に従ってエンティティを生成する。毎時λ=48 エンティティのポアソン到着過程を想定し、エンティティの到着時間間隔は平均 1.25 分の指数分布に従うよう Simio で設定する（**到着時間間隔**=1.25=60/48 は毎時 48 の**到着率**に相当する）。Simio オブジェクトモデルでは、到着時間間隔は Source オブジェクトの**プロパティ**の 1 つとして設定する。これらのプロパティは Properties ウィンドウで設定・編集できる。Source オブジェクトをクリックして選択し、右パネルに Properties ウィンドウを表示する（図 4.8）。

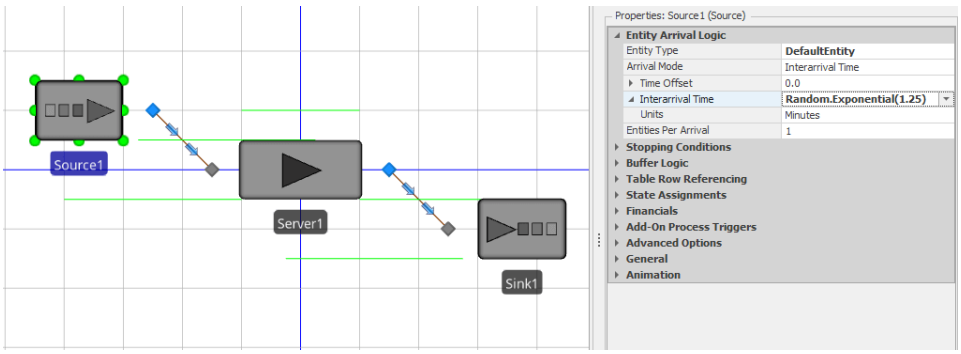


図 4.8　Source オブジェクトの到着時間間隔分布の設定

Source オブジェクトの到着時間間隔分布は、Interarrival Time プロパティを Random.Exponential(1.25)に、Units プロパティを Minutes と設定する。Units プロパティを表示するには、Interarrival Time の左側の矢印をクリックし、右側のプルダウンから Minutes を選択する。これにより、平均 1.25 分の指数分布に従って**次の**エンティティ生成時間が決定され、結果として時間当たりのエンテ

ィティ到着率$\lambda = 60 \times (1/1.25) = 48$が設定される。Random キーワードを使用した乱数生成については、4.4 節で詳しく説明する。Time Offset プロパティは最初のエンティティの生成時刻を決定するもので、通常は 0 に設定する。その他の Arrival Logic 関連のプロパティは、現時点ではデフォルト値のままとする。これらの設定により、シミュレーション実行中、エンティティが継続的に生成される。

オブジェクト名は変更可能である。デフォルトの名前（例：Source1）を変更するには、オブジェクト下のネームタグをダブルクリックするか、General プロパティセクションの Name プロパティを編集する。また、F2 キーを使用してほとんどの Simio アイテムの名前を変更できる。General セクションには、Description プロパティも含まれている。これはモデルの文書化に役立つため、各オブジェクトに重要な説明を記入することが望ましい。そこで入力された説明は、オブジェクトにマウスを合わせると、ツールチップとして表示される。

モデルの待ち行列のロジックを完成させるため、Server オブジェクトのサービスプロセスを設定する必要がある。Server モジュールの Processing Time プロパティでエンティティの処理時間を指定する。このプロパティは、Units プロパティを Minutes に設定し、Random.Exponential(1)とする。なお、ここで変更するのは Processing Time プロパティのみで、Process Type プロパティは変更しない（Process Type はデフォルトの Specific Time のままとする。他のオプションについては 10.4 節で説明する）。最後のステップとして、モデルの実行時間を 10 時間に設定する。この設定は、Run リボン／タブをクリックし、Ending Type プルダウンで Run Length オプションを選択して 10 Hours と入力する。

初期モデルを実行する前に、実行速度を設定する。**Speed Factor** はモデルの対話型実行の速度を制御するために使用する。Speed Factor を 50 に変更すると（Run リボンの Speed Factor フィールドで設定）、このモデルをより見やすい速度で実行できる。対話型実行での**最適**な Speed Factor は、コンピュータの性能、モデルやオブジェクトのパラメータ、個人の好みによって異なる。そのため、各モデルに応じて Speed Factor を調整することを推奨する（技術的には、Speed Factor は秒単位でのアニメーションフレーム間のシミュレーション時間間隔を表す）。

リボンの左側上部にある Run アイコンをクリックすることで、モデルを実行することができる。このモデルは**対話型**で実行している。モデルが実行されているとき、アプリケーションのフッター部分には、シミュレーションされた時間と実行済みパーセンテージが表示される。デフォルトの Speed Factor では、シミュレーション時間はかなりゆっくり進行するが、モデルの実行中も変更することができる。シミュレーション時刻が設定した実行期間の 10 時間に達すると、モデル実行は自動的に止まる。

対話型実行では、モデルを停止・一時停止するか、タブバーの Results タブをクリックすることで、モデルの結果をリアルタイムで確認できる。ここでは、モデルを 10 時間まで実行して、結果を見ることとする。Simio は基本的な結果を表示する方法として、Pivot Grid と Reports の 2 つを提供している。これらは左パネ

ル上部のアイコンをクリックして切り替えることができる。図 4.9 は、Model 4-1 を 10 時間実行後に停止した場合の Pivot Grid 表示である。

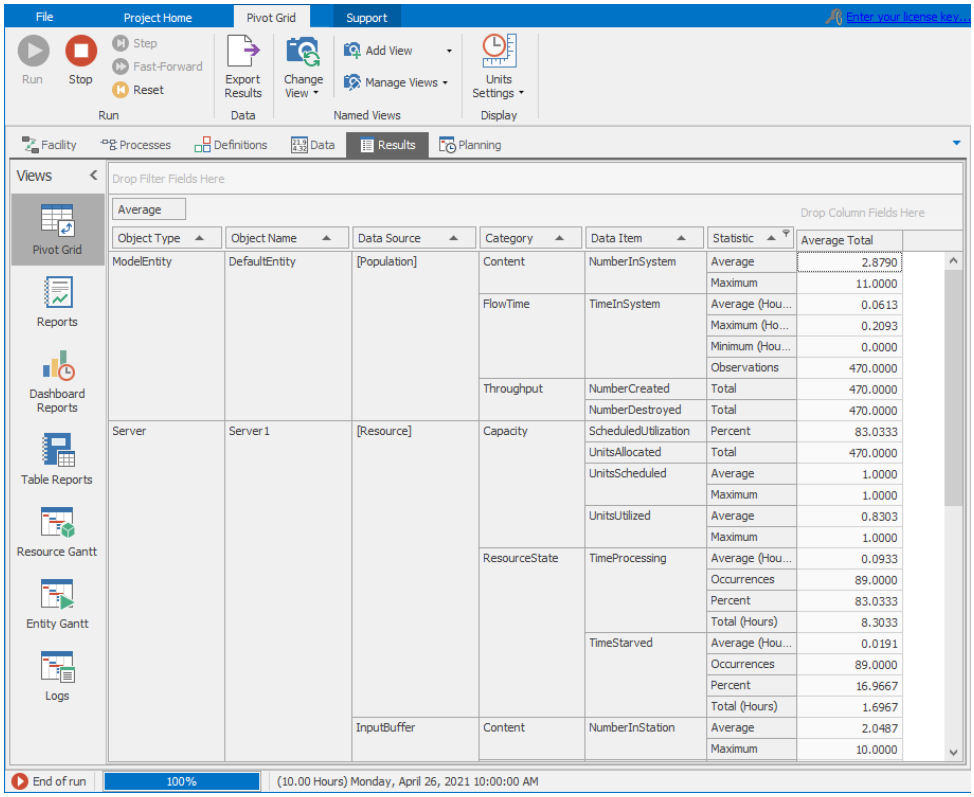


図 4.9　Model 4-1 の Pivot Grid レポート

Simio はアジャイル開発過程を用いており、頻繁に小さなアップデートや、時々主要なアップデートがある。そのため、対話型で例題を実行した際の出力値は、本書執筆時に得られた出力値と**合っていない**可能性がある。1.4 節で述べたとおり、これは同時に起こるイベントが処理される順序のような低レベルの挙動に関して、リリースされたバージョンごとにわずかな差異が存在することに起因する。これらの差異の理由にかかわらず、それらの存在によって、シミュレーション実験では適切な統計的設計と分析をすること、そして「答え」を得るために一度だけの実行で留めないことが重要である。これは、本書で繰り返し指摘しているポイントである。

Pivot Grid の形式は、非常に柔軟性があり、特定の結果を見つけるのに非常に迅速な手法を提供している。このタイプのレポートに慣れていないとはじめは大変そうに感じるが、それを使い始めると、すぐにそのありがたみがわかるだろう。また、Pivot Grid レポートは、容易に CSV（コンマ区切り値）形式のテキストファイルとして出力することができる。また、出力されたファイルは、Excel やその他のアプリケーションにインポートすることが可能である。

Pivot Grid のデフォルトでは、各列に以下の出力値が含まれている：

・Object Type
・Object Name
・Data Source
・Category
・Data Item
・Statistic

図 4.9 において、ModelEntity（Object Type）の DefaultEntity（Object Name）の TimeInSystem（Data Item）の Average（Statistic）は 0.0613 時間（0.0613 時間×60 分/時=3.6780 分）を示している。Pivot Grid の時間、長さ、割合の単位は、Pivot Grid リボンの Time Units、Length Units、Rate Units で設定可能である。Minutes に設定すると、TimeInSystem の Average は 3.6753 となり、手計算の 3.6780 分との間に微小な丸め誤差があることがわかる。また、TimeInSystem（Data Item）は FlowTime（Category）に属しており、エンティティ（動的オブジェクト）に基づく値であるため、Data Source は Dynamic Objects となっている。

画面で Pivot Grid をスクロールすると、この小さなモデルでも多くの出力パフォーマンス指標が確認できる。たとえば、TimeInSystem の Average 値 0.0613 時間から 3 行下を見ると、Throughput（Category）で合計 470 のエンティティが作成され（Source オブジェクトを通してモデルに進入）、その下の行では同じく 470 のエンティティが破棄された（Sink オブジェクトを通過して離脱）ことがわかる。このモデルの 10 時間実行では、到着した 470 のエンティティがすべて離脱したことになり、シミュレーション終了時にエンティティは存在しない。これは 10 時間実行後に一時停止し、アニメーションで確認できる（ただし、実行時間を 9 時間や 11 時間に変更すると、必ずしもこのような結果にはならない。これは Pivot Grid の Throughput（Category）で NumberCreated と Number Destroyed を見ても確認できる）。10 時間実行での平均システム時間 0.0613 時間は、これら 470 エンティティの個々のシステム時間の単純平均である。

シミュレーション実行期間をさまざまに変更する際は、8 分に変更してみてほしい。そして、3.3.1 節の図 3.8 で示した手動シミュレーションの結果と Pivot Grid 結果を比べてほしい。実は、手動シミュレーションで使用した「不思議な」到着時間間隔とサービス時間は、この 8 分の Simio 実行から生成され、Model Trace 機能を用いて数値を記録したものである。

Pivot Grid は 3 種類の基本的なデータ操作をサポートしている：

・グルーピング（Grouping）：列見出しを異なる位置にドラッグすることで、データのグループ化が変化する。
・ソーティング（Sorting）：列見出しをクリックすることで、その列の内容に基づいてデータを並べ替えする。
・フィルタリング（Filtering）：列見出しの右上隅にマウスをかざすと漏斗形のアイコンが表示される。このアイコンをクリックすると、データフィルタリン

グ用のダイアログが表示される。フィルタを適用した列には漏斗アイコンが表示される。フィルタリング機能により、データ量にかかわらず、必要な情報のみを閲覧することが可能である。

Pivot Grids では、フィルタ、ソート、分類の設定を複数の画面（**ビュー**）として保存できる。これらのビューは、パフォーマンス評価尺度の監視に役立つ。Simio のドキュメントでは、Pivot Grid の具体的な機能の使用方法について詳しく説明している。Pivot Grid 形式は、特に多数の出力列がある場合の情報検索に非常に効果的である。

Reports 形式では対話型実行の結果を表示する。このレポートは体系的に整理されており、詳細な形式での表示、印刷、他のファイル形式への出力、電子メール送信に対応している（これらのオプションは Ribbon の上の Print Preview タブから利用できる）。図 4.10 は、Print Preview タブが開いた状態の Reports 形式を示している。左の見出しを `TimeInSystem-Average(Hours)` まで進めると、0.06126 という値が表示される。この値は図 4.9 の Pivot Grid での出力値と（四捨五入の範囲内で）一致している。

4.2.2　初期の実験と分析

最初の Simio モデルが完成したので、待ち行列システムを理解するための初期実験と分析を行う。前述のように、システムの長時間実行や定常状態のパフォーマンスは、待ち行列解析で分析できる（詳細は 2 章を参照）。ただし、この種の正確な解析は単純なモデルにしか適用できないことに注意が必要である（これがシミュレーションを使用する理由である）。表 4.2 では、定常状態の待ち行列理論による結果と、図 4.9 の Pivot Table から得られたシミュレーション結果を比較している。

待ち行列の列とモデルの列の数値が異なることにすぐ気づくであろう。しかし、その違いの理由を説明する前に、より重要な問題について考察する必要がある。Facility ウィンドウに戻り（リボンの下の Facility タブをクリック）、モデルをリセットし（Run リボンの Reset アイコンをクリック）、再実行する。10 時間実行して停止させ、Pivot Grid を確認すると、前回の実行結果（図 4.9）とまったく同じ値が表示されることがわかる。このプロセスを何度繰り返しても、常に同じ結果が得られる。3.1.3 項で説明したように、モデルではエンティティの到着時間間隔とサービス時間に乱数を使用しているため、シミュレーション初心者にとって、これは不可解に思われるであろう。以下に、このコンピュータシミュレーションに関する重要な点を説明する：

1. 3.1.3 項で述べ、6.3 節で詳しく説明するように、使用される乱数は真にランダムではない。この文脈では、**疑似乱数**を使用しており、生成される数値の順序は決定的である。
2. 6.4 節で乱数生成プロセスについて説明するが、多くのシミュレーションソフトウェアでは疑似乱数生成を制御でき、この特性を活用することが可能である。

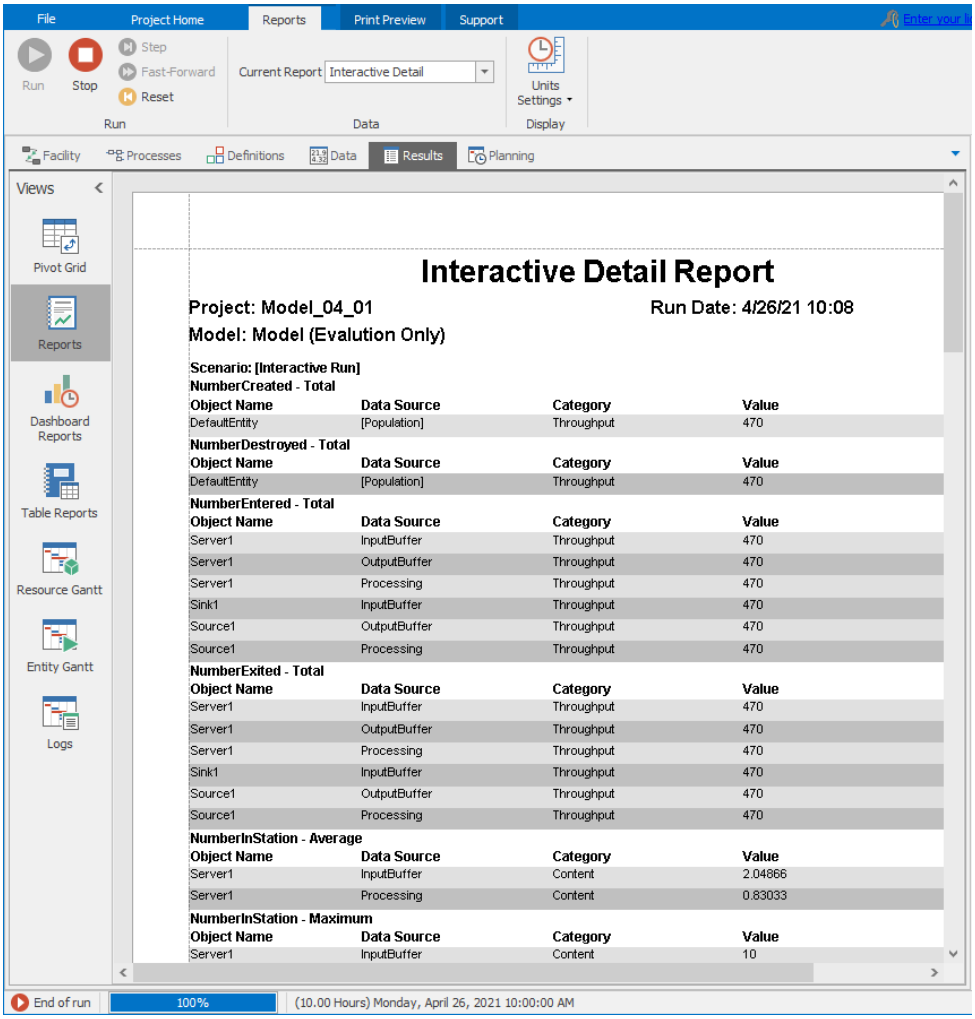


図 4.10　Model 4-1 の標準レポートビュー

表 4.2　待ち行列理論と初期モデルの結果比較

測定基準	待ち行列理論	モデル
稼働率(ρ)	0.800	0.830
システム内数(L)	4.000	2.879
待ち行列数(L_q)	3.200	2.049
システム内時間(W)	0.083	0.061
待ち時間(W_q)	0.067	0.044

「予測可能な乱数」という概念は、シミュレーションを初めて学ぶ人にとって不安を感じさせるかもしれない。しかし、この予測可能性は実際にはシミュレーションにとって有益である。これは、シミュレーション課題の採点を容易にするだけでなく（著者にとって重要）、より重要なモデルの**デバッグ**にも役立つ。たとえば、モデルに変更を加えた際、同じ「乱数」入力を使って同じシミュレーション

を実行することが**できる**ため、出力の変化（または変化の欠如）が、乱数の違いではなく、モデルの変更によるものだと明確にわかる。モデリングを進めていくと、デバッグに多くの時間を費やすことになるため、この機能の有用性を実感するだろう（4.9 節でデバッグプロセスと Simio のデバッグツールについて詳しく説明する）。さらに、この予測可能性により、**分散減少**（variance reduction）と呼ばれる手法を用いてシミュレーションの実行時間を短縮できる。これについては、一般的なシミュレーションの教科書（Banks et al.（2005）や Law（2015）など）で解説されている。Simio のデフォルトでは、モデル実行時に毎回同じ乱数系列（到着時間間隔やサービス時間などのモデル入力値）を使用する。そのため、特に別の動作を指定しない限り、モデルをリセットして再実行しても、常に同じ結果が得られる。

ここで、シミュレーションモデルの初期結果と待ち行列理論の結果（表 4.2）が一致しない理由について考察しよう。この差異には、次の 3 つの可能性が考えられる：

1. Simio モデルに誤りがある。つまり、モデルの設定のどこかが間違っている。
2. 期待値の解釈が誤っている。つまり、シミュレーション結果が待ち行列理論の結果と**一致するべきである**という前提自体が間違っている。
3. **サンプリング誤差**の影響がある。つまり、シミュレーション結果が確率的な意味での期待値と整合していても、じゅうぶんな実行時間や反復回数（同じ初期状態から開始するが、異なる乱数を使用する独立した実行）を確保していないか、結果の解釈を誤っている可能性がある。

実際、期待値とシミュレーション結果を比較して一致しない場合、モデルの種類に関係なく、**いつも**上記のいずれかの理由が考えられる。この事例では、期待値の解釈に問題があり、モデルの実行時間が不十分である。待ち行列理論の結果は、長時間実行による定常状態の値であることを思い出してほしい。つまり、理論上は無限の時間実行したシステムから得られる結果だが、このモデルではわずか 10 時間しか実行していない上、じゅうぶんな反復も行っていない（上記の項目 2 と 3 に関連）。期待値の算出とシミュレーション結果の比較、そして両者が収束するまでの繰り返しは、**モデル検証**と**妥当性の確認**において極めて重要な要素となる（4.2.5 項で詳しく説明する）。

4.2.3　反復と出力の統計分析

これまで示してきたように、**反復**とは、入力確率変数と乱数に対して、特定かつ個別の重複のない順列を用い、固定された開始・終了条件のもとでモデルを実行することを指す（このケースでは、到着時間間隔とサービス時間は指数分布に従う）。まず、開始と終了の状態は、シミュレーション時間によって定義されると仮定する（後述するが、他の種類の開始・終了状態の設定も可能である）。システムが空かつ遊休状態から開始し、10 時間実行すると、これが 1 回の反復となる。モデルをリセットして再実行すると、**同じ**反復が実行される。前述のように、**同じ**乱数が入力され、その結果として同じ確率変数が使用されるため、**同じ**結果が得ら

れる。**異なる**反復を実行するには、互いに異なる独立した乱数と確率変数を設定する必要がある。Simio ではこのプロセスを自動的に処理するが、対話型モードでは複数の反復実行はできない。そのため、Simio Experiment を作成して実行する必要がある。

Simio Experiments では、指定した回数の反復実行が可能である。各反復で使用される乱数は重複しないため、Simio が生成する**確率変数**は互いに統計的に独立している。これは統計分析において重要な特性である。実験を設定するには、Project ホームリボンから New Experiment アイコンをクリックする。新しい実験では、上部の Replications Required と右側の Default Replications をデフォルトの 10 から 5 に変更すると、図 4.11 のような Experiment Design ビューのようになる。

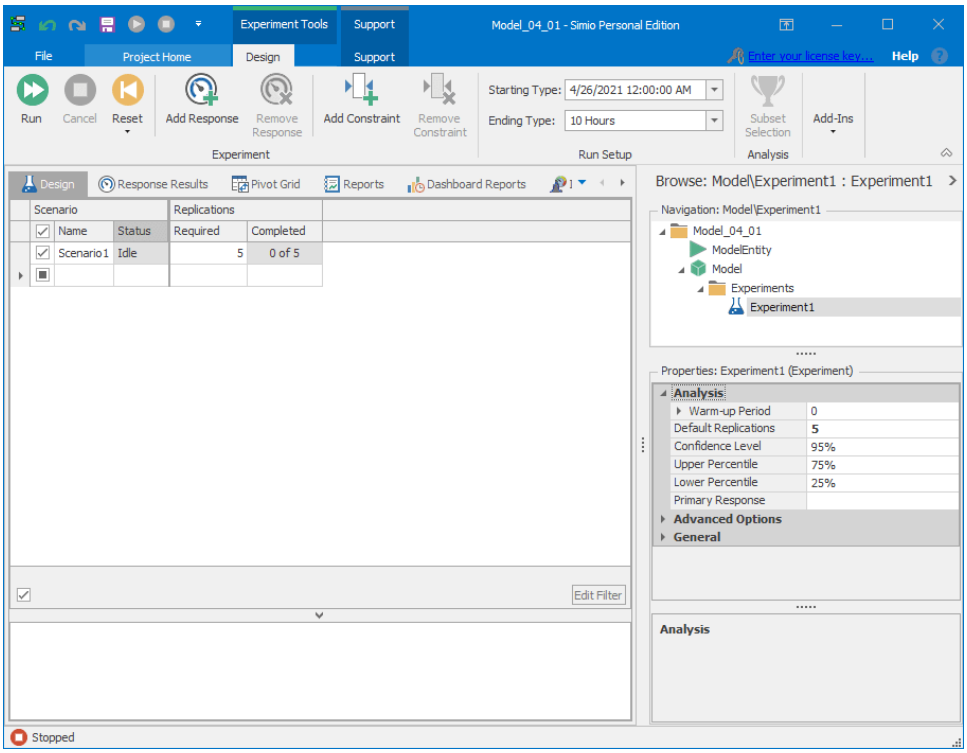


図 4.11　5 回の反復実行の初期実験計画

実験を実行するためには、Scenario1（デフォルト名）の列を選択し、Run アイコンをクリックする（Model ウィンドウの 1 つの白い右矢印ではなく、Experiment ウィンドウの 2 つの白い右矢印のアイコン）。Simio が 5 回の反復を実行した後、Pivot Grid レポート（図 4.12）を選択して表示する。

対話型実行時の Pivot Grid（図 4.9）と比較すると、新たに Minimum（最小値）、Maximum（最大値）、Half Width（半幅：期待値の 95%信頼区間で、Experiment Design プロパティの Confidence Level で編集可能）の列が追加されており、これは各出力統計値に 5 つの独立した観測値があることを反映している。

Design | Response Results | Pivot Grid | Reports | Dashboard Reports | Input Analysis

Drop Filter Fields Here

Average | Minimum | Maximum | Half Width

Object Ty...	Object Name	Data Source	Category	Data Item	Statistic	Scenario 1 Average	Minimum	Maximum	Half Width
ModelEntity	DefaultEntity	[Population]	Content	NumberInSystem	Average	3.6592	2.2958	6.7441	2.2029
					Maximum	15.8000	11.0000	28.0000	8.6636
			FlowTime	TimeInSystem	Average (Hou...	0.0762	0.0489	0.1306	0.0395
					Maximum (Ho...	0.2888	0.2018	0.5096	0.1601
					Minimum (Hou...	0.0003	0.0000	0.0009	0.0005
					Observations	464.4000	46.0000	479.0000	15.5208
			Throughput	NumberCreated	Total	471.8000	48.0000	507.0000	26.7909
				NumberDestroyed	Total	464.4000	46.0000	479.0000	15.5208
Server	Server1	[Resource]	Capacity	ScheduledUtilization	Percent	78.7339	76.3323	83.0333	3.2689
				UnitsAllocated	Total	465.2000	47.0000	480.0000	15.3912
				UnitsScheduled	Average	1.0000	1.0000	1.0000	0.0000
					Maximum	1.0000	1.0000	1.0000	0.0000
				UnitsUtilized	Average	0.7873	0.7633	0.8303	0.0327
					Maximum	1.0000	1.0000	1.0000	0.0000
			ResourceState	TimeProcessing	Average (Hou...	0.0794	0.0578	0.0933	0.0166
					Occurrences	101.4000	89.0000	132.0000	21.7982
					Percent	78.7339	76.3323	83.0333	3.2689
					Total (Hours)	7.8734	7.6332	8.3033	0.3269
				TimeStarved	Average (Hou...	0.0214	0.0181	0.0243	0.0034
					Occurrences	100.6000	89.0000	131.0000	21.5851
					Percent	21.2661	16.9667	23.6677	3.2689
					Total (Hours)	2.1266	1.6967	2.3668	0.3269
		InputBuffer	Content	NumberInStation	Average	2.8718	1.5325	5.9590	2.2020
					Maximum	14.8000	10.0000	27.0000	8.6636
			HoldingTime	TimeInStation	Average (Hou...	0.0593	0.0327	0.1148	0.0401
					Maximum (Ho...	0.2668	0.1500	0.5058	0.1764
					Minimum (Hou...	0.0000	0.0000	0.0000	0.0000

図 4.12　5 回の反復実行の Experiment Pivot Grid

これらの**反復実行間**の出力統計について理解することが重要である。ここでは、エンティティの TimeInSystem 値の 3-5 行を具体例として見てみることとする：

・TimeInSystem の平均（時間）の平均値 0.0762 は、5 つの反復それぞれの平均時間の平均である。最初の値は図 4.9 の単一実行から得られた 0.0613 である。95%信頼区間は 0.0762±0.0395（つまり[0.0367, 0.1157]時間）で、これは**反復内**でのシステムの平均時間の**期待値**を 95%の信頼度で含んでいる。この値は、10 時間の無限回の反復実行における平均システム時間を表していると解釈できる。また、この信頼区間は、シミュレーションが出力する確率変数の確率分布における期待値をカバーしているとも考えられる（信頼区間の詳細は表 4.4 で説明する）。

・Average(Hours)行の 0.1306 は、5 回の反復実行における平均システム内時間の最大値を示している。つまり、5 つの Average TimeInSystem 値の中でもっとも大きい値が 0.1306 であり、これが平均の最大となる。

・Maximum(Hours)行の 0.2888 は、5 回の反復におけるシステム内エンティティ時間の最大値の平均である。95%信頼区間 0.2888±0.1601 は、無限回の反復実行における最大システム内時間の期待値をカバーしようとしている。

・システム内時間の実際の最悪ケースとしてもっとも重要な値は、5 回の反復実行全体での最大値である 0.5096 時間である。

表 4.3 は、5 回の反復実行における待ち行列の測定基準とその標本平均・標本標準偏差を示している。

表 4.3　初期モデルの 5 回の反復実行データ

推定された測定基準	反復					平均	標準偏差
	1	2	3	4	5		
稼働率(ρ)	0.830	0.763	0.789	0.769	0.785	0.787	0.026
システム内数(L)	2.879	2.296	3.477	2.900	6.744	3.659	1.774
待ち行列数(L_q)	2.049	1.532	2.688	2.131	5.959	2.872	1.774
システム内時間(W)	0.061	0.049	0.075	0.065	0.131	0.076	0.032
待ち時間(W_q)	0.044	0.033	0.058	0.048	0.115	0.059	0.032

反復出力値を確認するには、Pivot Grid リボンの Export Details アイコンをクリックする。また、平均値や標準偏差などの反復実行間の結果を得るには、Pivot Grid の Export Summaries をクリックする。出力データは CSV 形式で保存され、Excel などのアプリケーションでそのまま利用できる。この表から特に注目すべき点は、反復間で値が大きく異なること（特にLとL_qにおいて顕著）である。このバラツキが、単一の反復実行結果だけでは信頼できる推論が得られない理由を示している。

モデルへの入力（エンティティ到着時間間隔とサービス時間）がランダムであるため、シミュレーションの出力パフォーマンス測定基準（ρ、L、L_q、W、W_qのシミュレーションベースの推定値をそれぞれ$\hat{\rho}$、$\hat{L}$、$\widehat{L_q}$、$\widehat{W}$、$\widehat{W_q}$とする）は確率変数となる。待ち行列解析では、これらの正確な定常状態の値としてρ、L、L_q、W、W_qが与えられる。反復実行では、同じモデルに対して別々の独立した確率変数を入力し、各反復で$\hat{\rho}$、$\hat{L}$、$\widehat{L_q}$、$\widehat{W}$、$\widehat{W_q}$の観測値が 1 つずつ生成される。統計用語では、n回の反復実行により、各確率変数についてn個の互いに独立で同一の分布に従う（IID）観測値が得られる。これにより、反復実行の標本平均から確率変数の平均値を推計できる。つまり、表 4.3 の平均列の値は、対応する確率変数の期待値の推定値を表している。ただし、この表からは推定値の**よさ**を判断することはできない。しかし、標本平均は**一致**推定量（nと共に分散が減少する）であり、**大数の強法則**（nが大きくなるにつれ、反復による標本平均は各確率変数の期待値に確率 1 で収束する）が成り立つため、反復回数を増やすほど推定の精度は向上する。

表 4.4 では 50 回と 5 回の反復実行結果を比較している。反復回数を増やすことで推定精度の向上が期待されるが、平均値だけでは推定値の**品質（精度）**に関する具体的な情報は得られない。そこで必要となるのが、サンプリング誤差（平均は**点推定**である）についての知見を与える**区間推定**である。半幅列は区間推定を示しており、これらは通常の正規分布アプローチ（統計テキストの入門書で説明さ

れているように、標本標準偏差と自由度 n-1 のスチューデントの t 分布を使用）に基づいて構成された、平均値における 95%信頼区間の半幅値を表している。

表 4.4　5 回の反復実行と 50 回の反復実行の比較

推定された測定基準	5 回の反復実行		50 回の反復実行	
	平均	h	平均	h
稼働率(ρ)	0.787	0.033	0.789	0.014
システム内数(L)	3.659	2.203	3.794	0.433
待ち行列数(L_q)	2.872	2.202	3.004	0.422
システム内時間(W)	0.076	0.040	0.078	0.008
待ち時間(W_q)	0.059	0.040	0.062	0.008

Lの 95%信頼区間（CI）について、5 回と 50 回の反復実行を比較してみよう。

5 回の反復実行　：　3.659±2.203 または[1.456, 5.862]
50 回の反復実行 ：　3.794±0.433 または[3.361, 4.227]

5 回の反復実行に基づくと、$\hat{L}$の真の平均（期待値あるいは母平均）が 1.456～5.862 の範囲にあると 95%で**信頼**できるものである。一方、50 回の反復実行では、真の平均が 3.361～4.227 の範囲にあると 95%で**信頼**できるものである（厳密には、このような 95%信頼区間を繰り返し作成した場合、その区間が未知の真の平均を含むという解釈となる）。つまり、出力統計値の平均値の信頼区間は、サンプリング誤差の指標と、確率変数の真の平均の推定精度を示すものである。反復実行（サンプル）数を増やすことで、半幅を任意に小さくすることができる。たとえば、250 回の反復実行では[3.788, 4.165]という信頼区間が得られる。明らかに、250 回の反復実行による推定値のほうが、5 回の反復実行による推定値よりも信頼できるものである。

独立した反復実行で真の平均を正確に推定したい場合、必要な反復回数は信頼区間の半幅によって決まるものである。信頼区間の半幅の公式の分母は$\sqrt{n}$であるため、半幅を半分にするには約 4 倍の反復実行が、1/10 にするには約 100 倍の反復実行が必要となる。しかし、「どの程度の半幅でじゅうぶんか」（つまり、与えられたシミュレーションモデルと意思決定の状況において、アナリストやクライアントが許容できる半幅の値）について、明確な基準は存在しない。

コンピュータ実行時間とサンプリング誤差の削減には明確なトレードオフが存在する。じゅうぶんな反復実行で半幅を任意に小さくできるものの、実行時間にはコストがかかるものである。追加の反復実行を行うかどうかを決める際には、次の 2 つの問題が重要である：

1.　サンプリング誤差により誤った判断をした場合の損失はどの程度か？
2.　追加の反復実行に必要な時間は確保できるか？

表 4.2 のシミュレーション結果が待ち行列の理論値と一致しない第一の理由は、モデルの**単一反復実行**の結果のみを使用したことにある。これは、サイコロを 1

回だけ振って 4（または他の値）が出たときに、その値を多数回の試行による**期待値**だと主張するようなものである。明らかに、これは個々の試行とは無関係な、不十分な推定値である。残念ながら、重大なリスクがあるにもかかわらず、シミュレーションの初心者はしばしば 1 回の反復実行の結果だけを使用してしまう。より適切なアプローチは、複数回の反復実行を行い、出力統計量の推定値として標本平均を使用することである。また、真の平均の推定に関心がある場合は、適切な反復実行回数を決定する際に 95%信頼区間の半幅を活用する。そして結果の分析では、単なる平均値（点推定）ではなく、信頼区間（区間推定）も併せて使用する。実験（図 4.12）の標準 Simio Pivot Grid レポートは、すべての出力統計量について標本平均と 95%信頼区間の半幅を自動的に提供することで、このアプローチを支援している。

期待値とモデル結果の不一致の第二の理由は、より複雑で、モデルの**ウォームアップ期間**の必要性に関係している。次節ではこの点について詳しく検討する。

4.2.4　定常状態と終結状態のシミュレーション

一般的に、待ち行列または待ち行列ネットワークの要素を含むシミュレーションモデルは、「**空かつ遊休**」状態で開始する。これは、システム内にエンティティが存在せず、すべてのサーバが稼働していない状態を意味する。単一サーバの待ち行列モデルを例に考えてみよう。最初に到着するエンティティは、サーバを**決して待つ必要はない**。また、2 番目に到着するエンティティは（前にいるのは最初のエンティティだけなので）、100 番目に到着するエンティティと比べて待ち行列での待ち時間は短くなるだろう。モデル化されるシステムの特性（この場合はサーバ稼働率の期待値）によっては、3 番目、4 番目、5 番目などのエンティティの待ち時間の分布や期待値が、**定常状態**の値とは大きく異なる可能性がある。ここでの定常状態とは、空かつ遊休の**初期状態**の影響が実質的に消失するまでじゅうぶんな時間が経過した後の状態を指す。実行開始時からモデルが「実質的な」定常状態に達するまでの期間を**初期過渡期間**と呼ぶ。

表 4.2 の結果を得るために使用した基本的な待ち行列解析（2 章参照）は、**定常状態**におけるシステムの正確な期待値を与える。しかし、これまで述べたように、ほとんどの待ち行列ネットワークのシミュレーションモデルは、定常状態に達する前に初期過渡期間を経験する。この初期過渡期間中のモデル統計を記録し、反復要約統計表でこれらの観測値を使用すると、**初期のバイアス**が生じる可能性がある。つまり、実測値が理論値と一致しない場合がある。例として、モデルの実行期間を 2、5、10、20、および 30 時間に設定した 5 つの実験を、それぞれ 500 回反復実行した。その推定値（95%信頼区間を含む）は以下のとおりである。

2 時間　：　3.232±0.168 または[3.064, 3.400]
5 時間　：　3.622±0.170 または[3.622, 3.962]
10 時間　：　3.864±0.130 または[3.734, 3.994]
20 時間　：　3.888±0.096 または[3.792, 3.984]
30 時間　：　3:926±0.080 または[3.846, 4.006]

2、5、10、および 20 時間の実行では、推定値が定常状態（定常状態値はL=4.000）に対してまだ下向きに偏っていることが明確である。30 時間の場合は信頼区間が 4.000 を含むものの、平均値がやや低いため確実な判断はできない。より多くの反復実行を行えば信頼区間の幅は減少するだろう。その結果、4.000 が信頼区間の外側となり、30 時間の実行にもまだ偏りがあると結論づけられる**かもしれない**が、現時点では判断できない。また、追加の反復実行によって、開始時の偏りが大きな影響を与えるという事実（これは統計的サンプリングの性質である）が証明されない可能性もある。幸いなことに、多くの科学的・社会学的実験とは異なり、反復回数と実行期間を完全に制御でき、満足のいく結果が得られるまで（あるいはコンピュータの処理能力か人間の忍耐力が尽きるまで）実験を続けることができる。ここで重要な点として、開始時の偏りは「複製」できないことを指摘しておこう。過渡期間はシステムに固有の特性であり、偶然による人工的な結果やサンプリング誤差ではない。

モデルを長時間実行して開始時の偏りを解消する代わりに、**ウォームアップ期間**を設定することができる。この方法では、モデル実行期間を 2 つに分割する。初期（ウォームアップ）期間中は、モデルは通常通り実行されるものの、統計データは収集されない。ウォームアップ期間後から統計データの収集が開始され、この時点でモデルは定常状態に近い状態となっている。このシンプルなモデルの場合、ウォームアップ期間後に最初のエンティティが到着する時点で、待ち行列内のエンティティの期待数は定常状態の値である 3.2 に近づいているはずである。この効果を確認するため、実行期間とウォームアップ期間をそれぞれ(20, 10)、(30, 10)、(30, 20)と設定して 3 つの追加実験を実施した（Simio では、希望するウォームアップ期間の長さを Experiment の Warm-up Period プロパティに設定する）。理論値 4.000 に対する実験結果は以下のとおりである。

(実行期間, ウォームアップ) ＝(20, 10)：4.033±0.155 または[3.978, 4.188]
(実行期間, ウォームアップ) ＝(30, 10)：4,052±0,103 または[3,949, 4,155]
(実行期間, ウォームアップ) ＝(30, 20)：3.992±0.120 または[3.872, 4.112]

ウォームアップ期間は、すべてのケースで開始時の偏りを減少または排除する効果があり、30 時間を超える総実行時間の増加を必要としないことがわかった。つまり、ウォームアップ期間を使用することで、コンピュータ負荷を増やすことなく推定値を改善できた。ここで「**ウォームアップ期間はどれくらいの長さが適切か？**」という疑問が生じる。一般に、モデルが定常状態に達するまでの時間を決定することは容易ではない。一つの実用的なアプローチとして、Simio モデルの Facility ウィンドウにダイナミックな Status Plots を挿入し（モデルの Facility ウィンドウで、Facility Tools 下の Animation リボンから選択。アニメーションの詳細は 8 章参照）、グラフの傾きが安定したように見える時点で判断を下す方法がある。ただし、アニメーションは一度に 1 回の反復しか表示できないため、これらの表示は非常に**ノイズの多い**（つまり変動の大きい）ものとなる。ウォームアップ期間の設定に関して、以下の点が考えられる：

・ウォームアップ期間が短すぎる場合、開始時の偏りが残存する（これは望ましくない）。
・ウォームアップ期間が長すぎる場合、サンプリング誤差が不必要に大きくなる（ウォームアップ期間が長くなるほど、実際に記録できるデータ量が減少する）。

したがって、「もっとも安全な」アプローチは、許容可能なサンプリング誤差（信頼区間の半幅で測定）を達成するために、ウォームアップ期間をじゅうぶんに確保し、実行時間を延長し、反復回数を増やすことである。このアプローチでは必要以上にコンピュータ時間を消費する可能性があるが、現代のコンピュータは高速である（実際には測定できない偏りは、気づかないうちに危険となりうる）。

もちろん、前段落のウォームアップの議論では、定常状態値が実際に必要だと仮定していた。しかし、必ずしもそうとは限らない。むしろ、過渡期間中の「短期的な」システム挙動に注目することも一般的である。たとえば、スポーツイベントの当日券販売では、空の状態から販売を開始し、予定された時間に終了するため、「定常状態」は存在しない。このような場合は、**終結シミュレーション**と呼ばれる。この場合、実験ではウォームアップ期間を単に無視し（デフォルトを 0 とする）、シミュレーションは関心のある期間におけるシステムの挙動と初期状態について、偏りのない視点を提供する。

定常状態と終結状態のどちらを目標とするかは、モデルの構造よりも研究の目的によって決まる。終結シミュレーションは設定、実行、分析が容易である。これは、各反復の開始と終了の規則が単なるアナリストの判断ではなく、モデルの一部となっているためである。実際の課題は、結果に対して許容可能な統計的精度を得るために必要な反復回数を決定することだけである。

4.2.5　モデルの検証

定常状態の挙動を推定したいときの反復問題とウォームアップ期間についてはすでに説明したので、更新したシミュレーションモデルの結果と待ち行列解析の結果を比較する。このモデルは、30 時間の実行期間と 20 時間のウォームアップ期間で 500 回反復実行した。表 4.5 に両方の結果を示す。表 4.2 の結果と比較すると、このモデルがより「正しい」と確信できる。言い換えれば、モデルが**検証された**という強い証拠がある（つまり、期待通りの反応を示している）。ただし、モデルは**実証可能な方法で**検証することはできないことに注意が必要である。できることは、誤りを見つけるか、あるいはモデルが正しいと確信できるまで証拠を集めることだけである。

表 4.5　待ち行列解析と最終シミュレーション実験の比較

推定された測定基準	待ち行列理論	シミュレーション
稼働率(ρ)	0.800	0.800±0.004
システム内数(L)	4.000	4.001±0.133
待ち行列数(L_q)	3.200	3.201±0.130
システム内時間(W)	0.083	0.083±0.003
待ち時間(W_q)	0.067	0.066±0.003

これまでの検証過程を要約すると、以下のとおりである：

1. モデル結果（待ち行列解析）に関して一連の**期待値**を算出した。
2. モデルを構築・実行し、モデル結果と期待値（表 4.2）を比較した。
3. 結果が一致しなかったため、以下の 3 つの可能性を検討した：
   a) Simio モデルに誤りがある（つまり、モデル自体にエラーがある）。この可能性は除外した。
   b) 期待値が不適切である（つまり、シミュレーション結果が待ち行列結果と**一致すべき**という仮定が誤っている）。分析の結果、開始時の偏りを除去し定常状態に到達させるためにウォームアップ期間が必要であることが判明した（過渡期間を含む分析結果を定常状態と比較することは不適切だった）。ウォームアップ期間の導入により修正した。
   c) サンプリング誤差の影響（つまり、シミュレーションモデルの結果は確率的に期待値と一致しているが、実行時間が不十分か、結果の解釈が不適切）。モデル出力のバラツキを適切に評価するため、**反復実行**と実行時間の延長が必要であることが判明した。
4. 最終的に、モデルの妥当性を確認した。

すべてのシミュレーションプロジェクトにおいて、この基本的な検証過程に従うことが推奨される。通常、求める厳密な結果を計算することはできない（もしできるのであれば、シミュレーションは不要であろう）。しかし、モデル化されるシステムの抽象的な見解に基づいていたとしても、常に**いくつかの**期待値を得ることは可能である。そして、これらの期待値と検証過程を用いることで、信頼性の高いモデルと期待値のセットへと収束させることができる。ここまで Simio におけるモデル検証と実験の基本を説明してきたが、本章の残りでは Simio のモデル化概念について説明する。なお、これらの基本的な問題については、本書全体を通じて繰り返し取り上げることになる。

4.3　Model 4-2：Processes を用いたモデル

高レベルの Simio オブジェクト（Standard ライブラリなど）を使用したモデル化は直感的で、多くのユーザにとって簡単かつ迅速である。しかし、低レベルの Simio Processes が必要となる場合がある。これは、モデルの構成、既存の Simio オブジェクトの拡張、詳細な専門的モデリング、実行速度の改善が必要な場合である。Simio Processes の使用には、Simio の詳細な理解と一般的な離散事象シミュレーションの知識が必要となる。本節では、単一サーバ待ち行列システムを例に、基本的な Simio Process モデルのデモンストレーションを簡潔に行う。次章で、さまざまなモデリング状況に応じた Simio Processes の詳細を説明する。

エンティティが有限容量リソース（待ち行列システムのサーバなど）を競合するシステムのモデル化には、Simio は **Seize–Delay–Release** モデルを採用している。これは標準的な離散事象シミュレーションのアプローチで、多くのシミュレーションツールで同様のモデルが使用されている。このモデルの基本的な仕組

みを理解することは、Simio Processes を効果的に活用する上で不可欠である。モデルは以下のように動作する：

- リソースは容量cとして定義される。これは、シミュレーション時刻の任意の時点で、同時に複数のエンティティに割当可能なリソースの単位数がcまであることを意味する。
- エンティティがリソースからのサービスを必要とするとき、エンティティはリソースから単位数sの容量を**占有**する。
- この時点で、リソースに他のエンティティに割り当てられていないs単位の容量があれば、直ちにその容量をエンティティに割り当てる。そして、エンティティはその割り当てられた単位を保持したまま、サービスタイムを表す**時間進行（遅延）**を開始する。もし容量が不足している場合、エンティティは必要な容量が利用可能になるまで自動的に待ち行列に配置される。
- エンティティのサービス時間が完了すると、エンティティはリソースの容量のs単位を**解放**し、プロセスの次のステップへ進む。リソース待ち行列で待機中のエンティティがあり、利用可能な容量（直前に退出したエンティティが解放した単位を含む）がじゅうぶんであれば、待ち行列から先頭のエンティティを取り出し、必要な単位数の容量を直ちに割り当てる。そして、そのエンティティは時間進行を開始する。

モデリングの観点から見ると、各エンティティは Seize-Delay-Release ロジックを通過し、シミュレーションツールがリソース容量の割当と待ち行列を管理する。モデル実行時には、Simio を含むほとんどのシミュレーションソフトウェアが、待ち行列、リソース、エンティティに関する統計を自動的に記録する。基本的な Seize-Delay-Release プロセスを図 4.13 に示す。この図では、「到着時間間隔」は連続するエンティティ間の時間間隔を、「処理時間」は処理に要する時間進行（遅延）を表している。**システム内数**は、シミュレーション時間における任意の時点でのシステム内のエンティティ数を追跡する。また、到着時刻を**記憶**・記録することで、各エンティティのシステム内滞在時間を追跡している。

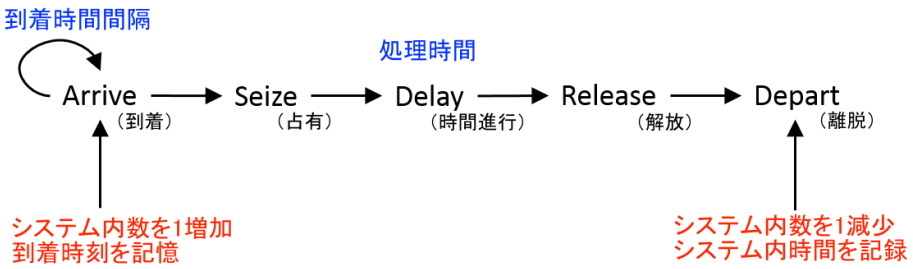


図 4.13　Seize–Delay–Release モデルの基本プロセス

単一サーバ待ち行列モデルでは、すべてのエンティティに対して$c = 1$および$s = 1$を設定する。図 4.13 が示すように、単一サーバモデルは基本的な Seize-Delay-Release ロジックを実装したものである。このモデルを Processes を用

いて構築する方法は、Standard ライブラリオブジェクトを使用する場合と比べてやや複雑だが、モデル開発やユーザ定義統計の収集に関するメカニズムを理解する上で有益である。Simio でこのモデルを実行するための手順は以下のとおりである（図 4.14）：

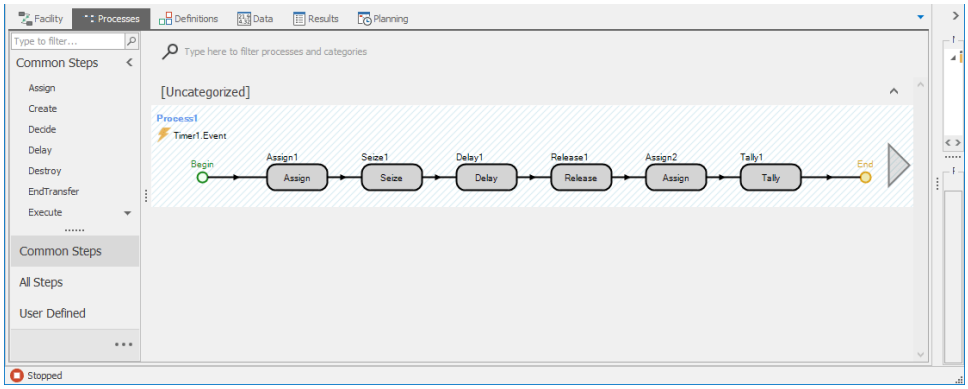


図 4.14　Model 4-2 の Process ビュー

1. Simio を開き、新しいモデルを作成する。
2. Standard Library から Facility ウィンドウに Resource オブジェクトをドラッグし、Resource オブジェクトを配置する。オブジェクトのプロパティにある Process Logic セクションで、Initial Capacity Type が Fixed で、Capacity が 1 であることを確認する（これらはデフォルト設定である）。General セクションのオブジェクト Name（デフォルトは Resource1）を確認しておく。
3. Navigation パネルで Model がハイライトされていることを確認し、Definitions タブをクリックして Definitions ウィンドウを表示する。左のパネルアイコンをクリックして States セクションを選択する。これにより、状態変数をモデルに追加する準備が整う。
4. States リボンの Discrete セクションで Integer アイコンをクリックし、新しい離散的な状態変数（States）を作成する。デフォルトの Name プロパティ（IntegerState1）を WIP に変更する。離散的な状態変数は数値の記録に使用される。この場合、Integer Discrete State を作成することで、モデル内のエンティティの現在数を蓄積するための場所を設定している（図 4.13 のシステム内数を参照）。
5. パネルアイコンをクリックして Elements セクションに移動し、Elements リボンの Timer アイコンをクリックして Timer エレメントを作成する（図 4.15）。Timer エレメントはエンティティ到着のトリガーとして機能する（ループバックについては図 4.13 を参照）。λ=48 エンティティ/時間のポアソン到着、または平均 1.25 分（1/0.8）の指数到着時間間隔を設定するため、Time Interval プロパティを Random.Exponential(1.25)に設定し、Units が Minutes になっていることを確認する。

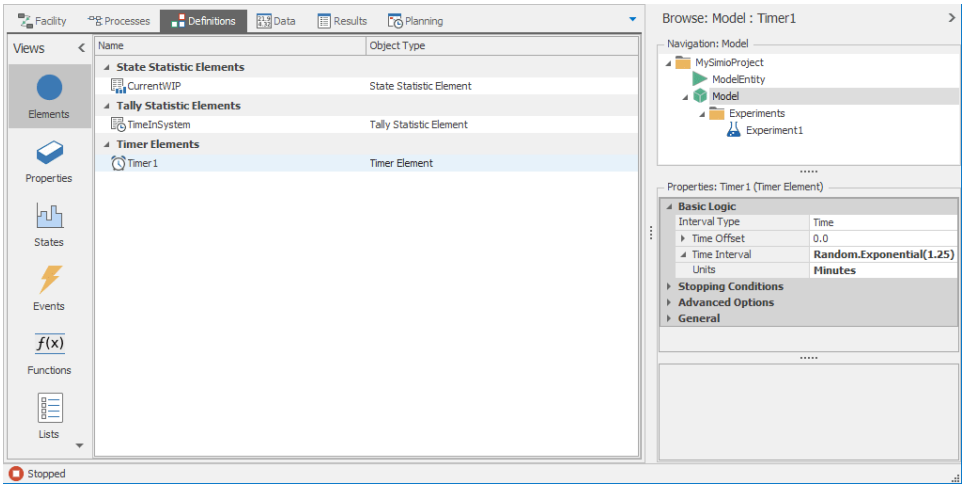


図 4.15　Model 4-2 の Timer エレメント

6. Elements リボンの Statistics セクションで StateStatistic アイコンをクリックして、StateStatistic を作成する。State Variable Name プロパティを WIP に設定する（以前に定義した Discrete State がプルダウンに表示される）。Name プロパティを CurrentWIP に設定する。これにより、Simio は時間経過における状態の値を追跡し、時間依存の統計値として記録する。
7. Elements リボンの Statistics セクションで Tally Statistic アイコンをクリックして、TallyStatistic を作成する。Name プロパティを TimeIn System に、Unit Type プロパティを Time に設定する。Tally Statistics は、観測統計量（離散時間型の統計量）を記録するために使用する。
8. Processes タブをクリックして Process ウィンドウに移動し、Process リボンの Create Process アイコンをクリックし新しい Process を作成する。
9. Triggering Event プロパティにタイマーイベントを設定する（図 4.16）。これにより、タイマーが切れるたびにプロセスが実行される。
10. Common Steps パネルから Assign ステップをドラッグし、プロセスの Begin インジケータの右側に配置する。State Variable Name プロパティを WIP、New Value プロパティを WIP+1 に設定する。これは、エンティティがシステムに到着した際に状態変数の値を増加させるためである（図 4.13 の「1 増加」を参照）。
11. Assign ステップの右側に Seize ステップを追加する。Resource1 を到着エンティティが占有できるようにするため、Basic Logic セクションの Seizes プロパティを選択し、右の「…」ボタンをクリックして Add をクリックする。そして、Resource1 が占有されるように設定する（図 4.17）。
12. Seize ステップの直後に Delay ステップを追加し、Delay Time プロパティを Exponential(1)分に設定する。これにより、エンティティの時間進行は平均 1 分（元のサービス率 60/時間と同等）の指数分布に従う。
13. Delay ステップの後に Release ステップを追加し、Releases プロパティを Resource1 に設定する。

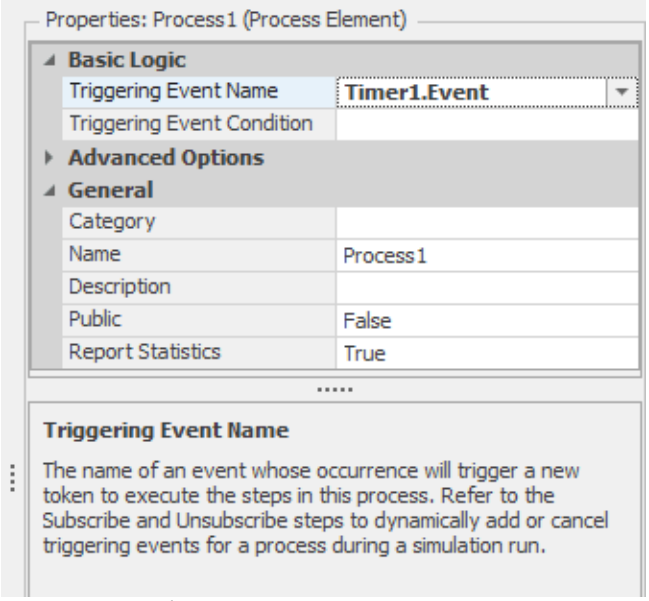


図 4.16　プロセスにおけるトリガーイベントの設定

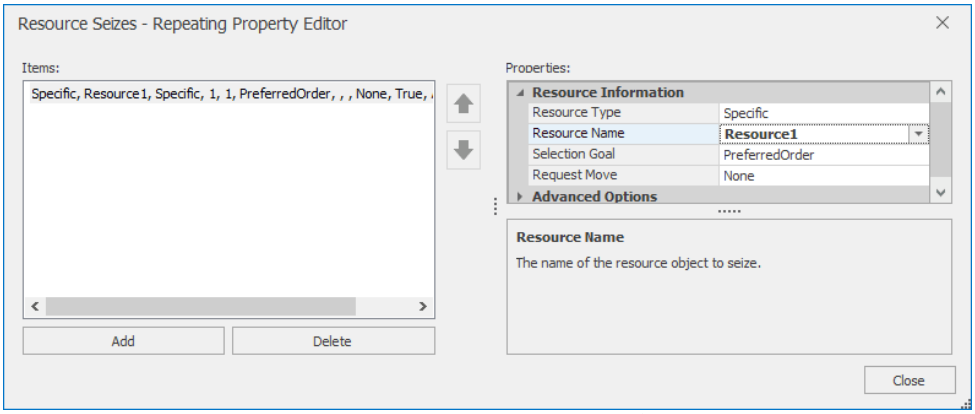


図 4.17　Resource1 が占有される Seize プロパティの設定

14. Release ステップの次に新しい Assign ステップを追加する。State Variable Name プロパティを WIP、New Value プロパティを WIP-1 に設定する。これにより、エンティティがリソースを解放してシステムを離脱する際に、状態変数の値が減少する。
15. Tally ステップを追加し、TallyStatisticName プロパティを TimeInSystem（すでに作成済みの Tally Statistic がプルダウンで選択可能）に、Value プロパティを TimeNow-Token.TimeCreated に設定する。この値は、現在のシミュレーション時刻からトークン作成時刻を引いた値であり、エンティティのシステム滞在時間を表す。この Tally ステップは図 4.13 の「記録」機能を実装している。なお、図 4.13 の到着時刻の「記憶」には、トークン状態変数 Token.TimeCreated を使用している点に注意が必要である。
16. 最後に、Facility ウィンドウに戻り、実行パラメータを設定する（この例で

は、Ending Type を **1,000 時間の固定実行期間**に設定）。

States（状態変数）、Properties（プロパティ）、Tokens（トークン）、およびその他の Simio Framework の詳細については 5 章で議論する。

モデルをテストするには、Project Home リボンの New Experiment アイコンをクリックして Experiment を作成する。図 4.18 は、500 時間のウォームアップ期間を設定したモデルを 10 回反復実行した Pivot Grid の結果を示している。レポートには CurrentWIP と TimeInSystem 統計を含む UserSpecified カテゴリが含まれている。これらの統計は、4.2 節の Standard ライブラリのオブジェクトモデルで自動収集される ModelEntity 統計（NumberInSystem や TimeInSystem）とは異なり、プロセスモデルで明示的に設定したものである。モデルが大規模化・複雑化するにつれて、デフォルトの統計量以外の情報が必要になることが多いため、ユーザ定義の統計を理解することは重要である。

Design　Response Results　Pivot Grid　Reports　Dashboard Reports　Input Analysis

Drop Filter Fields Here

Average　Minimum　Maximum　Half Width　　Scenario

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Scenario 1 Average	Minimum	Maximum	Half Width
Model	Model	CurrentWIP	UserSpecified	StateValue	Average	3.9636	3.4841	4.2981	0.1913
					FinalValue	0.9000	0.0000	3.0000	0.7872
					Maximum	32.8000	24.0000	44.0000	3.9439
		TimeInSystem	UserSpecified	TallyValue	Average (Hou...	0.0825	0.0730	0.0893	0.0037
					Maximum (Ho...	0.6255	0.4759	0.7466	0.0780
					Minimum (Hou...	0.0000	0.0000	0.0000	0.0000
					Observations	24,028.0000	37.0000	297.0000	130.0811
Resource	Resource1	[Resource]	Capacity	ScheduledUtilization	Percent	79.8263	78.7424	80.9127	0.4707
				UnitsAllocated	Total	24,027.9000	36.0000	298.0000	130.0166
				UnitsScheduled	Average	1.0000	1.0000	1.0000	0.0000
					Maximum	1.0000	1.0000	1.0000	0.0000
				UnitsUtilized	Average	0.7983	0.7874	0.8091	0.0047
					Maximum	1.0000	1.0000	1.0000	0.0000
			ResourceState	TimeBusy	Average (Hou...	0.0828	0.0777	0.0884	0.0023
					Occurrences	4,828.0000	77.0000	367.0000	107.3130
					Percent	79.8263	78.7424	80.9127	0.4707
					Total (Hours)	399.1316	93.7118	404.5634	2.3535
				TimeIdle	Average (Hou...	0.0209	0.0207	0.0211	0.0001
					Occurrences	4,827.9000	76.0000	367.0000	107.4499
					Percent	20.1737	19.0873	21.2576	0.4707
					Total (Hours)	100.8684	95.4366	106.2882	2.3535

図 4.18　Model 4-2 の結果

CurrentWIP 統計量は**時間従属**の統計量の例である。Simio の状態変数（ステップ 4）を定義し、プロセスを使用して必要なタイミング（ステップ 10 での増加、ステップ 14 での減少）で値を更新する。シミュレーション時間の進行に伴い、値の変化を追跡してサマリ統計量（$\hat{L}$、ステップ 4）を報告するよう設定している。

TimeInSystem 統計量は**観測**または**タリー**統計量の例である。各到着エンティティは 1 つの観測値（システム内滞在時間）を提供し、Simio はこれらの値のサマリ統計量（この場合、$\hat{W}$）を追跡・報告する。この統計値はステップ 7 で設定され、ステップ 15 で各観測を記録している。

プロセスモデルに関して注意すべき点は、対応する Standard ライブラリオブジェクトのモデルよりもかなり速く実行できるということである。この速度差は、Standard ライブラリオブジェクトが自動的な統計収集、アニメーション、経路における衝突探知、リソース故障などの追加的な機能を含んでいるためである。

上述のように、構築するほとんどの Simio モデルは Standard ライブラリオブジェクトを用いており、Simio のプロセスだけで完全なモデルを構築することは難しい。しかし、**プロセス**は Simio の基本的な要素であり、それらの動作原理を理解することが重要である。この点については 5.1.4 項で詳しく説明する。ここでは、Standard ライブラリオブジェクトを用いた初期モデルに戻ることとする。

4.4　Model 4-3：現金自動預払機(ATM)

初期 Simio モデルの構築では、単純な**エンティティ**と**サーバ**を持つ待ち行列システムに焦点を当てた。本節と 4.8 節では、シミュレーション分析で使用される「実際の」システムをより詳細に表現するため、モデルにいくつかの状況を追加する。以降の章では、シミュレーションモデリングの一般的な概念を説明し、Simio の特徴を探求しながら、モデルをさらに発展させていく。

Model 4-1 と 4-2 では、エンティティの到着間隔とサービス時間に指数分布からの観測値を用いた。これは、シミュレーションにおけるランダム性の基本を説明し、$M/M/1$待ち行列モデルの数学的な「妥当性」を示すためである。しかし、実際のモデリングでは、到着間隔とサービス時間が指数分布に従うことは少ない。Simio や他のシミュレーションパッケージでは、さまざまな分布からサンプリングすることで、より一般的なモデリングをサポートしている。Model 4-3 と 4-4 ではサービス時間に三角分布を使用し、5 章のモデルでは他の標準的な分布を使用する。6.1 節では、シミュレーションモデルが現実をより正確に表現できるよう、入力確率分布を実用的にどのように決定するかを解説する。

Model 4-3 は図 4.19 に示す ATM（現金自動預払機）をモデル化する。顧客は入口から入り、ATM まで歩き、操作を終えると出口まで歩いて退出する。このモデルでは、ATM 設置室がじゅうぶんに広く、複数の顧客が待機できると仮定する（これによりモデルが簡略化される。ただし、後の章で収容人数に制限のある待ち行列について検討する）。この仮定により、基本的には図 4.1 で示した単一サーバ待ち行列モデルと同様の構造となる。

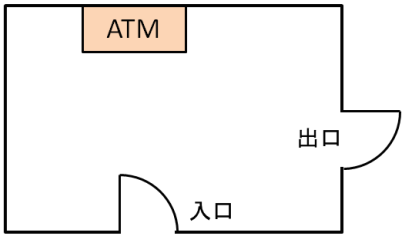


図 4.19　ATM の例題

Model 4-1 を基に、ATM モデルを作成していく（Model 4-1 のファイルを上書きしないよう、必ず Save Project As オプションを使用して Model 4-3 として保存すること）。完成した ATM モデル（Model 4-3）を図 4.20 に示す。

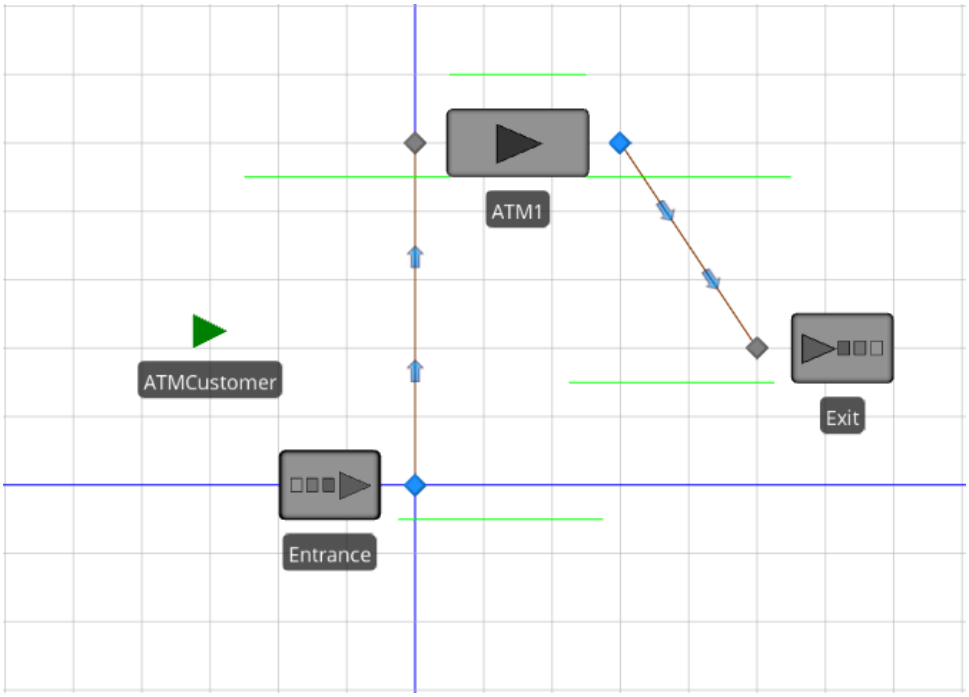


図 4.20　Model 4-3：ATM の例題

必要な変更は以下のとおりである：

1. システムの内容を反映するようオブジェクト名を変更する（エンティティを ATMCustomer、Source オブジェクトを Entrance、Server オブジェクトを ATM1、Sink オブジェクトを Exit とする）。
2. 図 20 に示すようにモデルを再配置する。
3. 顧客の移動時間を組み込むため、Connector とエンティティオブジェクトを修正する。
4. ATM の処理時間分布を、三角分布（最小値 0.25 分、最頻値 1.00 分、最大値 1.75 分）に変更する。

オブジェクト名の書き換えは、モデルの実行の特性やパフォーマンスに影響しないが、オブジェクトを命名することでモデルの可読性を（特に大きくて複雑なモデルでは）大いに改善することができる。そして、オブジェクトを命名し、Description プロパティを用いて重要な記述を加える習慣をつけておくことが望ましい。オブジェクトを選択し、F2 キーを押して新しい名前を入力するか、またはオブジェクトの Name プロパティを編集することでオブジェクトを改名できる。モデル化されるシステムらしく見えるようにモデルを再配置することは非常に容易である。Simio では、オブジェクトをドラッグするとき、オブジェクト間の接続は維持される。また、オブジェクトを動かすことに加えて、個々のオブジェクトの入出力ノードも動かすことができることに留意すべきである。

初期の待ち行列モデル（Model 4-1）では、エンティティは到着時にサーバに単に「現れる」としていた。Simio の Connector オブジェクトは、このタイプの

エンティティ移動を支援していた。ATM モデルのケースでは、顧客は入口から ATM までと、ATM から出口までとを徒歩で移動するため、このタイプの移動ではない（「本当」のシステムのほとんどのモデルは、似たようなエンティティ移動が含まれている）。幸い Simio は、エンティティ移動のモデル化を容易にするために、Standard ライブラリにいくつかのオブジェクトを提供している：

- **Connector（コネクタ）**：エンティティを瞬時に移動させ、シミュレーション時間はゼロとなる（つまり、無限速度での即時移動）。
- **Path（パス）**：オブジェクト間の距離とエンティティの速度に基づいて移動時間を計算し、エンティティを移動させる。
- **TimePath（タイムパス）**：ユーザが指定した時間式を使用して、オブジェクト間でエンティティを移動させる。
- **Conveyor（コンベア）**：物理的なコンベアシステムをモデル化する。

これらの移動手法は次章以降で詳しく説明する。本モデルでは Path を使用する（各オブジェクトの詳細な説明は、F1 キーまたは Simio ウィンドウ右上の?アイコンからアクセスできる Simio レファレンスガイドに記載されている）。現在のモデルは Model 4-1 を基にしているため、すでに Connector で接続されている。Connector から Path への変更は、Connector を右クリックして Convert to Type サブメニューから Path を選択するだけで完了する。別の方法として、既存の Connector を削除し、Standard ライブラリから Path を選択して、開始ノードと終了ノードを指定することでも可能である。

Path オブジェクトに沿ったエンティティの移動時間は、パスの長さとエンティティ速度によって決定される。Simio モデルはデフォルトで比例的に描画されるため、2 つのノード間にパスを追加すると、パスの長さは自動的にノード間の距離として設定される（長さなどの単位を持つプロパティを**入力する**際は、プロパティフィールドの+を押してフィールドを展開し、単位を指定できる。また、Run リボンの Unit Settings ボタンで、Facility ウィンドウのラベルや Pivot Grid の数値、出力の追跡などの**出力**単位を変更できる）。

Path オブジェクトの現在の長さは、General セクションの Physical Characteristics/Size グループ内の Length プロパティに表示される。また、グリッドを使用して経路の長さを推定することもできる。実寸での描画が不適切な場合は、論理パスの長さを手動で設定できる。これには、Drawn To Scale プロパティを False に設定し、Logical Length プロパティに希望の長さを入力する。

エンティティ速度は、エンティティプロパティの Travel Logic セクションにある Initial Desired Speed プロパティで設定する。Model 4-3 では、入口から ATM までのパス長を 10 meters、ATM から出口までを 7 meters とし、エンティティ速度を 1 meter/second に設定している。これにより、エンティティは入口から ATM までの移動に 10 秒、ATM から出口までの移動に 7 秒のシミュレーション時間を要する。モデル化するシステムに応じて、パスの長さとエンティティ速度は簡単に調整できる。

ATM モデルの最終的な変更は、サーバオブジェクトの処理時間の分布を変更することである。指数分布は ATM の取引時間のモデル化には適していない。その理由は、指数分布が多くの小さい値と少数の非常に大きい値を持ち、密度関数の最頻値がゼロだからである。ATM 取引では、すべての顧客がカードを挿入し、暗証番号を入力し、取引を選択する必要があり、取引の種類も限られているため、境界のある分布がより適切である。そこで、三角分布(0.25, 1.00, 1.75)を使用することにする。適切な分布の選択は**入力分析**の一部であり、これについては 6.1 節で詳しく説明する。当面は、この分布が適切であるとする。

処理時間の分布を変更するには、Processing Time プロパティを `Random.Triangular(0.25, 1, 1.75)`に設定し、Units プロパティは `Minutes` のままとする。`Random` キーワードを使用することで、約 19 の一般的な分布から統計的に独立な観測値を生成できる。これには、標準的な確率分布に適合しない場合に使用される連続型・離散型の経験分布も含まれる。これらの分布、必要なパラメータ、密度関数・確率質量関数のプロットについては、Simio レファレンスガイドの「Modeling in Simio」の「Expressions Editor, Functions and Distributions」の「Distributions」セクションで詳しく説明している。Simio の乱数生成と確率変数については、6.3 節と 6.4 節で説明する。

Model 4-3 の完成後は、4.2.5 項で説明したように、モデルの**検証**が必要である。検証プロセスには、期待値の設定、モデルの構築・実行、そしてモデル結果と期待値の比較が含まれる。期待値とモデル結果に不一致がある場合は、モデル、期待値、またはその両方の問題を特定し修正する必要がある。Model 4-1 では、待ち行列システム$M/M/1$をモデル化してパフォーマンス指標の厳密値を計算できたため、期待値の設定は比較的容易だった。しかし、Model 4-3 ではそれほど単純ではない。処理時間が指数分布ではなくなり、到着からサービス、そしてサービスから出発までのエンティティ移動時間が追加されたためである。これらの 2 つの変更は待ち行列指標において相反する効果を持つ。処理時間のばらつきが減少したことで、システム内時間と待ち行列内のエンティティ数は減少すると予想される。一方、エンティティ移動時間の追加により、システム内のエンティティ数と滞在時間は増加すると考えられる。そのため、検証可能な期待値の設定が困難である。これは複雑なモデルでは一般的な課題であるが、モデル検証には必要不可欠である。

この問題に対する 1 つの戦略は、期待値を容易に算出できる**修正モデル**を作成して検証を行うことである。Model 4-3 ではこのアプローチを採用する。モデル修正には 2 つの選択肢がある：Entity Transfer Times を `0` に設定して$M/G/1$の待ち行列近似を使用する（2 章参照）か、処理時間を指数分布に戻す。ここでは後者を選択し、ATM の Processing Time プロパティを `Random.Exponential(1)` に変更する。各エンティティに 17 秒の移動時間を追加しただけなので、ρ、L_q、W_qが表 4.2 の$M/M/1$の値と一致し、システム内時間Wは対応する値より 17 秒大きくなると予想される。表 4.6 は、500 回の反復実行（4.2.5 項と同じ条件の反復時間 30 時間、ウォームアップ時間 20 時間）の結果を示している。

表 4.6　Model 4-3（修正バージョン）の結果

推定された測定基準	シミュレーション
稼働率(ρ)	0.797±0.004
システム内数(L)	4.139±0.131
待ち行列数(L_q)	3.115±0.128
システム内時間(W)	0.086±0.003
待ち時間(W_q)	0.064±0.003

これらの結果は期待値と整合しているように見える（平均値と予測の間に小さな偏りが気になる場合は、反復時間を延長するか、反復回数を増やすことができる）。したがって、修正したモデルは適切に検証されたといえるが、Model 4-3 については、Processing Time プロパティの入力ミスか、Simio の Triangular 乱数ジェネレータの出力値に問題がある可能性がある。

ここで、プロパティが正しく入力されており、Simio の乱数および確率変量のジェネレータが正常に動作していると仮定しよう。表 4.7 は Model 4-3 の実験結果を示している（500 回の反復実行、30 時間の実行期間、20 時間のウォームアップ）。予想通り、エンティティの待ち行列数とシステム内時間は共に減少した（サービス時間の変動を減少させた結果として妥当である）。4.2.5 項で述べた重要な点を改めて強調しておく：モデルの検証は（一般的な意味での）**実証**ではない。むしろ、じゅうぶんな確信が得られるまで（おそらくプロセスの誤りを発見し修正しながら）証拠を積み重ねていく過程なのである。

表 4.7　Model 4-3 の結果

推定された測定基準	シミュレーション
稼働率(ρ)	0.800±0.003
システム内数(L)	2.791±0.064
待ち行列数(L_q)	1.764±0.061
システム内時間(W)	0.058±0.001
待ち時間(W_q)	0.036±0.001

4.5　平均値を超えて：Simio MORE(SMORE)プロット

確率的シミュレーションの結果はランダムな性質を持つため、適切な統計的手法による分析が必要だと、これまでも強調してきた。これまでは主に平均値に注目してきた。シミュレーションから得られる平均値を使用して、未知の母平均（または確率変数の期待値と関心のある出力応答の分布）を推定している。この推定には信頼区間の利用がもっとも効果的な方法であり、Simio では Experiment Pivot Grid と Reports でこれらを提供している。

平均値は重要な指標だが（シミュレーション出力に限らず）、それは定義上、関心のある確率変数の無限回の反復実行の平均に過ぎず、全体像（特に分布の両端）を示すものではない。つまり、平均システム内時間、最大待ち行列長、リソース利

用率といったシミュレーション出力について、データの分布範囲や、起こりやすい値・起こりにくい値を把握することはできない。この問題は、Sam Savage が著書『The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty』（Savage 2009）で指摘している。

3.2.3 項の 1 期間静的在庫シミュレーションでは、特に図 3.3 で結果のヒストグラムを検討し、平均値に加えて、利益の上限・下限リスク、特に損失が発生するリスクを確認した。たとえば、5,000 個の帽子を注文する場合、**平均**利益は良好に見えたものの（95%信頼区間で 8,734.20 ドル±3,442.36 ドル）、30%の確率で**損失**が発生する**リスク**があった。しかし、平均値だけではこれらの重要な情報を捉えることができない。

そこで、Experiment Pivot Grid と Report に加えて、Simio には出力統計を報告するための新しい図表形式が含まれている。この形式には、推定平均値に信頼区間の情報が加えられている。**Simio MORE（SMORE）プロット**は、John Tukey が 1977 年に考案し改良した**箱のプロット**を基にしており、ヒストグラムと個々の反復サマリ応答の点プロットを組み合わせたものである（Tukey 1977）。この SMORE プロットは、Barry Nelson（2008）が開発した **Measure of Risk and Error（MORE）プロット**を発展させたもので、図 4.21 にその主要な要素を示している。

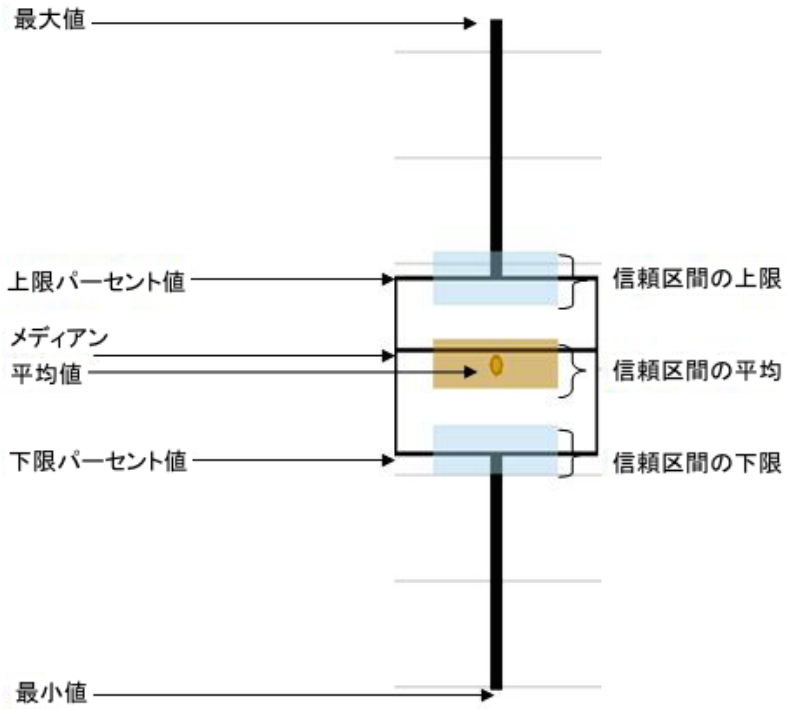


図 4.21　SMORE プロットの構成（Simio レファレンスガイドより引用）

SMORE プロットは、出力パフォーマンス尺度（**応答**）のサマリを視覚化したものである。ここでの応答とは、複数回の反復実行から得られる平均システム内時

間、待ち行列の最大数、リソース稼働率などを指す。標準の箱のプロットの機能に加えて、最小・最大の観測値、標本平均、標本中央値、そして下限・上限のパーセント値も表示する。なお、ここでの「標本」は**反復からの**個別の観測値ではなく、**反復を通した**サマリ測定値から構成される。このプロットは主に、複数回反復される終結シミュレーションや、適切なウォームアップ期間を設定した定常状態シミュレーションでの使用を想定している。SMORE プロットでは、平均値の信頼区間、上下両方の**パーセンタイル**値、観測値のヒストグラム、各反復からの個別の応答を必要に応じて表示できる。

SMORE プロットは実験の Responses から自動的に生成される。たとえば、Model 4-3 では、顧客のシステム滞在時間（ATM への到着から退出までの時間間隔）の平均値に注目できる。Simio はこの統計値を自動的に追跡し、式 `ATMCustomer.Population.TimeInSystem.Average` で反復中の平均値にアクセスできる。これを Response として追加するには、（Experiment ウィンドウの）Design リボンの Add Response アイコンをクリックし、出力のラベルとして Name（`AvgTimeInSystem`）を、そして Expression プロパティ（`ATMCustomer.Population.TimeInSystem.Average`）を指定する（図 4.22）。「Expression に `ATMCustomer.Population.TimeInSystem.Average` と入力する必要があることをどう知るのか」というのはよくある質問であるが、Simio はさまざまな場面でコンテキストに応じたヘルプを提供している。

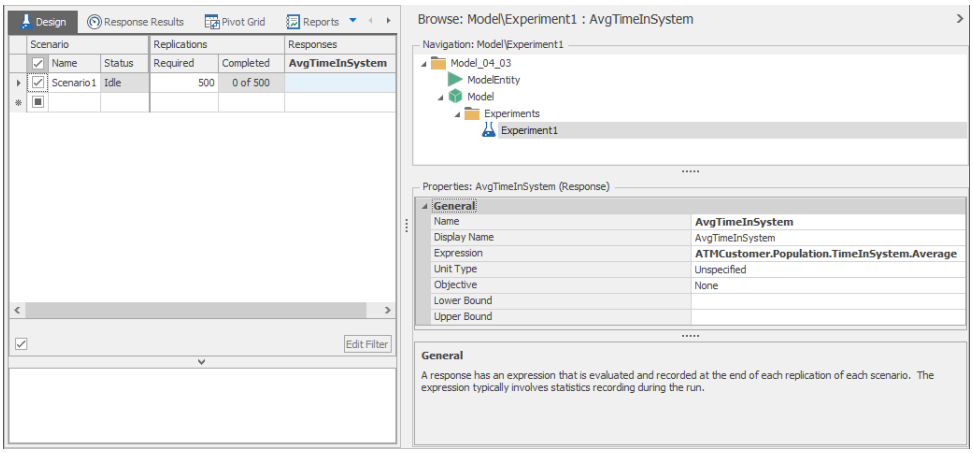


図 4.22　Model 4-3 の平均システム内時間のための実験 Response 定義

Expression フィールドをクリックすると、右側に下向き矢印が表示される。この矢印をクリックすると、赤色の×印と緑色のチェックマークがついた新しいフィールドが開く。これが Simio の**式ビルダ**である。詳しい説明は 5.1.7 項を参照されたい（図 4.23）。

式ビルダの使い方を説明する。まず空の Expression フィールドをクリックし、式の左端でキーボードの**下矢印キー**を押す。表示されるリストから `ATMCustomer`（エンティティの名前）を見つけて**ダブルクリック**する。この名称が Expression フィールドにコピーされる。次に、ATMCustomer の直後にピリオドを入力する

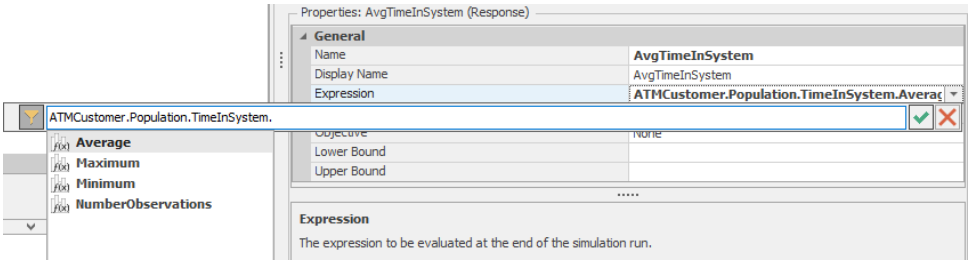


図 4.23　SMORE プロットにおける平均システム内時間 Response への式ビルダの利用

と、続くオプションのリストが表示される。ここでは ATMCustomer エンティティ全体の統計値を取得したいので、Population をダブルクリックする。新しい選択リストから、TimeInSystem（ATMCustomer エンティティの滞在時間）をダブルクリックする。リストが見えなくなった場合は、下矢印キーを再度押す。これまでと同様に、式の右側にピリオドを入力し、表示されるリストから Average を選択する。これは、平均システム内時間を取得したいためである（最大値や最小値が必要な場合は、それらを選択する）。最後に、緑色のチェックマークをクリックして式を確定する。

このプロセスを繰り返してさらに応答を追加できる。SMORE プロットの表示時には、ドロップダウンから選択した Names に切り替えることができる。ここでさらに 2 つの Response を追加する：1 つ目は、Name=MaxQueueLength、Expression=ATM1.AllocationQueue.MaximumNumberWaiting である。2 つ目は、Name=ResourceUtilization 、 Expression=ATM1.ResourceState.PercentTime(1)である。特に最後の(1)については、式ビルダの使用を推奨する。式ビルダのドロップダウンでは、図 4.23 のように、選択肢にマウスを合わせると有用な説明が表示される。これには(1)を含むすべてのエントリの説明が含まれている。実験プロパティで信頼区間の Lower、Upper Percentiles、Confidence Level の割合を設定できる（図 4.24）。デフォルトでは、下限と上限のパーセンタイルは従来の箱ひげ図と同じく 25%と 75%（四分位数）に設定される。表示される「箱」は、デフォルトでは反復の中央の**半分**のみの結果を含むため、より広い範囲を表示したい場合がある。たとえば、10%と 90%に設定すると、反復結果の 80%を含む箱が表示される。

SMORE プロットを表示するには、Experiment ウィンドウの Response Results タブを選択する。図 4.25 は、Model 4-3 の 500 回の反復実行（各反復は 30 時間の実行期間と 20 時間のウォームアップ）における平均システム内時間を SMORE プロットで示したものである。このプロットは Confidence Intervals と Histogram が表示されており、個々の反復の観測値は含まれていない。パーセンタイルはデフォルトの上限 75%と下限 25%に設定されている。Rotate Plot ボタンを使用すると、プロットを垂直方向から水平方向に変更することができる。また、SMORE プロットウィンドウ下部の Raw Data タブをクリックすることで、グラフの基となる数値データを直接確認することができる。

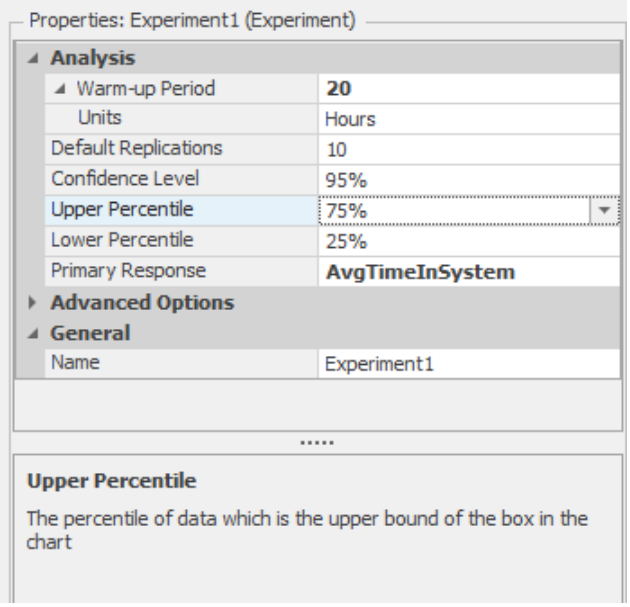


図 4.24　パーセンタイルと信頼区間の水準の設定

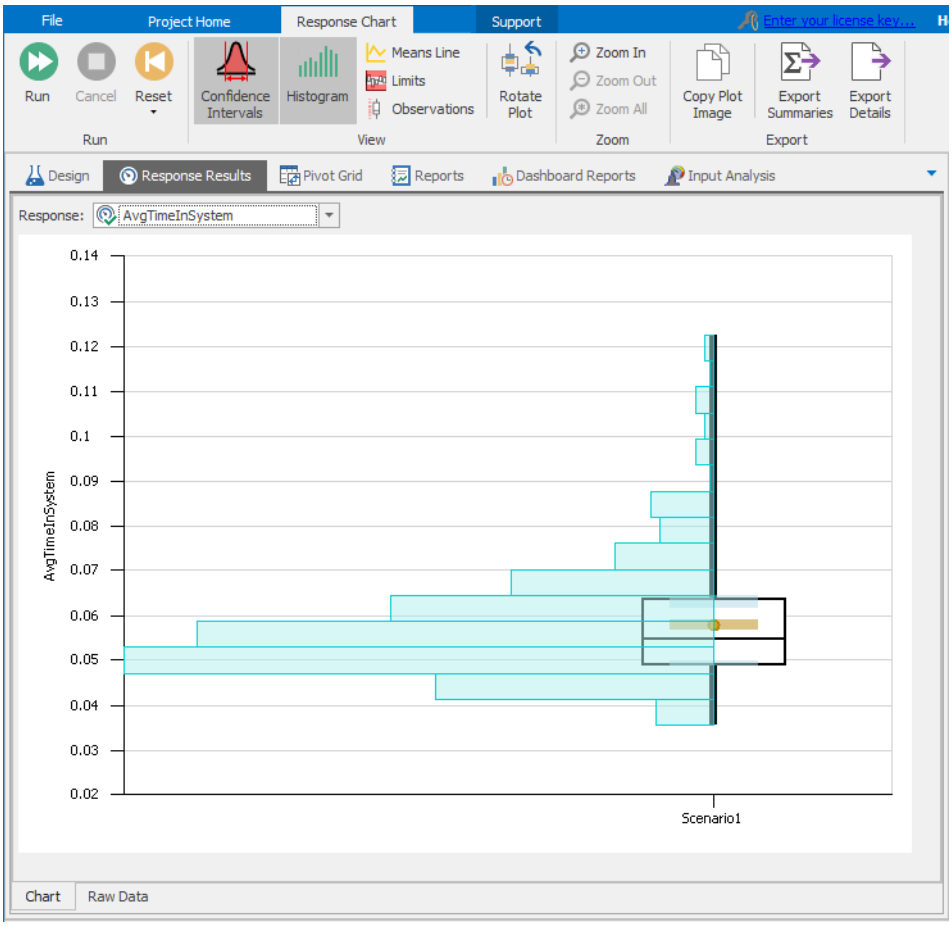


図 4.25　Model 4-3 における平均システム内時間の SMORE プロット

図 4.25 から、システムの予測平均時間は 0.058 時間（3.5 分）未満であることがわかる。中央値はやや低く、ヒストグラムは右に歪んだ分布を示している。箱のプロット（75 パーセンタイル）の上限が約 0.064 時間（3.8 分）であることから、平均システム内時間がこの値を超える確率は 25%である。500 回の反復実行により得られた信頼区間はじゅうぶんに狭く、これは結論の信頼性が高いことを示している。

図 4.26 は最大の待ち行列の長さを SMORE プロットで示している（同じ要素を表示）。

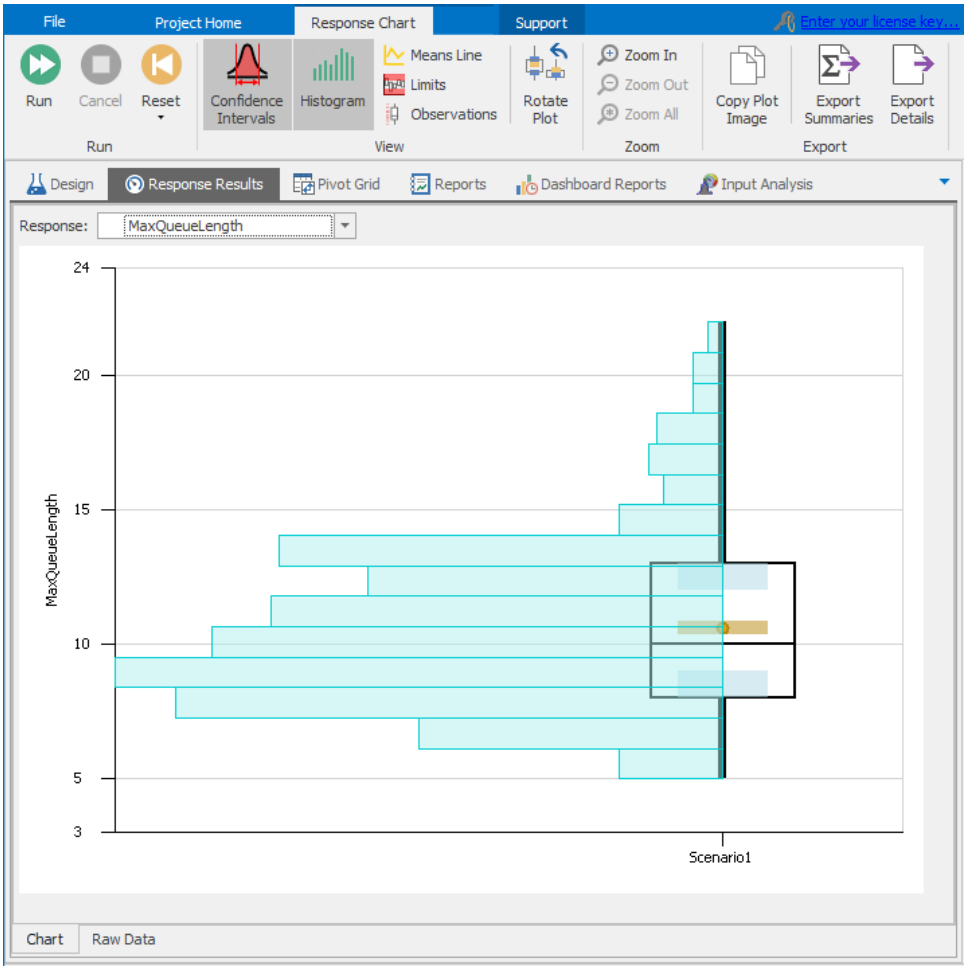


図 4.26　Model 4-3 におけるシステムの最大待ち行列の長さの SMORE プロット

このプロットでは、Upper Percentile をデフォルトの 75%から一時的に 90%に変更している。各反復における**最大**（平均ではない）の待ち行列の長さを示しており、これは全反復を通して必要な待ち行列スペースを表している。ATM のロビーでは、14～15 人分のスペースを確保すれば、反復の約 90%で待ち行列を収容できることがわかる。Experiment Design ウィンドウで設定される Upper Percentile と Lower Percentile を変更して試してみてほしい。これらのパーセンタイルを 0%と 100%の極端な値に設定すると、箱の端も外側に移動する。た

だし、興味深いことに、信頼区間は広くなり、精度が低下する。これは、極端なパーセント点が本質的に変動しやすく、推定が困難なためである。その結果、より広い信頼区間となる。

図 4.27 では、観測されたサーバ稼働率の分布を示している。

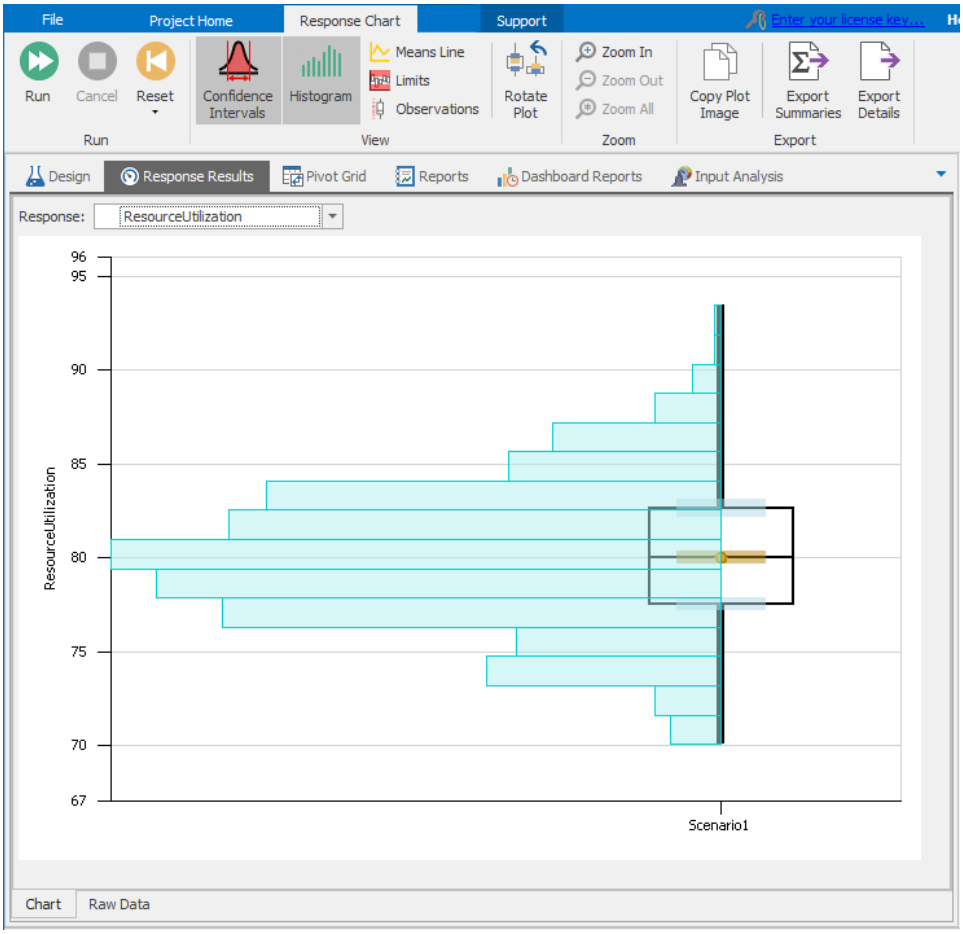


図 4.27　Model 4-3 におけるサーバ稼働率の SMORE プロット

稼働率は時間大半で 77%から 83%の間であり、これは待ち行列理論で予測される値と一致する(Simio リファレンスガイドによれば、パラメータが**最小**、**最頻**、**最大**である三角分布の期待値は、(**最小値** + **最頻値** + **最大値**)/3である)。

$$\frac{\mathrm{E}(\text{サービス時間})}{\mathrm{E}(\text{到着時間間隔})} = \frac{(0.15 + 1.00 + 1.75)/3}{1.25} = 0.80$$

ただし、ヒストグラムの広がりから、稼働率が 90%程度まで上昇する可能性もある（500 回の反復実行での最大稼働率は 90.87%で、下部の Raw Data タブで確認可能)。

Nelson（2008）で説明されているように、SMORE プロットは、シミュレーションにおけるシステムの**リスク**とサンプリング**誤差**を解釈しやすいグラフで表示

する。これは単なる反復実行の平均値やその信頼区間よりもはるかに多くの情報を提供する。SMORE プロットの信頼区間は、百分位数と平均値の推定におけるサンプリング誤差を表す。反復回数を増やすことで、信頼区間の幅（サンプリング誤差）を小さくできる。目視で信頼区間の幅が広すぎると判断した場合は、反復回数を増やす必要がある。信頼区間がじゅうぶんに狭くなれば（つまり、サンプリング誤差が許容範囲内になれば）、上下の百分位数を使って応答の変動性を把握できる。また、ヒストグラムを使って応答の分布の特徴を理解できる（たとえば、図 4.25 から図 4.27 では、平均システム内時間と最大待ち行列の長さは右に歪んだ分布を示すが、稼働率の分布はほぼ対称であることがわかる）。5 章で見るように、SMORE プロットは、複数の代替シナリオ間の違いを、平均値だけでなく分布の広がりと形状の面からも比較するのに非常に有用である。

4.6　追加的統計分析に対する出力データのエクスポート

Simio は統計分析パッケージではなく、シミュレーションパッケージである。信頼区間と SMORE プロットなどの統計的ツールを提供しているが（後の章でさらに機能を紹介する）、その主な特徴は出力データの柔軟なエクスポート機能にある。シミュレーション結果は CSV ファイルとして出力でき、SAS、JMP、SPSS、Stata、S-PLUS、R などの専門的な統計分析パッケージで後処理することができる。

簡単な統計分析においては、Simio から CSV ファイルを直接 Excel にエクスポートして処理することができる。Excel の基本関数（AVERAGE や STDEV など）やデータ分析ツール、さらに Palisade 社の StatTools のような高度な統計分析アドインも利用可能である。このエクスポート機能により、仮説検定、分散分析、回帰分析など、好みの統計パッケージで自由に分析することができる。

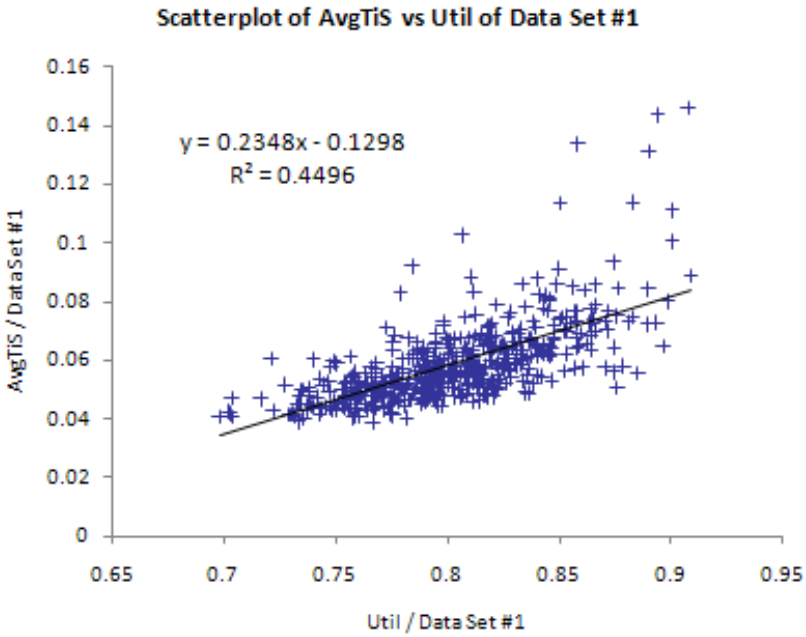
重要な注意点として、各反復から得られるのは個々のエンティティの結果ではなく、1 つのサマリ値（平均システム内時間、最大待ち行列長、サーバ稼働率など）である。これらのサマリ値は互いに独立で同一の分布に従う観測値であり、標準的な統計手法を適用することができる。つまり、統計用語での「サンプルサイズ」は反復回数に相当する。11.4 節で説明されるように、アドオンのプロセスロジックの Write ステップや、Standard ライブラリオブジェクトのカスタマイズにより、個々のエンティティの観測値も収集することができる。また、タリーと状態変数の統計量はログとして記録され、Dashboard Report として表示できる。

4.2.3 項で説明したように、Experiment ウィンドウから Pivot Grid ウィンドウの Export Details アイコンを使用して CSV ファイルへエクスポートすることができる。モデルの規模や反復回数によっては、エクスポートされる CSV ファイルが大きくなることがある。データの再配列や抽出が必要になる場合もあるが、一度この形式でデータが保存されれば、あらゆる種類の統計分析が可能となる。出力データからパターンや関係性を見出すデータマイニングも実施できる。

たとえば、4.5 節の SMORE プロットに用いた 500 回の反復データをエクスポートし、Excel スプレッドシートの 1 列目に平均システム内時間、2 列目に稼働率を配置する。各行の 1 列目は各反復での平均システム内時間、2 列目はその**同**

じ反復でのサーバ稼働率である。このデータに対して StatTools 統計分析アドインを使用して、図 4.28 の散布図と相関分析を作成した。

サーバ稼働率が高くなるほど平均システム内時間が長くなる傾向が見られるが、データにはランダムな「ノイズ」が多く含まれているため、例外も多く存在する。図 4.28 では Excel の内蔵グラフツールを使用して線形回帰直線を追加し、その方程式と R^2 値を表示した。分析の結果、平均システム内時間とサーバ稼働率には正の相関関係があり、平均システム内時間の変動の約 45%がサーバ稼働率の変動で説明できることがわかった。これは、Simio のデータを CSV ファイルにエクスポートして分析できる一例である。適切なデータ抽出（カスタムマクロやスクリプト言語の使用も考えられる）と強力な統計パッケージを組み合わせることで、モデルの挙動や関係性についての洞察を得ることができる。



Correlation　　0.670

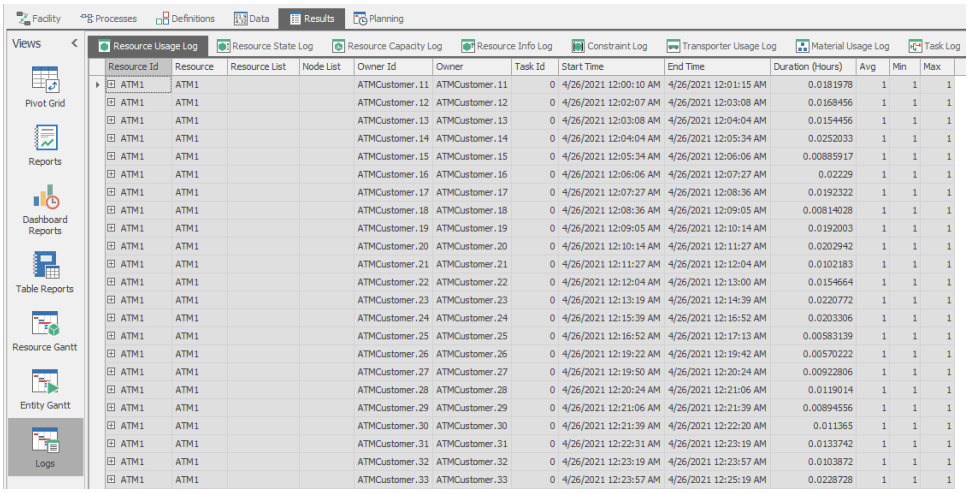
図 4.28　Model 4-3 の平均システム内時間対サーバ稼働率の StatTools 散布図

4.7　インタラクティブ・ログとダッシュボードレポート

Simio は、シミュレーション実行中に各種データを記録し、**ダッシュボードレポート**を作成する機能を提供する。すべてのライセンスタイプで基本的なログ機能が利用でき、Academic RPS ライセンスではさらに高度な機能を使用できる。Simio のログとダッシュボード機能は非常に強力だが、ここではシンプルな例を 1 つ紹介する。より詳細な機能説明は Simio Help と SimBits で確認できる。

この例では、Model 4-3 を使用して ATM リソースの稼働率を記録し、関連するダッシュボードレポートを作成する。まず、ATM リソースの稼働率の**ログを記録する**ための手順は以下のとおりである：

1. Run リボンの Advanced Options 項目から、Interactive Logging をオンにする。
2. ATM サーバのリソース稼働率ログをオンにするため、ATM オブジェクトのインスタンスを選択し、Advanced Options リソースグループの Log Resource Usage プロパティを True に設定する。
3. モデルをリセットし、早送りモードで実行する。
4. Results タブの Logs セクションから、Resource Usage Log をクリックする。図 4.29 は、最新の Resource Usage Log の一部を示している。このログは、指定されたリソースのすべての「使用中」状態を記録し、各エンティティ、開始時間、終了時間、および期間中の稼働率を特定できる。



Resource Id	Resource	Resource List	Node List	Owner Id	Owner	Task Id	Start Time	End Time	Duration (Hours)	Avg	Min	Max
ATM1	ATM1			ATMCustomer.11	ATMCustomer.11	0	4/26/2021 12:00:10 AM	4/26/2021 12:01:15 AM	0.0181978	1	1	1
ATM1	ATM1			ATMCustomer.12	ATMCustomer.12	0	4/26/2021 12:02:07 AM	4/26/2021 12:03:08 AM	0.0168456	1	1	1
ATM1	ATM1			ATMCustomer.13	ATMCustomer.13	0	4/26/2021 12:03:08 AM	4/26/2021 12:04:04 AM	0.0154456	1	1	1
ATM1	ATM1			ATMCustomer.14	ATMCustomer.14	0	4/26/2021 12:04:04 AM	4/26/2021 12:05:34 AM	0.0252033	1	1	1
ATM1	ATM1			ATMCustomer.15	ATMCustomer.15	0	4/26/2021 12:05:34 AM	4/26/2021 12:06:06 AM	0.00885917	1	1	1
ATM1	ATM1			ATMCustomer.16	ATMCustomer.16	0	4/26/2021 12:06:06 AM	4/26/2021 12:07:27 AM	0.02229	1	1	1
ATM1	ATM1			ATMCustomer.17	ATMCustomer.17	0	4/26/2021 12:07:27 AM	4/26/2021 12:08:36 AM	0.0192322	1	1	1
ATM1	ATM1			ATMCustomer.18	ATMCustomer.18	0	4/26/2021 12:08:36 AM	4/26/2021 12:09:05 AM	0.00814028	1	1	1
ATM1	ATM1			ATMCustomer.19	ATMCustomer.19	0	4/26/2021 12:09:05 AM	4/26/2021 12:10:14 AM	0.0192003	1	1	1
ATM1	ATM1			ATMCustomer.20	ATMCustomer.20	0	4/26/2021 12:10:14 AM	4/26/2021 12:11:27 AM	0.0202942	1	1	1
ATM1	ATM1			ATMCustomer.21	ATMCustomer.21	0	4/26/2021 12:11:27 AM	4/26/2021 12:12:04 AM	0.0102183	1	1	1
ATM1	ATM1			ATMCustomer.22	ATMCustomer.22	0	4/26/2021 12:12:04 AM	4/26/2021 12:13:00 AM	0.0154664	1	1	1
ATM1	ATM1			ATMCustomer.23	ATMCustomer.23	0	4/26/2021 12:13:19 AM	4/26/2021 12:14:39 AM	0.0220772	1	1	1
ATM1	ATM1			ATMCustomer.24	ATMCustomer.24	0	4/26/2021 12:15:39 AM	4/26/2021 12:16:52 AM	0.0203306	1	1	1
ATM1	ATM1			ATMCustomer.25	ATMCustomer.25	0	4/26/2021 12:16:52 AM	4/26/2021 12:17:13 AM	0.00583139	1	1	1
ATM1	ATM1			ATMCustomer.26	ATMCustomer.26	0	4/26/2021 12:19:22 AM	4/26/2021 12:19:42 AM	0.00570222	1	1	1
ATM1	ATM1			ATMCustomer.27	ATMCustomer.27	0	4/26/2021 12:19:50 AM	4/26/2021 12:20:24 AM	0.00922806	1	1	1
ATM1	ATM1			ATMCustomer.28	ATMCustomer.28	0	4/26/2021 12:20:24 AM	4/26/2021 12:21:06 AM	0.0119014	1	1	1
ATM1	ATM1			ATMCustomer.29	ATMCustomer.29	0	4/26/2021 12:21:06 AM	4/26/2021 12:21:39 AM	0.00894556	1	1	1
ATM1	ATM1			ATMCustomer.30	ATMCustomer.30	0	4/26/2021 12:21:39 AM	4/26/2021 12:22:20 AM	0.011365	1	1	1
ATM1	ATM1			ATMCustomer.31	ATMCustomer.31	0	4/26/2021 12:22:31 AM	4/26/2021 12:23:19 AM	0.0133742	1	1	1
ATM1	ATM1			ATMCustomer.32	ATMCustomer.32	0	4/26/2021 12:23:19 AM	4/26/2021 12:23:57 AM	0.0103872	1	1	1
ATM1	ATM1			ATMCustomer.33	ATMCustomer.33	0	4/26/2021 12:23:57 AM	4/26/2021 12:25:19 AM	0.0228728	1	1	1

図 4.29　Model 4-3 の Resource Usage Log

以上の手順で、個々のリソースの稼働率を記録する。次に、稼働率の期間を出力するダッシュボードレポートを作成する。手順は以下のとおりである：

1. Results タブの Dashboard Reports セクションを選択し、Dashboards リボンの Create をクリックする。Add Dashboard ダイアログボックスが開いたら、ダッシュボードの名前を入力する。図 4.30 は、ダッシュボード・ダイアログボックスの初期画面である。Data Source ドロップダウンメニューから Resource Usage Log を選択する。
2. Chart Tools/ Home リボンの Chart オプションから、新しいチャートを作成する。図 4.31 は、作成された新しいダッシュボードレポートのチャートを示している。Resource Usage Log から必要な項目をドラッグし、DATA ITEMS にドロップしてチャートをカスタマイズする。
3. Duration (Hours)をドラッグし、Values (Pane 1)データ項目にドロップする。
4. データ項目の右側にある棒グラフアイコンをクリックし、チャートの種類を折れ線グラフに変更する。

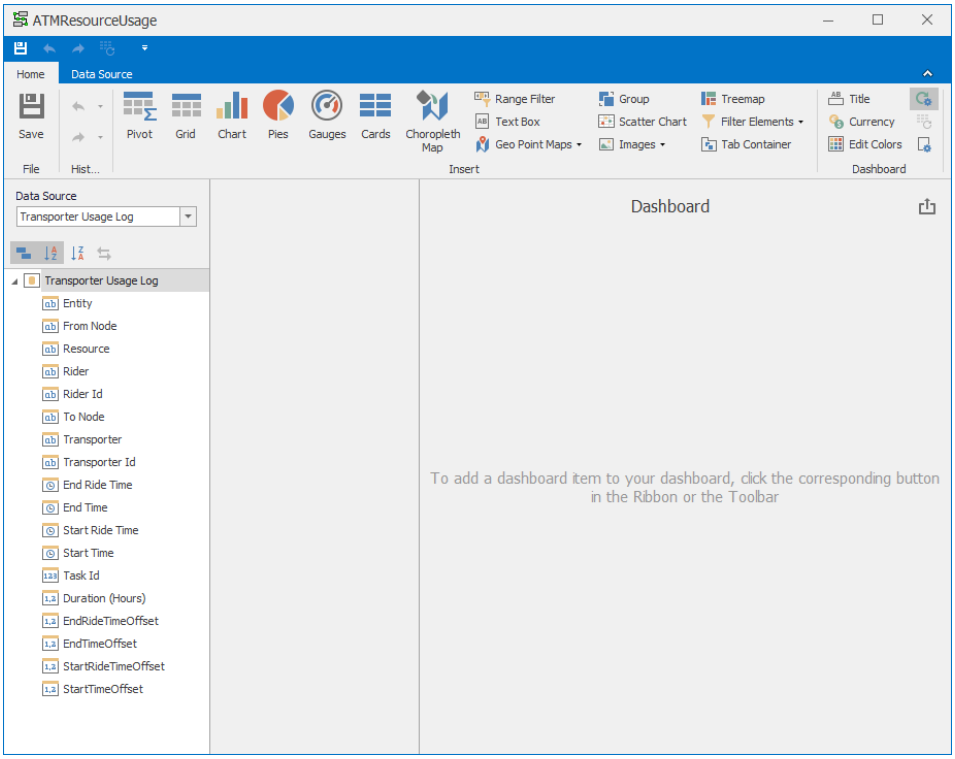


図 4.30　ダッシュボードレポートのダイアログボックス初期画面

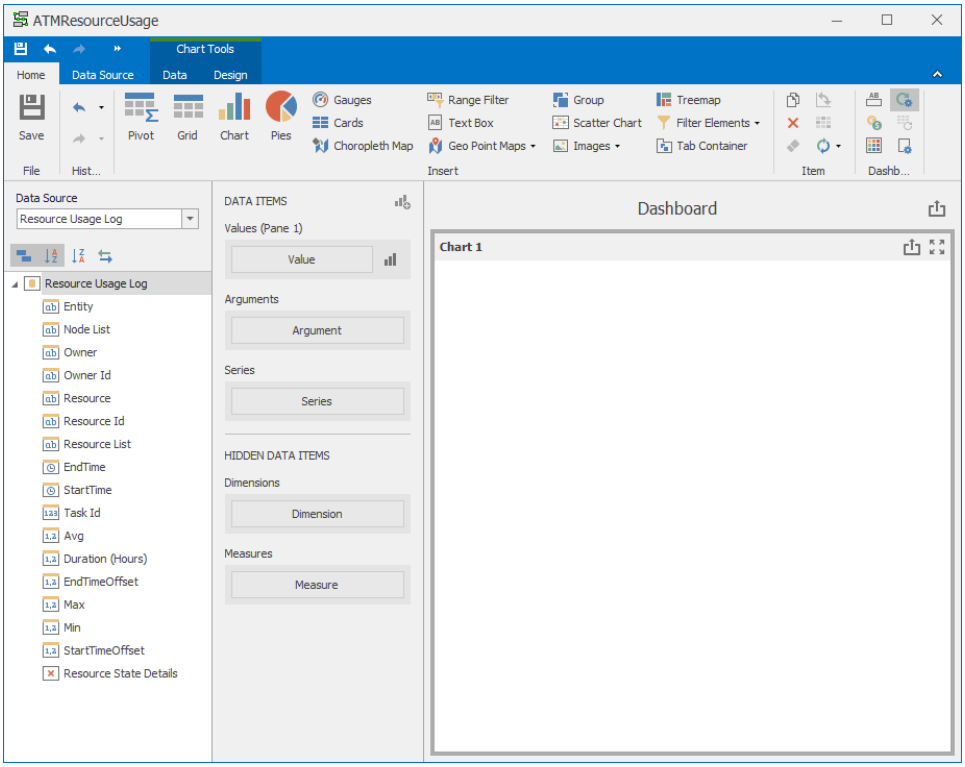


図 4.31　ダッシュボードで作成した新しいチャート

5. Start Time をドラッグし、Arguments データ項目にドロップする。
6. ドロップした Start Time のドロップダウンリストを開き、Date-Hour-Minute を選択する。図 4.32 に作成されたリソース稼働率チャートを示す。このチャートでは、リソースの利用開始時間に対応する稼働率を確認できる。
7. 作成したダッシュボードを保存するには、リボンの Save をクリックする。

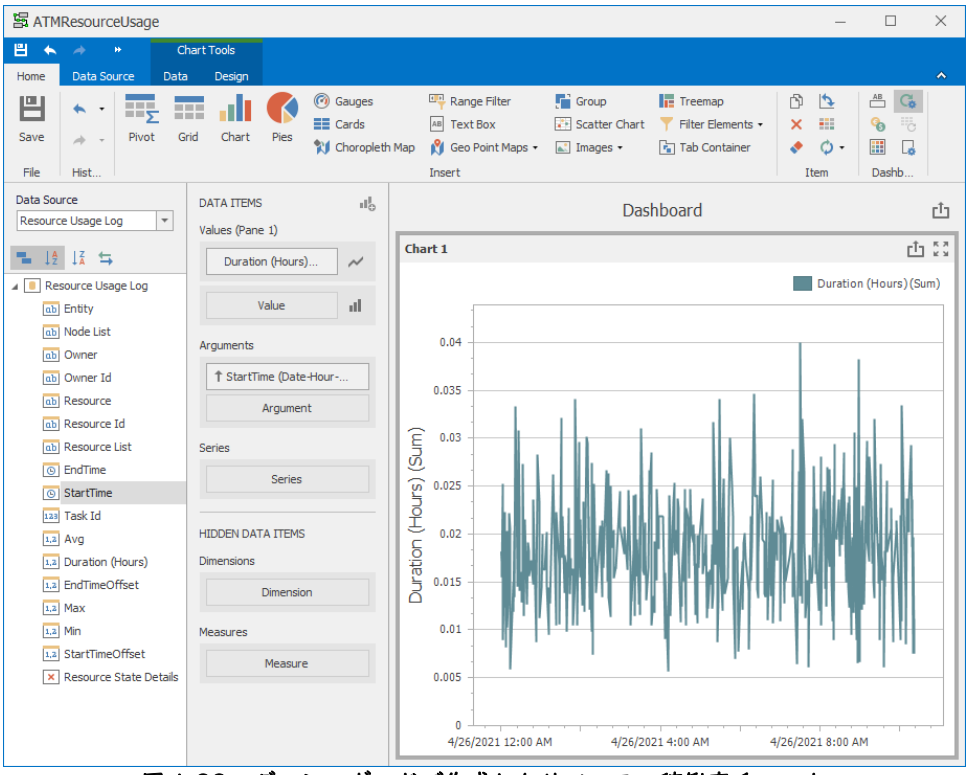


図 4.32　ダッシュボードで作成したリソースの稼働率チャート

Simio のログとダッシュボード機能は非常に広範である。そのため、実際に機能を試しながら Simio ヘルプや SimBits で詳細を確認することを推奨する。ダッシュボードの詳細情報、具体例、手順については、Simio ヘルプおよび https://docs.devexpress.com/Dashboard/ を参照されたい。

4.8　モデルアニメーションの基礎

これまでアニメーションについてほとんど触れてこなかったが、実はすでに実用的なアニメーションを作成している。本節では、アニメーションの主要な機能を紹介し、詳細は 8 章で説明する。アニメーションは主に Facility ビューで操作する。Facility ウィンドウで H キーを押すと、アニメーションに関するキーボードとマウス操作のヒントの表示/非表示を切り替えることができる。操作に慣れるまでは、このヒントを表示したままにしておくとよいだろう。

Simio の大きな特長は、Standard ライブラリでモデルを構築すると同時にアニメーションも自動的に作成される点である。図 4.8 と図 4.20 で示したように、2

次元（2D）のアニメーションモデルが表示されている。さらに、通常は 2D ビューで作業するが、3D モデルも自動的に生成される。2D と 3D の切り替えは、キーボードの 2 または 3 キーを押すか、View リボンから 2D/3D オプションを選択するだけである。図 4.33 は Model 4-3 の 3D モードを示している。3D モードではマウスを使って視点の移動、ズーム、回転が可能である。図 4.33 のモデルでは、ATM1 のサーバに 1 人の顧客（Processing.Contents 待ち行列）、ATM の待ち行列に 2 人の顧客（ATM1 サーバの InputBuffer.Contents 待ち行列）、そして入口から ATM 待ち行列までの経路に 1 人の顧客がいる。

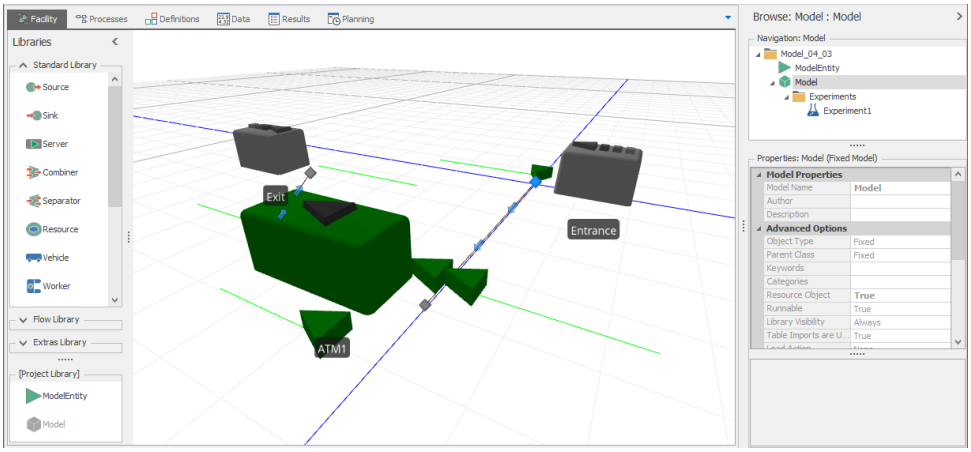


図 4.33　Model 4-3 の 3D ビュー

Model 4-4 では、Model 4-3 のアニメーションを充実させる。まず、Model_04_03.spfx に新しい名前をつけて保存する必要がある。Model_04_04.spfx として、ダウンロードしたファイルと混同しないよう、システム上の別のディレクトリに保存することを推奨する。Simio の Save As 機能は、Project Home タブ左の黄色いプルダウンタブにある。

シンボルやオブジェクトをクリックすると、Symbols リボンが表示される。このリボンでは、シンボルの色やテクスチャの変更、新規シンボルの追加、デフォルトシンボルの置き換えなど、さまざまな操作が可能である。まずは基本的な作業として、Simio の内蔵シンボルライブラリから新しいシンボルを選択し、エンティティの見た目をデフォルトの緑色三角形から人物の姿に変更する。

最初に、ATM Customer という名前の Entities オブジェクトを選択する。Symbols リボンの Apply Symbols ボタンをクリックすると、図 4.34 に示すように既存のライブラリが上部に表示される。ライブラリには 27 カテゴリ、300 以上のシンボルが収録されている。Domain、Type、Action の 3 つのフィルタを使用して、シンボルを効率的に検索できる。これらのフィルタは組み合わせて使用でき、必要なシンボルを絞り込むことができる。たとえば、Type フィルタで People オプションを選択すると、ライブラリ内の人物シンボルがすべて表示され、最上部に People カテゴリが表示される。シンボル上にマウスを重ねると拡大プレビューが表示されるため、クリックする前に確認できる。エンティティの新し

いシンボルとして Female を選択すると、図 4.35 のようにモデルのエンティティが変更される。なお、People フォルダ内の Animated サブフォルダには、Walking、Running、Talking などの動作を表現したリアルな人物シンボルも用意されているが、この小規模な ATM モデルでは必要ない。

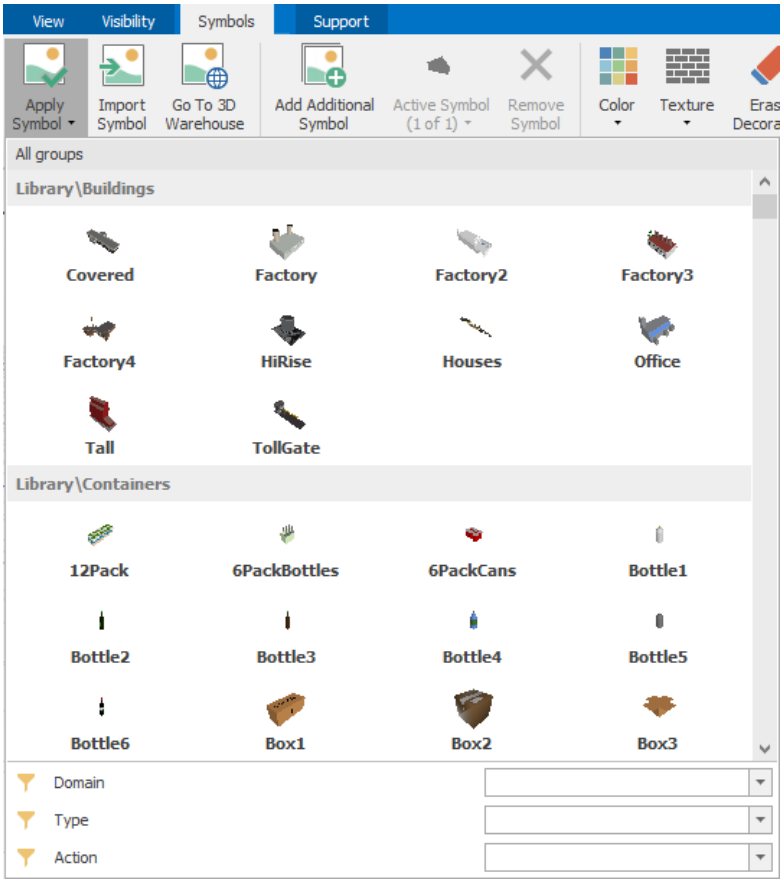


図 4.34　Simio シンボルライブラリのナビゲート

また、図 4.35 では、ATM 機のシンボルが含まれていることに気づくだろう。このシンボルはライブラリに標準で収録されているものではない。もし手元に適切な ATM の画像があればインポートすることができ、絵心があるならば自分で描くことも可能である。しかし、Simio はより簡単な解決策を提供している。それは、**Trimble 3D Warehouse** からのダウンロードである。これは無料で利用できる豊富なシンボルの宝庫であり、Simio から直接アクセスが可能である。では、サーバのピクチャを ATM 機に変更してみよう。

まず、ATM1 という名前をつけた Server オブジェクトを選択し、Symbol リボンから Go To 3D Warehouse アイコンをクリックする。すると、デフォルトの Web ブラウザで 3D Warehouse の Web ページ（https://3dwarehouse.sketchup.com）が開かれる。検索ボックスに ATM と入力し、虫眼鏡アイコンの Search ボタンをクリックして Models タブを選択する。ATM という単語を含む数百個のシンボルが表示され、図 4.36 のようなものを見つけることができる。ただし、ライブ

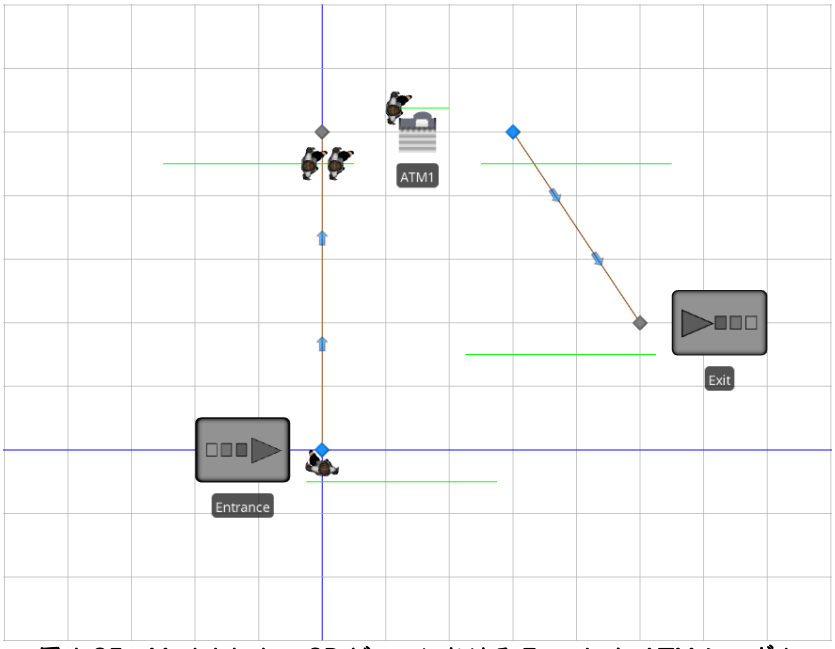


図 4.35　Model 4-4 の 2D ビューにおける Female と ATM シンボル

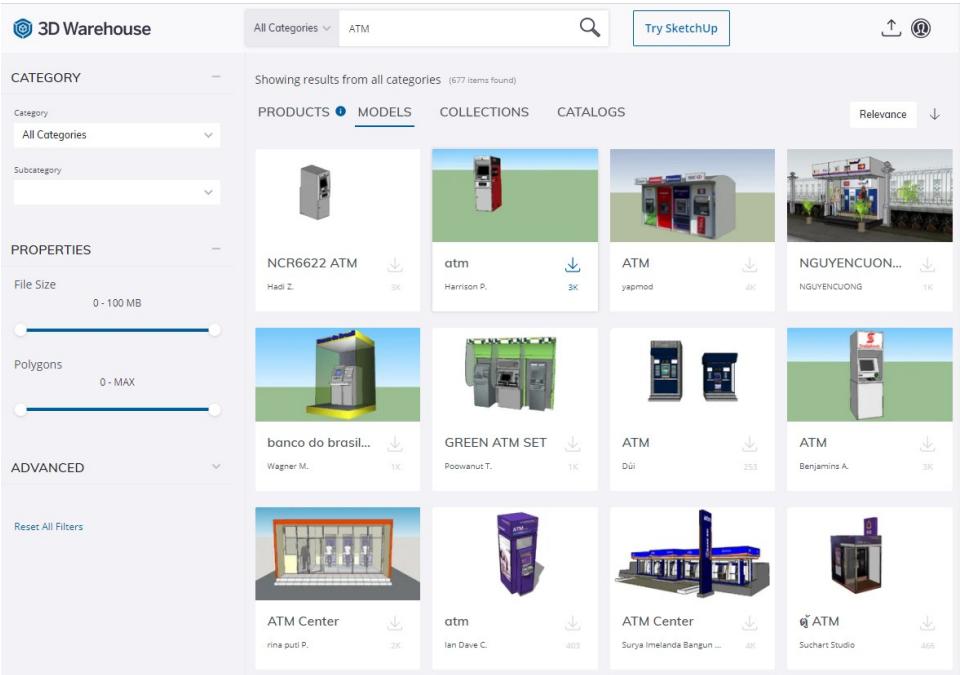


図 4.36　Trimble Warehouse における ATM の検索結果

ラリは定期的に更新されるため、実際の検索結果は図と異なる場合もある。また、Web ページのデザインや操作方法も、本書の説明と異なる可能性がある。

　これらの多くは自動 ATM ではないが、最初の画面で少なくとも１つは見つかるであろう。目的のオブジェクト（ここでは Mesin ATM）をクリックすると、フ

ァイルサイズなどの基本情報が表示される。See more details をクリックすると、ブラウザで 3D モデルを表示し、回転させることができる。問題がなければ、Download を選択して skp ファイルをコンピュータに保存する（シンボルをダウンロードするには 3D warehouse のアカウントが必要である）。Simio に戻り、Server オブジェクトのインスタンスを選択した状態で Import Symbol アイコンを使用し、選択した Server にシンボルをインポートして適用できる。これにより 3D モデルが Simio にインポートされ、シンボルの名前、サイズ、向きを調整してオブジェクトインスタンスに適用できる。一度インポートしたシンボルは、Apply Symbol アイコンを使用して他のオブジェクトインスタンスにも適用でき、再インポートは不要である。

インポートする際の重要なポイントの 1 つは、メートル単位でのサイズが適切かどうかの確認である。寸法の比率は固定されているが、全体的なサイズは自由に変更できる。今回の ATM は幅 0.7 メートル、高さ 2 メートルであり、これは適切なサイズといえよう。OK をクリックすると、ATM サーバに新しいシンボルが適用される。

2D ビューで実行すると、新しい Model 4-4 は図 4.35 のように表示される。3 キーを押して 3D ビューに切り替えると、図 4.37 のように新しいシンボルが立体的に表示される。

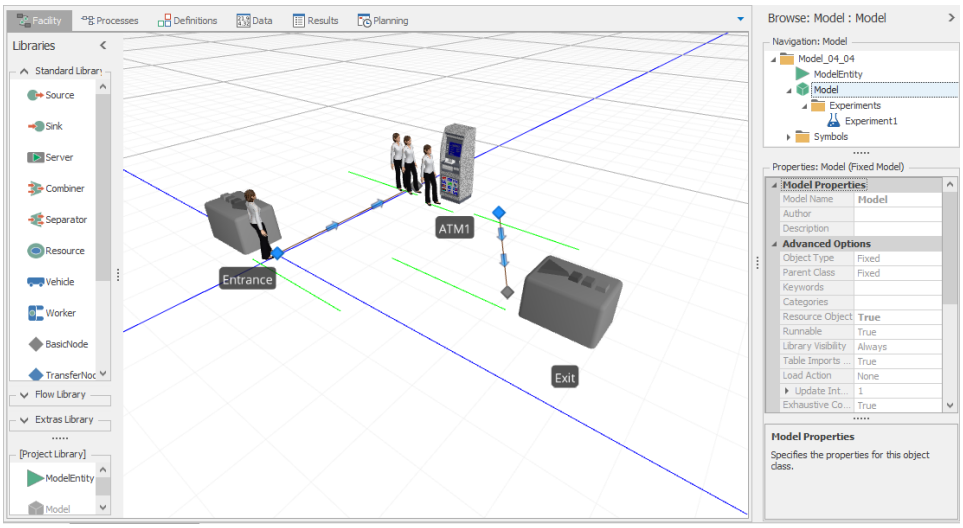


図 4.37　Model 4-4 の 3D ビューにおける Female と ATM シンボル

もちろん、ここで紹介したアニメーションの可能性は、ほんの一部に過ぎない。Drawing リボンを使って壁を描いたり、扉や植物などの装飾を追加したりできる。また、より現実的なモデル表現のために、概略図や背景画像をインポートすることも可能である。これらの詳細については 8 章で説明する。

4.9　モデルのデバッグ

人間は誰でもミスを犯すものだ。ソフトウェアに存在する誤りは**バグ**と呼ばれ

る。バグの原因はさまざまで、入力ミス、システムの仕組みへの誤解、シミュレーションソフトウェアの動作に関する理解不足、またはソフトウェア自体の問題などがある。経験豊富なシミュレーション専門家でさえバグに遭遇する。実際、**モデリングにおいて、もっとも労力を要するのはバグの解決である**。複雑なシステムのモデル化には複雑なソフトウェアの使用が不可欠で、初めて作成する実用モデルにバグが含まれるのは避けられない。そのため、バグを効果的に特定し除去する能力が、モデル作成者の生産性を左右する。本節では、デバッグ能力を向上させるための方法を解説する。

もっとも効果的な方法は、バグの影響を最小限に抑える反復型開発手法である（1.5.3 項参照）。モデルの動作確認をせずに長時間の構築作業を続けると、必ず複雑で発見困難なバグに遭遇する。そのため、頻繁にモデリングを中断して検証することを推奨する。問題を見つけたら、その原因を徹底的に考察することで、バグの迅速な特定と修正の能力が身につく。

バグに遭遇した際のもっとも一般的な第一反応は、ソフトウェア自体の問題だと考えることである。優れたプログラムでじゅうぶんな検証を経ていても、複雑なソフトウェアにはバグが存在するものだが、大半の問題はユーザ側の誤り、つまりモデル作成者自身に起因する。他に原因があると証明できない限り、問題は自分にあると考えてすぐにバグ修正に着手するのが最善の方法である。

4.9.1　気づきにくい問題の発見

問題をどのように発見するか。多くの問題は一目瞭然である。Run をクリックすると、何も起こらないか、明らかな異常が発生する。しかし、もっとも厄介な問題は見つけにくく、慎重に探し出す必要がある。

- 4.2.2 項で述べたように、モデル実行前に結果を予測することが重要である。モデルの結果と予測を比較することが、問題発見のもっとも効果的な方法となる。4.2.5 項で解説した基本的な検証プロセスに加え、以下でより詳細な検証方法を説明する。
- アニメーションを観察する。動作すべき要素が実際に動いているか確認し、動いていない場合はその原因を特定する。
- アニメーションに詳細情報を追加する。浮動ラベルやフロアラベルを使用して、エンティティや他のオブジェクトに診断情報を表示する。
- 出力統計結果を綿密に確認する。出力値や値間の関係は論理的か検証する。たとえば、低稼働率のリソースの前に長い待ち行列ができているような矛盾はないか。必要に応じて、独自の統計量を追加して詳細な情報を収集する。
- 最後に、問題が残っていないかを確認するため、後述するデバッグツールを活用する。

4.9.2　デバッグプロセス

バグを発見したら、自身の作業に起因する誤りを特定し修正する責任がある。状況に応じて、以下のような手順を試みることができる。

- すべてのオブジェクトを確認する。特に最近追加したものや頻繁に変更したオ

ブジェクトに注意を払う。
  - デフォルト値から変更したプロパティをすべて確認する（Simio では、変更値は太字で表示され、そのカテゴリは展開される）。各変更が意図的であり、正確に行われたことを確認する。
  - デフォルトのままのプロパティも確認する。デフォルト値が想定と異なる場合があるため、値が適切かどうかを判断する。
- エンティティの流れを単純化する。まずモデルのエンティティを 1 つだけにして問題が再現するか確認し、必要に応じて 2 つ目を追加する。少数のエンティティでも多くの問題が特定できる。この手法は他のデバッグ作業でも有効だ。Simio では、ソースの Maximum Arrivals を 0、1、2 に設定してエンティティ数を制限できる。
- モデルを最小化する。モデルを保存した上で、問題に無関係と思われる要素を削除する。削除しすぎた場合は元に戻し、別の部分を削除する。モデルが単純なほど、問題の特定と解決が容易になる。
- 警告やエラーメッセージを注意深く確認する。わかりにくい場合でも、通常は有用な情報が含まれているため、よく読み解く。
- エンティティの流れを段階的に追跡する。エンティティの動作とその理由を正確に把握する。意図した経路を進まない場合、誤った指示をしているか、必要な指示が不足している可能性がある。出力結果から手がかりを得ることもできる。
- 視点を変える。問題を異なる角度から見直す。プロパティを確認する際は、先頭からではなく最後から確認してみる。オブジェクトの場合も、通常と逆の順序で確認する。これにより新しい気づきが得られることがある。銀行では、数値を左から右へ読む行員と右から左へ読む行員でダブルチェックを行う。これは、人が期待に基づいて認識しがちなパターンを打破する方法である。
- 他者の協力を得る。同分野に詳しい仲間がいれば、助言を求める。また、専門知識のない人に詳しく説明するのも効果的だ。一人でも、熱帯魚やぬいぐるみに説明するような形で試せる。一見滑稽に思えるが、声に出して説明することで問題を新たな視点で捉えられる。
- RTFM（Read The Friendly Manual：親切な（とされる）マニュアルを読む）。マニュアルを読むのは気が進まないかもしれないが、他の方法で解決できない場合は、テキスト、マニュアル、対話型ヘルプを活用する場面である。

これらの手順は必ずしも順番通りに行う必要はない。最後から始めたり、一部を省略したりすることで、よりよい結果が得られることもある。次項では、デバッグプロセスを効率化するためのツールについて説明する。

4.9.3　デバッグツール

アニメーションと数値出力結果はデバッグの手がかりとなるが、高機能なシミュレーションソフトウェアには、モデル作成者がモデルの動作を理解するための支援ツールが備わっている。以下では、基本的なツールとして Trace（追跡）、Break（中断）、Watch（監視）、Step（ステップ実行）、Profiler（プロファイル）、

Search Window（検索ウィンドウ）について順に説明する。

Trace（追跡）は、モデル実行中に発生する事象の詳細な説明を提供する機能である。この機能は、エンティティの流れやイベント、およびそれらが引き起こす結果を記録する。SimioのTraceは、Process Stepレベルで動作し、各プロセスステップが１つ以上の追跡命令を生成する。SimioのTraceは、プロセスの仕組みに慣れるまでは理解が難しい場合があるが、Stepの概念を把握すれば、その詳細な情報の有用性が実感できるようになる。また、Simio Traceには使いやすいフィルタ機能が搭載されており、モデル実行後の分析用に外部ファイルへのエクスポートも可能である。

Break（中断）は、特定の条件でシミュレーションを一時停止する機能である。基本的な機能として、指定した時点やエンティティが特定の場所（例：サーバ）に到着した時点でシミュレーションを停止できる。より高度な機能では、「3つ目のエンティティがAという場所に到着する」や「時間125以降に到着する最初のエンティティ」といった条件つきの一時停止も可能である。Simioでは、基本的なBreak機能はオブジェクトやステップの右クリックメニューから、高度な機能はBreakウィンドウからアクセスできる。

Watch（監視）は、モデルのシステム状態をチェックする機能である。シミュレーションを一時停止した際に、モデルやオブジェクトの状態を確認し、モデル内の決定やアクションの理由、過程、影響についての理解を深めることができる。SimioのWatch機能は、すべてのオブジェクトの右クリックメニューから利用できる。この機能では、プロパティ、状態変数、関数などの各種要素にアクセスし、オブジェクトの階層構造に沿って詳細な分析が可能である。

Step（ステップ実行）は、小単位の活動ごとに時間を進め、モデルの実行を制御する機能である。これによりモデルのアクションとその結果を詳細に検証できる。Simioには２つのステップモードがある。Facilityビューでは、Stepボタンでアクティブエンティティを次の活動時間まで進められる。Process ウィンドウでは、エンティティ（トークン）を次のプロセスステップに移動できる。

Profiler（プロファイル）は、実行速度の問題解決を支援する。この機能は、処理速度低下の原因を特定するための内部分析を行う。プロセッサに負荷がかかるステップを特定することで問題点が明確になり、別のモデリング手法で実行速度を改善できる機会を見出せる。

Search（検索）は、検索キーワード（たとえば、シンボル名）を入力すると、プロジェクト内のすべての箇所を動的に検索する機能を提供する。たとえば、「Teller」というオブジェクト名が使用されているすべての箇所を見つけるのに、Search Windowは効果的なツールとなる。また、状態変数として使用している「Counter」を別の表現に変更したい場合にも活用できる。

Trace、Break、Watch、Stepは、強力なデバッグツールセットとして組み合わせて利用できる。これらのツールを上述のデバッグプロセスと組み合わせることで、モデルの理解を深め、最適な結果を導き出すための優れた手段となる。

Trace、Errors、Breakpoints、Watch、Search、Profileのウィンドウには、

Project Home リボンからアクセスできる。これらのウィンドウは、通常ユーザのアクションに応じて自動的に開く。たとえば、Run リボンの Model Trace を操作すると Trace ウィンドウが開き、構文エラーが発生すると Errors ウィンドウが自動的に表示される。また、必要に応じてこれらのボタンを使用し、ウィンドウの再表示や終了（Errors ウィンドウなど）、あるいは Breakpoints ウィンドウの新規表示なども可能である。

図 4.38 に、これらの機能を活用する典型的な画面を示す。ハイライトされている Model Trace ボタンは、Trace ウィンドウを表示し、モデル追跡を生成するために使用される。Server2 の進入ノード（赤のシンボル）に設定された Break ポイントに達すると、モデルの実行は自動的に一時停止する（実行された最後の操作は濃い色で強調表示される）。この時点で Step ボタン（左上）をクリックすると、エンティティは次の時間進行まで進み（黄色の背景）、さらに 10 行の追跡が生成される。画面右側の Watch ウィンドウでは、Server2 の入力バッファとバッファ内の各エンティティを監視できる。赤く強調表示されたテキストは、最近変更された箇所を示している。

デバッグウィンドウは、デフォルトで同一画面上の複数のタブとして表示される。各ウィンドウは図 4.38 のようにドラッグして再配置できる。4.1.8 項で述べたように、ユーザのニーズに合わせたウィンドウ配置も可能である。これらのウィンドウは他の画面上に重ねて配置できるため、まれにウィンドウの位置を見失うことがある。その場合は、Project Home リボンの Reset ボタンをクリックすることで、すべてのウィンドウの位置をリセットできる。

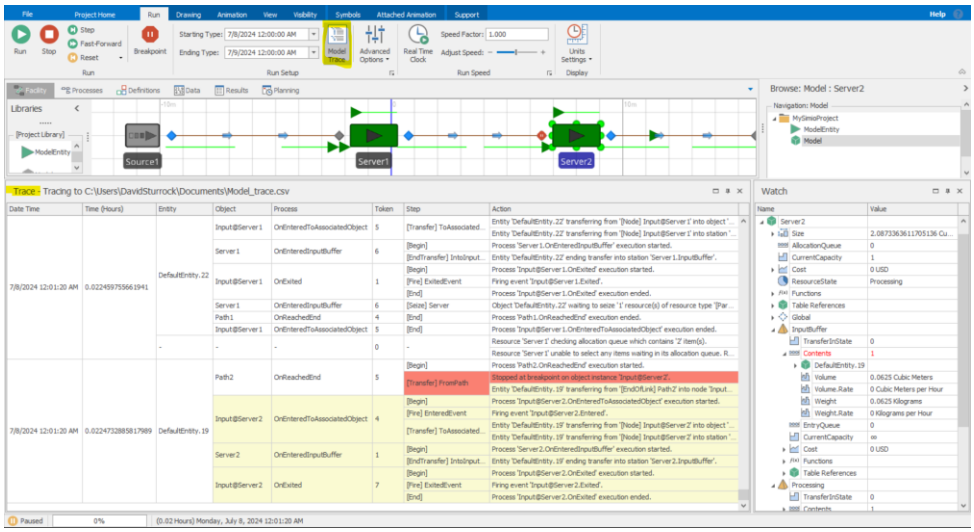


図 4.38　カスタムレイアウトによる Trace、Watch、Break ウィンドウの利用

4.10　まとめ

本章では、Simio を導入し、Standard ライブラリと Simio プロセスを活用して、いくつかの基本的なモデルを構築した。シミュレーションの出力における統計分析について詳しく説明し、実際のプロジェクトにおいて、モデル化と同様に

重要な反復、実行期間、ウォームアップ、モデル検証、そして SMORE プロットによる分析能力について解説した。待ち行列の理論モデルを出発点とし、現実的な待ち行列システムのモデル化に必要な重要な背景知識を加えた。また、Simio の Path を使用したエンティティ移動のモデル化やアニメーションの基礎についても取り上げた。次章以降では、さらに高度なモデルを用いて Simio とその関連トピックについて探求していく。

4.11　問題

1．Model 4-1 と同様の状態のモデルを作成しなさい。毎時 120 のエンティティが到着する到着率λと毎時 190 のエンティティを処理するサービス率μを用いて、100 時間実行しなさい。生成されたエンティティの数、サービスを終了した数、およびエンティティがシステムで費やした平均時間を報告しなさい。

2．問題 1 の Simio モデルを待ち行列モデルに展開し、100 時間におけるエンティティのシステム滞在時間と処理されるエンティティ数の期待値を計算しなさい。

3．問題 1 のモデルを用いて、100 回の反復実験を行いなさい。エンティティのシステム滞在時間を SMORE プロットで観測し、さまざまな設定（上限・下限の百分位数の変更、プロットの回転、ヒストグラムの確認）で実験を行いなさい。

4．問題 3 の実験を複数回（5 回程度）実行すると、到着時間間隔とサービス時間がランダムであるにもかかわらず、常に同じ結果が得られる。この現象が起こる理由を説明しなさい。

5．あるシステムのモデルを構築し、検証の一環として期待される結果を設定する。しかし、モデル実行時の結果が期待値と一致しない。このような不一致が生じる可能性のある理由を 3 つ挙げなさい。

6．シミュレーションモデリングにおいて、**反復**とは何か。また、特定のモデルで必要な反復回数をどのように決定するか？

7．**定常状態**シミュレーションと**終結**シミュレーションの違いを説明しなさい。

8．定常状態シミュレーションにおける**初期過渡期間**と**ウォームアップ期間**について説明しなさい。

9．問題 1 のモデルを Simio プロセスを使用して複製しなさい（Standard ライブラリのオブジェクトは使用しない）。100 時間の実行を 50 回反復し、問題 1 のモデルとの実行時間を比較しなさい。

10．ATM モデル（Model 4-3）を 240 時間（10 日間）にわたって 10 回反復実行しなさい。システム内の顧客の最大数と平均最大数を求めなさい（このモデルでは ATM の物理的なスペースを考慮していないことに注意）。この仮定は妥当であったか（物理的なスペースを考慮する必要がなかったか）？

11．SMORE プロットが、モデルの実行に関連したシステムの**リスク**と**サンプリング誤差**の全体像をどのように示すか説明しなさい。

12．問題 1 のモデルにアニメーションを追加しなさい。ファストフード店のレジ係をモデル化し、エンティティを顧客、サーバをレジ係として表現する。アニメーションには、Simio の標準のシンボルを使用すること。

13．問題 1 のモデルを次のように変更しなさい。鋼板の穴あけ加工をモデル化する。ボール盤の容量は一度に 3 つの部品までとする（待ち行列用語では$c = 3$）。部品の到着率は毎時 120 個、加工率は毎時 50 個とする。エンティティ（鋼板）とサーバ（ボール盤または穴あけ装置）の適切なシンボルは Trimble 3D Warehouse から選択すること。アニメーシ

ョン上に、現在処理中の部品数を示すラベルを追加すること。

14．次の 4 つの待ち行列モデルについて、定常状態の待ち行列理論による結果を確認・検証するための Simio モデルを構築する。定常状態の出力性能指標は、2.3 節に記載されている。Simio モデルは空かつ遊休の状態から開始され、統計的な変動の影響を受けやすいことに注意する。これらの課題に対処するため、適切な Simio Experiments を設計・実行する。実験では、試行錯誤を通じて適切な実行期間、反復回数、ウォームアップ期間を決定する。各モデルについて、まず 2.3 節の方法を用いて待ち行列理論による定常状態の出力性能指標ρ、L、L_q、W、W_qを計算する。その後、シミュレーションによる推定値および信頼区間と比較する。なお、すべての時間単位は分とし、Simio モデルも同様に設定する。

a) $M/M/1$の待ち行列で、到着率は毎分$\lambda = 1$、サービス率は毎分$\mu = 1/0.9$である。
b) $M/M/4$の待ち行列で、到着率は毎分$\lambda = 2.4$、4 つの各サーバのサービス率は毎分$\mu = 0.7$である（図 2.2 の mmc.exe コマンドラインプログラムと同じパラメータを使用）。
c) $M/G/1$の待ち行列で、到着率は毎分$\lambda = 1$、サービス時間はガンマ分布$(2.00, 0.45)$に従う（形状パラメータと尺度パラメータ）。ガンマ分布の特性については、6.1.3 項にウェブリンクを参照すること。
d) $G/M/1$の待ち行列で、到着時間間隔は 1 分から 5 分の間の一様分府に従い、サービス率は毎分$\mu = 0.4$である（図 2.3 と同じ設定）。

15．4.3 節で構築したプロセスによるモデルでは、システム内時間の決定に標準の Token.TimeCreated トークン状態変数を使用した。このモデルを修正し、到着時刻を手作業で**記憶**する方式で同様のモデルを構築しなさい（図 4.13 に示すように、到着時刻をシステム内時間の記録に使用する）。

ヒント：この値を保持するために、状態変数を持つ独自のトークンを作成する必要がある。また、トークン生成時のシミュレーション時刻を保存するために Assign ステップを使用する（カスタムトークンの追加は Definitions タブの Tokens セクションで行い、実装する Process のプロパティ Token Class Name を新しく作成したトークンに設定する。詳細な手順は Simio ヘルプを参照）。

第 5 章　Simio の中級モデリング

本章では、4 章で学んだ Simio の基本的なモデリングと分析の概念を発展させ、より現実的なシステムのモデル開発・実験に取り組む。はじめに Simio の仕組みと全体的なフレームワークについて詳しく解説し、続いて前章の単一窓口待ち行列モデルを拡張して、複数のプロセスや条件による分岐・合流などの機能を追加する。モデル開発の過程で新しい Simio の機能を順次紹介し、また共通目標に対する複数の代替シナリオを比較する観点から、統計的なシミュレーション実験の計画・分析方法を再検討する。本章を修了する頃には、Simio を用いてある程度複雑なシステムをモデル化し、分析する方法をじゅうぶんに理解できるようになる。

5.1　Simio のフレームワーク

これまでに多くの Simio 固有の概念を紹介してきた。本節では、以降の内容を理解する上で重要となる概念についてより詳しく解説する。プログラミング経験のない方には、本節の一部が詳細すぎると感じられるかもしれないが、本章および本書の後半で全体像が明らかになっていく。

5.1.1　オブジェクトの概要

コンピュータプログラミングの世界では、**オブジェクト指向プログラミング**（OOP）がモダンなソフトウェア開発の事実上の標準となっている。C++、Java、C#などの主要な言語でもオブジェクト指向がサポートされており、離散型イベントシミュレーション（DES）も同様の道をたどっている。多くの DES 製品が OOP で開発されているものの、シミュレーション担当者にとって重要な点は、シミュレーションモデルの構築過程で真にオブジェクト指向の利点を活かせる製品が少ないことである。その中で Simio、AnyLogic、FlexSim、Plant Simulation の 4 製品は、モデル開発者に真の OOP ツールキットを提供している。

なぜこれが重要なのか疑問に思うかもしれない。OOP がソフトウェア開発業界に革命を起こしたのと同様の理由で、シミュレーション業界でも革新をもたらしているからである。オブジェクトを活用することで、大規模な問題をより管理しやすい小さな問題に分割できる。オブジェクトによってモデルの信頼性、堅牢性、再利用性、拡張性、保守性が向上し、結果として全体的なモデリングの柔軟性と効率が大幅に改善される。場合によっては必要なモデリングの専門知識レベルも低減できる。

本節の残りでは、Simio のオブジェクトの種類と重要な用語を紹介し、続いて Simio オブジェクトの特徴をより詳しく説明する。オブジェクトの概念に初めて触れる方には、オブジェクト指向の理解と効果的な活用がいかに重要であるか、本節を通じて理解していただけるだろう。

5.1.1.1　オブジェクトとは？

Simio はモデリングにオブジェクトアプローチを採用しており、システムの物理的な構成要素を表すオブジェクトを組み合わせてモデルを構築する。**オブジェ**

クトは自己完結型のモデリング要素であり、特性、データ、ふるまい、ユーザインタフェース、アニメーションを定義する。オブジェクトはモデル構築におけるもっとも基本的な構成要素であり、Standard Library（Simio に標準搭載された汎用オブジェクトのセット）を使用したモデル作成では、すでにオブジェクトを活用している。Model 5-1、5-3、5-4 では、Simio の Standard Library から Source、Server、Sink、Connector、Path オブジェクトを使用した。

オブジェクトは、システム内の**イベント**に応答する独自のふるまいを持ち、それはオブジェクトの内部モデルで定義される。たとえば、生産ラインのモデルでは、機械、コンベヤ、フォークリフト、通路などを表すオブジェクトを配置してモデルを構築する。一方、病院のモデルでは、職員、病室、ベッド、治療機器、手術室などを表すオブジェクトでモデル化する。Standard Object Library のオブジェクトを使用したモデル構築に加え、特定の応用分野向けに**独自の**オブジェクトライブラリを作成することもできる。Standard Library のオブジェクトのふるまいは、プロセスロジックを用いて変更・拡張が可能である。独自のオブジェクト構築方法については 11 章で説明する。

5.1.1.2　オブジェクトに関する用語

これまで「オブジェクト」という言葉を一般的に使用してきた。ここでオブジェクトに関する用語をより詳しく見ていく。まず、**Simio オブジェクト階層構造**における 3 つの層を説明する：

- **オブジェクト定義**（Object Definition）：オブジェクトの外観やふるまい、他オブジェクトとの相互作用を定義する。これは**プロパティ**、**状態変数**、**イベント**、**外観**、**ロジック**から構成される。オブジェクト定義は、プロジェクトまたはライブラリの一部となる。Server や ModelEntity がオブジェクト定義の例である。オブジェクト定義を編集するには、Navigation ウィンドウでそのオブジェクトを選択する。この操作により、他のモデルウィンドウ（Processes や Definitions ウィンドウなど）がすべてそのオブジェクトに対応する。
- **オブジェクトインスタンス**（Object Instance）：モデルにオブジェクトをドラッグ（または**インスタンス化**）する際に作成される。オブジェクトインスタンスはプロパティ値を含み、1 つ以上のシンボルを定義できるが、その他のすべての要素は対応するオブジェクト定義から参照される。オブジェクトインスタンスはモデル内のみに存在するが、単一のオブジェクト定義から複数のインスタンスを作成できる。Server1、ATM1、DefaultEntity、ATMCustomer などがオブジェクトインスタンスの例である。オブジェクトインスタンスを編集するには、Facility ビューでそのインスタンスを選択する。これによりプロパティ値とアニメーションを変更できる。
- **オブジェクト実行空間**（Object Runspace）：オブジェクト実行空間（または**オブジェクトの実現**）は、オブジェクトの状態変数の現在値を保持する第 3 の層である。各実行空間は固有の ID を持ち、可変のシンボルを参照できる。オブジェクト実行空間は実行開始時に作成され、実行中に動的に作成・破棄される

場合もある。実行終了時にはすべての実行空間が破棄される。動的オブジェクト(エンティティなど)の実行空間の状態変数は、InstanceName[InstanceID].StateName の形式で参照できる。たとえば、LiftTruck[2].Priority は 2 番目の LiftTruck の優先順位の状態変数を参照する。

Entity オブジェクトの 3 つの層を図 5.1 に示す。1 つのエンティティ定義（図では人）があり、そのエンティティインスタンス（大人（Adult）と子供（Child））がモデルに配置される。モデル実行時には多くの大人と子供の実行空間が作成され、それぞれが動的なエンティティの状態変数を保持する。Simio オブジェクト構造については 11.1 節、特に図 11.1 で詳しく説明する。

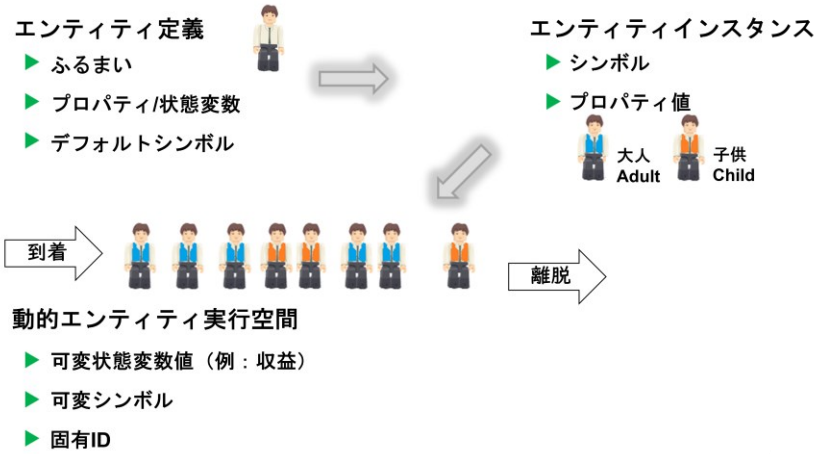


図 5.1　エンティティオブジェクトの 3 つの層

ここでは、オブジェクトを説明する際によく使用される 3 つの重要な用語について説明する：

- **モデル**（Model）：モデルはオブジェクト定義の別名である。システムやサブシステムのモデルを構築する際は、実質的にオブジェクトを構築していることになる。個々の構成要素を説明する際は「オブジェクト」という用語（例：Server オブジェクト）を、最上位のオブジェクトを指す際は「モデル」という用語（例：前章の Model 4-3 における Model_04_03 という名称）を使用することが一般的である。ただし、最上位のオブジェクトが他のモデルの構成要素として使用されることもある。
- **プロジェクト**（Project）：プロジェクトはオブジェクト定義（またはモデル）の集合体である。プロジェクト内のオブジェクトは通常、連携して動作したり、階層的な関係を持つように設計され、相互に関連づけられている。ファイル拡張子 SPFX は Simio プロジェクトファイルを表す。一般的に「プロジェクト」という技術用語を「モデル」と呼ぶことも多く、本書の例題もこの慣習に従っている（例：Model 4-1）。
- **ライブラリ**（Library）：ライブラリはオブジェクト定義の集合体であり、実質的にプロジェクトの別名である。すべてのプロジェクトファイルは、Project

Home リボンの Load Library ボタンを使用してライブラリとして読み込むことができる。

これらの概念は、既存の Simio オブジェクトを使用する場合だけでなく、独自のカスタムオブジェクトを作成する場合にも重要である。次の数章では既存の Simio オブジェクトを使用してモデルを構築し、11 章ではカスタム Simio オブジェクトの作成方法を説明する。技術的には、モデルを構築する時点ですでにカスタム Simio オブジェクト定義を開発していることになるが、これらは単に**モデル**と呼ぶ。オブジェクト指向の考え方に慣れていない場合、モデルがオブジェクト定義であるという概念は、最初はわかりにくいかもしれないが、Simio の使用経験を重ねるうちに理解できるようになるだろう。

5.1.1.3　オブジェクトの特徴

前述のように、オブジェクトはプロパティ、状態変数、イベント、外観、およびロジックによって定義される。以下でそれぞれについて詳しく説明する。

プロパティ（Property）は、モデル内にオブジェクトを配置（またはインスタンス化）する際に設定できる入力値である。たとえば、窓口を表すオブジェクトには、サービス時間を指定するプロパティがある。Model 5-1 では、Source オブジェクトの Interarrival Time プロパティで到着間隔の分布を、Server の Processing Time プロパティで窓口の処理時間の分布を指定した。

状態変数（State）は、モデルの実行中に変化する動的な値である。たとえば、窓口オブジェクトの稼働・遊休状態は、Status という状態変数によって管理され、顧客へのサービス開始・終了時に更新される。Model 4-2 では、`WIP` という状態変数を使用してシステム内のエンティティ数を追跡し、エンティティの到着時に増加、離脱時に減少させた。また、シミュレーション中の任意の時点で、特定の状態変数が全体的なオブジェクト状態の一部となることがある。単一窓口待ち行列システムの例では、システム状態は `WIP` モデル状態変数（システム内エンティティ数）と `Entity.TimeCreated` エンティティ状態変数（各エンティティの到着時刻）で構成される。

イベント（Event）は、オブジェクトが選択された時点で「発火する」事象である。3.3.1 項で説明した動的シミュレーションのイベントの概念に基づいている。単一窓口待ち行列モデルでは、エンティティの到着とサービス完了をイベントとして定義した。Simio ではイベントはオブジェクトに紐づけられ、たとえば窓口オブジェクトでは顧客対応完了時に、オイルタンクオブジェクトでは満タンまたは空になったときにイベントが発火する。イベントは重要な出来事を他のオブジェクトに通知したり、カスタムロジックを組み込む際に役立つ。

外観（External View）は、オブジェクトの 2D または 3D の視覚的表現である。これは Facility ビューで表示されるグラフィックスのことで、Standard Library のオブジェクトをモデルに追加すると確認できる。

ロジック（Logic）は、特定のイベント発火時にオブジェクトがどのように応答するかを定義する。たとえば、窓口オブジェクトでは顧客到着時の処理を指定で

き、これがそのオブジェクト固有の動作となる。

Simio の特徴的な機能の 1 つは、システムモデルの構築時に、入力プロパティと外観を追加するだけでモデルをオブジェクト定義に変更できることである。これにより、モデルを上位モデルのサブモデルとして再利用できる。この機能によって Simio では自然に階層的なモデリングが可能である。詳細は 11 章で扱う。

5.1.2　プロパティと状態変数

プロパティと状態変数についてはすでに簡単に触れたが、頻繁に使用される重要な概念であるため、さらに詳しく説明する。

5.1.2.1　プロパティ

プロパティは、オブジェクトの動作をカスタマイズし、ユーザから情報を収集するためにオブジェクト内で定義される（図 5.2）。ここでの**ユーザ**とは、オブジェクトを配置（インスタンス化）してプロパティ値を設定する人を指す。Standard ライブラリのオブジェクトを使用する場合は、Simio LLC が作成したオブジェクトを使うため、**モデル作成者自身**がユーザとなる。11 章で説明する方法で独自のオブジェクトを作成した場合は、そのオブジェクトを使用する開発者以外の人がユーザとなる。

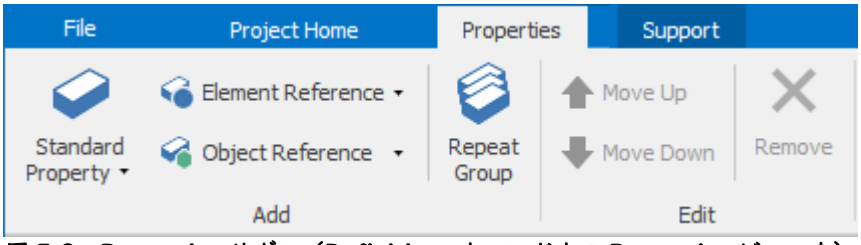


図 5.2　Properties リボン（Definitions ウィンドウの Properties ビュー内）

オブジェクトのプロパティを作成または表示するには、Navigation ウィンドウ（Facility ウィンドウ**ではない**）でオブジェクトを選択し、Definitions タブの Properties 項目を選択する。オブジェクト固有のプロパティは（存在する場合）ウィンドウの最上部に表示され、他のオブジェクトから継承されるプロパティは、その下の展開可能な領域に表示される。

プロパティには、以下の種類がある：

- Standard：Integer、Real、Expression、Boolean、DateTime、String…（整数、実数、式、真理値、日付、文字列など）
- Element：Station、Network、Material、TallyStatistic…
- Object：Entity、Transporter、その他のオブジェクト参照
- Repeat Group：上記いずれかのリピートグループ

ヒント：新しいプロパティを作成すると、対象オブジェクトの Properties ウィンドウに表示される。新しいプロパティが見当たらない場合は、正しいオブジェクトの Properties ウィンドウを表示しているか確認してほしい（例：ModelEntity オブジェクトに追加したプロパティは、Model オブジェクトや Server オブジェ

クトには表示されない)。プロパティにカテゴリを設定しない場合、デフォルトで General カテゴリに表示される。

プロパティは、実行時に**定義された値**を変更できない。たとえば、ProcessTime プロパティに値 5.3 を定義した場合、実行を停止しない限り、そのプロパティは常に 5.3 を返す。ただし、プロパティの定義に確率変数や動的状態変数を含めることで、定義自体は不変でも異なる値を返すことが可能である。たとえば、Model 4-3 では、Server オブジェクトの ProcessTime プロパティに `Random.Triangular(0.25, 1, 1.75)` を使用した。この場合、プロパティ定義自体は変わらないが（常にパラメータ 0.25、1、1.75 の三角分布)、各エンティティには指定された確率分布からの個別の**サンプル値**が割り当てられる。同様に、モデル実行中に変化する状態変数をプロパティ定義に指定すると、参照時点の状態変数の現在値が返される。

また、プロパティ値は各オブジェクトのインスタンス（Facility ウィンドウに配置されたオブジェクトのコピー）に固有のものである。たとえば、Facility ビューに Project Library から ModelEntity オブジェクトをドラッグして 3 つのインスタンス（PartA、PartB、PartC）を配置する場合、各インスタンスの Initial Desired Speed プロパティには異なる値を設定できる。ただし、プロパティは実行中に値を変更できないため、生成される動的エンティティ（実行空間）の数に関係なく、インスタンスに保存された初期値が使用される。

5.1.2.2　状態変数

状態変数はオブジェクト内で定義され（図 5.3)、モデルの実行中に値が変化する。状態変数（state）は「状態変数の変数（state variables)」とも呼ばれるが、本書ではこれらを同義として扱う。状態変数には離散型と連続型の 2 種類がある。**離散型状態変数**（Discrete State；離散変化型状態変数）は、特定の時点でのみ値が変化する（例：時刻 1.2457 と時刻 100.2 での変化)。**連続型状態変数**（Continuous State；連続変化型状態変数）は、変化率または加速度が 0 でない場合、連続的に自動更新される。

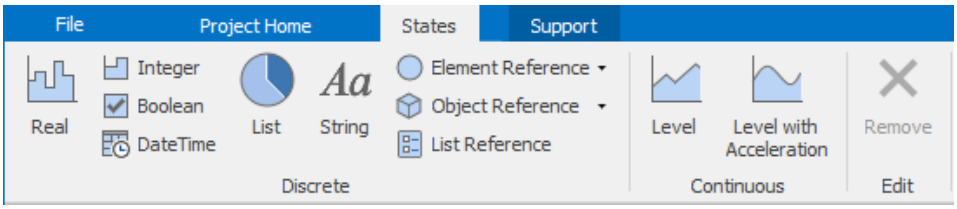


図 5.3　States リボン（Definitions ウィンドウの States ビュー内）

状態変数には、次に挙げる種類がある：

- **Real**（実数）：任意の数値をとる離散型状態変数
- **Integer**（整数）：整数のみをとる離散型状態変数
- **Boolean**（真理値）：**真**（1）または**偽**（0）の値をとる離散型状態変数
- **DateTime**（日付）：日付型で値を代入する離散型状態変数
- **List**（リスト）：String リストのエントリと整数値を対応づける（7.5 節で扱

う）。統計値やアニメーションでよく使用される

- **String**（文字列）：文字列（例："Hello"）を代入できる離散型状態変数で、String.Length のような**文字列操作関数**を用いて操作する
- **Element Reference**（エレメント参照）：エレメント参照値が代入される離散型状態変数（たとえば、TallyStatistic や Material のようなエレメント）
- **Object Reference**（オブジェクト参照）：オブジェクト参照値が代入される離散型状態変数（たとえば、一般的なオブジェクトやエンティティのような特定のオブジェクトを参照する）
- **List Reference**（リスト参照）：リスト参照値が代入される離散型状態変数（たとえば、LocalTrucks や LDTrucks のようなリストを参照する）
- **Level**（レベル）：実数値（レベル）および変化率を持つ連続型状態変数。変化率が 0 でないとき、状態変数は連続的に変化する。変化率は離散的にのみ変更されることに注意してほしい
- **Level with Acceleration**（加速度のあるレベル）：実数値（レベル）、変化率および加速度を持つ連続型状態変数。状態変数は、変化率と加速度の値に基づいて、時間とともに連続的に変化する。加速度が 0 でないとき、変化率は連続的に変化する。変化率と加速度の両方は、離散的に変更される

状態変数は、任意のオブジェクトに関連づけることができる。エンティティに関連づけられた状態変数は、エンティティの**属性**としてみなされることが多い。また、モデル上の状態変数は、モデル内のグローバル変数として捉えられる（たとえば、Model 4-2 ではモデルの状態変数として WIP が用いられた）。ただし、スループットやコストを記録するようなタスクを実行するために、他のオブジェクトの状態変数を保持することもできる。

状態変数は、基本的に Properties ウィンドウには**表示されない**が、初期値を指定するために使用されたプロパティについては Properties ウィンドウに**表示される**。オブジェクトの状態変数を作成または表示するには、Navigation ウィンドウ（Facility ウィンドウ**ではない**）でオブジェクトを選択し、Definitions タブの States 項目を選択する。たとえば、ModelEntity オブジェクトに状態変数を追加するには、最初に Navigation ウィンドウで `ModelEntity` をクリックする。適切なオブジェクトを選択すると、`Definitions` タブと `States` 項目を選択できる。このオブジェクト固有の状態変数が（もしあれば）ウィンドウの最下部に表示される。他のオブジェクトから継承される状態変数は、ウィンドウの上部にある展開可能な領域に表示される。

状態変数の値は、各オブジェクトの実行空間（すなわち、実行中に動的に生成されるオブジェクトのコピー）に固有であり、オブジェクトインスタンスに対しては**そうではない**。たとえば、モデルに設置した ModelEntity のインスタンス（PartA と呼ぶ）は、PartA のエンティティすべてに共通する Initial Desired Speed という名称のプロパティを保有できる。しかし、ModelEntity のすべてのインスタンスは、Speed という名称の状態変数も同時に保持できる。これらの状態変数は、モデルを移動する際に値を割り当てられるが、生成された各エンティ

ティ（動的実行空間）は異なる値の状態変数 Speed を持つかもしれない。したがって、状態変数 Speed の値は、動的なエンティティ実行空間に保持する必要がある。状態変数を保持するにはメモリ領域が必要となるため、実行中に値を変化させる必要がない限り、状態変数ではなくプロパティを用いるべきである。

5.1.2.3　オブジェクトプロパティの編集

Facility ウィンドウでオブジェクトを選択（クリック）すると、選択したオブジェクトのプロパティが画面右の Browse パネルにある Properties ウィンドウに表示され、編集できる。たとえば、Server オブジェクトを選択すれば、Server のプロパティが Properties ウィンドウに表示される。プロパティは展開と折りたたみが可能なカテゴリに分類されている。Standard Library の Server の場合は、Process Logic カテゴリが展開されており、他のものは折りたたまれている。プロパティ名の左側にある三角形のシンボルをクリックすると、カテゴリを展開または折りたたむことができる。プロパティを選択すると、グリッドの下部にプロパティの説明が表示される。上述のように、プロパティは文字列、数値、リストあるいは式などさまざまな種類をとりうる。これらは多くの場合、プロパティに反映される。たとえば、Server の Ranking Rule はドロップダウンリストから選択される一方、Processing Time は式として指定される。プロパティフィールドに無効なエントリを入力すると、フィールドがオレンジ色になり、画面下部にエラーウィンドウが自動的に開く。エラーウィンドウのエラーをダブルクリックすると、エラーが発生したプロパティフィールドが自動的に選択される。エラーを修正すれば、エラーウィンドウは自動的に閉じる。

多くの OOP と同様に、Simio はオブジェクトのデータを示すために**ドット表記**を用いている。一般的な形式は「xxx.yyy」であり、yyy が xxx の構成要素であることを示す。下位の構成要素、あるいはさらに下位の構成要素を示すために、繰り返して用いられることもある。たとえば、Server1 に割り当てられた能力の平均を得るには、`Server1.Capacity.Allocated.Average` と記述する。このような表記方法は Properties ウィンドウでは隠されているが、ドロップダウンリストや式ビルダを用いる際に目にすることがある（式ビルダについては、4.5 節のはじめに簡単に、5.1.7 項で詳しく解説する）。慣れれば式ビルダを用いずに、直接ドット表記で入力できるようになる。

5.1.2.4　プロパティ参照

プロパティ参照は、一般に他のプロパティの定義内で活用するための、プロパティの特殊な利用方法である。たとえば、`PaintingTime` という名称のプロパティ参照を定義し、Server オブジェクトの ProcessingTime の定義として `PaintingTime` を用いることができる。このように既存のプロパティを利用するか、あるいは図 5.4 に示すとおり、プロパティを右クリックして新規のプロパティ参照を作成できる。

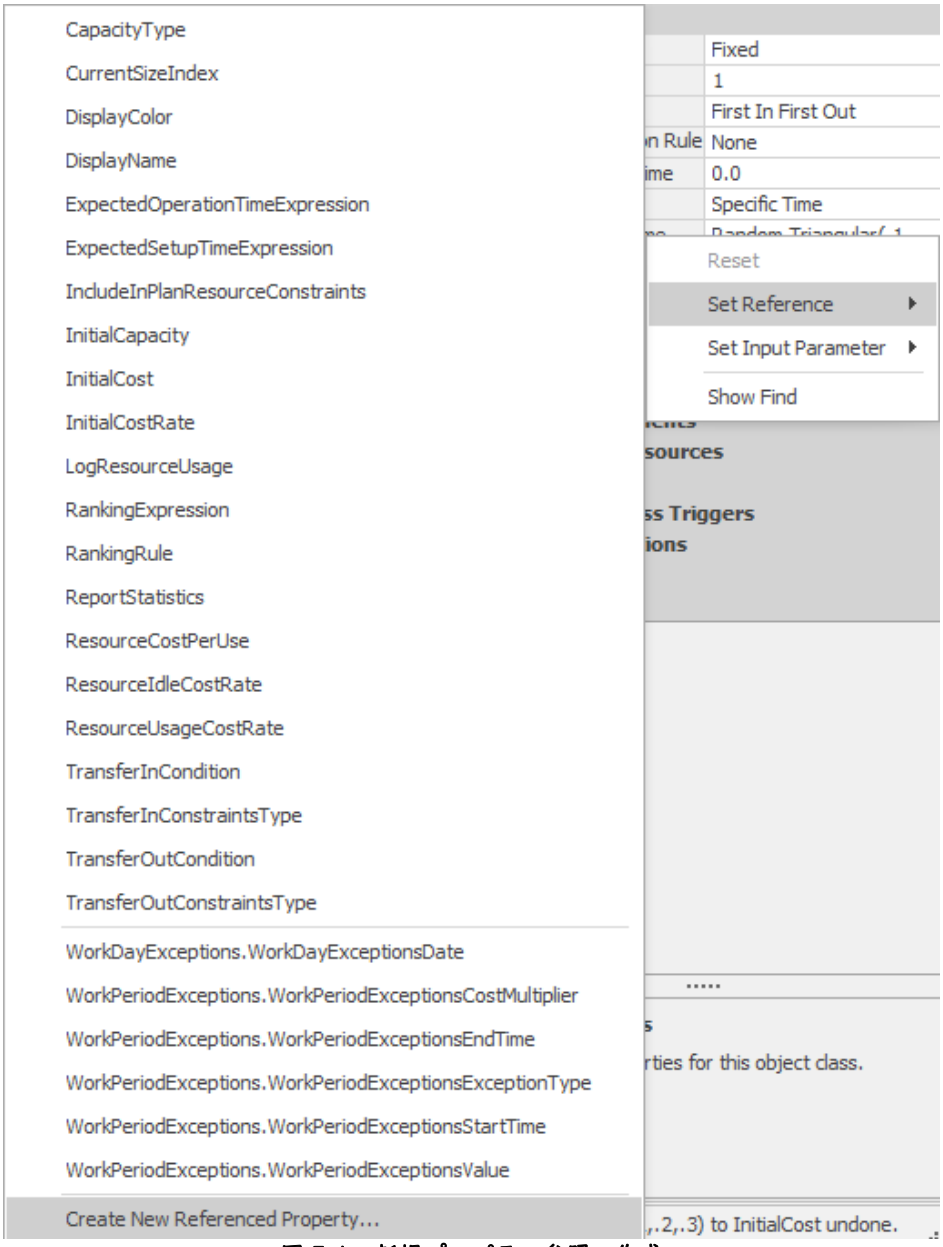


図 5.4　新規プロパティ参照の作成

プロパティ参照を用いる主な理由は以下の 3 つである：

- モデル作成者以外のユーザが、主要なパラメータを（Controls として）容易に検索・変更できる。プロパティ参照はすべて、モデルの Properties ウィンドウの Controls カテゴリに表示される（Navigation ウィンドウで Model を右クリックし、Properties を選択）。さらに、Definitions の Properties ウィンドウで Default Value、Display Name、Category Name などを変更することで、表示方法をカスタマイズできる。
- プロパティを共有する効率的な手段となる（例：一度プロパティで値を指定す

れば、複数の箇所でその値を参照できる）。これにより、異なるプロパティ値での実験が容易になる。

・Experiment ウィンドウでモデル実験を実施する際、プロパティ参照は**決定変数**（Controls）として自動的に表示される（Visible の値が False でない場合）。これらの決定変数は、窓口能力の変更などのように、複数の代替シナリオで変化させる項目として使用される（5.5 節参照）。

プロパティ値としてプロパティ参照を使用すると、緑色の矢印が表示される（図 5.5）。モデルのインタラクティブ実行時には、コントロールの値をモデルの Properties ウィンドウで設定できる。実験時には、モデルの Properties ウィンドウのコントロール値は無視され、代わりに Experiment Design ウィンドウの各シナリオに対応する値が使用される。プロパティ参照の具体例については、本章および後続の章のモデルで示す。

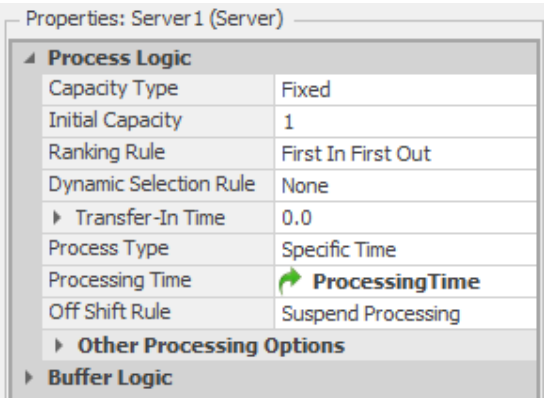


図 5.5　プロパティ参照の使用例

5.1.2.5　プロパティと状態変数のまとめ

表 5.1 に、プロパティと状態変数の主な違いを示す。初学者はプロパティと状態変数を混同しがちであり、片方に偏った使用をしてしまうことがある。ただし、さまざまな機能や構成要素を用いてモデルを構築し、その過程でプロパティと状態変数の使い方を実践的に学ぶことで、理解は深まっていく。

表 5.1　プロパティと状態変数の主な違い

	プロパティ	状態変数
基本データ型	22	9
実行中の変更	不可	可能
配列の利用	不可	可能
保有場所	オブジェクトインスタンス	オブジェクト実行空間
窓口での使用例	処理時間	処理数
エンティティでの使用例	初期速度	現在速度
コストでの使用例	時間当たりのコスト	賦課コスト
故障での使用例	故障率	最後の故障時刻
バッチでの使用例	希望バッチサイズ	現在のバッチサイズ

5.1.3　トークンとエンティティ

多くのシミュレーションパッケージには、エンティティという概念が存在する。エンティティとは、一般にシステム内を移動する部品や人間などの物理的な「モノ」を表す。ただし、システム制御ロジックのようなエンティティを含まない低位のロジックや、処理時間の進行中にイベントを待つといった複数のアクションを同時に行うエンティティの表現には、別の概念が必要とされる。Simio は、このような柔軟性を実現するためにトークンという概念を提供している。

5.1.3.1　トークン

トークン（token）は、プロセスのステップを実行するオブジェクトの代理である。通常、トークンはエンティティの代理となるが、実際にはプロセスを実行する**任意の**オブジェクトをトークンとして利用できる。トークンはプロセスの開始時に生成され、終了時に破棄される。プロセスを移動する際、トークンは各ステップで指定されたアクションを実行する。単一のプロセスは複数のトークンを並行して移動させることができ、また単一のオブジェクトは複数のトークンによって表現できる。

トークンは独自のプロパティと状態変数を保持できる。たとえば、トークンがロジックの特定ポイントを通過した回数を追跡する状態変数などである。多くの場合、Simio のデフォルトトークンでじゅうぶんであるが、独自のプロパティや状態変数が必要な場合は、Definitions ウィンドウの Token パネルで作成する。

トークンは親オブジェクトと関連オブジェクトの両方への参照を持つ(図 5.6)。

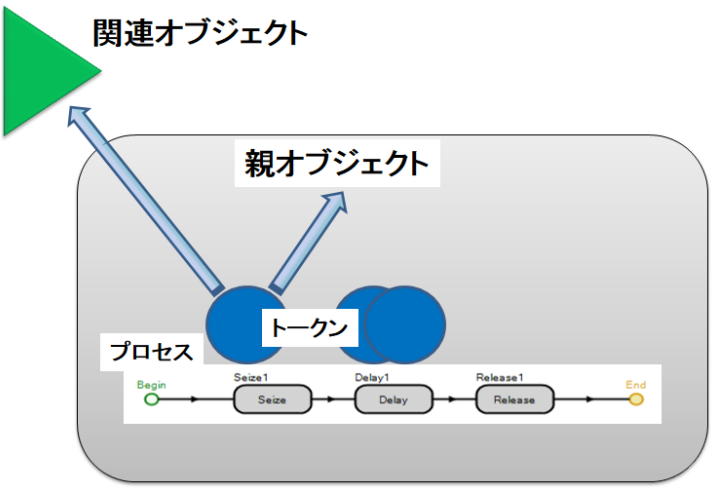


図 5.6　トークン、オブジェクトおよびプロセスの相互関係

親オブジェクトは、プロセスが定義されたオブジェクトのインスタンスである。たとえば、Server オブジェクト定義内のプロセスは、その親として窓口（Server1 など）を持つ。**関連オブジェクト**は、プロセス実行の起点となる（親オブジェクトとは異なる）オブジェクトである。たとえば、エンティティの到着によって開始されるプロセスでは、そのエンティティが関連オブジェクトとなる。処理ロジックは、親オブジェクトと関連オブジェクトの両方のプロパティ、状態変数、関数を参

照できる。そのため、トークンは Delay ステップの時間を指定する際に到着エンティティを参照できる。関連オブジェクトを参照する場合は、オブジェクトのクラス名をプロパティまたは状態変数の前につける必要がある。たとえば、`ModelEntity.TimeCreated` は、ModelEntity 型の関連オブジェクトの Time Created 関数の値を返す。

5.1.3.2　エンティティ

Simio において、**エンティティ**はオブジェクトモデルの一部として独自の知的なふるまいを持つ。エンティティは意思決定、要求拒否、休憩の決定などを行うことができ、モデル内の他のオブジェクトと同様にオブジェクト定義を持つ。エンティティオブジェクトは動的に生成・破棄され、リンクやノードのネットワークを移動し、3D 空間内や固定オブジェクトの内外を移動できる。

エンティティオブジェクトの例として、顧客、患者、部品、加工物などがある。エンティティはトークンとは異なり、プロセス内を直接移動しない。代わりに、プロセスを実行するためにエンティティの代理としてトークンが生成される。エンティティがオブジェクト内外を移動すると、それがイベントの引き金となってプロセスが実行され、トークンが生成されてプロセスのステップを流れていく。

エンティティは Facility ウィンドウ内に物理的な**位置**を持ち、Free Space、Station、Link、Node のいずれかとして存在する（8 章で詳述）。エンティティの位置は常に最新の位置が参照される。Simio 内のステーションの例として、Node の駐車ステーション（`TransferNode1.ParkingStation`）、Server オブジェクトの入力バッファ（`Server1.InputBuffer`）、Combiner オブジェクトの処理ステーション（`Combiner1.Processing`）がある。

5.1.4　プロセス

4.3 節では、Process のみを用いて全体のモデルを構築する極端な例を示した。実際には、すべてのオブジェクトのロジックはプロセスを利用して設定されるため、すべてのモデルがプロセスを使用している（11 章で詳述）。本項では、より一般的な用法であるアドオンプロセスについて説明する。

プロセスは、（シミュレートされる）時間の経過に伴って発生する動作のセットで、システムの状態変数を変更することもある。Simio では、プロセスはトークンによって実行される**ステップ**を用いたフローチャートとして定義され、複数の要素の状態変数を変更できる。図 5.6 に示した 3 ステップから成るプロセスを含むオブジェクト（親オブジェクト）を見ると、最初の青いトークンから出る矢印は、親オブジェクト（サーバなど）とその関連オブジェクト（プロセス実行を開始したエンティティなど）への参照を示している。他の青いトークンも同様の参照を持っている。

プロセスは以下の方法で**発火**される：

- **Simio 定義プロセス**は、Simio エンジンによって自動的に実行される。たとえば、`OnInitialized` プロセスは初期化時に各オブジェクトに対して実行される。
- **イベント発火プロセス**は、モデル内で発火するイベントによってトリガーされ

る、ユーザ定義のプロセスである。

・**アドオンプロセス**は、オブジェクトの**標準的な**ふるまいにカスタムロジックを追加できるよう、オブジェクト定義に組み込まれている。

以下では、アドオンプロセスについて詳しく説明する。

5.1.4.1　アドオンプロセス

オブジェクトを基盤としたツールは使いやすいが、一般的に重要な課題がある。オブジェクトが適用先に完全に適合していない場合（これが通常）、不一致を無視するか（妥当性や利用可能性に問題が生じる）、オブジェクトを修正するか、独自に構築する必要がある（ソフトウェアによっては非常に困難）。この課題に対応するため、Simio は**アドオンプロセス**という概念を導入した。アドオンプロセスは独自の動作を追加して標準オブジェクトのロジックを補完でき、そのための**トリガー**が Standard Library の全オブジェクトに組み込まれている。これにより、モデル構築者に高い柔軟性が提供される。

アドオンプロセスは通常のプロセスと似ているが、別のオブジェクトによって発火される。たとえば、Server にはエンティティが入る際に発火されるアドオンプロセスがある。オブジェクト（この場合 Server）は、実行したいプロセスの名前を Add-on Process Triggers に設定する。これらのプロセスはシンプルながら大きな柔軟性を提供する。たとえば、Simio の Server は複数の故障タイプに対応しているが、修理開始前の外部リソース（電気技術者など）の利用可能性は考慮していない。この制約を無視するか（望ましくない）、独自のオブジェクトを作成するか（煩雑）、または 2 つの単純なプロセスを追加して実装するかを選択できる。Server には、修理時間開始前に電気技術者を確保する Seize ステップの Repairing トリガーと、修理完了後に解放する Release ステップの Repaired トリガーがある。

これらのプロセスの作成は実際には非常に簡単である。複数のオブジェクトで共有・参照される場合は、Processes タブでプロセスを作成してロジックを設定できる。一度だけ使用する場合は、より簡単な方法がある。以下で具体例を示す。

新しいモデルを作成し、Source、Server（Server1）、Sink を設置する。これらをパスで接続し、Worker（Worker1）を追加する。Server1 の Failure タイプを Calendar Time Based に変更し、Uptime Between Failures を 10 Minutes、Time to Repair を 1 Minute に設定する。他のプロパティはデフォルトのままとする。これで、Standard Library オブジェクトを使用した基本的な故障モデルが完成する。次に、修理時に Worker を利用するよう Server の動作をカスタマイズする。

1. Server1 プロパティの Add-on Process Triggers で、Failed をダブルクリックする。Simio は自動的にアドオンプロセスを作成し、適切に命名（Server1_Failed）して分類（Server1 Add-on Processes カテゴリ）し、Process ウィンドウを開く。
2. Seize ステップをプロセスへドラッグする。

3. Seize ステップのプロパティを設定する：Resource Seizes の 0 Rows をクリックして横の「…」ボタンを押し、Add ボタンをクリックしてリソースダイアログを開く。占有するオブジェクト名として Worker1 を選択し、ダイアログを閉じる。
4. Facility ウィンドウの Server1 プロパティに戻り、Add-on Process Triggers 下の Repaired をダブルクリックする。Process ウィンドウが自動的に開く。
5. Release ステップを Server1_Repaired プロセスへドラッグする。
6. Release ステップのプロパティを設定する：Resource Releases の 0 Rows をクリックして横の「…」ボタンを押し、Add ボタンをクリックしてリソースダイアログを開く。解放するオブジェクト名として Worker1 を選択し、ダイアログを閉じる。

完成したモデルは図 5.7 のようになる。実行結果を見ると、サーバの故障時間が 10%となり、それに応じて Worker の稼働時間も実行時間の 10%となっている。修理作業のない時間に Worker に別の作業をさせることで、このモデルをさらに実用的にできる。タスク間での Worker の移動方法や、作業時の現在位置の考慮については後述する。ここでは、アドオンプロセスが高い柔軟性を提供し、その設定が比較的容易であることを理解すればじゅうぶんである。

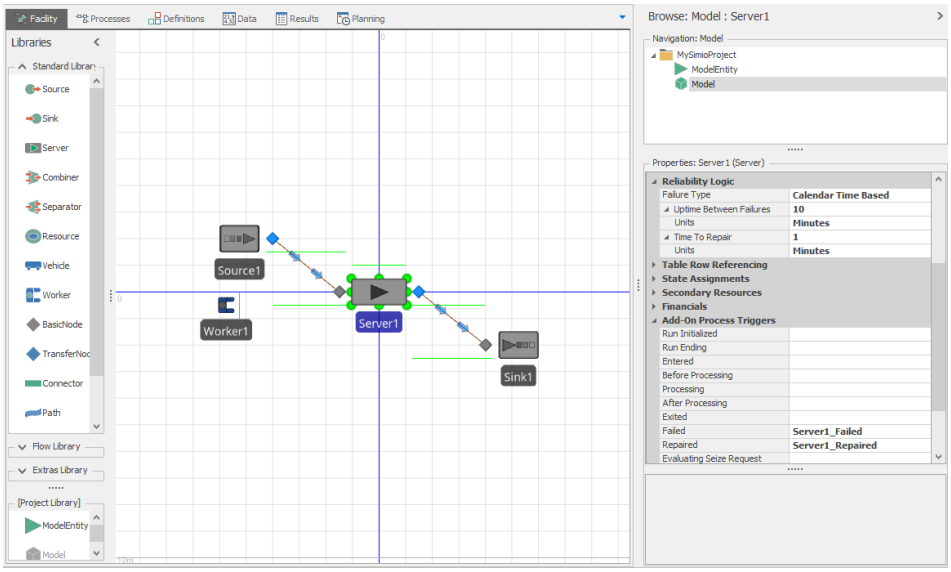


図 5.7　アドオンプロセスを用いた修理リソースの追加方法

5.1.5　リソースとしてのオブジェクト

前項で、システムにおける制約を表現する**リソース**の概念について議論した。これまでのモデル（4.3 節）では、Standard Library から主に Resource オブジェクトを使用した。Resource オブジェクトは便宜上ライブラリに提供されているが、このオブジェクトだけに限定されるわけではない。実際には、**リソースとしてどの Simio オブジェクトも使用できる**（注：オブジェクト開発者は、オブジェ

クト定義においてプロパティ Resource Object を False に設定するか、リソース関連のプロパティの一部またはすべてを非表示にすることで、その機能を無効化できる)。

リソースとして利用可能なオブジェクトは、以下の能力を持つ：

・能力（容量）制約がある。
・能力が利用可能になるまでトークンが待機する待ち行列がある。
・計画された、または割り当てられた能力に関する統計量を自動収集できる。
・占有しようとするオブジェクトと相互作用できる。

それぞれの能力について詳しく見ていこう。

リソースには能力制約がある。能力には 0（利用不可能)、Infinity（無制限)、またはその間の任意の整数を指定できる。また、能力はスケジュールに従わせることもできる（5.3.3 節と 7.2 節参照)。カレンダに基づくスケジュールは、Data タブの Schedules ウィンドウで定義できる。ここでは、日や週などの定期的なスケジュールを構成したり、休日や計画残業などのスケジュール例外（Exceptions）も設定できる。

柔軟性を最大限に発揮するため、リソースの能力は Process ロジックや Assign ステップで制御される。このアプローチにより、状況に応じて、また任意の遷移ルール（例：稼働中のリソースがオフシフトに移る際の挙動）を考慮して、能力を動的に調整できる。

リソースには、能力が利用可能になるまでトークンが待機する待ち行列がある。エンティティは直接リソース能力を取得しない。代わりに、エンティティは、リソースが能力を割り当てるよう要求するプロセス内で、Seize ステップを実行するトークンを持つ。すべてのトークンは、特定のリソースに対する単一の割当待ち行列で待機する。要求されたリソースが利用できない場合、トークン（および対応するエンティティ）は待機しなければならない。複数リソースまたはリソースリストの 1 つを占有したいエンティティは、各リソースに対する割当待ち行列で待機するトークンを生成する。

割当待ち行列の**ランキングルール**には、静的および動的な順位づけがある。**静的ランキングルール**には、First In First Out、Last In First Out、Smallest Value First、Largest Value First がある。後者の 2 つのルールは通常、Entity.Priority などの状態変数による Ranking Expression（順位づけの式）を必要とする。待ち行列が適切な順位を決定する際に、この式が評価される。静的ランキングは非常に効率的だが、「最適な」割当戦略としてリアルタイムの意思決定が必要な場合は、動的ランキングがより優れた柔軟性を提供する。

動的ランキングルールは、割当の実施時にすべての待機エンティティを評価する。動的ランキングは順位決定ごとにすべてのエンティティを評価する必要があるため、待ち行列が非常に大きい場合は**実行速度が低下する**可能性がある。Smallest Value First や Largest Value First に加え、Critical Ratio、Least Setup Time、Campaign Sequence Up などのより精巧なルールが **Standard**

Dispatching Rule のオプションとして含まれている（**キャンペーン**は塗装や圧延装置を使用する産業でよく用いられる。これらの産業では、順に濃い色を塗装したり、徐々に鉄鋼を薄くしていくような正確な工程順序が極めて重要である）。上級ユーザは独自の動的な割当ルールを開発することもできる（11 章参照）。

リソースは、計画された、あるいは配置された能力に関する統計量を自動収集できる。これらは状態変数の統計量（Idle、Busy、Blocked、Failed など）で補完できる。すべてのリソースは自動的に、計画された能力ユニットおよび割り当てられた能力ユニットの平均、最小、最大値を追跡する。多くの標準オブジェクトは、状態変数に基づく詳細を追跡する List State 機能を活用している。たとえば、Resource オブジェクトは以下の状態変数を追跡する：

- **Idle**：リソースが任意のタスクに割り当てられていない。
- **Busy**：リソースが 1 つ以上のタスクに割り当てられた能力を保持している。
- **Failed**：リソースが故障し、タスクに割り当てられていない。
- **OffShift**：リソースがシフトを外れており、タスクに割り当てられていない。
- **FailedBusy**：リソースは故障しているが 1 つ以上のタスクに割り当てられており、故障中でも「Busy」状態とみなされる。
- **OffShiftBusy**：リソースはシフトを外れているが 1 つ以上のタスクに割り当てられており、オフシフトでも「Busy」状態とみなされる。

また、出力統計に加えて、これらのデータの円グラフをモデルやコンソールに表示できる。

リソースは、占有しようとするオブジェクトと相互作用できる。この機能は大きなモデリングの自由度を提供する。オブジェクトは自律的な挙動を持つ可能性があるため、この機能により「インテリジェントな」リソースの概念が実現される。Simio では、リソースの占有は 2 つのオブジェクト間の交渉であり、占有の前に両方のオブジェクトの「同意」が必要となる。リソースは自身が占有される状況とタイミングを選択できる。たとえば、シフト終了が近い場合は占有を防ぐことができる。また、より高い優先順位の要求が後に到着することがわかっている場合は遊休状態を維持することを選択したり、現在地の近くにある要求のみを考慮したりすることもできる。

上述のように、この機能によりリソースに意味のある知性を組み込むことができる。割り当ての知的制御は、標準プロセス `OnEvaluatingAllocation` にロジックを追加することで実現される。また、いくつかの Standard Library オブジェクトには、Evaluating Seize Request アドオンプロセストリガーがある。この使用方法については 11 章で詳しく学ぶ。

5.1.6　データ範囲

オブジェクトは定義時点では使用方法や配置先のオブジェクトが不明なため、あるオブジェクト定義は、それを包含するオブジェクトや同等のオブジェクトについて何も知ることができない。たとえば、モデル上で定義された `DayOfWeek` という状態があっても、モデル内のオブジェクトから直接参照されていない場合が

ある。これはプロパティの項目において間接的な参照を許可することで機能する。実際、これは Standard Library オブジェクトの大部分のプロパティで採用されている仕組みである。

オブジェクトがパブリックとして選択した情報のみが、そのオブジェクト外で利用可能となる。たとえば、カレンダを追跡するように設計されたオブジェクトが DayOfYear という状態変数を持ち、それがパブリックと定義されている場合、モデルと他のオブジェクトはその値を参照できる。DayOfYear がパブリックでない場合は、カレンダオブジェクトのみがその値を参照できる。

アドオンプロセスは包含するオブジェクトの範囲に含まれる。これは、親オブジェクトや関連オブジェクトと同様、アドオンプロセスの Step が包含するモデル（メインモデルなど）の情報を直接参照できることを意味する。11 章で独自オブジェクトの構築を学ぶまでは、データ範囲の問題をあまり意識する必要はない。

5.1.7　式ビルダ

式フィールド（Processing Time など）において、Simio では複雑な式の入力を簡素化するために**式ビルダ**を提供している。4.5 節で基本的な使用方法を例示したが、ここでその他の詳細を説明する。式フィールドをクリックすると、右側に下向き矢印のある小さなボタンが表示される。このボタンをクリックすると、右に赤色の×と緑色のチェックマーク、左に漏斗記号がある式ビルダが開く。デフォルトでは漏斗が押下されており、フィルタが自動適用されたことを示す。ほとんどのモデルでは選択肢がじゅうぶんにあるが、フィルタをオフにするとさらに多くの選択肢が表示される。

式ビルダはマイクロソフト製品群の IntelliSense に似ており、5.1.2.3 で説明したドット表記での入力により、合致する名前やキーワードが表示される。式リストの多くの項目には、必須または任意のサブ項目がある。サブ項目を持つ項目は右端に「>>」が表示され、追加の選択肢があることを示す。**太字でない**項目は、まだ有効な選択でないことを示す。これらは単独では有効な式を表現できないため、サブ項目の選択が必要である。項目自体が**太字**の場合は、有効な選択であることを示す。これらの項目にはサブ項目がないか、任意のサブ項目のみがある。たとえば、図 5.8 の Color、DateTime、DirectDistanceTo、Elements、Enum は太字でないため有効な選択ではない。これらを使用するにはサブ項目を選択する必要がある。

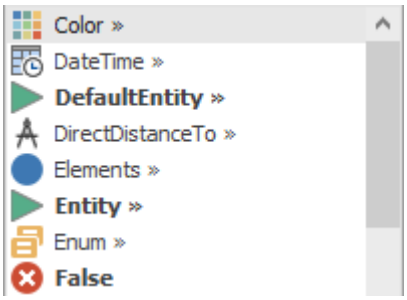


図 5.8　式ビルダの項目例

その他の３つの項目（すべて太字）は有効な選択である。最初の２つ（Default Entity と Entity）には矢印で示されるように任意のサブ項目があり、残りの太字項目（False）にはサブ項目がない。

2 * (3 + 4) ^ (4 / 3)のように、式を作るために、**数学演算子**（+、-、*、/、^）や計算順序を制御する括弧を用いることができる。<、<=、>、>=、!=、==、&&、||、!などの**論理演算子**を用いて論理式を作成することもできる。論理式は真のとき 1、偽のとき 0 の数値を返す。たとえば、10*(A>2)は、A が 2 より大きければ 10 を返し、そうでなければ 0 を返す。配列（最大 10 次元）は 1 から始まる添え字を持ち、角括弧で指定する（例：B[2,3,1]は 3 次元配列 B）。モデルのプロパティは、プロパティ名で参照できる。たとえば、Time1 と Time2 がモデルのプロパティの場合、(Time1+Time2)/2 のように、式に入力できる。

式ビルダでよく使用されるのが確率分布の入力である。これらは Random.DistributionName(Parameter1, Parameter2, …, ParameterN)という書式で指定される（パラメータの数と意味は分布により異なる）。たとえば、Random.Uniform(2,4)は 2 から 4 の間の連続一様からサンプルを返す。この式を入力するには、まず「Random」と入力すると、ドロップリストが Random までジャンプする。次にピリオドを入力するとすべての分布名が表示される。「U」を入力すると Uniform(min, max)にジャンプするため、Enter か Tab キーを押して式に追加し、パラメータ min と max に数値を入力する。分布名やその他の項目にマウスを合わせると、説明が自動表示される。式ビルダの引数リストには表示されないが、すべての分布は最後に乱数系列を表すパラメータを持っている。通常は既定値のままとするが、特殊な場合は系列番号を指定することもある。

Sin、Round、Min、Log などの**数学関数**は、キーワード Math で同様にアクセスできる。「Math」と入力し、ピリオドを入力すると数学関数の一覧が表示される。関数をハイライトすると説明が自動表示される。また、**日時関数**（キーワード DateTime）を用いて、任意の時刻（TimeNow など）を月、日、時などの要素に変換できる。

オブジェクト関数も関数名で参照できる。これは Simio が内部的に管理または計算する値であり、式では使用できるが代入できない値である。たとえば、すべてのオブジェクトがもつ ID 関数は、モデル内の各アクティブオブジェクトに割り当てられた一意の整数を返す関数である。動的エンティティの場合、大きな数の生成を避けるため ID 番号は再利用される。また、リソースとして使用されるすべてのオブジェクトには、現在の状態を数値で返す ResourceState 関数がある。たとえば、Server1 が Processing 状態の場合、Server1.ResourceState は 1 を返す。ResourceState 関数をハイライトすると、取り得る値が表示される。

オブジェクト関数と数学関数の詳細は、Simio ヘルプと Reference Guide（Support リボン）に記載されている。Support リボンの Expression Reference Guide は、現在のモデルのすべてのコンポーネントを説明する HTML ベースのヘルプである。

5.1.8　費用計算

モデル化の重要な目的の一つは、期待費用の予測や費用に基づくシナリオ比較である。単純な場合は、シナリオ間の資本投資を比較することで達成でき、状態変数による費用追跡やモデル外での計算が可能である。しかし、運用経費の測定が必要なケースも多く、直接費用と間接費用、製品ミックス、各サブシステムのスループット、総スループットなど、生産面で大きく異なることがある。このような複雑なケースでは、**活動基準原価計算（ABC；**Activity-Based Costing）の包括的アプローチが有効である。ABC は消費された資源に基づいて製品やサービスに費用を配賦する手法であり、より多くの費用を直接費として認識し、さまざまなシナリオにおける個々の製品やサービスの費用への貢献をより正確に理解できる。

シミュレーションは ABC を実行する理想的なツールである。活動が費用測定と同じ詳細さでモデル化されるためである。モデルは通常、各エンティティが使用するリソースとその使用時間を把握している。一部のシミュレーション製品では、ABC 実行のために費用追跡用のロジックとデータ項目を追加する必要があるが、より包括的な製品には ABC が組み込まれている。Simio は後者に該当する。

Simio Standard Library のほとんどのオブジェクトには、**Financials カテゴリ**がある。そのプロパティはオブジェクトの複雑さと機能により異なり、単純な時間当たり稼働費用（Usage Cost Rate）から、より複雑な費用計算まで対応できる。たとえば、多くの作業者は次の作業を待つ間も給与が発生し（Idle Cost Rate）、一部のリソースにはエンティティ処理ごとの固定費（Cost Per Use）がある。また、航空機のような高価なリソースやエンティティは、未使用時でも高額な保管費用（Holding Cost Rate）が発生する。多くの場合、費用の正確な測定は容易ではないが、適切な部門への配賦や配分（Parent Cost Center）が必要である。

図 5.9 では、Server と Worker に類似のリソース費用計算プロパティがあることがわかる（占有・保持費用、待機中の費用など）。Server には入出力バッファでの待機費用計算プロパティがあり、Worker にはエンティティ運搬時の費用や移動中の運搬費用がある。通常、一つのオブジェクトインスタンスがこれらすべての費用を持つことはまれであり、使用ごとの費用が発生するリソースや、稼働・遊休時の原価率を持つリソースなど、さまざまである。

図 5.9 の左側は自動化された収縮包装機（Server）を示している。それは、33,500 ユーロの資本費用と 1 回当たり 13.852 ユーロの使用固定費が必要である。この費用は ShippingCenter コストセンターに配賦される。右側は、稼働状態に関わらず時給 20US ドルのフォークリフト作業者（Worker）を示している。作業者が輸送にフォークリフトを使用する際は、追加で時間当たり 31.50US ドルの輸送機械費用が発生する。これらの費用は Transportation コストセンターに配賦される。

最後に、多国籍企業やプロジェクトでは、世界中で業務が行われ、複数の通貨が関係することが一般的である。Simio は主要な世界通貨に対応している（ゴートやビットコインは除く）。Advanced Options の Run タブの Financials カテゴリで、デフォルトの通貨を指定でき、各通貨との為替レートも設定できる。

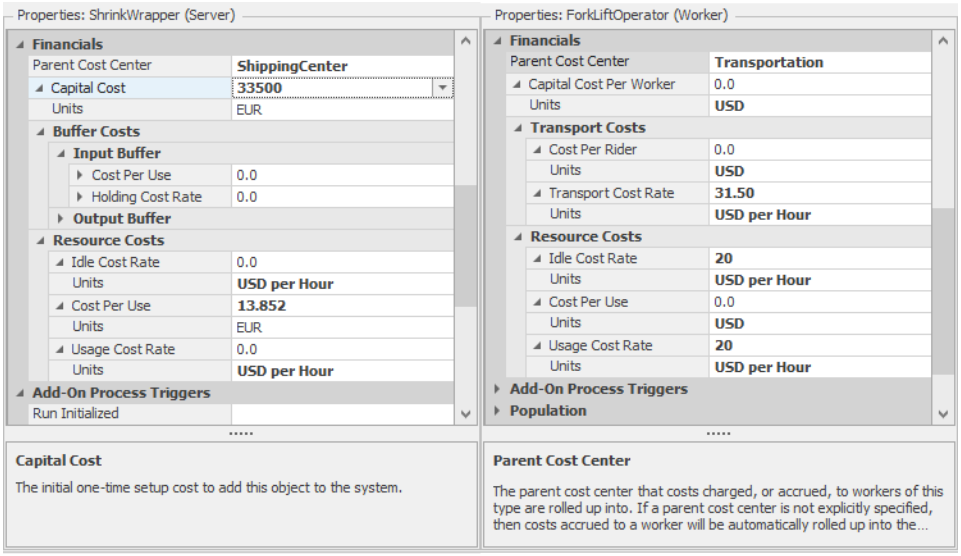


図 5.9　Server（左）と Worker（右）の Financials プロパティ

最後に、多国籍企業やプロジェクトでは、世界中で業務が行われ、複数の通貨が関係することが一般的である。Simio は主要な世界通貨に対応している（ゴートやビットコインは除く）。Advanced Options の Run タブの Financials カテゴリで、デフォルトの通貨を指定でき、各通貨との為替レートも設定できる。

5.2　Model 5-1：PCB 組立

ここまで Simio のフレームワークについて理解を深めてきたので、モデリングと分析に焦点を移そう。本章のモデルは、プリント基板（PCB）の簡易組立作業を扱う。PCB は、実質的にすべての電子機器において不可欠な構成部品である。PCB 組立では、電子部品間の信号伝達を最適化するよう設計された特殊基板に、さまざまな電子部品を実装する。図 5.10 にプリント基板の例を示す（画像は Auburn 大学の Center for Advanced Vehicle Electronics（CAVE）の厚意により提供：cave.auburn.edu）。PCB 組立工程は通常、複数の連続した加工、検査、再加工、包装作業で構成される。本章では、これらの工程を段階的にモデルに組み込んでいくが、まずは 4 章で扱った単一窓口待ち行列モデルの拡張から始める。

Model 5-1 では、表面実装部品（たとえば、図 5.10 の基板上の黒い長方形の部品）を基板に配置し、検査する作業を扱う。基板は上流工程から毎時 10 枚の速度で部品配置に到着する（当面、到着過程はポアソン分布、つまり到着時間間隔は指数分布に従うと仮定する）。部品配置機械の処理時間は三角分布(3, 4, 5)分に従う。部品配置後、すべての構成部品が正しく配置されているかを検査員が確認する。検査時間は 2 分から 4 分の一様分布に従う。過去の記録によると、検査された基板の 92%が「良品」、8%が「不良品」となっている。

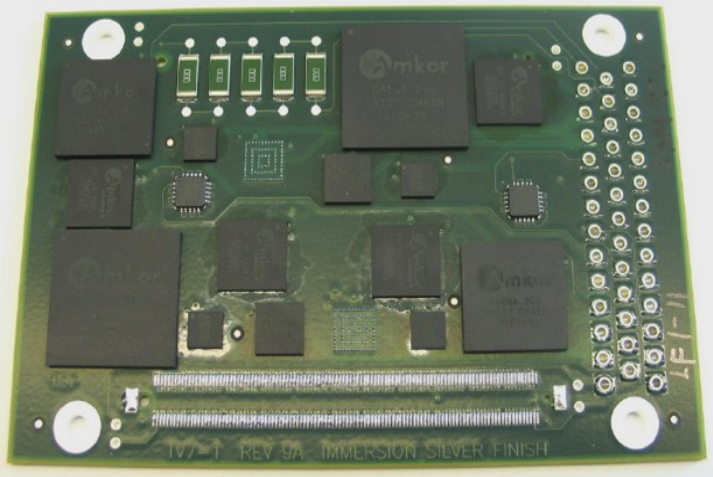


図 5.10　プリント基板（PCB）の例

到着時間間隔、部品配置、検査時間はすべてランダムであるため、複数の待ち行列が発生すると予想される。そこで、部品配置と検査工程における待ち行列長と稼働率の推定を行いたい。また、基板がシステム内で費やす時間と待ち時間、さらに良品・不良品の数も把握したい。

この時点でモデル構築に移れるが、4.2.5 項で述べたように、パフォーマンス尺度を**推定**する前にモデルの検証が必要である。そのため、まずシステムの**期待値**を予測しよう。基本的な検証プロセスでは、この期待値とモデルの結果を比較する。図 5.11 は、初期 PCB 組立システムの基本的な**待ち行列**モデルを示している。

基板の到着率（$\lambda = 10$）と各サービス率（$\mu_p = 15$および$\mu_i = 20$）が与えられている場合（単位はすべて基板枚数/時間）、定常状態の窓口稼働率を正確に計算できる。この稼働率は特定の到着間隔や処理時間の分布に依存しない$\rho_p = \lambda/\mu_p = 0.667 < 1$であるため、検査員への到着率も毎時 10 基板となり（なぜか？）、したがって$\rho_i = \lambda/\mu_i = 0.500$であることがわかる。両サーバの稼働率が 1 未満なので、システムは**安定**している（到着よりも処理が速いため、システム内の基板数は有限となり、無限に増加することはない）。部品配置と検査時間が指数分布に従う場合、システム内の基板の期待値$L = 3.000$と、基板の平均システム滞在時間$W = 0.3000$を計算できる（詳細は 2 章参照）。これらはいずれも定常状態の値である。一方、指数分布に従わない場合は、4 章で述べたように、検証用のシミュレーションモデルを最初から構築すればよい。

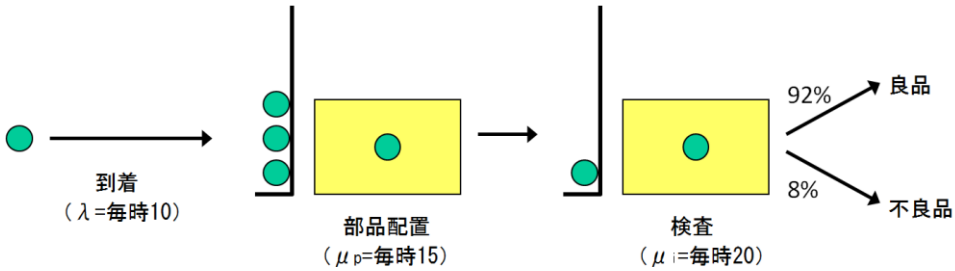


図 5.11　Model 5-1 の待ち行列モデル

期待値が判明したので、Simio モデルを構築できる。Model 5-1 は 4 章のモデルから 2 つの基本的な機能を拡張している：(部品配置から検査へと続く) **連続したプロセス**と（検査後に基板を「良」または「不良」に分類する）**分岐**である。これら 2 つの拡張機能は Simio で広く活用され、実装も容易である。図 5.12 は完成した Simio モデルを示しており、Entity オブジェクト（PCB)、Source オブジェクト（Source1)、2 つの Server オブジェクト（Placement と Inspection)、そして 2 つの Sink オブジェクト (GoodParts と BadParts) で構成されている。

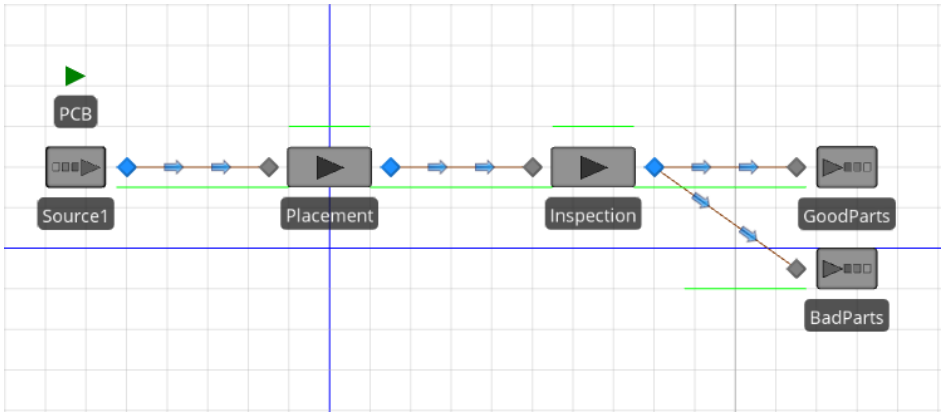


図 5.12　Model 5-1

初期モデルでは、オブジェクトを接続してエンティティの流れを指定するために Connector（コネクタ）を使用した。Connector は、あるオブジェクトの出力ノードから別のオブジェクトの入力ノードまで、シミュレーション時間ゼロでエンティティを移動させることを思い出してほしい。連続したプロセスは、2 つの Server オブジェクトを接続するだけでモデル化できる（Placement オブジェクトから出た基板を Inspection オブジェクトへ移動させる)。分岐は、同じ出力ノードから 2 つのコネクタを伸ばすことでモデル化される（検査を終えた基板を GoodParts または BadParts のいずれかの Sink へ移動させる)。

特定のエンティティがどの分岐を選択するかは、出力ノードの Routing Logic で設定する（図 5.13 は Model 5-1 の設定を示している)。

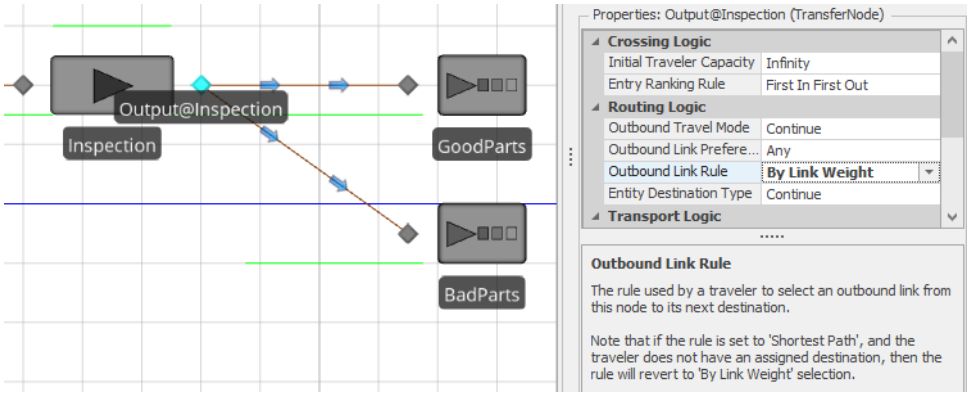


図 5.13　出力ノードの Routing Logic

このケースでは、Inspection オブジェクトの出力ノードを選択し、Routing Logic グループの Outbound Link Rule プロパティを By Link Weight に設定している。これにより、コネクタの Selection Weight プロパティを使用して各リンクの選択確率を決定し、ランダムに出力リンクを 1 つ選択するよう Simio に指示している（エンティティに目的地が設定されている場合は Shortest Path がデフォルトルールとなるが、目的地が指定されていない場合は By Link Weight が使用される）。

このモデルの仕様では、検査された基板の 92%が良品であるため、GoodParts オブジェクトのリンクの Selection Weight プロパティを 0.92 に、BadParts オブジェクトのリンクを 0.08 に設定する（図 5.14）。このリンク重みづけのメカニズムは、各出力リンクに相対的な重みを割り当てることで、n種類のランダムな分岐を容易にモデル化できる。リンクの重みには、92 と 8、9.2 と 0.8、あるいはその他の比率（例：ケーキのレシピで小麦粉 5、砂糖 2、ふくらし粉 1 など）のいずれも使用できる点に注意が必要である。Simio はこれらの重みを、エンティティを各経路に振り分ける割合の決定にのみ使用する。Simio には基本的な Routing Logic を指定する他の手法も存在するが、これらは以降のモデルで紹介することとし、ここではランダム選択のみを扱う。

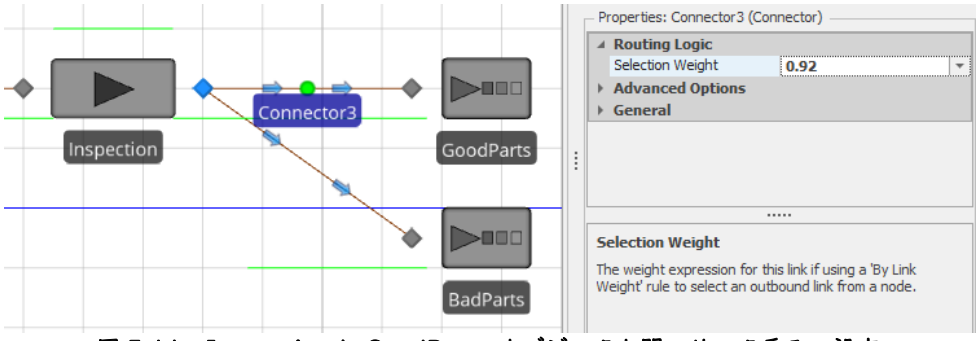


図 5.14　Inspection と GoodParts オブジェクト間のリンク重みの設定

Model 5-1 を完成させる最後のステップは、4 章のモデルと同様に、Interarrival Time と Processing Time プロパティを使用して到着時間間隔とサービス時間の分布を設定することである。まず、モデル検証のステップとして、待ち行列理論から得られた結果と比較するため、処理時間を指数分布に従うように設定する。表 5.2 は、シミュレーション期間 1,200 時間、ウォームアップ期間 200 時間での 25 回の反復実行結果を示している。これらの結果は 4.2 節で説明した Pivot Grid レポートから取得したものである。すべての結果が期待値の信頼区間内に収まっており、これはモデルが期待通りに動作していることの確かな証拠となる（つまり、検証されたといえる）。これにより、処理時間の分布を指定された値に設定してモデルを完成させることができる。

表 5.3 に、検証実験と同じ実行条件に基づく結果を示す。予想通り、稼働率および良品・不良品の割合はほとんど変化しなかったが、エンティティ数（L）および滞在時間（W）のシミュレーションによる推定値が共に小さくなったことに注

目すべきである。実際の**数値**は変化したものの、**推定値**は確率的な観点からは変化していない。ここで、平均値と信頼区間の半幅は確率変数の観測値であることを思い出してほしい。また、指数分布から三角分布あるいは一様分布への変更により、両プロセスの分散が減少することは予測可能である（これは三角分布と一様分布が、指数分布と異なり、右側に有界であるためである）。この分散の減少がLおよびWの値を小さくする要因となっている。

表 5.2　Model 5-1 の初期結果（指数分布に従う処理時間）

推定する指標	シミュレーション結果
Placement の利用率（ρ_p）	0.666 ± 0.004
Inspection の利用率（ρ_i）	0.498 ± 0.003
システム内数（L）	2.992 ± 0.060
滞在時間（W）	0.280 ± 0.005
良品の平均個数	9178.480 ± 44.502
不良品の平均個数	803.280 ± 11.401

表 5.3　Model 5-1 の初期結果（三角分布または一様分布に従う処理時間）

推定する指標	シミュレーション結果
Placement の利用率（ρ_p）	0.668 ± 0.002
Inspection の利用率（ρ_i）	0.501 ± 0.002
システム内数（L）	1.849 ± 0.020
滞在時間（W）	0.185 ± 0.002
良品の平均個数	9205.400 ± 34.588
不良品の平均個数	806.320 ± 12.357

5.3　Model 5-2：PCB 組立の拡張

本節では、実システムで一般的に見られる特徴を追加し、PCB 組立モデルを拡張する。追加する機能は以下のとおりである：

- **再加工**（Rework）：検査済み基板は良品または不良品に分類される。検査済み基板の 26%は専用の再加工ステーションでの処理が必要と仮定する。このステーションでは、作業員が基板から配置済みの部品を取り外す。これらの再加工対象基板は、前節で良品としていた基板から選別される。そのため、本モデルでは検査後の分類が良品 66%、不良品 8%、再加工 26%となる。再加工の処理時間は三角分布(2, 4, 6)に従う。再加工後、基板は部品配置工程に戻される。再加工された基板は新規基板と同様に扱われ、検査時にも同じ確率で良品・不良品・再加工の判定がなされる（つまり、再加工の回数制限はなく、基板は部品配置と検査を何度も繰り返す可能性がある）。
- **作業者のスケジュール**（Worker Schedules）：検査と再加工は人間の作業員が担当するため、就業時間には「食事休憩」を考慮する必要がある。検査は操業時間中常時実施されるが、再加工は「2 番目のシフト」でのみ行われる。部品配置

は完全自動化されており、作業員を必要としない。

・**機械故障**（Machine Failures）：自動化された部品配置工程では、修理を要する故障がランダムに発生する。

このモデルには作業者スケジュールの概念を導入したため、**操業時間**を明確に定義する必要がある。Model 5-2 では 24 時間操業とし、125 日間連続して実行する。

5.3.1　再加工ステーションの追加

図 5.15 に完成したモデルを示す。再加工ステーションは新しい Server オブジェクト Rework として追加する。Processing Time と Name プロパティを設定し、サーバの入力ノードと Inspection オブジェクトの出力ノードを接続して、検査後の 3 種類の分岐を作成する（リンクの重みも更新が必要）。再加工された基板は再処理のために部品配置に戻されるため、Rework オブジェクトの出力ノードを Placement オブジェクトの入力ノードに接続する。これは**合流**プロセスの例である。Placement オブジェクトには 2 つの異なるエンティティの「流れ」が入力される：Source1 オブジェクトからのもの（新規基板）と、Rework オブジェクトからのもの（検査不合格で再加工された基板）である。Placement オブジェクト（およびその入力ノード）では、どちらの経路から到着したかに関わらず、エンティティを同様に扱う。到着エンティティと状態変数に応じて異なる処理を行う方法についても後で説明する。

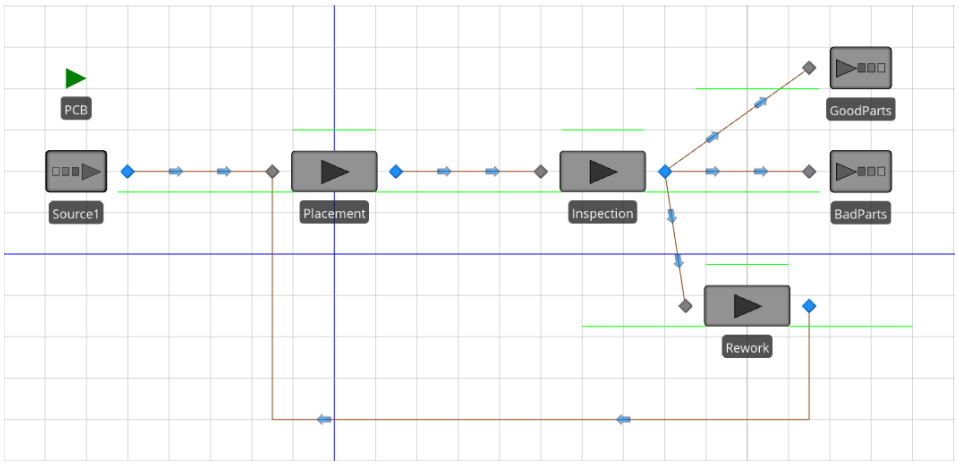


図 5.15　Model 5-2

システムとモデルの構造から明らかなように、部品配置と検査工程では新規基板と再加工基板を区別しないため、基板は複数回再加工される可能性がある。各基板が部品配置で処理された回数を追跡し、その統計量を収集したい（なお、再加工回数の追跡という、より単純な方法も選択可能だが、この場合の最小値は 1 ではなく 0 となる）。これは**ユーザ定義統計量**の典型例だが、Simio では自動的には提供されていない。この機能を実装するには、以下の手順が必要である：

1.　エンティティの**状態変数**を定義し、部品配置での処理回数を追跡する。

2. 部品配置での処理完了時に、状態変数値を増加させるロジックを実装する。
3. 実行結果の要約統計量を報告する**タリー統計量**（Tally Statistic）を作成する。
4. エンティティが（良品または不良品として）システムを離れる際に、タリー値を記録するロジックを実装する。

これは 4.3 節で実装した **TimeInSystem** 統計値と同様の仕組みである。これらは**観測統計量**（または離散時間過程）の例で、各エンティティが 1 つの観測値を生成する。このモデルでは部品配置工程の通過回数とシステム内時間を観測する。タリー統計量は、全観測値の平均値や最大値などの要約統計量を報告する（**標準偏差**は適切ではない。計算は容易そうだが、偏りがあり正確な結果が得られない。詳細は 3.3.1 項を参照）。

状態変数は、オブジェクト内で定義され、シミュレーション実行中に値が変化する変数である（5.1.2.2）。エンティティの部品配置工程通過回数の追跡には、状態変数が必要となる。エンティティが工程を通過（離脱）するたびに、状態変数の値を増加させる。エンティティに関連する状態変数を定義するには、Navigation ウィンドウで ModelEntity を選択し、Definitions タブを開き、パネルから States を選択して、リボンから Integer アイコンをクリックする（図 5.16）。

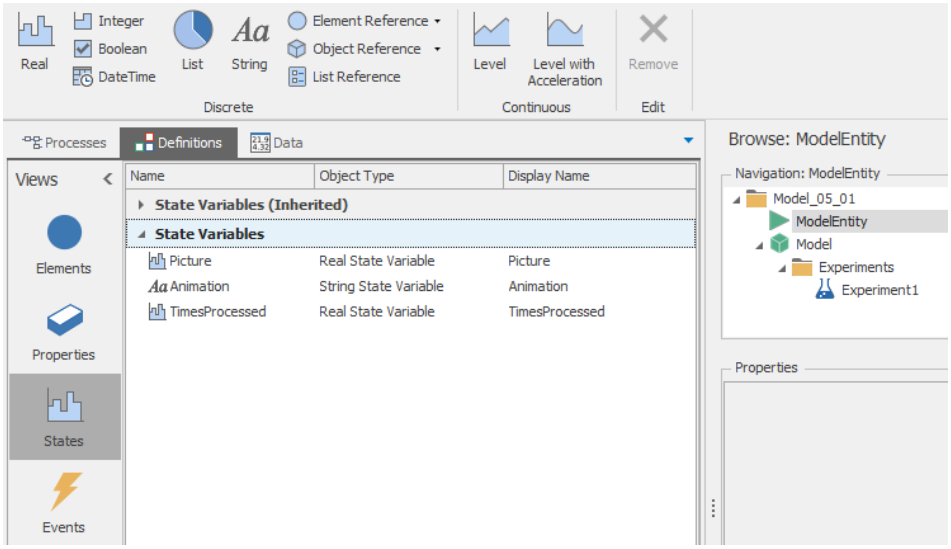


図 5.16　エンティティの状態変数 TimesProcessed の作成

ここでは、状態変数を TimesProcessed と命名した。ModelEntity タイプのすべてのエンティティオブジェクトがこの TimesProcessed 状態変数を持ち、シミュレーション実行中にその値が更新される。このモデルでは、部品配置工程が完了するたびに値を 1 増加させる必要がある。Simio でこのロジックを実装する方法はいくつかあるが、本モデルでは Server オブジェクトの State Assignments プロパティを使用する。このプロパティを使用することで、エンティティがサーバに入るときまたは出るときに、複数の状態変数に値を割り当てることが可能である（図 5.17）。

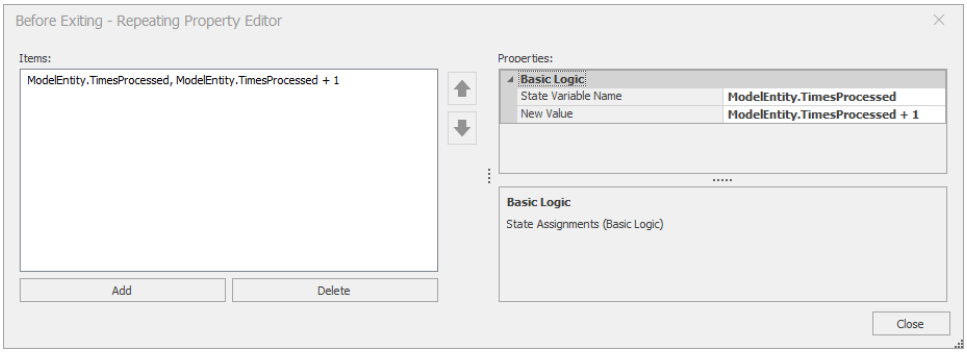


図 5.17　Placement オブジェクトの State Assignments プロパティの設定

Placement オブジェクトを選択し、State Assignments セクションの Before Exiting リピートグループで、状態変数 TimesProcessed の値を増加させる割り当てを追加する。これにより、エンティティがオブジェクトを離れる直前に、指定した状態変数（この場合は単一の変数）に値が割り当てられる。その結果、どの時点でも、エンティティの状態変数 TimesProcessed は、そのエンティティが Placement オブジェクトで処理を完了した回数を正確に示すことになる。

続いて、エンティティがシステムを離れる直前に、状態変数 TimesProcessed の値を観測（タリー）統計量として記録するよう設定する。この設定は以下の 2 段階で行う：

1. タリー統計量を定義する。
2. モデル内の適切な場所で、エンティティ状態変数 TimesProcessed の値を記録する。

タリー統計量の定義手順は次のとおりである。まず、Navigation ウィンドウで Model を選択して Definitions タブを開く。次に、Elements ペインを選択し、リボンから Tally Statistic アイコンをクリックしてタリー統計量を定義する（図 5.18）。Name プロパティに NumTimesProcessed と入力して命名する。最後に、タリー統計値を記録する 2 つの方法を説明する。**アドオンプロセス**を使用する方法と、Sink オブジェクトの入力ノードの Tally Statistics プロパティを使用する方法である（通常はどちらか一方を使用するが、ここでは両方の方法を紹介する）。

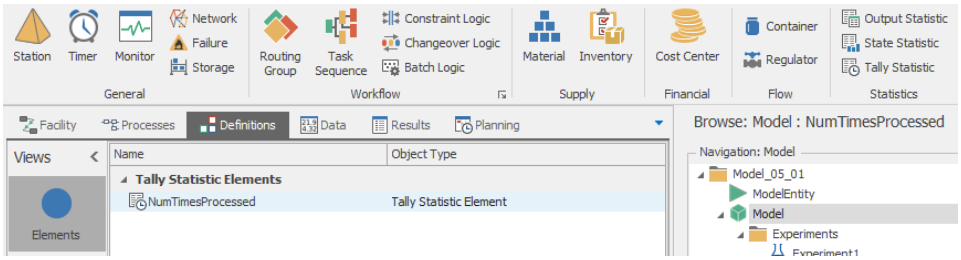


図 5.18　タリー統計量 NumTimesProcessed の定義

最初に、Sink オブジェクト GoodParts に向かうエンティティでアドオンプロセスの方法を使用する。このプロセスは、エンティティが Sink オブジェクトに入

る時点で実行される。アドオンプロセスを作成するには、Sink オブジェクト GoodParts を選択し、モデルのプロパティパネルで Add-On Process Triggers グループを展開する。ここには、Run Initialized、Run Ending、Entered、Destroying Entity の 4 つのトリガーが定義されている。各エンティティの値を記録する必要があるため、Entered トリガーを選択する。アドオンプロセスには任意の名前をつけられるが、より簡単な定義方法がある。トリガーのプロパティ名（この場合は Entered）をダブルクリックするだけで、Simio が適切な名前を自動的に定義し、新しく作成されたプロセスを選択した状態で Processes パネルが開く。このアドオンプロセスは、タリー統計値を記録する単一ステップの簡単なものであり、Tally ステップを使用して実装する（図 5.19）。

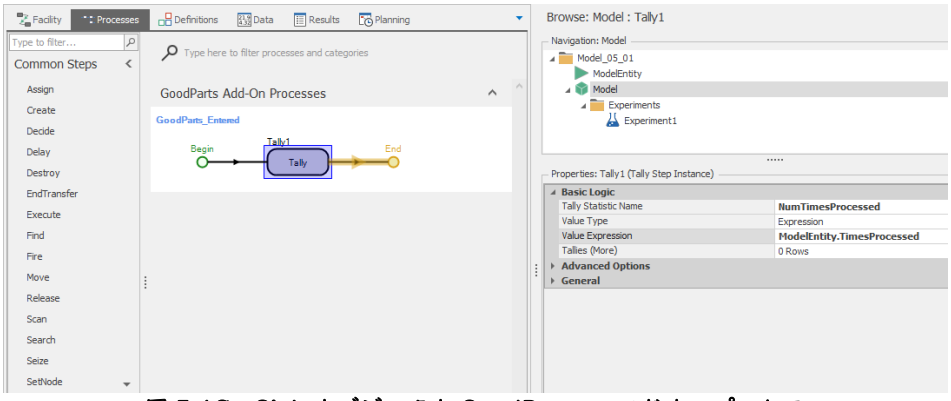


図 5.19　Sink オブジェクト GoodParts のアドオンプロセス

ステップを追加するには、Common Steps パネルから Tally ステップを選択し、プロセスにドラッグして Begin ノードと End ノードの間に配置する。Tally ステップの主要なプロパティは TallyStatistic Name で、ここで先に定義した Tally Statistic（NumTimesProcessed）と Value を指定し、記録する値（この場合、エンティティ状態変数 TimesProcessed）を設定する。また、Value Type プロパティを Expression に設定することで、値の評価や記録に式を使用することを指定する。

もう一つの方法として、Standard Object ライブラリの Basic Node にある Tally Statistics プロパティを使用する方法がある。これを Sink オブジェクト BadParts に適用する。図 5.20 にこの設定を示す。

最初に Sink オブジェクト BadParts の入力ノードを選択し、Tally Statistics プロパティグループを展開する（図 5.20 の上部）。次に、On Entering プロパティの横にある「…」をクリックしてリピートグループエディタを開き、図 5.20 の下部に示すようにタリー統計量を追加する。両方の部品タイプで同じタリー統計量を使用して、すべての部品の処理回数を追跡する。なお、部品タイプ（Good と Bad）ごとに統計量を個別に追跡したい場合は、2 つの異なるタリー統計量を設定する。モデル実行後は、標準の Simio Pivot Grid と Reports でタリー統計量を確認でき、プロセスロジックでもその値を使用できる。

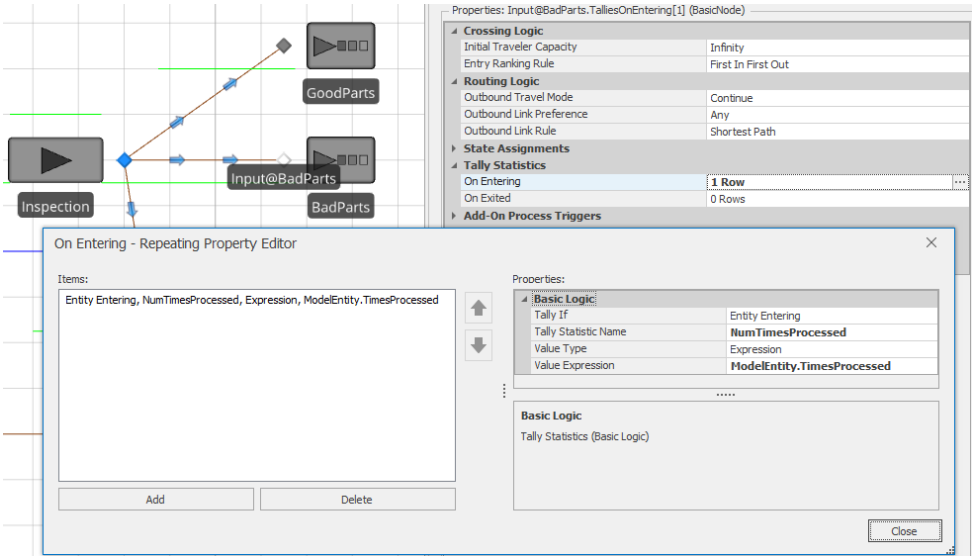


図 5.20　Sink オブジェクト BadParts における Tally Statistics プロパティ

これまでの説明は、ユーザ定義の観測統計量を作成する一般的なプロセスの具体例である。この作成プロセスは以下の手順にまとめられる：

1. 観測する**式**を特定する。多くの場合、観測される式を計算・格納するための新しいモデルやエンティティ状態変数の定義が必要となる。
2. 各観測に対する式の値を設定するロジックを追加する（モデルにまだ組み込まれていない場合）。
3. タリー統計量を作成する。
4. モデル内の適切な場所で、アドオンプロセスに Tally ステップを使用して式の値を記録するか、ノードオブジェクトインスタンスの Tally Statistics プロパティを設定する。

ユーザ定義の観測統計量を組み込む際は、常にこの一連のプロセスに従う。

5.3.2　リンクの重みに対する式の利用

Model 5-2 の続きに入る前に、コネクタのリンクの重みに**式**を利用する方法を説明する。これは、リンク選択のロジックを簡素化できる強力な機能であるが、見落とされがちである。先の例では、検査結果の 3 つの可能性（良品：66%、再加工：26%、不良品：8%）に定数の確率を指定し、それに従ってリンクの重みを設定した。ここで、基板の処理回数に制限を設けることを考える。具体的には、次のルールを適用する：

・基板は最大 2 回まで処理される。3 回処理された基板は不合格となる。
・検査された基板の 95%は良品である。
・検査された基板の 5%は不良品である。

このロジックをモデルに実装する。ここでのリンク選択は純粋な確率的選択ではない。基板が不合格（3 回処理済み）かどうかを判定する条件があり、その条件

を満たさない場合にのみ、残りの 2 つの選択肢から確率的に選択する。図 5.21 は、このロジックを実装するためにリンクの重みに式を使用する簡単な方法を示している。基板の加工回数が 3 回未満の場合のみ確率が適用され、3 回目の加工後は基板が不合格となることに注意する。これは括弧内の要素でリンクの重みが 0 または 1 と評価されることで実現される。

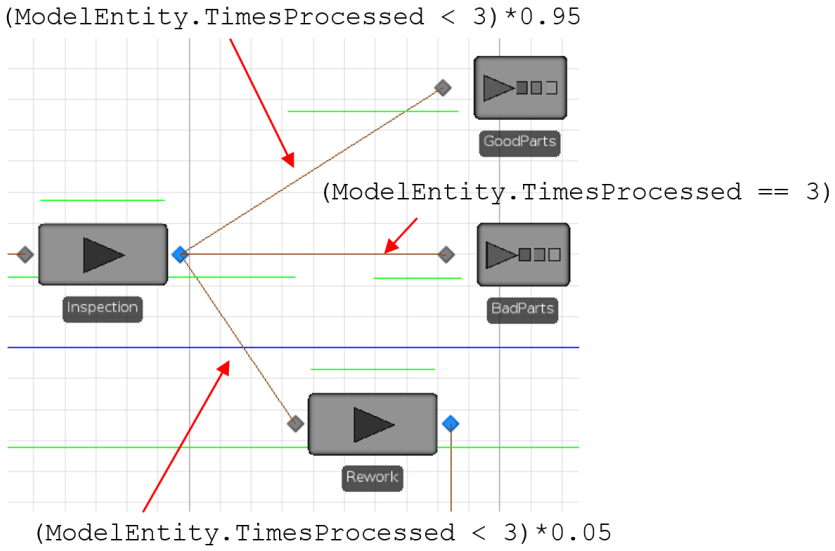


図 5.21　式を用いたリンクの重みづけの例

この例では、リンクの重みづけルールに式を使用することの利点を示した。初心者は、このようなロジックを実装する際に、不要な複雑なアドオンプロセスを使用しがちである。ここで示した式の使用方法は、モデル開発を大幅に簡素化するため、この例をよく理解することを推奨する。ただし、実際のシステム運用では、基板が 3 回処理される**前に**不合格判定を行うことが望ましい。つまり、2 回目の不合格判定直後、再加工される前に判断すべきである。この課題については、読者の演習問題として用意している（章末の問題 7 を参照）。

5.3.3　リソーススケジュール

PCB 組立システムの再加工と検査工程は、どちらも人間の作業員が担当している（部品配置工程は自動化されている）。作業員は通常**シフト**制で働き、勤務時間中に休憩を取る。作業者スケジュールの Simio での実装方法を説明する前に、Simio がどのように 1 日の時間を追跡するかを理解する必要がある。Simio は標準で、各「日」を（深夜）12 時から始まる 24 時間の時計で管理している。これまでのモデルでは実行期間とウォームアップ期間に注目してきたが、1 日 24 時間の区分については触れていなかった。

PCB の例では、施設は 1 日 3 シフト制で運営され、各シフトは 8 時間勤務に 1 時間の食事休憩が含まれる（つまり、実質的な 1 日の勤務時間は休憩を含めて 9 時間となる）。シフトは以下のとおりである。

・1 番目のシフト：
　午前 8 時～午後 12 時、午後 12 時～午後 1 時（休憩）、午後 1 時～午後 5 時
・2 番目のシフト:
　午後 4 時～午後 8 時、午後 8 時～午後 9 時（休憩）、午後 9 時～午前 1 時
・3 番目のシフト：
　午前 12 時～午前 4 時、午前 4 時～午前 5 時（休憩）、午前 5 時～午前 9 時

また、複数シフトがある工程（この例では検査工程）では、各シフトの最初の 1 時間は事務作業とシフト準備に充てられる。そのため、シフトが重複する時間帯でも、実際に検査作業を行う作業員は 1 名となる。

この作業スケジュールでは、検査工程は 24 時間体制で運営され、3 回の 1 時間食事休憩（正午～午後 1 時、午後 8 時～午後 9 時、午前 4 時～午前 5 時）が設定されている。これは、1 日 8 時間勤務の検査作業員 3 名体制と同義である。Simio は午前 12 時起点の 24 時間時計を採用しているため、このスケジュールは表 5.4 のような検査工程の**リソース能力スケジュール**として表現される。

表 5.4　検査工程のリソース能力スケジュール

時間帯			リソース能力（容量）
午前 12:00	－	午前 4:00	1
午前 4:00	－	午前 5:00	0
午前 5:00	－	午後 12:00	1
午後 12:00	－	午後 1:00	0
午後 1:00	－	午後 8:00	1
午後 8:00	－	午後 9:00	0
午後 9:00	－	午前 12:00	1

この作業スケジュールでは、3 回の「食事休憩期間」中は検査員が不在となるため、検査ステーションの前に基板の待ち行列が発生することが予想される。モデルが完成すれば、アニメーションでこの待ち行列の挙動を確認できる。このようなリソース能力スケジュールを Simio に実装する方法は複数あるが、Model 5-2 では内蔵の Schedule テーブルを使用する。このテーブルを使用するには、まずスケジュールを定義し、その後 Inspection オブジェクトのリソース能力を決定するためにそのスケジュールを適用する。

作業スケジュールを作成するには、Navigation ペインで Model を選択し、Data タブを選択して、Data パネルの Schedules アイコンをクリックする。表示される 2 つのタブのうち、Work Schedules タブには StandardWeek という名前のサンプルスケジュールが、Day Patterns タブには StandardDay というサンプルがすでに作成されている。

Day Patterns タブから作業を開始し、1 日の能力値と作業サイクルを定義する。まず、StandardDay を選択して Remove をクリックし、デフォルトのパターンを削除する。次に、リボンの Create カテゴリから Day Pattern をクリックし

て新規パターンを作成する。新しいパターンに DayPattern1 という名前をつけ、名前の左側にある「+」をクリックして Work Period（作業期間）の定義を展開する。最初の作業期間では、Start Time を 12:00:00 AM、Duration（持続期間）を 4 に設定する。End Time は自動計算され、Value（リソース能力）は 1 がデフォルト値となる。同様の手順で、表 5.4 に示された残りの 3 つの作業期間を追加する。完成した 1 日のパターンは、図 5.22 のとおりとなる。

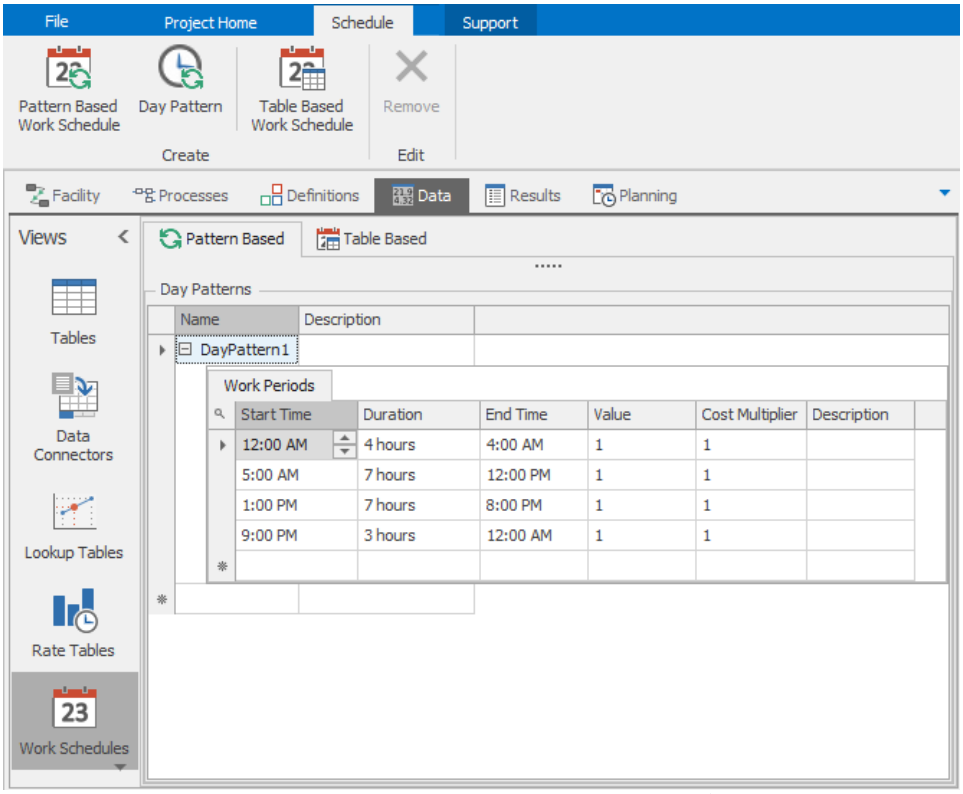


図 5.22　検査工程の 1 日のパターンの定義

次に、Work Schedules タブに移動する。サンプルスケジュールを削除して新規作成することも可能だが、ここでは既存のものを修正して使用する。Work Schedule Name を InspectionSchedule に、Description を Daily schedule for inspectors に変更する。このモデルでは作業パターンが毎日同一のため、パターンの繰り返し日数を示す Days プロパティ値を 1 に設定する。StandardDay パターンは既に削除済みのため、Day 1 列にエラーが表示されている。これを解消するため、プルダウンリストから先ほど作成した DayPattern1 を選択する。これで作業スケジュールの設定が完了する（図 5.23）。

作成したスケジュールを使用するため、Inspection オブジェクトの設定を変更する。具体的には、Capacity Type プロパティを WorkSchedule に設定し、Work Schedule プロパティに InspectionSchedule を指定する（図 5.24）。

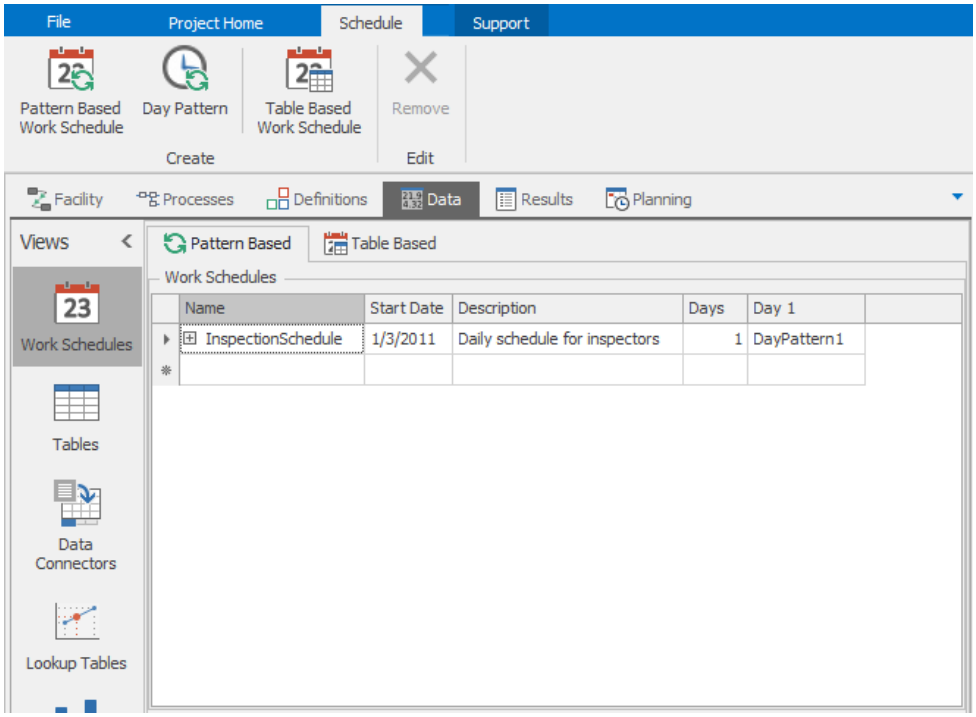


図 5.23　検査工程のスケジュール

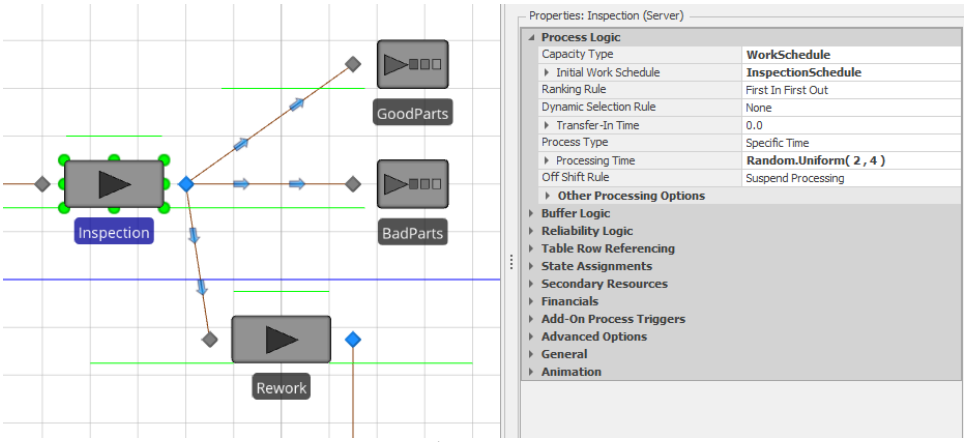


図 5.24　Inspection オブジェクトに WorkSchedule を適用

次に、再加工の作業員についても同様に能力スケジュールを指定する必要がある。前述のとおり、再加工工程は第 2 シフトのみ操業する。具体的には、午後 4 時から午後 8 時まで、午後 8 時から午後 9 時まで（食事休憩）、そして午後 9 時から午前 1 時までである。先ほどと同様に、新しい日のパターン DayPattern2 を作成し、新しい作業スケジュール ReworkSchedule でこのパターンを参照した後、Rework オブジェクトにこのスケジュールを適用する（図 5.25）。スケジュールの詳細については 7.2 節で説明する。

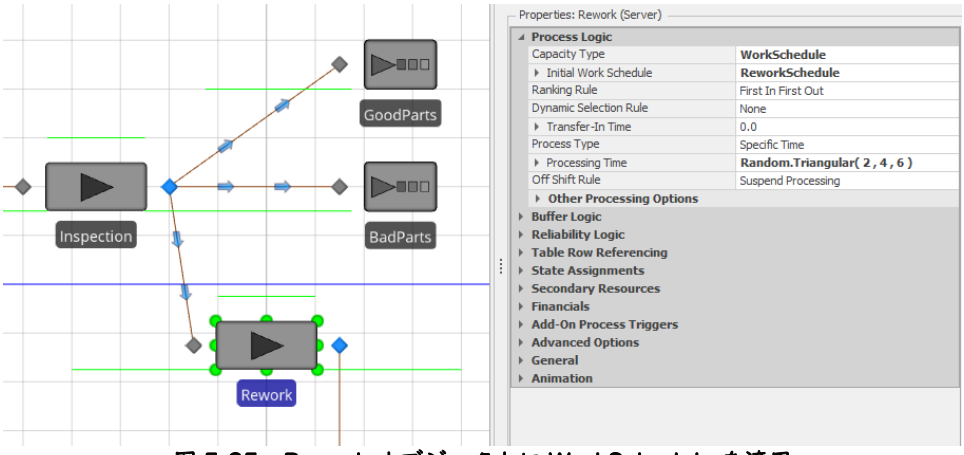


図 5.25　Rework オブジェクトに WorkSchedule を適用

部品配置機械の故障をモデルに組み込む前に、リソース能力スケジュールがサーバ稼働率の統計量にどのような影響を与えるかを検討することが重要である。これまでのモデルでは、利用率を単にサーバの稼働時間の割合（またはそのパーセンテージ）として定義していた。これはリソースの能力が一定であった場合には適切に機能した。しかし、リソース能力が変動するサーバの場合、利用率の計算はより複雑になり、その**解釈**も単純ではない。特に、リソースの「オフシフト」時間をどのように評価するかが問題となる。

たとえば、再加工作業員がある日 6 時間 28 分稼働していたとする。利用率を計算する一つの方法は、この稼働時間を 1 日（24 時間）で割って 26.17%とする方法である。しかし、作業員からすれば「待ってください。私はもっと忙しかったはずです！」という反論が予想される。この違和感の原因は、再加工作業員が 24 時間のうち第 2 シフトの 8 時間だけが**作業可能である**という事実を、この計算方法が考慮していないことにある。もう一つの計算方法は、稼働時間を実際の利用可能時間である 8 時間で割って 78.50%とする方法である。この値であれば、作業員も納得するだろう。

ここで重要なのは、これらの解釈はどちらも技術的に「正しい」ということである。これらは単にリソース利用率という概念に対する 2 つの異なる解釈であり、Simio はデフォルトでこの両方の値を出力する。

図 5.26 は、ウォームアップ期間 25 日、実行期間 125 日で行った Model 5-2（部品配置機械に故障なし）の 25 回反復実行による標準 Pivot Grid レポートの一部を示している。ここには、Inspection オブジェクトの Server Resource 統計量が含まれており、稼働率に関する重要な値を表 5.5 に抽出している。

統計量は以下のとおり定義される：

・ScheduledUtilization(Percent)：実行中のオブジェクトのスケジュール能力に対する利用率（%）。計算式は 100*(Capacity.Utilized.Average/Capacity.Average)。
・UnitsScheduled(Average)：実行中のオブジェクトにおけるスケジュール済み能力の平均値。

[Resource]	Capacity	ScheduledUtilization	Percent	77.3062	76.3032	78.7940	0.2442
		UnitsAllocated	Total	32,462.8800	53.0000	136.0000	104.2211
		UnitsScheduled	Average	0.8750	0.8750	0.8750	0.0000
			Maximum	1.0000	1.0000	1.0000	0.0000
		UnitsUtilized	Average	0.6764	0.6677	0.6894	0.0021
			Maximum	1.0000	1.0000	1.0000	0.0000
	ResourceState	TimeOffShift	Average (Minutes)	60.0000	60.0000	60.0000	0.0000
			Occurrences	300.0000	00.0000	300.0000	0.0000
			Percent	12.5000	12.5000	12.5000	0.0000
			Total (Minutes)	18,000.0000	00.0000	000.0000	0.0000
		TimeProcessing	Average (Minutes)	5.5445	5.3928	5.6699	0.0291
			Occurrences	17,569.4000	81.0000	942.0000	58.6694
			Percent	67.6429	66.7653	68.9447	0.2137
			Total (Minutes)	97,405.8470	42.0028	280.3872	307.6782
		TimeStarved	Average (Minutes)	1.6466	1.5457	1.7258	0.0166
			Occurrences	17,365.3600	71.0000	747.0000	60.2163
			Percent	19.8571	18.5553	20.7347	0.2137
			Total (Minutes)	28,594.1530	19.6128	357.9972	307.6782

図 5.26　Inspection オブジェクトの Pivot Grid の Resource 部分

表 5.5　検査の稼働率関連の統計量

カテゴリ	統計量	値
Capacity	ScheduledUtilization (Percent)	77.3062
	UnitsScheduled (Average)	0.8750
	UnitsUtilized (Average)	0.6764
Resource State	Offshift Time (Percent)	12.5000
	Processing Time (Percent)	67.6429
	Starved Time (Percent)	19.8571

・UnitsUtilized(Average)：実行中に実際に使用された能力単位の平均値。
・Offshift Time(Percent)：リソースが Offshift 状態にある時間の割合（%）。
・Processing Time(Percent)：リソースが Processing 状態にある時間の割合（%）
・Starved Time(Percent)：リソースが Starved 状態にある時間の割合（%）。ここで **Starved** とは、リソースは利用可能だが処理すべきエンティティが存在しない状態を指す。

この例題の再加工作業員について、26.17%は UnitsUtilized*100 に相当し、78.50%は ScheduledUtilization に相当する。表 5.5 の値から、検査作業員は勤務時間中の 77.3%が稼働状態であり、24 時間全体では 67.7%が稼働時間となっている。UnitsScheduled の 0.8750 という値は、1 日 24 時間のうち 21 時間（つまり 1 日の 0.8750）が勤務時間としてスケジュールされており、残りは 3 回の 1 時間食事休憩であることを示している。

Capacity 値と Resource State 値は必ずしも同じ関係性を持つわけではない。特に、スケジュールのリソース能力値は 0 か 1 のため、以下の関係式が成り立つ：

$$UnitsScheduled(Average) = \frac{Processing\ Time(\%) + Starved\ Time(\%)}{100}$$

$$UnitsUtilized(Average) = \frac{Processing\ Time(\%)}{100}$$

リソース能力が 1 以上（例：2～3 人の検査作業員がシフトで勤務）の場合、Resource State 値は稼働リソース数に依存せず、Capacity 値はリソース能力数を反映する。たとえば、検査工程の処理率を半分（Processing Time プロパティを `Random.Uniform(4, 8)`）にし、勤務サイクル中の検査作業員を 1 人から 2 人に増やすと、表 5.6 の結果が得られた。

このモデルでは、ScheduledUtilization と Offshift Time の割合は変化しなかったが、他の値は変化した。ProcessingTime の割合が増加したのは、リソース能力ユニットの一方または両方が稼働している時間を測定するためである。同様に、Starved Time の割合が減少したのは、検査リソースが Starved 状態になると両方の能力ユニットが Starved になるためである。

表 5.6　変更後の検査の稼働率関連の統計量

カテゴリ	統計量	値
Capacity	ScheduledUtilization (Percent)	77.4024
	UnitsScheduled (Average)	1.7500
	UnitsUtilized (Average)	1.3545
Resource State	Offshift Time (Percent)	12.5000
	Processing Time (Percent)	82.5084
	Starved Time (Percent)	4.9916

5.3.4　機械の故障

Model 5-2 の最終的な拡張として、部品配置機械にランダムな故障を組み込む（5.1.4 項でも機械故障の例は示した）。部品配置機械は複雑な装置であるため故障が発生しやすく、基板の処理を続けるには修理が必要となる。Standard Library の Server オブジェクトでは、以下の 3 種類の故障がサポートされている：

- **カレンダ時刻基準**（Calendar Time Based）：カレンダ時刻に基づく稼働時間で、故障間の稼働時間を指定する。
- **イベント数基準**（Event Count Based）：発生イベント数に基づく稼働時間で、イベント数を指定する。
- **処理数基準**（Processing Count Based）：サーバのエンティティ処理回数に基づく稼働時間で、故障までの処理回数を指定する。

このモデルでは、部品配置機械の故障に Calendar Time Based を採用する。過去の記録によると、部品配置機械は平均 6 時間の稼働後に故障し、修理には約 30 分を要する。稼働時間と修理時間は大きく変動するため、両方を指数分布と仮定する。この仮定を Model 5-2 に実装するには、FailureType プロパティを `Calendar Time Based` に、Uptime Between Failures プロパティを `Random.`

Exponential(6) Hours に、Time to Repair プロパティを Random.Exponential (30) Minutes に設定する（図 5.27）。

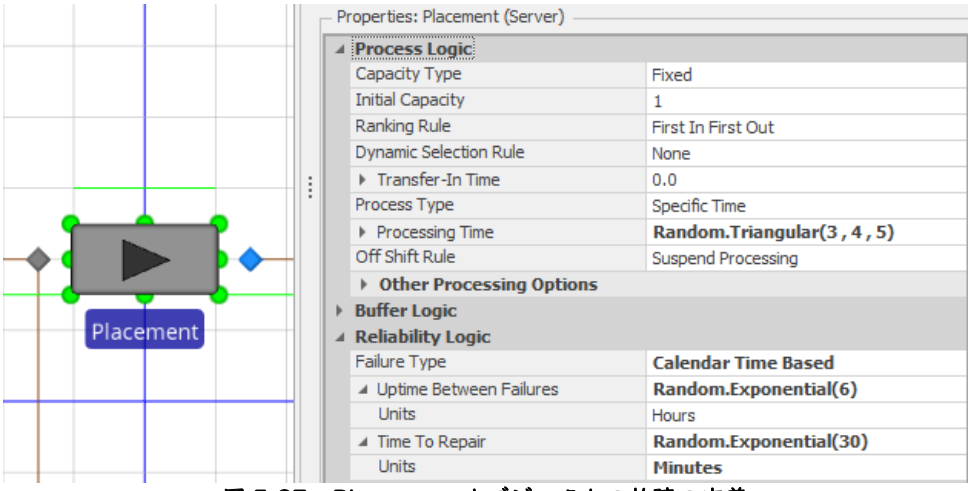


図 5.27　Placement オブジェクトの故障の定義

モデルを実行すると（前項と同じパラメータを使用）、Placement オブジェクトの ResourceState カテゴリに TimeFailed セクションが追加され、機械の故障状態時間の割合は約 7.54%と推定される（図 5.28）。

ResourceState	TimeFailed	Average (Min...	29.3480	25.9614	32.0560	0.6455
		Occurrences	370.2000	334.000	400.0000	7.8205
		Percent	7.5403	6.5220	8.3968	0.2030
		Total (Minutes)	10,858.1037	9,391.6	12,091.3	292.2791
	TimeProcessing	Average (Min...	141.7151	94.7266	192.7511	10.2053
		Occurrences	941.2400	678.000	1,350.00	64.0321
		Percent	90.1313	88.8062	90.8988	0.2306
		Total (Minutes)	129,789.1041	127,880	130,894.	332.0899
	TimeStarved	Average (Min...	5.7642	5.1519	6.0391	0.0919
		Occurrences	580.6000	291.000	1,028.00	68.2165
		Percent	2.3283	1.0411	4.2332	0.2829
		Total (Minutes)	3,352.7922	1,499.2	6,095.77	407.3246

図 5.28　故障導入後の Placement オブジェクトの ResourceState の結果

Calendar Time Based の故障は、故障発生間隔が時間の関数となるランダムな機械の故障をモデル化する際に一般的に使用される。一方、Processing Count Based の故障は、故障発生間隔が処理回数の関数となる故障のモデル化に適している。たとえば、原材料の補給が必要な状況がこれに該当する。この場合、機械は部品加工に使用する原材料の在庫を持っており、在庫が空になると補給されるまで処理を続行できない。つまり、故障（この場合は原材料切れ）の発生間隔は、処理された部品数（Processing Count）に依存する。Event Count Based の故障は、Simio 定義イベントまたはユーザ定義イベントの発生回数に基づいて故障を発生させる方式である。

5.3.5　Model 5-2 の検証

システム分析にモデルを使用する前に、モデルの正確性について一定の確信が必要である。この文脈での検証とは、モデルが意図したとおりに動作する（つまり、仕様が正しく Simio モデルに変換されている）ことの確認を意味する。Model 5-1 と 4 章のモデルの検証では、モデル出力と比較するための予測値を導き出すという基本方針をとった。ただし、それらのモデルでの予測に使用した待ち行列解析は非常に単純なものだった。Model 5-2 は依然として比較的単純なモデルではあるものの、標準的な待ち行列モデルでは扱いにくい要素が追加されている。もちろん、待ち行列理論や他の分析手法でシステムを完全に分析できるのであれば、シミュレーションは不要である。Model 5-2 の待ち行列解析を複雑にする要素は、以下の 2 点である：

1. 再入可能な流れ：再加工ステーションから出た基板は再処理のために戻されるため、Placement ステーションへの到着率は外部からの到着率（毎時 10 基板）を上回る。
2. 実行中に変動する能力：リソース能力スケジュールと機械故障により、Placement、Inspection、Rework リソースの能力が時間とともに変化する。

これらの問題を順に検討していく。Placement ステーションへの到着率が Model 5-1 の毎時 10 基板を超えることから、Placement リソースの利用率も上昇すると予測される。新しい利用率を推定するには、実際の到着率を測定（推定）する必要がある。図 5.29 は再入可能な流れを示している。

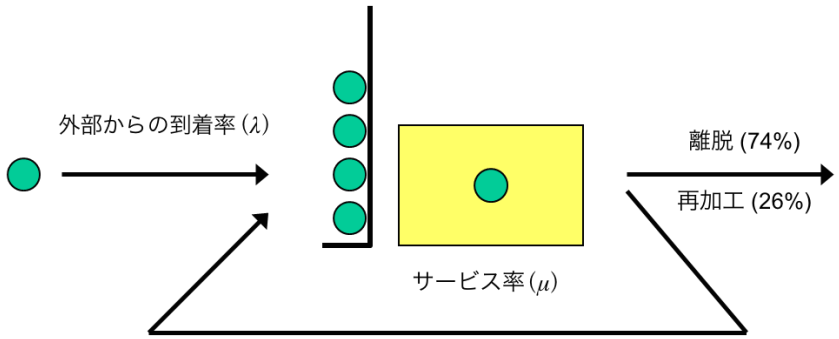


図 5.29　再入可能な流れ

ここで、λ' を**有効な到着率**（システムが実際に処理する到着率）とすると：

$$\lambda' = \lambda + 0.26\lambda'$$

λ' について解くと、以下を得る：

$$\begin{aligned}\lambda' &= \lambda/(1-0.26)\\ &= 1.3514 \times \lambda\end{aligned}$$

機械の故障を便宜的に無視すると、部品配置機械の稼働率は $13.514/15 = 0.9019$ と予測できる。この稼働率は厳密に 1 以下であるため、離脱率は到着率と

等しくなり、検査ステーションの期待稼働率は（リソース能力スケジュールを考慮しない場合）$13.514/20 = 0.6757$となる。同様に、検査の利用率も 1 以下であるため、再加工ステーションの期待利用率は（同じくリソース能力スケジュールを考慮しない場合）、$(13.514 \times 0.26)/15 = 0.2342$となる。リソース能力スケジュールと機械故障を無視して Model 5-2 を実行すると、モデルの結果が予測値と一致することが確認できる。

リソース能力スケジュールと機械故障は、リソースの利用可能な能力を減少させる。リソース能力スケジュールでは、検査と再加工に対する **1 日当たりの**到着率とサービス率を考慮するとよい。検査オペレータは（3 回の 1 時間の食事休憩により）1 日 21 時間働いているので、1 日当たりの処理率は$21 \times 20 = 420$となり、期待利用率は$(24 \times 13.514)/420 = 0.7722$である。同様に、再加工作業員は 2 番目のシフトのみ働いているので、1 日当たりの処理率は$8 \times 15 = 120$となり、期待利用率は$(24 \times 13.514 \times 0.26)/120 = 0.7027$である。

機械の故障は異なるアプローチで考える必要がある。部品配置機械の期待稼働時間は 6 時間で、期待修理時間は 30 分である。したがって、機械は 390 分中約 30 分、つまり 7.692%の割合で故障する。部品配置機械が予測遊休時間の割合$(1 - 0.9010) \times 100 = 9.91\%$は期待故障時間の割合より長いため、システムは安定状態を維持できると予測される。

表 5.7 は、100 日間のウォームアップ期間を経た 600 日間の実行における、50 回の反復実験の結果を示している。これらの値は予測と一致しており、モデルの正確性を強く示唆している。したがって、このモデルをシステム分析に使用することができる。

表 5.7　Model 5-2 の検証実験結果

推定する指標	シミュレーション結果
Placement の利用率（ρ_p）	0.9006 ± 0.0825
Inspection の利用率（ρ_i）	0.7719 ± 0.0716
Rework の利用率（ρ_r）	0.7025 ± 0.1198
故障時間の割合（Placement）	7.6926 ± 0.0591

5.4　Model 5-3：工程選択のある PCB モデル

PCB モデルの最終的な拡張として、既存の部品配置機械の後に第 2 の部品配置工程を追加する。この新しい工程では、**ファインピッチ**部品を PCB に実装する。これらの部品は高精度な配置が必要なため、通常の部品配置よりも処理時間が大幅に長くなる。そこで、通常の部品配置、検査、および再加工工程の遅延を防ぐため、**並行して** 3 台の部品配置機械を使用する。図 5.30 に示す完成モデルは、異なる処理速度を持つ工程からなる連続生産システムの典型的な構成例である。

モデルの現実性を高めるため、3 台のファインピッチ部品配置機械には異なる処理速度を設定する。具体的には、高速機、中速機、低速機の処理時間が以下の分布に従うと仮定する：

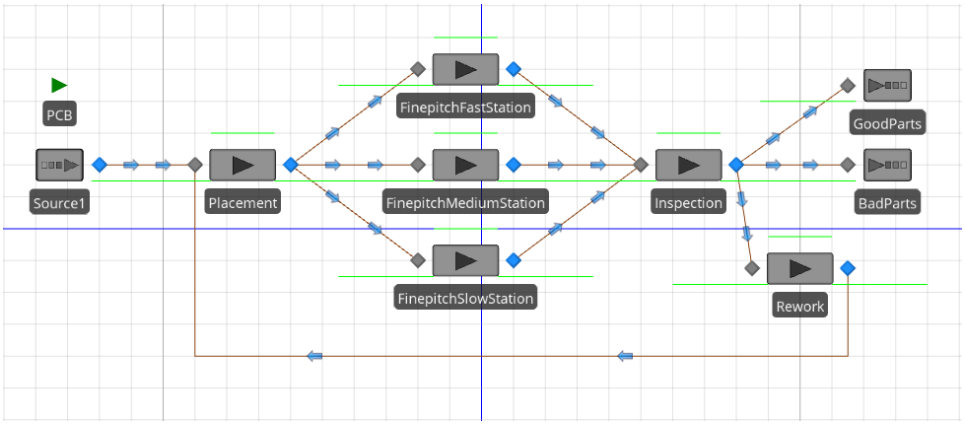


図 5.30　Model 5-3

・高速機：Triangular(8, 9, 10)
・中速機：Triangular(10, 12, 14)
・低速機：Triangular(12, 14, 16)

また、高速機についてはランダムな故障も想定する。故障の発生間隔は平均 3 時間の指数分布に従い、修理時間は平均 30 分の指数分布に従うものとする。なお、中速機と低速機は故障しないと仮定する。

図 5.30 に示すように、Placement オブジェクトと Inspection オブジェクト間の Connector オブジェクトを削除し、3 台のファインピッチ部品配置機械（FinepitchFastStation、FinepitchMediumStation、FinepitchSlowStation）を表す新しい Server オブジェクトを追加した。各ステーションの Capacity Type プロパティを Fixed、Initial Capacity プロパティを 1 に設定し、前述の処理時間分布を Processing Time プロパティに設定する。さらに、FinepitchFast Station オブジェクトには故障設定を追加した（図 5.31）。

Placement オブジェクトの出力ノードから 3 個の新しい Server オブジェクトの入力ノードへ Connector オブジェクトで接続し（**分岐**）、その後、新しい Server オブジェクトの出力ノードから Inspection オブジェクトの入力ノードへ接続した（**合流**）。

モデルには現在 5 つの Server オブジェクトインスタンスがあるため、**プロパティスプレッドシートビュー**の使用方法を説明する。このビューでは、特定のオブジェクトの全インスタンスに共通するプロパティを、スプレッドシート形式で効率的に表示できる。図 5.32 は Model 5-3 のプロパティスプレッドシートビューを示している。

このビューを開くには、オブジェクトインスタンスを右クリックしてコンテキストメニューから「Open properties spreadsheet view for all objects of this type …」を選択するか、Project Home リボンの Windows セクションの Properties ボタンを使用する。このビューには標準の並べ替えとフィルタリング機能が備わっており、同じオブジェクトの複数インスタンスを扱う際に特に便利である。

Properties: FinepitchFastStation (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	**Random.Triangular(8 , 9 , 10)**
Off Shift Rule	Suspend Processing
Other Processing Options	
Buffer Logic	
Input Buffer	
Capacity	**0**
Balking & Reneging Options	
Output Buffer	
Capacity	Infinity
Balking & Reneging Options	
Reliability Logic	
Failure Type	**Calendar Time Based**
Uptime Between Failures	**Random.Exponential(3)**
Units	Hours
Time To Repair	**Random.Exponential(30)**
Units	**Minutes**

図 5.31　FinepitchFastStation オブジェクトのプロパティ

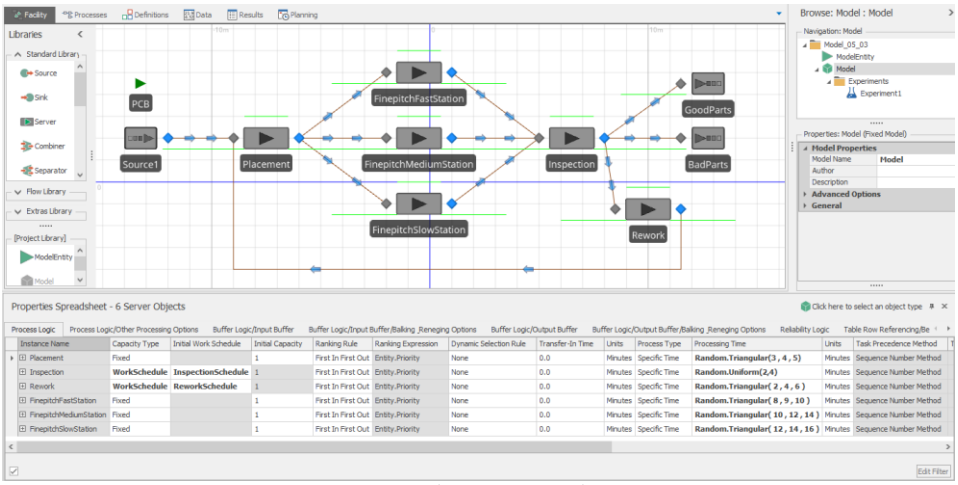


図 5.32　Model 5-3 のプロパティスプレッドシートビュー

5.2 節の最初の分岐例では、エンティティの目的地を**リンクウェイト**で無作為に選択した。ファインピッチ機械の選択でも同様のロジックを使用できるが、今回は異なるアプローチをとる。機械をランダムに選択するのではなく、「最初に利用可能な」機械を選択することにする。このロジックには、Simio の**ノードリスト**を使用する。ノードリストの実装には、まずリストを定義し、次に Placement オブジェクトの出力ノードで PCB オブジェクトの目的地決定にそのリストを使用

するよう設定する。リストの定義手順は以下のとおりである：Navigation パネルで Model を選択し、Definitions タブを選択して、Definitions パネルから Lists アイコンを押す。Simio は、String、Object、Node、Transporter の各リストタイプをサポートしている。リボンの Node アイコンをクリックして新しいノードリスト FinePitchStations を作成し、3 つのファインピッチ・オブジェクトの入力ノード（Input@FinepitchFastStation、Input@FinepitchMediumStation、Input@FinepitchSlowStation）をメインウィンドウ下部枠のノードリストに追加する（図 5.33）。なお、下部枠の Node リストの各項目は、モデル内の全ノードを含むプルダウンリストから選択できる。

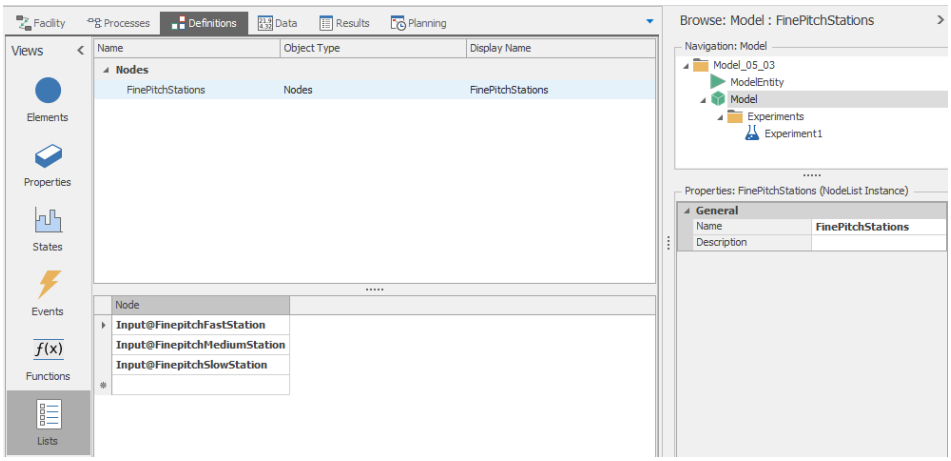


図 5.33　FinePitchStations ノードリスト

次に、定義したリストから要素を選択する方法を Simio に設定する必要がある。実際のシステムでは、最初に利用可能になる機械に PCB を送ることが望ましい。しかし、処理時間がランダムで機械が偶発的な故障を起こす可能性があるため、どの機械が最初に利用可能になるかを事前に判断することは困難である。そこで、代替の評価基準として、待機中または移動中の基板数がもっとも少ない機械を選択することにする。このロジックを実装するため、Placement オブジェクトの出力ノードを選択し、図 5.34 に示すパラメータ値を設定する。

Entity Destination Type と Node List Name プロパティは、FinePitch Stations リスト（図 5.33）を使用するように設定し、Selection Goal プロパティでは、Selection Expression の最小値を選択するよう指定する。エンティティがノードに到着すると、Simio はリスト内の各候補ノードについて式を評価し、その値が最小となるノードへエンティティを送る。

式の値 Candidate.Node.AssociatedStationOverload について説明しよう。Candidate.Node というキーワードは、リストから各候補ノードを評価することを示す。関数 AssociatedStationOverload は、各ノード（この場合はファインピッチ機械オブジェクト）に対応するステーションの過負荷状態を返す。Simio Reference Guide では、このプロパティを次のように定義している：

Properties: Output@Placement (TransferNode)

Crossing Logic	
Initial Traveler Capacity	Infinity
Entry Ranking Rule	First In First Out
Routing Logic	
Outbound Travel Mode	Continue
Outbound Link Preference	Any
Outbound Link Rule	Shortest Path
Entity Destination Type	**Select From List**
Node List Name	**FinePitchStations**
Selection Goal	**Smallest Value**
Selection Expression	Candidate.Node.AssociatedStationOverload
Selection Condition	
Blocked Destination Rule	Select Available Only
Other Routing Out Options	
Transport Logic	
Ride On Transporter	False
State Assignments	
Tally Statistics	
Add-On Process Triggers	
Advanced Options	
General	
Animation	

図 5.34　Placement オブジェクトの出力ノードのプロパティ

外部の入力ノードについて、そのノードに対応するオブジェクト内のステーション位置における、負荷値と能力との差の現在値を返す（正の値は過負荷（Overload）を意味する）。

ステーションの「負荷」は、(1) ステーションに向かっているエンティティ数、(2) ノードで待機中のエンティティ数、(3) ステーションを使用中のエンティティ数の合計として定義される。この関数は、負荷と現在のステーション能力の差を返す（$Overload = Load - Capacity$）。

つまり、エンティティが Placement オブジェクトの出力ノードに到達すると、Simio は最小の過負荷（機械の待ち行列の長さを示す指標）を持つファインピッチ機械の入力ノードを選択し、そこへエンティティを送る。複数のノードが同じ過負荷値を持つ場合（例：最初の基板到着時は全機械が未使用）、Simio はリストの順序に従って最初のノードを選択する。

重要な注意：Input Buffer Capacity を無限（Infinity）に設定すると、AssociatedStationOverload や AssociatedStationLoad などの関数は正しく動作しない。無限大に数を加えても無限大のままという基本的な性質を考えれば、これは当然である。このモデルでは、同着の場合の優先順位を確保するため、高速機を最初に配置し、バッファ容量を 0 に設定している。容量を 0 にすることで、宛先の選択をできるだけ遅らせることができる。なお、1、2、3 など異なるバッファ容量でも実験し、モデルの挙動への影響を確認することを推奨する。

AssociatedStationOverload 以外にも、キーワード AssociatedStation はさまざまな値と組み合わせて使用できる。主な例を以下に示す（式ビルダで値を展開し、矢印で選択肢を確認可能）：

・AssociatedStation.Capacity：外部入力ノードのステーション現在容量を返す。
・AssociatedStation.Capacity Remaining：外部入力ノードのステーションにおける、現在利用可能な容量を返す。
・AssociatedStation.Contents：外部入力ノードのステーションで待機中のエンティティ数を返す。

ファインピッチ部品配置処理のロジックをモデルに実装したので、次は各機械（高速・中速・低速）の選択頻度を追跡するためのユーザ定義統計量を追加する。この統計量には 2 つの目的がある。第一に、モデル検証のためである。理論的には、高速・中速・低速機の使用割合はそれぞれ約 38%、33%、29%となるはずである（章末の問題 8 参照）。第二に、ユーザ定義統計量の設定方法を説明するためである。統計量の作成手順は以下のとおりである：

1.　各機械による基板処理回数を記録するモデル**状態変数**を定義する。
2.　各ファインピッチ機械が処理を完了した際、対応する状態変数を増加させる。
3.　各使用割合を Output Statistic として報告するように設定する。

最初のステップとして、Navigation ウィンドウで Model を選択し、Definitions タブを開き、Definitions パネルから States アイコンを選ぶ。そして、3 つの Discrete States（NumFast、NumMedium、NumSlow）を追加する（図 5.35）。

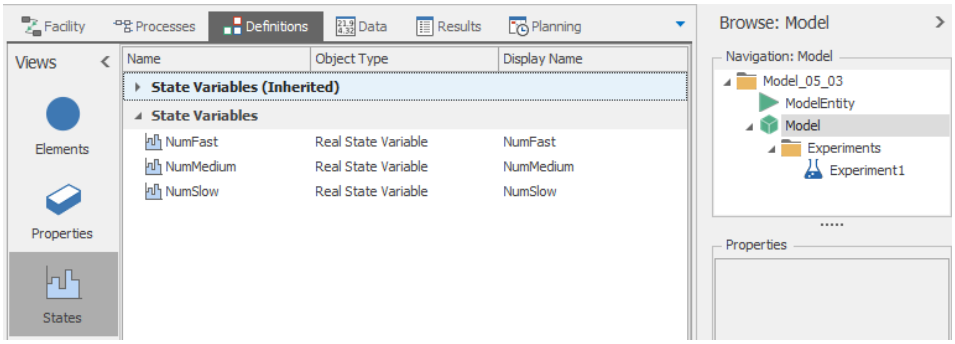


図 5.35　ファインピッチ機械の処理件数を追跡するモデル状態変数の設定

ユーザ統計量を定義して部品配置機械による基板処理数を追跡した際（5.3.1 節）、エンティティ状態変数を使用し、モデル状態変数は使用しなかった。これは、前回は**エンティティ**の特性を追跡していたのに対し、今回は**モデル**の特性を追跡しているためである。この区別は、ユーザ定義統計量を作成する際に非常に重要となる。

2 つ目のステップでは、5.3.1 項と同じ手法を使用する。Server オブジェクトに対する State Assignments プロパティを使用し、各リピートグループの BeforeExiting で NumFast、NumMedium、NumSlow を増加させる。

最後に、モデルの Definitions パネルから Elements アイコンを選択し、各機械の処理完了割合を計算するために 3 つの Output Statistics を追加する。Output Statistic は反復実行終了時に式の値を報告する。高速機の利用割合を求める式は以下のとおりである：

```
NumFast / (NumFast + NumMedium + NumSlow)
```

この設定により、Simio は反復終了時に割合を計算し、反復間の統計量を追跡・報告する。中速機と低速機の出力統計量も、分子の NumFast をそれぞれ NumMedium、NumSlow に変更する以外は同様に定義する。ただし、モデルが基板処理前に終了した場合、分母が 0 となりゼロ除算エラーが発生する。これを防ぐため、Math.Max()関数を使用して分母が 0 にならないようにできる（例：Math.Max(1, NumFast + NumMedium + NumSlow)）。特別な理由がない限り、これは推奨される方法である。

また、Simio が提供する処理数のカウントを使用して出力統計量を定義することも可能である。具体的には、以下の Simio プロパティを使用できる：

```
FinepitchFastStation.Processing.NumberExited
```

は、FinepitchFastStation オブジェクトから出るエンティティの数を返す。したがって、以下の式を用いることによって、上で指定した同じ割合を指定できる：

このプロパティは FinepitchFastStation オブジェクトから出たエンティティの数を返す。したがって、高速機の利用割合は以下の式でも計算できる：

```
FinepitchFastStation.Processing.NumberExited /
(FinepitchFastStation.Processing.NumberExited +
FinepitchMediumStation.Processing.NumberExited +
FinepitchSlowStation.Processing.NumberExited)
```

これら 2 つの手法を比較すると、独自の状態変数を使用する利点は、値の追跡方法を**正確に**把握でき、カスタマイズの自由度が高いことである。一方、Simio が提供する状態変数を使用するほうが、状態変数の定義や増加処理が不要なため簡便である。どちらの方法も理解しておくことが重要である。なお、この方法でもゼロ除算エラーの可能性があるため、前述のように Math.Max()関数の使用を検討すべきである。

5.5　Model 5-4：複数の代替シナリオの比較

シミュレーションの主たる用途の 1 つは、代替システムのデザインを評価することである。例として、PCB 組立システムへのファインピッチ部品配置機械の追加購入を検討している場合を考える。高速、中速、低速のいずれかの機械を 1 台購入するかどうかを判断したい（各機械の価格は異なる）。本節では、Model 5-3 を改良して Model 5-4 を作成し、各代替案におけるシステムの予測パフォーマンスを比較できるようにする。

従来の方法では、ファインピッチ機械ステーションの能力を 1 つずつ増加させ、モデルを実行して結果を保存し、これを異なる機械タイプで繰り返すことになる。

この手動での比較も可能だが、ここでは Simio の実験機能を活用した効率的なアプローチを紹介する。最初のステップとして、3 つのファインピッチ Server オブジェクトの Initial Capacity プロパティに対してプロパティ参照(5.1.2.4 参照)を定義し、実験での操作を容易にする。

図 5.36 は、FinepitchFastStation サーバのプロパティ設定を示しており、Initial Capacity にプロパティ参照 InitialCapacityFast を設定している(緑色の矢印はプロパティ参照を示す)。プロパティ参照を定義するには、Properties ウィンドウの Initial Capacity プロパティの**ラベル上で右クリック**し、Set Referenced Property から Create New Referenced Property を選択する。表示されるダイアログボックスで Reference Property 名を入力する(ここでは InitialCapacityFast)。同様にして、中速・低速機のプロパティ参照も定義する。

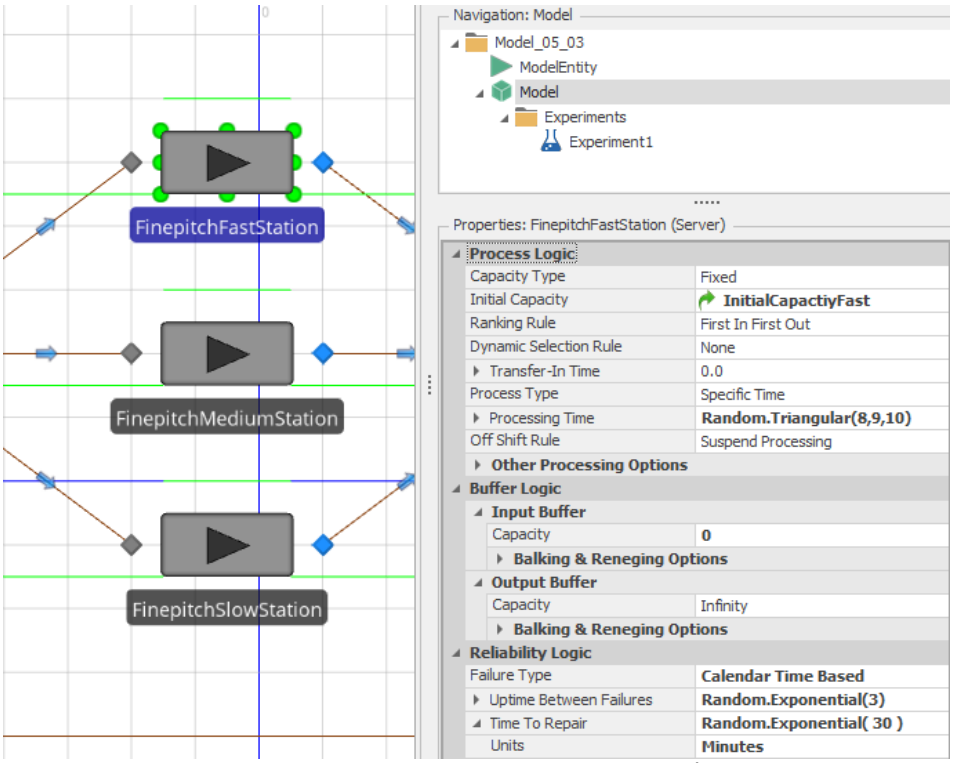


図 5.36　高速のファインピッチ機械ステーションにおけるプロパティ参照の設定

3 つのプロパティ参照を定義後、Navigation ウィンドウから Model を選択し、Definitions タブの Properties を選択すると、新しく定義したプロパティ参照が表示される(図 5.37)。

ここで、Default Value プロパティを設定する(Model 5-4 では 1 に設定)。これにより、3 つのファインピッチ機械の Initial Capacity プロパティがプロパティ参照を使用するよう設定された。インタラクティブ実行時のプロパティ参照値は、Navigation ウィンドウで Model を右クリックして Properties を選択することで設定できる。

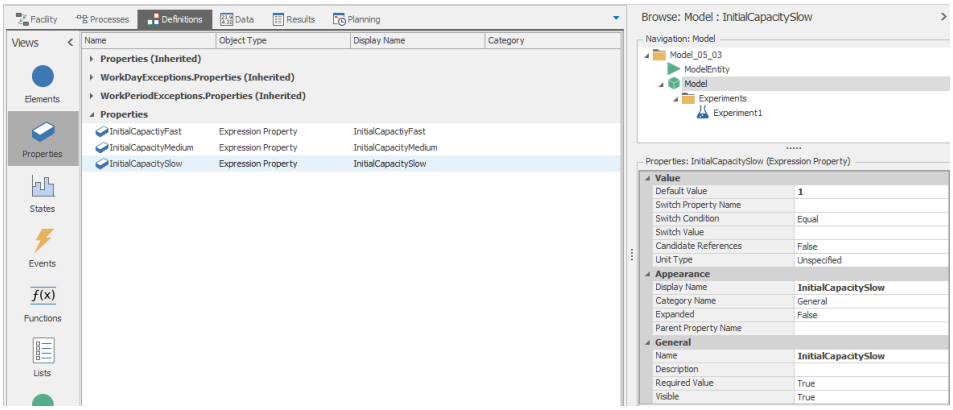


図 5.37　Model 5-4 におけるプロパティ参照

次のステップは、3 つの代替案を比較するための実験を定義することである。実験を定義する前に、パフォーマンスの差をより明確にするため、3 つのファインピッチ機械ステーションの Processing Time プロパティを調整する。新しい処理時間分布は、高速が Triangular(8, 9, 10)、中速が Triangular(18, 20, 22)、低速が Triangular(22, 24, 26)である。

更新されたモデルの Experiment Design を図 5.38 に示す。

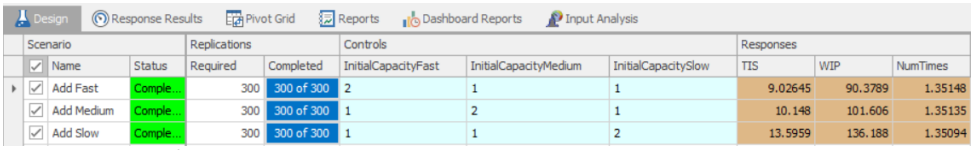


図 5.38　プロパティ参照と結果が表示された Model 5-4 の Experiment Design

ここで、実験の 3 つの側面、**コントロール**(Controls)、**レスポンス**(Responses；実験テーブルの列)、および**シナリオ**（Scenarios；実験テーブルの行）について説明する。4.5 節では実験に**レスポンス**(応答)を導入し、Simio MORE(SMORE)プロットでレスポンスを用いる方法を解説した。Model 5-4 では、以下のレスポンスを定義した：

- TIS（システム内時間；Time in System）：エンティティ（PCB）のシステム内時間の平均。Simio は各エンティティタイプについて、この統計量（PCB オブジェクトの PCB.Population.TimeInSystem.Average）を自動的に追跡する。
- WIP（システム内数；Work in Process）：システム内のエンティティ（PCB）数の平均。Simio は各エンティティタイプについて、この統計量（PCB オブジェクトの PCB.Population.NumberInSystem.Average）を自動的に追跡する。
- NumTimes：エンティティが部品配置機械によって処理された回数の平均。5.3.1 項で対応するタリー統計量を定義し、その平均値を得るために式 NumTimes Processed.Average を用いる。

図 5.38 では、実験の実行中に**現在**のレスポンス値が表示される。Controls 列には、3 つのファインピッチ部品配置機械オブジェクトの初期能力を指定するためのプロパティ参照が表示されている。実験では、これらのプロパティ値と実行

する反復回数を設定してシナリオを定義できる。図 5.38 に示すように、プロパティ参照を用いて 3 つのシナリオ（高速機 1 台追加、中速機 1 台追加、低速機 1 台追加）を定義した。実験テーブルの左列のチェックボックスで実行するシナリオを選択でき、コンピュータの CPU 性能に応じ複数シナリオを同時に実行できる。

図 5.39 は、TIS レスポンスを選択した 3 シナリオの Response Chart タブ（4.5 節の SMORE プロット）を示している。直接比較できるよう、3 つの SMORE プロットは同じスケールで表示されている。これらのプロットでは、Upper Percentile を 90%、Lower Percentile を 10%に設定しており、箱ひげ図の箱は 300 回の反復におけるレスポンスの中央 80%を示している。また Response Chart リボンの Rotate Plot ボタンで値の軸を水平表示にした。予想通り、Add Fast シナリオが他の 2 つより優れており（平均システム内時間が短い）、Add Medium は Add Slow より良好である。図の左上のプルダウンリストから WIP レスポンスを選択しても、同様のパターンが見られる（図省略）。

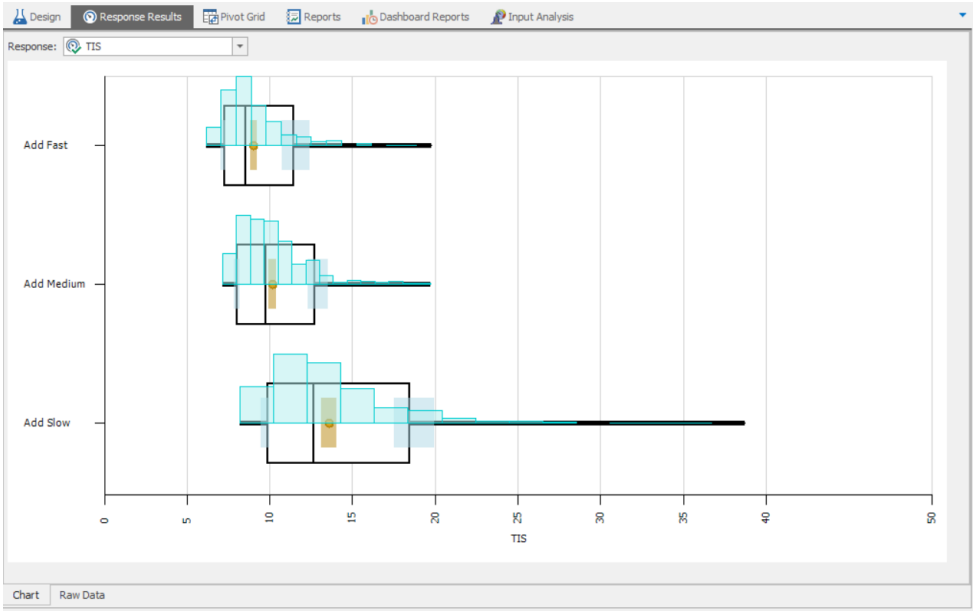


図 5.39　Model 5-4 における平均システム内時間の SMORE プロット

図 5.40 は、部品配置機械でエンティティが処理された平均回数（NumTimes レスポンス）の SMORE プロットを示している。予想通り、基板の処理回数は購入する機械の種類（高速、中速、低速）に影響されない。

TIS や WIP の SMORE プロットで、あるシナリオが他より優れて「見える」かもしれないが、正式な結論を導く前に統計的に妥当な比較を行うべきである。一つの方法は、Simio の Export Details 機能で各反復のレスポンスを CSV ファイルに出力し、専用の統計分析パッケージで分析することである（4.6 節参照）。**平均値**の比較では、古典的な対応のある*t*検定アプローチが使える。このアプローチでは、**2 つ**のシナリオの平均値の差の信頼区間を構築するか、**2 つの**平均値の差の仮説検定を行う。ここでは全シナリオで同じ乱数を使用するため標本が独立でな

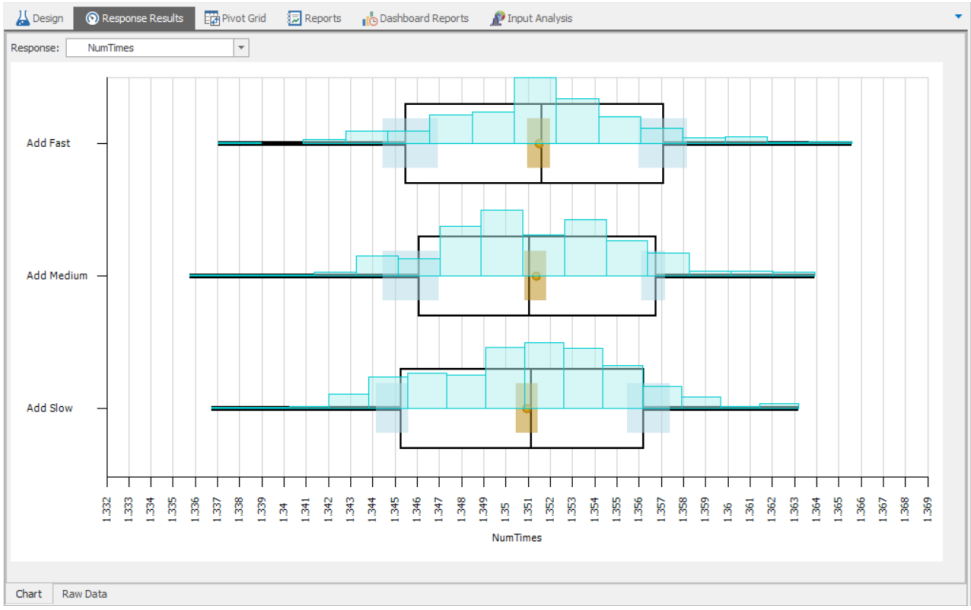


図 5.40　Model 5-4 における平均処理回数の SMORE プロット

いことから、**2 標本による*t*検定**ではなく**対応のある*t*検定**を用いる。対応のある*t*検定は非独立性を許容できるが、2 標本の*t*検定では独立した標本が必要で、各シナリオに別々の乱数が必要となる。この方法では **2 つ**のシナリオしか同時に比較できず、3 つ以上のシナリオではすべての組み合わせ（高速と中速、高速と低速、中速と低速）を比較することになる。ただし、多数の統計的比較を行うと全体の信頼水準が低下するため注意が必要である。同時にすべての平均値を比較するには、分散分析（ANOVA）のような一般的な多重比較手法（Tukey、Scheffé、Bonferroni など）を用いる。詳細は標準的な統計学の教科書を参照されたい。

シミュレーションの応用では複数の代替シナリオの比較が一般的であり、そのための特別な統計的手法が開発されている。図 5.38 の TIS レスポンス列では、Add Medium と Add Slow シナリオの Response 値は薄茶色で、Add Fast シナリオは濃茶色で表示されている。これは、Simio Experiments の Subset Selection Analysis 機能の結果である。この分析は Boesel et al.（2003）の統計的手法を用いて、応答の期待平均値に基づきシナリオを 2 つの部分集合に分類する。300 回の反復実験に基づき、「おそらく最善」なシナリオ（濃茶色）と「棄却」シナリオ（薄茶色）に分けられる。Nelson（2013）によれば、この手法の重要な解釈は「おそらく最善の部分集合に分類されるものはすべて、標本の最適値から統計的に有意な差がない」ということである。Subset Selection を実行するには、Experiment 実行後に Experiment Design ウィンドウで TIS レスポンスを選択し、Properties グリッドの Objective を `Minimize` に設定する（平均システム内時間は小さいほうが望ましいため）。次に、Experiment Design リボンの Analysis から `Subset Selection` を選択すると、アルゴリズムが「おそらく最善」（この場合は Add Fast）と「棄却」（Add Medium と Add Slow）を判定する。なお、この分析の統計的信頼水準は、Experiment Properties の Confidence Level で

90%、95%、98%、99%のいずれかに設定する必要がある。WIP レスポンスについても同様の分析が可能で、同じく Objective を Minimize に設定する。実際に分析すると、TIS レスポンスの場合と同じ「最善」「棄却」の分類結果となったが、これは偶然の一致であり、すべてのレスポンスでこのような一致は期待できない。なお、NumTimes レスポンスについては、その値の大小どちらが望ましいか明確でないため、Subset Selection Analysis は実施しなかった。

部分集合の選定は必ずしも明確な結論を導けるわけではない。「おそらく最善」シナリオの部分集合に複数のシナリオが含まれる可能性があるためである。このような場合、選択肢を絞り込むために、Experiment リボンの Select Add-In に「Select Best Scenario using KN」アドインが用意されている。これも平均値に基づく手法で、Kim and Nelson（2001）のアルゴリズムを使用する。このアドインは、選択したレスポンス（一度に 1 つ）について、「最善」にもっとも近い単一のシナリオを特定するため、反復実行回数を増やしながら実験を実行する。設定項目として、信頼水準と**検査続行域**（真の最善シナリオとの差の許容範囲）、および反復実行回数の上限を指定する必要がある。この手順の詳細は Simio Reference Guide の「Select Best Scenario Using KN Add-In」を、統計的手法の詳細は Kim and Nelson（2001）を参照されたい。また、9.1.2 項では大規模な例でこの KN アドインを使用した最善シナリオの選択を解説する。

5.6　まとめ

本章では新しい Simio の概念を用いた中級モデリングを扱い、Simio のオブジェクト指向の定義について基礎を固めた。また、複数の代替シナリオを比較する際の効果的な統計分析手法について説明した。これらの内容は、より高度な Simio シミュレーションへの強固な基盤となる。続く章では、モデルデータ、アニメーション、高度なモデリング技法について解説する。

5.7　問題

1．オブジェクト**プロパティ**とオブジェクト**状態変数**の違いを説明しなさい。

2．Server オブジェクトのプロセスにおいて、トークンの**親**オブジェクトと**関連**オブジェクトの違いを説明しなさい。

3．表 5.2 の Model 5-1 において、両ステーションの処理時間が指数分布の場合と同様の定常状態値を示す待ち行列モデルを開発しなさい。

4．運転免許証を発行するオフィスについて考える。顧客は受付/申請、視力検査、筆記試験の 3 段階のプロセスを経る。顧客の到着は 1 時間当たり平均 6 人のポアソン過程に従う（つまり、到着間隔は平均 10 分の指数分布）。表 5.8 に各プロセスの処理時間分布を示す。オフィスには受付/申請担当者 1 名と視力検査担当者 1 名がいる。筆記試験は 3 台のコンピュータで実施される。オフィスは午前 9 時から午後 5 時まで営業する。このシステムの Simio モデルを開発し、次の指標を評価しなさい：(1) 顧客のシステム内滞在時間、(2) 担当者とコンピュータの稼働率、(3) 各待ち行列（受付/申請、視力検査、筆記試験）の平均待ち人数と最大待ち人数。信頼性の高い結果を得るために必要な実験の反復回数を提案し、その理由を説明しなさい。

表 5.8　問題 4 の処理時間パラメータ

プロセス	処理時間の分布
受付/申請	Triangular(5, 8, 11)
視力検査	Triangular(2, 4, 6)
筆記試験	Triangular(15, 15, 20)

5. 問題 4 のモデルをアニメートしなさい。まだ実施していない場合は、合理的なオフィスレイアウトを設定し、顧客の移動には Path オブジェクトを使用する。ステーション間の距離がオフィスとして現実的であり、エンティティの移動速度が人の歩行速度として適切となるよう注意すること。

6. Model 5-2 では、検査作業員のシフトが重複しないと仮定していた（つまり、2 人の検査員が同時に作業することはなかった)。あるシフトの開始 1 時間と次のシフトの終了 1 時間で作業員が**重なって**作業するようにモデルを変更しなさい。2 つのモデルの結果を比較し、作業員のシフト重複がシステム運営に有効かどうかを検討しなさい。

7. Model 5-2 では、基板検査での再加工回数に制限を設けていなかった。基板が検査に 3 回不合格となった場合、不良品として処理されるように Model 5-2 を修正しなさい。検査に 3 回不合格し不良品となった基板の数を計測すること。

8. Model 5-3 の検証過程で示された、高速・中速・低速のファインピッチ部品配置機械への部品振り分け比率（それぞれ 38%、33%、29%）を推定するための待ち行列モデルを開発しなさい。

9. 調剤薬局での処方薬受け取りについて考える。顧客は 2 つの方法で処方薬を受け取ることができる。1 つは医師から処方箋を事前にファクスで送ってもらい後から薬局を訪れる方法、もう 1 つは処方箋を持参しその場で調剤を待つ方法である。送信された処方箋は薬剤師が直接受け取り、薬を調合してカウンタ裏の保管場所に置く。**ヒント**：このソリューションでは、初心者向けに 7.3 節で説明する 2 つの基本的なレートテーブルを使用している。

・データによると、約 59%の顧客が処方箋を事前にファクスで送っており、薬剤師の調剤時間は三角分布(2, 5, 8)分である。事前に処方箋を送信した顧客の場合、レジ係は保管場所から調剤済み薬を取り出し、支払い処理を行う。レジ係が処方箋を探して支払い処理を完了するまでの時間は、三角分布(2, 4, 6)分である。処方箋を持参した顧客の場合、レジ係は支払い処理を行い、処方箋を薬剤師に渡して調剤を依頼する。この場合も、レジ係の処理時間は三角分布(2, 4, 6)分、薬剤師の調剤時間は三角分布(2, 5, 8)分である。ファクスによる処方箋の到着、顧客の来店、および薬局スタッフの配置は 1 日を通じて変動する。表 5.9 に、到着率と人員配置のデータを示す（C：レジ係の人数、P：薬剤師の人数、λ_1：ファクスによる処方箋の到着率、λ_2：顧客の到着率）。

表 5.9　問題 9 の到着率と人員配置のデータ

時間帯	C	P	λ_1	λ_2
午前 8 時　－　午前 11 時	1	2	10	12
午前 11 時　－　午後 3 時	2	3	10	20
午後 3 時　－　午後 7 時	2	2	10	15
午後 7 時　－　午後 10 時	1	1	5	12

・この薬局の Simio モデルを作成しなさい。評価すべき性能指標は、ファクスされた処方箋の平均調剤時間、顧客の平均システム滞在時間、およびレジ係と薬剤師の稼働率である。薬局は午前 8 時から午後 10 時まで営業し、閉店時にシステム内に残る処方箋と顧

客は無視できるものとする。分析には 500 回の反復を用い、各性能指標について SMORE プロットを作成すること。

10. 図 5.41 に示す直列生産システムの Simio モデルを作成する。部品は時間当たり 120 個の割合でポアソン過程に従って到着する。各機械の特性を表 5.10 に示す。

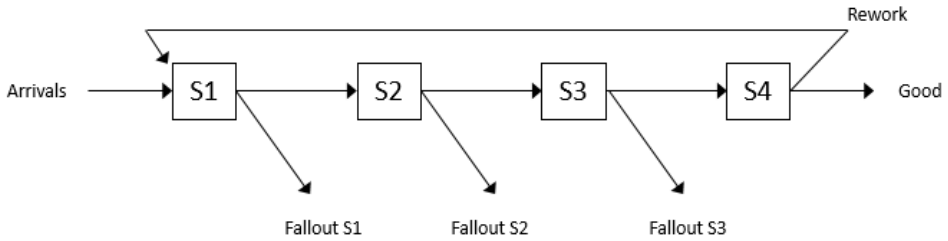


図 5.41　問題 10 の直列生産システム

表 5.10　問題 10 の機械の特性

	サービス時間（秒）	窓口（容量）	不良/再加工の確率
S1	UNIF(10, 20)	1	0.15
S2	TRIA(20, 30, 40)	1	0.3
S3	UNIF(60, 84)	2	0.33
S4	EXPO(40)	1	0.08

モデルには、以下の特性を含める必要がある：

a) 部品（エンティティ）は、機械間を瞬時に（0 シミュレーション時間で）移動する。

b) モデル内のオブジェクトインスタンスに、システムの意味がよくわかるような名前をつける。

c) システム内の「Reworks（再加工）」の数を追跡する**ユーザ定義統計量**を作成する。「Rework」とは、少なくとも 1 回再加工された（つまり、S1 に 2 回以上入った）部品を指す。

d) 次の情報を表示する 25 時間分のステータスプロットを作成する：（1）現時点のシステム内エンティティ数、（2）システム内のエンティティ数の平均、（3）現時点のシステム内再加工数。

e) モデルの魅力的なアニメーションを作成する。

f) 以下の条件で実験を実施する：
- 実行時間：200 時間
- ウォームアップ：50 時間
- 20 回の反復実行

g) 以下の**実験応答**を作成する：
- システム内の平均部品数
- システム内の最大再加工数
- システム内の部品滞在平均時間（分）
- S4 の稼働率
- 良品（Good）の完成率（1 時間当たりの部品数）
- S3 における不良（Fallout）率（1 時間当たりの部品数）

第 6 章　入力解析

3 章から 5 章のスプレッドシートモデルでは、モデル作成の過程で入力データの確率分布の特定が必要な箇所がいくつかあった。たとえば、Model 3-3 では、帽子の需要を整数$\{1000,1001,\cdots,5000\}$の離散一様確率変数とした。Model 3-5、4-1、4-2 では、待ち行列システムの到着間隔は平均 1.25 分の指数分布、サービス時間は平均 1 分の指数分布とした。Model 4-3 と 4-4 では、サービス時間の分布を最小 0.25、最頻 1.00、最大 1.75 の三角分布に変更した。これは指数分布（最頻値が 0）よりも現実的なサービス時間の形といえるであろう。5 章のモデルでは、到着間隔や加工時間、検査時間、再加工時間に指数分布、三角分布、一様分布を用いた。本章のモデルでは、固定値ではなく不確実性を含むべきさまざまな入力データに対して、多くの確率分布の特定が必要である。

これについて、次の 2 つの疑問を持たれるだろう：

1. どのように入力分布を実際に特定するのか？
2. 入力分布を特定した後、シミュレーションを実行するために、どのようにその分布から値を「生成」するのか？

本章では両方について説明するが、モデル構築の一環として説明するため、大部分が最初の問題の説明となる。2 番目の問題は、幸いにも Simio のようなシミュレーションソフトウェア内では、機能として提供されている。それでも、シミュレーション実験を適切に設計し分析するためには、それらの仕組みの基本を理解する必要がある。本章（と本書全体）では、読者はすでに**確率統計**の基本を理解していると想定している。これには、以下の内容が含まれる：

・2 章の冒頭で箇条書きされたすべての確率のトピック。
・**ランダムサンプリング**と、平均、分散、標準偏差の**推定**。
・**独立同一分布**（IID）データの観測値や確率変数（RV）。
・平均、分散、標準偏差の推定値の**標本分布**。
・平均、分散、標準偏差の**点推定**と**不偏性**。
・平均や他のパラメータの**信頼区間**およびその解釈。
・平均や他のパラメータの**仮説検定**（本章では仮説検定の一種である**適合度検定**について簡単に説明する）。仮説検定の$\boldsymbol{p}$**値**（観測された有意水準）の概念とその解釈。

これらの概念が理解できない場合は、先に進む前に確率・統計の本で復習されることを強く勧める。すべての式や正規分布表を覚える必要はないが、これらの概念を理解しておく必要がある。マルコフ過程や他の確率過程、回帰分析、因果分析、パス解析、データマイニングなど、他にもさらに多くの確率・統計に関するトピックが存在する。しかし、本章で本当に必要なものは上のリストに挙げたものだけである。

6.1 節では、一変数だけ（**単一変量**）の入力確率分布の特定方法を説明する。

6.2 節では、相関のある変数や時間によって変化する到着率（例：**非定常ポアソン過程**）など、より複雑な入力データの種類を紹介する。6.3 節では、**乱数**（0 と 1 の間に一様に分布する連続観測値）の生成方法に移る。乱数は多くの人が考えるよりもはるかに厄介であるが、Simio には高品質の乱数ジェネレータが組み込まれている。6.4 節では、乱数を使って確率分布から**観測値**（あるいは**実現値**または**サンプル**）に変換する方法を説明する。最後に 6.5 節では、Simio の Input Parameters 機能と入力解析の活用法を解説する。

6.1　単一変量の入力確率分布の特定

この節では、シミュレーションの入力として使う単一変量確率変数の分布を特定する方法について、一般的な手順を説明する。6.1.1 項では全体的なテーマを説明し、6.1.2 項では実際のデータをどのように使用するかを説明する。6.1.3 項と 6.1.4 項では、候補となる分布の選び方とデータへの当てはめ方を解説する。6.1.5 項では、選んだ分布が本当にデータに適合しているかを確認する方法を詳しく説明する。最後に 6.1.6 項では、分布を特定する際によくある問題点を紹介する。

6.1.1　一般的アプローチ

ほとんどのシミュレーションでは、ランダムな値を表す確率分布を決める必要がある。たとえば、商品の需要、到着間隔、サービス時間、機械の稼働時間や修理時間、移動時間、バッチサイズなどさまざまなものがある。各入力には、**単一変量**確率分布（つまり 1 次元の RV）を自分で決める必要がある。通常、これらの入力分布はシミュレーションモデル内で互いに**独立**していると考える(6.2.2 項では、相関のある入力や複数の値を持つ入力について簡単に説明する)。

たとえば、多くの待ち行列シミュレーションでは、サービス時間の分布を決める必要がある。これには工場での機械の加工時間や救急室での患者の診察時間などが含まれる。通常、すでに集めた実際のデータを持っているか、シミュレーションのために新たにデータを集める。図 6.1 は、Excel ファイル Data_06_01.xls の一部を示している（付録 C で説明される方法で本書のウェブサイトからダウンロード可能)。A 列には 47 個のサービス時間が記録されている。これらのデータは実際のサービス時間の IID 観測値で、関心のある期間の定常的で代表的な部分から集められたものである。そして、以下に示す統計的な検定に「合格する」ような、これらのデータを適切に表す確率分布を見つける。シミュレーション実行時には、この分布から**確率変量**（サービス時間の RV を元にする観測値（サンプル、実現値)）を「生成」してシミュレーションを動かす。

6.1.2　観測された実データの使用方法

シミュレーションを実施する際、観測された実際のサービス時間をそのままシミュレーションに直接利用するという自然なアプローチではなく、確率分布を当てはめてそこから確率変数を生成するという間接的な方法をとることに疑問を感じることもあるだろう。この間接的なアプローチには、以下のような理由がある：

	A	B	C	D	E
1	34.2	実際のサービス時間に対する47の観測値			
2	28.4				
3	26.9				
4	23.5				
5	21.9				
6	21.5				
7	32.6				

図 6.1　データファイルにある 47 件のサービス時間観測値（分）

・後の章で説明するように、一般的には非常に長時間のシミュレーションを実行したいものである。また、出力データから統計的に有効かつ正確な結論を導くため、膨大な回数の IID **反復実行**を行うことがある。そのため、実世界の入力データはすぐに不足してしまう。

・図 6.1 と Data_06_01.xls の観測データは、データが収集された時点での状況のみを表している。別のときには異なるデータが観測され、特に異なる範囲（最小値と最大値）のデータが単なる代表例として得られることもある。シミュレーションの駆動に直接観測データを使用した場合、偶然得られた値だけに制限され、観測範囲外の値を生成できない。特にサービス時間をモデル化する際、たとえまれであっても、大きな値は待ち行列の混雑指標（システム平均時間や最大待ち行列長など）に大きな影響を及ぼすことがある。シミュレーションで使用するサービス時間の分布の右側を適切に「丸める」ことで、この影響をシミュレーション結果に反映させることができる。

・サンプルサイズが極めて大きくない限り、観測データには、現実には起こり得る値であっても、たまたま観測できなかった範囲のギャップが残る。そのような値は、直接データを使用したシミュレーションでは決して発生しない。

このように、確率分布を適合させるいくつかの理由がある。シミュレーションを駆動するために確率変数からサンプルを生成することが、直接観測された実世界のデータを使用するよりも優先されることが多い。観測された実データから直接シミュレーションを駆動させる状況としては、モデルの**妥当性の確認**が考えられる。シミュレーションモデルが実システムのパフォーマンスを正確に再現するかを確かめる場合である。しかし、この場合には、シミュレーション出力に対応する実システムからの出力データを**保有**している必要がある。ただし、データ収集活動が**入力**分布の特定に向けられているなら、出力に関する実データを持っていることはまれである。なお、確率分布の適合よりも観測データの集合からサンプリングするほうがよいとする最近の研究もある（Song et al. 2014、Song and Nelson 2015）。この話題と、Simio で観測データの集合からサンプリングする方法については、6.5 節で簡単に解説する。本書での主題は、**どの**確率分布が観測データを適切に表しているかを見つけ出すことである。

6.1.3　確率分布の選択

モデルへのランダムな入力をモデル化するために**選択**可能な確率分布は多数あり、その一部にはすでに精通しているだろう。一般的な連続分布には正規、指数、連続一様、三角分布があり、離散分布には離散一様、二項、ポアソン、幾何、そして負の二項分布がある。これらの分布に関する詳細は他の資料で容易に入手できるため、ここでは詳しく言及しない。Simio Reference Guide（Simio 内で F1 キーを押すか右上隅の?アイコンをクリック）では、約 20 の分布について基本情報を提供している（Simio モデルでこれらの分布を指定すると、Simio はそれらから確率変量を生成する）。Simio Reference Guide Contents→Modeling in Simio →Expression Editor, Functions and Distributions→Distributions と進み、参照したい分布を選択する。図 6.2 は左側にこのパスを示し、右側にガンマ分布のエントリを表示している。そこには、PDF のプロット（この例では連続分布なので滑らかな連続線）、この分布に良好に適合したデータのヒストグラム、および Simio での構文とパラメータなどの基本情報が上部に示されている。

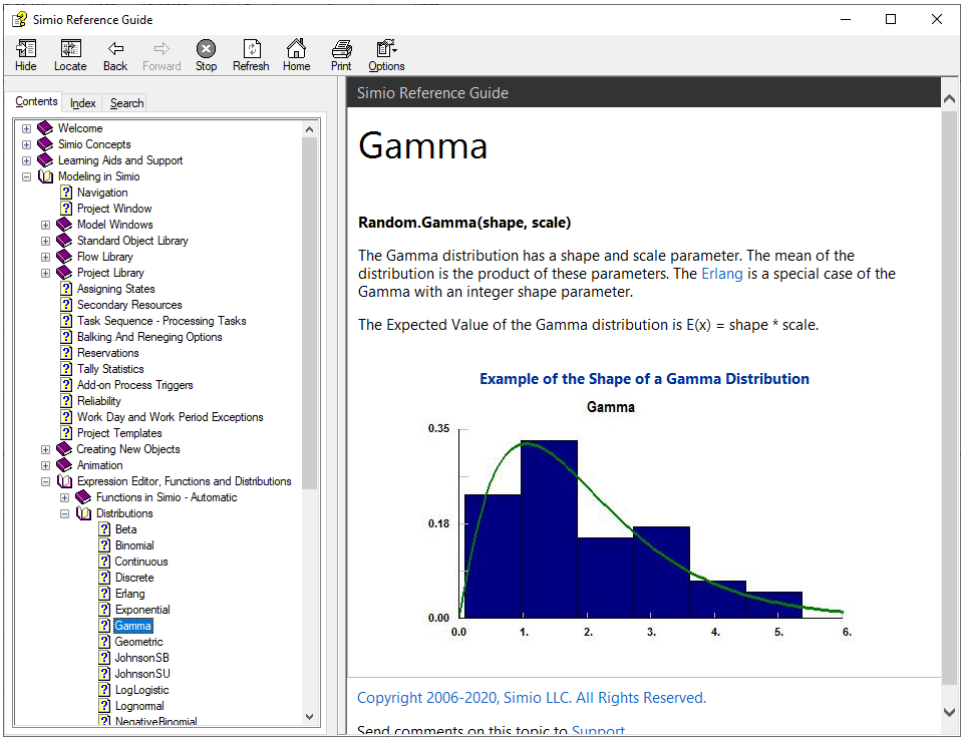


図 6.2　Simio Reference Guide におけるガンマ分布の説明

Selia et al.（2003）や Law（2015）などの文献には、シミュレーション入力モデリングに有用な確率分布に関する発展的な議論や章がある。さらに、複数巻の書籍（Johnson et al. 1994、1995、2005、Evans et al. 2000 など）では、多くの確率分布について詳細に記述されている。もちろん、多数のウェブサイトでも分布の概要が提供されている。検索エンジンで「probability distributions」を検索すると、6,900 万件以上の結果が表示される。https://en.

wikipedia.org/wiki/List_of_probability_distributions では、約 100 種の単変量分布へのリンクがあり、離散分布と連続分布の両方が範囲（または support（台））に基づいてカテゴリ分けされている。https://en.wikipedia.org/wiki/Gamma_distribution は、図 6.2 と同じガンマ分布を示している。分布のパラメータ指定は常に同じ様式とは限らないため注意が必要である。そのため、Simio での対応関係に気をつける必要がある。たとえば、Simio では指数分布は**平均**$\beta > 0$でパラメータ化されているが、平均速度λで発生するイベントに関連するポアソン過程の**割合**が$\lambda = 1/\beta$と表現される場合、Simio ではイベント間隔を平均$\beta = 1/\lambda$とする指数分布として指定することになる。

何百もの確率分布から、どのように **1 つ**を選択すればよいのか？最初に、**質的な**意味を確認する必要がある。それは、次のようなことを意味する：

- 入力量が本質的に離散的か連続的かを判断する必要がある。バッチサイズは連続的な確率変数としてモデル化すべきではなく（生成した数値をもっとも近い整数に丸める場合を除く）、持続時間は一般的に離散的な確率変数としてモデル化すべきではない。
- 確率変数がとりうる値の範囲に注意する。具体的には、右と左に有限か無限かどうか、そして特定の入力に対して適切かどうかである。前述したように、サービス時間を表す分布の右裾が有限か無限かは、待ち行列やシステム内の時間（または数）などの待ち行列モデルの結果に非常に大きな影響を与える。
- 上述の点に関連して、両方に、特に左側に無限の裾を持つ分布の使用には注意が必要である。もちろんこれには正規分布が含まれる（それを理解した上で使用するとしても）。その PDF は常に無限の左（および右）裾を持ち、ゼロの左側にまで伸びているため、負の値を生成する可能性がある。これは、サービス時間や他の持続時間の入力データでは意味をなさない。ただし、平均がゼロより 3 または 4 標準偏差以上離れている場合、負の値が実現する確率は「小さい」。基本的な統計の教科書では、それを無視できると書かれている。負の値を取得する実際の確率は、平均がゼロを 3 または 4 標準偏差上回る場合、それぞれ 0.00134990 と 0.00003167 である。しかし、**コンピュータシミュレーション**では、入力確率分布から数十万、さらには数百万もの確率変量を容易に生成できる。したがって、最終的に負の値が生成される可能性がある。そのような場合、シミュレーションで負の値をどう扱うかによって、望ましくない、あるいは誤った結果が生じるかもしれない（Simio はこの状況が発生すると、実行を停止し、エラーメッセージを表示して、実行内容が正しいかを確認する）。上記の 2 つの確率は、それぞれ約 741 分の 1 と 31,574 分の 1 であり、シミュレーションでじゅうぶん起こりうる範囲内である。もし形状的に正規分布を使用したいと考え（そして実データによく適合している場合）、他の選択肢があることも知っておくとよい。特にワイブル分布は正規分布に形状が似ているが、その PDF はゼロの左側に裾が**伸びていない**ため、負の値を**生成することはない**。

したがって、離散的か連続的か、そして範囲が適切かどうかを判断することが

最初のステップであり、これにより適切な分布を絞り込むことができる。

しかし、まだ選択可能な相当数の分布があるため、次にその形状（離散分布の PMF または連続分布の PDF）が観測された実データの**ヒストグラム**の形状におおよそ似ているものを探す必要がある。これは、ヒストグラムがデータの根底にある PMF または PDF の経験的なグラフィカルな推定となるためである。図 6.3 は、図 6.1 と Data_06_01.xls の 47 のサービス時間観測値のヒストグラムを示している（詳細は後述するが、分布適合ソフトウェアの Stat::Fit®で作成）。これはサービス時間であるため、連続分布を考慮し、（負の値の生成を回避するために）左側を有限とする必要がある。

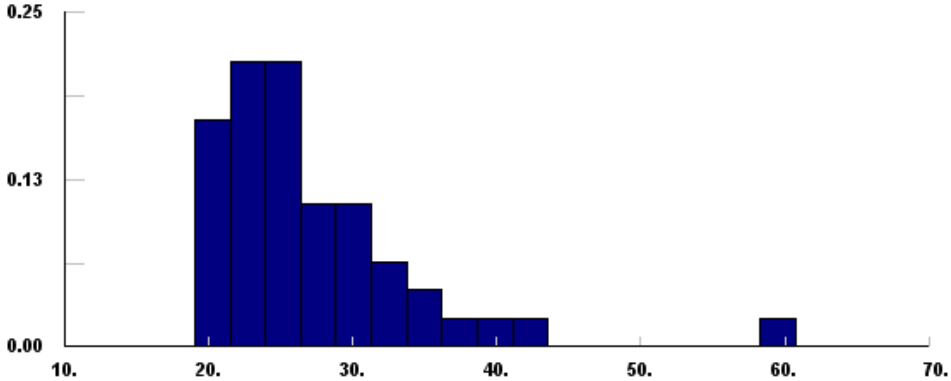


図 6.3　データファイルにある 47 件のサービス時間観測値（分）のヒストグラム

サービス時間がどれほど長くなりうるかという上限の情報がない場合、おそらく右側は無限となる。（上述の Simio Reference Guide や、Simio 内で F1 キーを押すか右上隅の?アイコンをクリックして）分布の PDF プロットのリストを確認すると、候補となりうるのは、アーラン、ガンマ（アーランの一般化）、対数正規、ピアソン VI、またはワイブル分布である。ただし、これらの分布にはそれぞれ推定が必要なパラメータ（ガンマとワイブルでは形状やスケールなど）がある。また、データから推定したパラメータを用いて、その分布が許容できる適合かどうか（つまり、データをじゅうぶんに表現できているか）を**検定**する必要がある。この推定と適合度検定の過程は、観測データに分布を**当てはめる**ことを意味する。

6.1.4　観測データへの確率分布の適合

このような分布の適合とそれに続く**適合度検定**のためのツールがすでに多く存在するため、Simio ではこの機能を提供していない。そうしたツールの一つが、Geer Mountain Software Corporation（www.geerms.com）の Stat::Fit®である。付録 C で説明される方法で本書ウェブサイトから無料の教科書バージョンをダウンロードできるので、その一部を紹介する。Stat::Fit を選んだもう一つの理由は、適切なパラメータと構文に合致した分布の仕様を Simio に単純なコピー・貼り付けでエクスポートできることである。Stat::Fit の完全なマニュアルは pdf ファイルとして教科書バージョンのダウンロードに付属しているため、ここでは Stat::Fit の機能すべてを説明せず、基本的な部分のみを解説する。他のパッケー

ジとしては、@RISK®（Lumivero Corporation、lumivero.com、旧 Palisade Corporation）や ExpertFit®（Averill M. Law & Associates、www.averill-law.com）に分布適合ツールが含まれている。

Stat::Fit のヘルプメニューは Windows ヘルプで作成されているため、Windows Vista 以降ではサポートされていない。Microsoft のウェブサイト support.microsoft.com/kb/917607 には、詳しい説明と、この問題を解決して Stat::Fit のヘルプシステムへのアクセスを可能にする WinHlp32.exe をダウンロードするためのリンクがある。また、このテキストの Web サイトからダウンロードできる Stat::Fit と同じ ZIP ファイルに pdf マニュアルも含まれている。

図 6.4 は、左側の Data Table にサービス時間データを貼り付けた Stat::Fit の画面である。Excel から直接、観測データをコピー・貼り付けでき、47 の値のうち最初の 19 が表示されているが、47 の全データがそこに存在し、Stat::Fit で使用される。

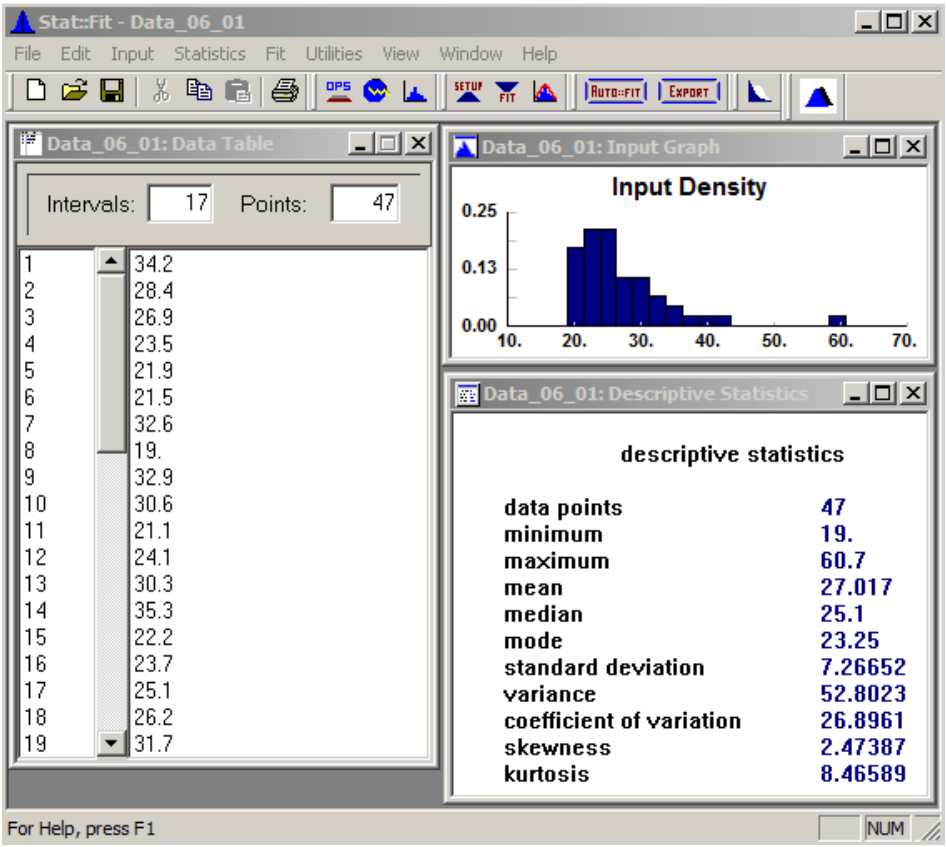


図 6.4　Stat::Fit のデータファイル、ヒストグラム、記述統計量（単位：分）

右上に表示されているのは図 6.3 のヒストグラムで、Input→Input Graph（またはツールバーの Graph Input ボタン）で作成できる。また、Input→Options、またはツールバーで OPS と書かれている Input Options ボタンから、間隔の数をデフォルトの 7 から 17 へ変更した。右下のウィンドウは観測データの基本的な記述統計量で、Statistics→Descriptive で生成される。

サポートされている分布の PMF および PDF を参照するには、メニューの Utilities→Distribution Viewer へ進み、ウィンドウ右上のプルダウンフィールドから選択する。教科書バージョンの Stat::Fit には 7 つの分布（離散の二項とポアソン分布、連続の指数、対数正規、正規、三角、連続一様分布）が含まれており、100 個のデータポイントに制限されている。一方、商用のフルバージョンでは 33 の分布がサポートされ、観測データのポイント数は 8,000 まで拡張できる。

サービス時間をモデル化するための連続分布が必要で、さらに負の値を生成する**わずかな可能性も**許容したくない場合、このリストから質的に適切な選択肢は、指数、対数正規、三角、一様分布である（これは無料の教科書バージョンからの選択であり、フルバージョンでは他にも可能性がある）。ヒストグラムの形状から一様分布は明らかに適合しないように見えるが、適合度の低い分布を適合させた場合の結果を示すためにリストに含めている。適合度検定を行うには、これら 4 つの分布を選択する必要がある。メニューの Fit→Setup、またはツールバーの Setup Calculations ボタン（SETUP と表示）をクリックして、Setup Calculations ウィンドウを起動する。Distributions タブ（図 6.5）では、左側の Distribution List で適合したい分布を 1 つずつクリックすると、右側の Distributions Selected リストに追加される。分布を削除したい場合は、右側のリストでその分布をクリックするだけでよい。

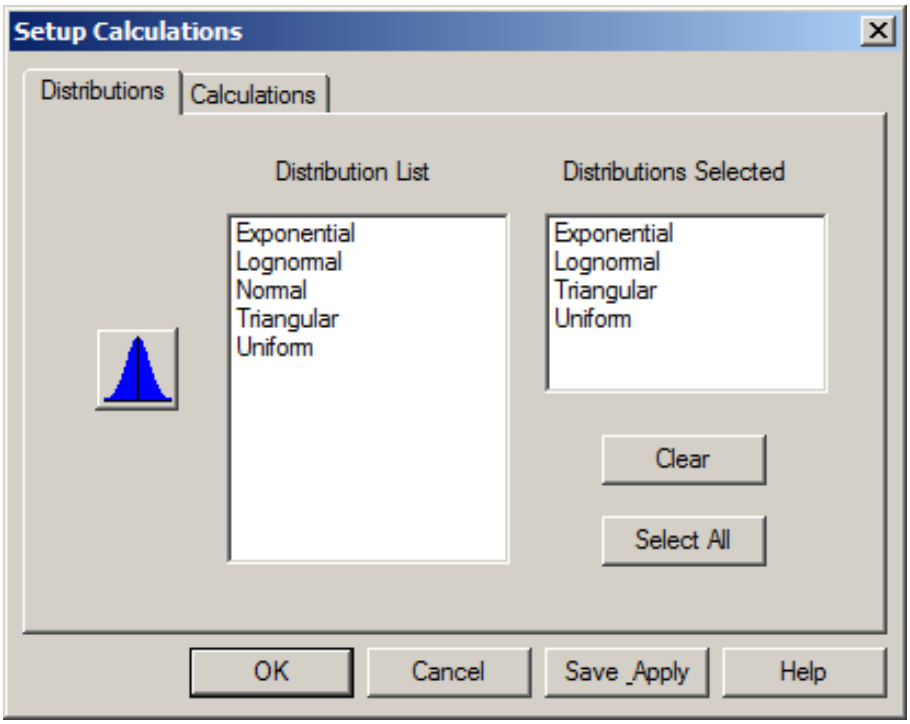


図 6.5　Stat::Fit の Setup Calculations ウィンドウの Distributions タブ

Calculations タブ（図 6.6）では、Estimates（推定方法）（MLE；Maximum Likelihood Estimation、つまり最尤推定を推奨）、分布の下限（Lower Bound）（unknown（未知）を選択して Stat::Fit にデータにもっとも適合するものを選択

させるか、または下限を強制する場合は fixed（固定）に下限を指定する）を選択する。また、Calculations タブでは、Tests（適合度検定）の種類（カイ二乗検定、コルモゴロフ-スミルノフ検定、アンダーソン-ダーリング検定）を選択できる。カイ二乗検定を選択した場合には、利用する間隔の種類（通常は、Equal Probability（等確率））を選択する。適合度検定については 6.1.5 項を参照してほしい。詳しくは Banks et al.（2005）または Law（2015）を参照のこと。

Lower Bound（下限）の仕様はもう少し説明が必要であろう。リストに挙げた指数や対数正規のような多くの分布は、それらの慣習的な下限として、0 とされることがよくある。しかし、データのヒストグラムに対して、よりよく合致させるために、PDF を左または右（サービス時間のように正のデータではたいてい右）へ移動させることにより、0 でない下限のデータに適合する可能性がある。ここで fixed（固定）を選択すると、分布の下端を指定することができる（その際、下のフィールドがアクティブになり、値を入力できるようになる。デフォルトは、データセットの最小値であり、0 とすれば慣習的な下限となる）。しかし、unknown（未知）を選択した場合、Stat::Fit にこの決定をさせるので、数値フィールドには入力できない。その数値は、よりよい適合を得るために、データセットの最小値よりもたいてい少しだけ小さくなる。すべての選択を保存して適用するには、Save Apply をクリックする。

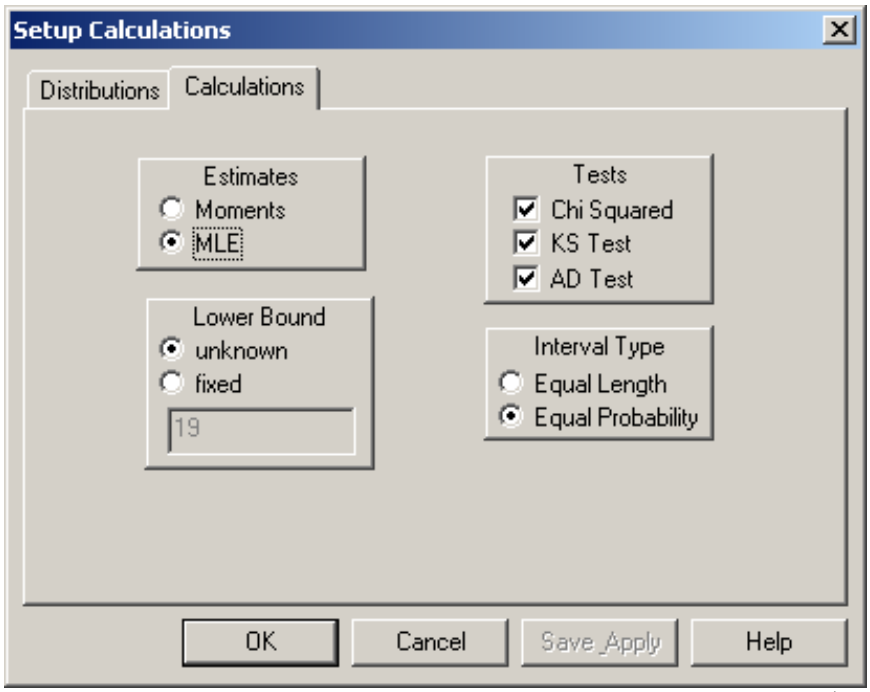


図 6.6　Stat::Fit の Setup Calculations ウィンドウの Calculations タブ

サービス時間データを表すために選択した 4 つの分布を適合させるためには、Fit→Goodness of Fit のメニューを選ぶか、またはツールバーにある Fit Data ボタン（FIT のラベル）をクリックすると、結果の詳細ウィンドウが表示される。図 6.7 がその最初の一部を示している。

Data_06_01: Goodness of Fit

goodness of fit

data points	47
estimates	maximum likelihood estimates
accuracy of fit	3.e-004
level of significance	5.e-002

summary

distribution	Chi Squared	Kolmogorov Smirnov	Anderson Darling
Exponential	4.89 (8)	0.116	1.28
Lognormal	0.298 (8)	3.05e-002	5.65e-002
Triangular	24.4 (8)	0.339	10.5
Uniform	67.7 (8)	0.539	30.

detail

Exponential

minimum =	19.
beta =	8.01702
Chi Squared	
total classes	17
interval type	equal probable
net bins	9
chi**2	4.89
degrees of freedom	8
alpha	5.e-002
chi**2(8,5.e-002)	15.5
p-value	0.769
result	DO NOT REJECT
Kolmogorov-Smirnov	
data points	47
ks stat	0.116
alpha	5.e-002
ks stat(47,5.e-002)	0.194
p-value	0.511
result	DO NOT REJECT
Anderson-Darling	
data points	46
ad stat	1.28
alpha	5.e-002
ad stat(5.e-002)	2.49
p-value	0.24
result	DO NOT REJECT

図 6.7　Stat::Fit の Goodness of Fit の結果ウィンドウ

検定の概要と、各分布に適用された 3 種類の検定結果が上部に示されている。これらすべての検定では、検定統計量の値が小さいほど適合度がよいことを示している（Chi Squared（カイ二乗検定）統計量の後にある括弧内の値は自由度を表す）。対数正規分布が、明らかにもっとも適しており、続いて、指数、三角となり、そして一様は最悪の値（もっとも大きい検定統計量）である。

さらに下の detail セクションで、順番にそれぞれの分布の適合の詳細を確認できる。図 6.7 は、指数分布のみを示しており、他の 3 つの分布の適合の詳細については、ウィンドウを下にスクロールしてみることができる。詳細レポートのも

っとも重要な部分は、各検定のp値である。指数の適合においては、カイ二乗検定は 0.769、コルモゴロフ-スミルノフ検定は 0.511、アンダーソン-ダーリング検定は 0.24 であり、それらのそれぞれの下に DO NOT REJECT（棄却されない）という結論が示されている。（任意の仮説検定のための）p値は、帰無仮説が真であれば、実際に得たサンプルよりも、代替仮説を採択してより多くのサンプルを得られる確率であるということを思い出してほしい。適合度検定では、帰無仮説は、候補の分布がじゅうぶんにデータへ適合することである。したがって、このように非常に大きいp値は（p値は確率であるので、常に 0 と 1 の間をとる）代替仮説を採択することがかなり容易になることを示している。換言すれば、データから得られる代替仮説をあまり採択しないので、指数分布によってじゅうぶんに適合する帰無仮説は、非常に合理的にみえる。

p値を見るもう 1 つの方法は、従来の仮説検定においてである。事前にα（通常は 0.01 から 0.10 の間）を第 1 種の誤り（誤って真の帰無仮説を棄却する）の確率として選択する。p値がαよりも小さい場合にのみ、帰無仮説を棄却する。指数分布の適合度に対して、3 つの検定すべてのp値はαの合理的な値を超えているので、指数分布が適合しているという帰無仮説を棄却することはない。Stat::Fit で、ウィンドウを下にスクロールすると、対数正規の 3 つの検定のp値がさらに大きいことがわかる。それらは 1 として表示されているが、実際には丸め誤差により 1 よりも少しだけ小さい。いずれにせよ、対数正規がよい適合を提供していることに反論する根拠はまったくない。しかし、三角分布や一様分布の適合の詳細が示されている部分まで下にスクロールすると、p値が極めて小さく、これらの分布はデータへの適合がよくないことがわかる（一様分布の場合はすでにヒストグラムから明らかであったが、三角分布についてはそれほど明確ではなかった）。

どの分布がデータに適合しているか、そしてどの分布がそうでないのかについて簡単なサマリを得るには、Fit→Auto::Fit、またはツールバーの Auto::Fit ボタンをクリックする。Auto::Fit ダイアログが最初に表示されるので、サービス時間に対しては（無限の左裾は必要ないため）、continuous distributions（連続分布）と lower bound（下限）ボタンを選択する。そして、OK を押すと、図 6.8 の Automatic Fitting の結果ウィンドウが表示される。

Data_06_01: Automatic Fitting

Auto::Fit of Distributions

distribution	rank	acceptance
Lognormal(17.4, 2.04, 0.672)	100	do not reject
Exponential(19., 8.02)	12.2	do not reject
Triangular(19., 61.7, 19.)	0.	reject
Uniform(19., 60.7)	0.	reject

図 6.8　Stat::Fit の Auto::Fit 結果

これは、p値は表示されないが、適合の帰無仮説について、各分布の acceptance（受け入れるか否か）の結論と、さらに適合した分布のパラメータを与える

（Stat::Fit のパラメータの表記規則は、普遍的に合意されたものではないため、すべてのケースで Simio の規則と同一とは限らない）。rank の列は、Stat:: Fit によって与えられた内部スコアであり、数値が大きいほど、より適合していることを示している（詳細レポートのp値の順序と一致しており、対数正規が最適である）。

図 6.9 に示すように、ヒストグラム上に適合した密度関数をグラフィカルに比較するには、Fit→Result Graphs→Density、またはツールバー上の Graph Fit ボタンによって行う。右上にある分布上でクリックすると、（下部にある凡例ごとに異なる色で）それらすべてを重ね合わせることができる。右下のウィンドウでクリックすると、個別の分布を表示から除外できる。これは、適合度検定を**図で確認する**機能を提供している（サービス時間に対して、一様分布がいかに不適切か、また三角分布もあまりよい適合ではないことが視覚的に確認できる）。

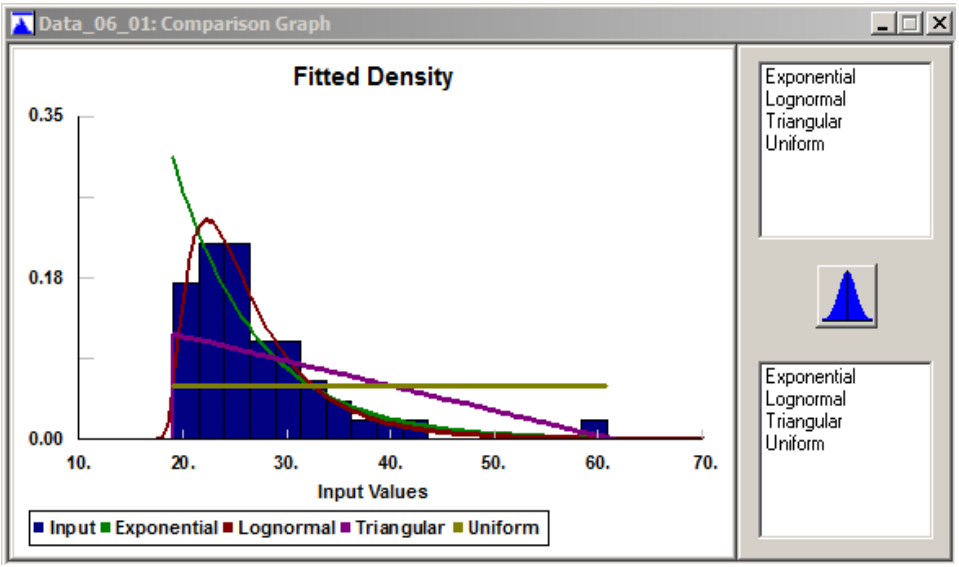


図 6.9　Stat::Fit のデータヒストグラム上に重ね合わせた適合分布の密度関数

最後のステップは、結果を適切な構文に変換し、コピーして直接 Simio の expression フィールドに貼り付けることである。File→Export→Export fit、または Export ツールバーボタンを選択すると、図 6.10 に示す EXPORT FIT ダイアログが表示される。

左上のドロップダウンには、いくつかのシミュレーションモデリングパッケージのリストがあり、Simio もそこに含まれている。右上のドロップダウンは、検定に適合した分布を示している。利用したい分布（ここでは対数正規（Lognormal））を選択し、そのすぐ下のクリップボード（Clipboard）のラジオボタンを選択する。すると、パネルの下に、Simio で有効な式として 17.4 + Random.Lognormal(2.04, 0.672)が表示される。この式は、Windows のクリップボードにコピーされているため、Simio アプリケーションに移動し、確率分布を式のフィールドへ貼り付ける（CTRL-V）だけで設定が完了する。Stat::Fit では、データによりよく適合させるために、対数正規分布で習慣的に使われる 0 の下端ではなく、17.4 だけ対数正規を上に（右に）シフトすることを推奨している。このシフトに用いる 17.4 という

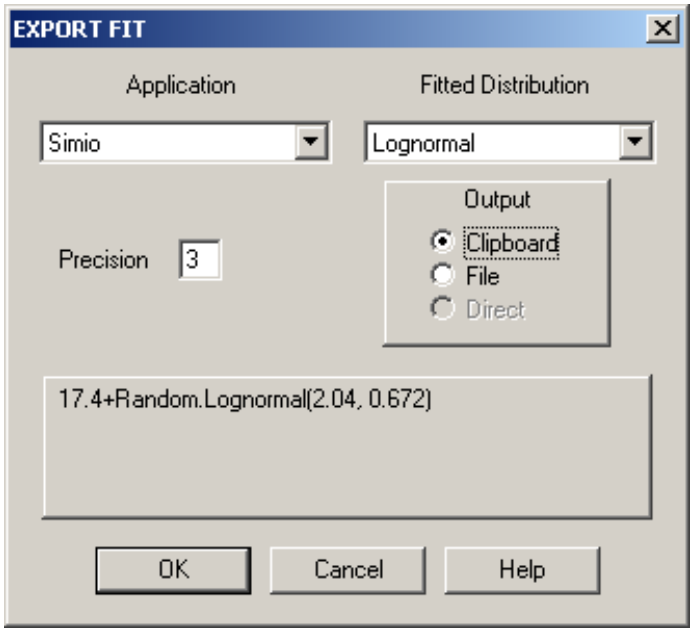


図 6.10　Stat::Fit の EXPORT FIT ダイアログ

値は、データセットの最小値 19.0 よりわずかに小さい値である。そのため、適合した対数正規の確率密度関数は、最小データ値のわずかに左が 0 となる（したがって、変数を生成するとき、最小の可能性がある値は 17.4 となる）。

データ、分布の選択、結果、そしてグラフを含めたすべての Stat::Tools での作業を保存することができる。いつものように、File→Save または Save　As を選択するか、Save ボタンをクリックする。デフォルトのファイル名の拡張子は、.sfp（Stat::Fit プロジェクト）である。この例では、ファイル名を Data_06_01.sfp とした。

6.1.5　適合度検定の詳細

6.1.4 項で、Stat::Fit を用いて分布をデータに適合し、候補となる分布が観測データをどの程度よく表現できているかを検定する方法について概要を示した。適合度を評価する方法は多様であり、本書ですべてを扱うことはできない。詳細は、Banks et al.（2005）や Law（2015）を参照してほしい。本稿では、これまでに述べた主要な方法の背後にある、正式な仮説検定や非公式な図によるヒューリスティクスなどの理論をいくつか説明する。ただし、アンダーソン-ダーリング検定は扱わないため、Banks et al.（2005）あるいは Law（2015）を参考にしてほしい。

6.1.5.1　カイ二乗適合度検定

図 6.9 では、当てはまりの程度を判断するために、ヒストグラム（青棒）と適合された分布の PDF（下方の曲線群）との一致をもって良好（少なくとも合理的である）としていた。この議論では、連続分布を前提としている。これはいくぶん直感的であるが、そこには数学的裏づけもある。

ここで、nを観測された実データの数（この例では$n = 47$）とし、kを横軸方向

にヒストグラムを作成する際の区間数（この例では$k = 17$）とする。以下では、ヒストグラム内のすべての区間の幅をwで一定とするが、後述するようにカイ二乗検定や他の多くのケースで、その前提は実際には必要ない。さて、区間kの左端を$x_0, x_1, x_2, \cdots, x_{k-1}$とし、$k$番目の区間の右端を$x_k$とすると、$j = 1,2,\cdots,k$について、区間$j$を$[x_{j-1}, x_j)$と表現できる。$n$個の観測値のうち$O_j$が区間$j$に分類される場合（これをその区間におけるデータの**観測度数**という）、O_j/nを区間jに分類されるデータの**出現率**という。

ここで、適合を検討中の分布の PDF を$\hat{f}(x)$とする。これがデータを表現する正しい PDF ならば、観測サンプルのデータ点がj番目の区間$[x_{j-1}, x_j)$に入る確率は

$$p_j = \int_{x_j-1}^{x_j} \hat{f}(x)dx$$

であり、これは観測出現率O_j/nにほぼ等しくなる。したがって、この分布が実際に良好な適合を示す場合、すべての区間jについて$O_j/n \approx p_j$を期待できる。これにnをかければ、すべての区間jについて$O_j \approx np_j$となる。つまり、各区間の**観測度数**および**期待度数**（$\hat{f}$がデータに対して真に正しい密度関数の場合）は、互いに「近い」値になるはずである。そのため、多くの区間で実質的に一致しない場合は、不十分な適合として疑うべきである。これは、実際には先ほど「見た目で」検定したときに行ったことである。

しかし、本当に良好な適合であっても、ランダムサンプリングの自然な変動により、すべての区間jについて（正確に）$O_j = np_j$となることは期待できない。この概念を形式化するため、次のカイ二乗検定統計量を定式化する：

$$\chi^2_{k-1} = \sum_{j=1}^{k} \frac{(O_j - np_j)^2}{np_j}$$

そして、観測データが適合分布$\hat{f}$に従うという帰無仮説H_0の下で、χ^2_{k-1}は（より正確には**漸近**的に$n \to \infty$のとき）自由度$k-1$のカイ二乗分布となる。ここで、検定統計量χ^2_{k-1}の値が「大きすぎる」（つまり「**膨張**する」）場合、適合度の帰無仮説H_0を棄却する。「大きすぎる」の判断基準はカイ二乗分布表に基づく。すなわち、有意水準αで$\chi^2_{k-1} > \chi^2_{k-1,1-\alpha}$ならば$H_0$を棄却する。ここで、$\chi^2_{k-1,1-\alpha}$は確率$\alpha$で自由度$k-1$のカイ二乗分布に含まれる臨界値である。もう一つの結論を導く方法は、p値による検定である。ここで、p値は自由度$k-1$のカイ二乗分布に検定統計量χ^2_{k-1}が含まれる確率である。一般にp値を用いる場合、p値が有意水準αよりも小さければH_0を棄却する。

検定統計量の値χ^2_{k-1}、そして検定結果も、区間の選び方に依存することに留意すべきである。この区間をどう選ぶべきかは多くの研究がなされているが、一般的に受け入れられる「正しい」答えは存在しない。共通点があるとすれば、次の２点である。（1）区間は**等確率**で選ぶべきである。つまり、積分の確率値p_jが互い

に等しく（または少なくともほぼ等しく）なるように選ぶ。（2）すべての区間jについて、np_jを少なくとも（約）5 とする（この条件は基本的に、サンプルサイズが極めて小さい場合、カイ二乗適合度検定の適用を制限する）。等確率の区間の端点を見つける一つの方法は、$x_j = \hat{F}^{-1}(j/k)$とすることである。$\hat{F}$は適合分布の CDF であり、上つきの-1は逆関数を表す（算術的な逆数ではない）。これはx_jについて$\hat{F}(x_j) = j/k$として解くことができるが、$\hat{F}$の形状によっては、割線法やニュートン法などの数値近似による求根アルゴリズムが必要になるだろう。

カイ二乗適合度検定は、（バッチサイズのような）値が離散的なデータに離散確率分布を適合する場合にも適用可能である。上記の議論では、p_jを、j番目の区間に含まれる適合 PMF 値の**和**に置き換えるだけで、手順は同様である。離散分布では、一般に区間の選択にいて正確な等確率性を達成できないことに注意すべきである。

6.1.5.2　コルモゴロフ-スミルノフ適合度検定

カイ二乗検定が適合 PDF（離散の場合は PMF）と**経験** PDF または PMF（ヒストグラム）を比較するのに対し、コルモゴロフ-スミルノフ（K-S）検定は、適合 CDF をデータから直接定義される**経験** CDF と比較する。経験 CDF を定義するにはいくつかの方法があるが、ここでの目的には、$F_{\mathrm{emp}}(x)$=（すべてのxに対して$\leq x$である観測データの割合）を用いる。これは有限個の階段関数であり、（順序化された）観測データ値の各々は高さ$1/n$のステップで発生する。前と同様に、$\hat{F}(x)$を特定の適合分布の CDF とする。K-S 検定統計量は、xの取り得るすべての値について、$F_{\mathrm{emp}}(x)$と$\hat{F}(x)$との間の垂直方向の乖離の最大値である。数学的に表現すると、次のようになる：

$$V_n = \sup_x \left| F_{\mathrm{emp}}(x) - \hat{F}(x) \right|$$

ここで「sup」は**上限**であり、すべてのxの値についての最小上界である。max（最大値）演算子を用いないのは、最大の縦の乖離が$F_{\mathrm{emp}}(x)$の「ジャンプ」の前に起こる可能性があるためである。その場合、上限は特定のxの値について正確に到達できない。

K-S 検定統計量V_nを評価する有限の（すなわち、計算可能な）アルゴリズムは Law（2015）に示されている。$X_1, X_2, \cdots, X_n$が観測サンプルを表すものとし、$i = 1,2,\cdots,n$について、$X_{(i)}$をデータの中でi番目に小さな値としよう（したがって、$X_{(1)}$が最小の観測値であり、$X_{(n)}$が最大の観測値となる）。$X_{(i)}$は観測データの**i番目の順序統計量**と呼ばれる。以上より、K-S 検定統計量は次式で表される：

$$V_n = \max\left\{ \max_{i=1,2,\cdots,n} \left[\frac{i}{n} - \hat{F}\left(X_{(i)}\right) \right], \max_{i=1,2,\cdots,n} \left[\hat{F}\left(X_{(i)}\right) - \frac{i-1}{n} \right] \right\}$$

直感的には、V_nが大きいほど、適合はよくない。どの程度の大きさが「大きすぎる」のかを決めるためには、有意水準 α における検定の臨界値を求める表や参照分布が必要であり、あるいは検定のp値を決定する必要がある。K-S 検定の欠点は、カイ二乗検定とは異なり、異なる仮説分布や異なるサンプルサイズnに対して、

別々の表（あるいは参照分布）が必要とされる点である（これは K-S 検定統計量 V_nにnを含めた理由である）。さらに詳しくは Gleser（1985）や Law（2015）を参照してほしい。Stat::Fit のような分布適合パッケージは、K-S 検定のp値を提供する参照分布を機能として備えている。カイ二乗検定に対する K-S 検定の利点は、恣意的な区間の選択に依存しない点と、小さなサンプルサイズnに対しても正確である点である（$n \to \infty$の場合だけではない）。

Stat::Fit では、メニューで Fit→Result Graphs→Distribution と選択することで、図 6.11 のプロットが作成される。青色の曲線が経験 CDF であり、さまざまな適合分布の CDF が下部の凡例に示された色で表示されている。適合分布の追加・削除は図 6.9 の場合と同様に行える。K-S 検定統計量は図 6.11 に表示されていないが（各適合分布については、経験 CDF と適合分布間の最大の乖離を、縦軸の長さから判断するしかない）、一様分布および三角分布の CDF は乖離が大きいため経験 CDF との適合度が低く、対数正規分布は良好に適合していることが見てとれる（実際、図 6.11 では、対数正規分布は経験分布を見分けるのが難しいほど重なり合っている）。

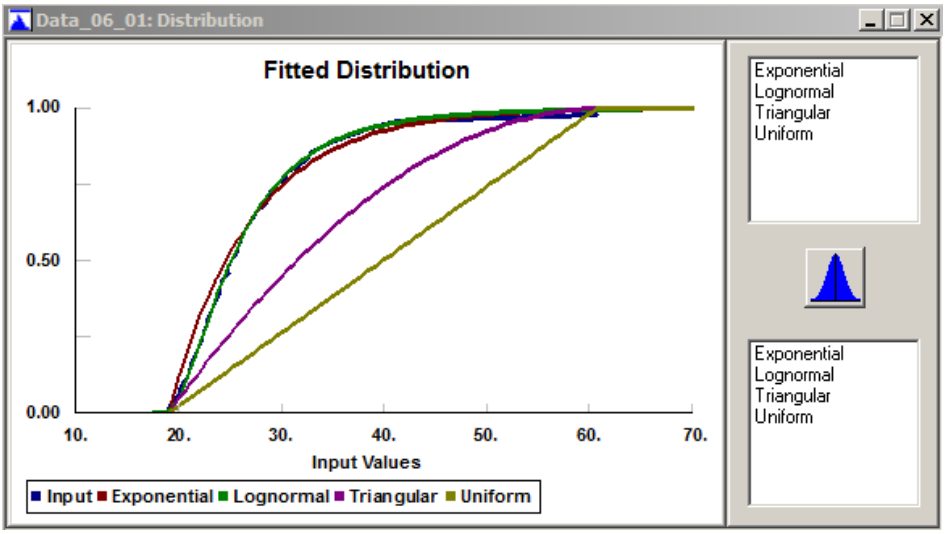


図 6.11　Stat::Fit による経験分布への適合 CDF の重ね合わせ

6.1.5.3　P-P プロット

ここで、適合分布の CDF を表す$\hat{F}$が、観測データに内在する未知で真の CDF に良好な適合を示すものとしよう。この場合、各$i = 1,2,\cdots,n$について、$\hat{F}(X_{(i)})$は、$X_{(i)}$に重なる、あるいは含まれるデータ点の経験出現率に近くなっているはずである。その出現率はi/nであるが、計算上の便宜のために、0 および 1 を出現率から除外する。そのため、少し調整して、この経験出現率の代わりに$(i-1)/n$を用いる。$\hat{F}$がデータに対して実際に良好な適合を示すかどうかを（正式な仮説検定ではなく）ヒューリスティックに確認するために、$i = 1,2,\cdots,n$について点$\left((i-1)/n, \hat{F}(X_{(i)})\right)$をプロットし、$i = 1,2,\cdots,n$について$(i-1)/n \approx \hat{F}(X_{(i)})$であるかを調べる。実際に良好な適合であれば、これらの点はプロット上で$(0,0)$から$(1,1)$への対角線

上に並ぶ。これらの点のx軸およびy軸は確率（それぞれ経験および適合）であるため、このプロットは**確率プロット**または **P-P プロット**と呼ばれる。

Stat::Fit では、P-P プロットはメニューの Fit→Result Graphs→PP Plot を選択するだけでよい。図 6.12 は、47 個のデータに対して 4 つの適合分布を試した P-P プロットである。図 6.9 および図 6.11 と同様に、右上のボックス内をクリックすることで適合分布を追加でき、右下のボックスでは選んだ分布を削除できる。対数正規分布の P-P プロットは対角線に極めて近く、優れた適合を示している。一方、三角分布および一様分布の P-P プロットは対角線から大きく離れており、適合しているとは言い難い。指数分布も完全に不適切とまではいえないものの、対数正規分布ほど良好な適合は示していない。

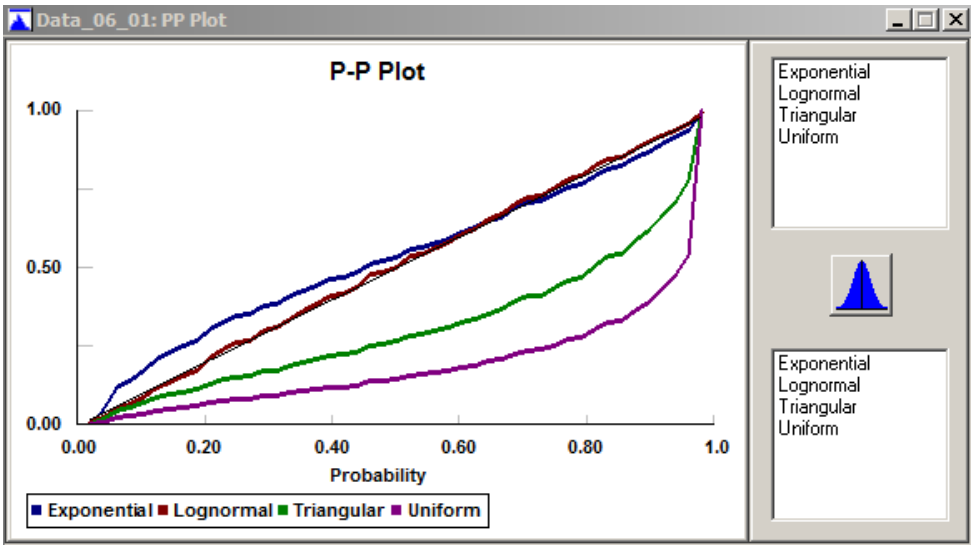


図 6.12　Stat::Fit の P-P プロット

6.1.5.4　Q-Q プロット

P-P プロットは、$i = 1,2,\cdots,n$において、$(i-1)/n \approx \hat{F}(X_{(i)})$か否かを確かめるものである。この近似式に対して$\hat{F}^{-1}$（適合分布の CDF を表す$\hat{F}$の逆関数）を適用すると、$i = 1,2,\cdots,n$について$\hat{F}^{-1}((i-1)/n) \approx X_{(i)}$が得られ、これは P-P プロットと対称的な関係にある。ここで、左辺は適合分布の**分位数**である（任意の確率または出現率（この場合、$(i-1)/n$）以下となる値）。なお、$\hat{F}^{-1}$について閉じた形の表現が利用できない場合は、数値近似を行う必要がある。

$\hat{F}$がデータに対して良好な適合を示すならば、$i = 1,2,\cdots,n$について点$\left(\hat{F}^{-1}((i-1)/n), X_{(i)}\right)$をプロットすると、おおよそ対角線が現れるはずである。ただし、今度は$(0,0)$と$(1,1)$ではなく、$\left(X_{(1)}, X_{(1)}\right)$と$\left(X_{(n)}, X_{(n)}\right)$間の対角線となる（実際には、Stat::Fit は順序を逆にして、点$\left(X_{(i)}, \hat{F}^{-1}((i-1)/n)\right)$をプロットするが、直線が現れるという結果は変わらない）。これらの点のx軸およびy軸はともに分位数（それぞれ経験分位数と適合分位数）であるため、このプロットは**分位数プロット**または **Q-Q プロット**と呼ばれる。

Stat::Fit のメニューで Fit→Result Graphs→QQ Plot を選ぶと Q-Q プロッ

トを作成できる。この例のデータおよび適合分布について、図 6.13 に示す。図 6.9、6.11 および 6.12 と同様に、どの適合分布を表示するかを選択できる。対数正規と指数分布の Q-Q プロットは、右裾を除いて対角線に極めて近く、良好な適合を示している。一方、三角分布と一様分布の Q-Q プロットは、これまでの結果と同様に、非常に当てはまりが悪いことを示している。Law (2015) によれば、P-P プロットは分布の中央部において乖離がより強く現れるのに対して、Q-Q プロットは分布の両裾でデータと適合分布との乖離が大きくなる傾向にある。

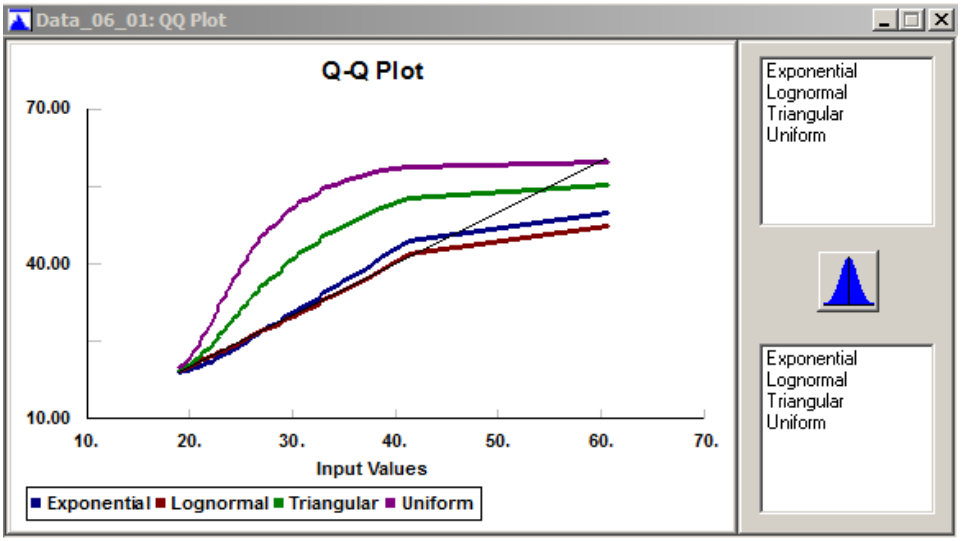


図 6.13　Stat::Fit の Q-Q プロット

6.1.6　分布の適合に関する問題

この項では、観測データに分布を適合させる際に頻繁に生じる問題や疑問について簡潔に述べる。

6.1.6.1　データが独立同分布か否か、そうでない場合の対処法

6.1.3 項から 6.1.5 項で議論されたすべての分布適合手法および適合度検定の根底にある基本的かつ重要な仮定は、モデル入力に対して観測される実世界のデータが IID（独立同分布；Independent and Identically Distributed）であることである。論理的観点からみると、この仮定は次の 2 つの要素から構成されており、6.1.3 項から 6.1.5 項で扱った手法が妥当であるためには、いずれの要素も満たされなければならない：

- 独立性（I）：各データ点が、データセット内の別のデータ点と、確率的・統計的に独立していること。この仮定は、データが経時的に任意のプロセスから連続的に収集される場合に疑わしくなる。なぜなら、ある観測値が次あるいは後の観測値と何らかの関連性を持ったり、「物理的な」因果関係や明らかに統計的な相関関係を持つ可能性があるからである。
- 同一分布（ID）：データを生じさせる確率分布または確率過程が、各データ点に関して同一であること。この仮定は、データの基となる分布に影響を与える

ような形でデータ収集中に条件が変化する場合や、観測値のソースの観点から、データセット内に異質性が認められる場合に疑わしくなる。

モデル化されるシステムや、データ収集の方法についての文脈を理解しておくことは常に重要である。独立同分布の「独立性」と「同一分布」の双方について、正式な統計的検定は存在するが、ここでは、これらの仮定を非公式に確認するための簡単なグラフィカルな方法を紹介し、いずれかの仮定が満たされない場合の方針を述べたい。

これまでと同様に、$X_1, X_2, \cdots, X_n$が観測値を表すが、添字iを観測値の時間順とする。すなわち、X_1は時間的に 1 番目の観測値、X_2は 2 番目の観測値である。IID 性を評価する簡単な第一歩は、縦軸にX_iをとり、横軸にiをとったデータの**時系列プロット**を作成することである。図 6.14 に 6.1.1 項から 6.1.5 項で対象とした 47 のサービス時間の時系列プロットを示す。このプロットは Excel スプレッドシート Data_06_02.xls に含まれている。Data_06_01.xls にあるオリジナルデータが列 B に再録されており、列 A に観測順iが追加されている（他の列はここでは無視してほしい）。IID のデータであれば、このプロットはまとまりのない雲のような形状になるはずで、持続的に上昇もしくは下降するような傾向や周期性（これが見られる場合は、「同一分布」が侵害されている）や、連続する点の間に強い体系的な関連性（これが見られる場合は、「独立性」が侵害されている）がないはずである。このデータにおいては、後者の問題が生じているように思われる。

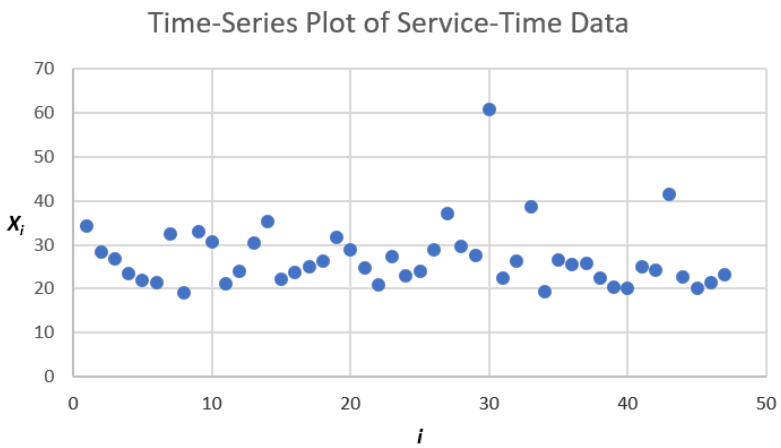


図 6.14　47 のサービス時間の時系列プロット

もう一つの簡単なプロットとして、特にデータ内の隣接する観測値（時系列データの**ラグ 1**（遅れ 1 または lag-1）と呼ぶ）間の非独立性を検出するための**散布図**がある。図 6.15 に 47 のサービス時間について、$i = 1, \cdots, n-1$の隣接するペア(X_i, X_{i+1})に対する散布図を示す。このプロットも Data_06_02.xls に含まれているが、Excel でこの図を作成するために、$i = 1, \cdots, n-1$ におけるX_{i+1}のデータとして新しく列 C を設けた。ラグ 1 に正の**相関**があれば、右上がりの直線の周りに点が集まり（負の相関の場合は、右下がりの直線の周辺に点が集まる）、独立したデータならば（図 6.15 でみられるように）そのような集まりは形成されな

い。なお、Stat::Fit ではメニューの Statistics→Independence→Scatter Plot で、ラグ 1 の散布図を作ることができる。$k > 1$のラグにおける非独立性を検討したい場合も、同様に$i = 1, \cdots, n-k$のペア(X_i, X_{i+k})に対する散布図を描けばよい（ただし Stat::Fit では作成できないため、Excel で工夫する必要がある）。

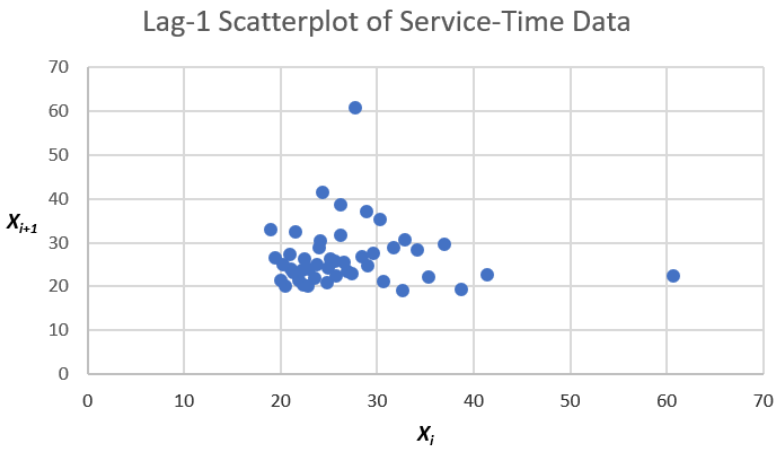


図 6.15　47 のサービス時間の散布図（ラグ 1）

時系列プロットおよび散布図は、独立性の欠如について直感的に可視化できるが、数値的な尺度もある。おそらくもっとも明らかな尺度は、さまざまなラグの数値的な**自己相関**（同じデータ系列内にあるため**自己**相関と呼ぶ）を推定する**相関行列**を作成することだろう。多くの標準的な統計分析パッケージではこれを自動的に行えるが、Excel でも作成できる。図 6.16 に、Data_06_02.xls の列 A～E を示す（行 9～43 は省略）。図の列 C、D、E は、列 B のデータをそれぞれ 1、2、3 行だけシフトしたもので、観測時間順にラグを表現している。

	A	B	C	D	E
1	i	X_i	X_{i+1}	X_{i+2}	X_{i+3}
2	1	34.2	28.4	26.9	23.5
3	2	28.4	26.9	23.5	21.9
4	3	26.9	23.5	21.9	21.5
5	4	23.5	21.9	21.5	32.6
6	5	21.9	21.5	32.6	19.0
7	6	21.5	32.6	19.0	32.9
8	7	32.6	19.0	32.9	30.6
44	43	41.4	22.8	20.0	21.3
45	44	22.8	20.0	21.3	23.2
46	45	20.0	21.3	23.2	
47	46	21.3	23.2		
48	47	23.2			

図 6.16　自己相関行列のためのデータ列の配置

自己相関行列を作成するには、Excel の**データ分析**パッケージを利用する。これは[**データ**]タブの[**分析**]から起動できる([データ分析]アイコンが表示されていない場合、分析ツールを導入するために、まず[**ファイル**]から[**オプション**]を選び、[**Excel のオプション**]ウィンドウを開く。それから、[**アドイン**]を選択し、[**Excel アドイン**]を[**設定**]する必要がある)。データ分析ウィンドウでは、[**相関**]を選んで[**OK**]を押す。次に入力範囲として$B$1:$E$45（青色のセル。ただし、ラグ 3 の自己相関を計算するため、最後の 3 行は除外している）を選択し、[**先頭行をラベルとして使用**]をチェックする。そして結果を表示したい場所を選ぶ(この例では、[**出力先**]を$G$37 とした)。図 6.17 に結果を示す。

Autoorrelation Matrix Though Lag 3				
	X_i	X_{i+1}	X_{i+2}	X_{i+3}
X_i	1			
X_{i+1}	-0.07	1		
X_{i+2}	-0.05	-0.06	1	
X_{i+3}	0.24	-0.04	-0.05	1

図 6.17　データファイルの自己相関行列

ラグ 1 の自己相関について 3 つの推定量があり、(左上から右下への対角線上に）それぞれ−0.07、−0.06、−0.05 である。これらは、ほとんど同じデータ（各系列の最後の数個のデータが異なるのみ）を用いているため互いに非常に近い値であり、いずれも極めて小さい（相関は常に−1 から＋1 の間の値をとる)。同様に、ラグ 2 の自己相関については 2 つの推定量があり、−0.05 および−0.04 である。こちらも互いに極めて近く、非常に小さい値である。ラグ 3 の自己相関に対する 1 つの推定量は＋0.24 である。これは、3 つ離れた観測値間に何らかの正の関係が存在することを示しているが、いずれにせよ弱い関係性である(なお、これらの自己相関の推定量が統計的に有意に 0 から離れているかについて、ここでは議論しない)。自己相関の可視化は、横軸にラグ（1、2、3、...)、縦軸に対応する相関係数をとったグラフで行われる（**コレログラム**と呼ぶ。ここでは 1 つのデータ系列を用いるため**オートコレログラム**とも呼ぶ)。Stat::Fit ではメニューの Statistics→Independence→Autocorrelation でこのグラフを作成でき、図 6.18 に示すとおり、自己相関行列と同様の傾向を表している。

また、データセット内の**独立性**に関して、多くの正式な統計的仮説検定があり、その中にはさまざまな**連検定**がある。この検定は、データの**連**におけるデータ順序(上昇連または下降連)について、ある連の長さおよび頻度の期待値を計算し、それがデータ内の実際の連の長さおよび頻度と一致するかを検討するものである。Stat::Fit では 2 種類の連検定（中央値および転換点を上回る・下回る）をサポートしており、メニューの Statistics→Independence→Runs Test から利用できる（読者自身で確かめてほしい)。

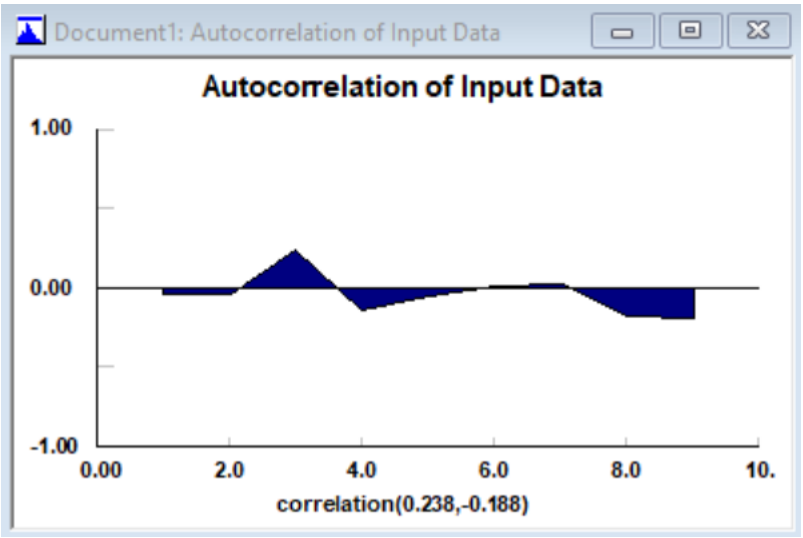


図 6.18　Data_06_02.xls の Stat::Fit によるコレログラム

ここで検討中の 47 のサービス時間に関する独立性の仮定は、反証する強いエビデンスがないため、少なくとも妥当といえる。データの独立性を明確に示す証拠がない場合、モデリングを開始するか、さらなるデータ収集を行うかは、モデル作成者の判断に委ねられる。シミュレーションにおける重要な特殊ケースとして、**時間によって変動する到着**パターン（6.2.3 項で扱うテーマ）があり、Simio ではこれを効果的にモデリングできる。しかし一般に、シミュレーションで非独立の入力過程をモデリングすることは困難であり、本書の範囲を超える。興味のある読者は、時系列に関する書籍（たとえば Box et al.（1994））を参照されたい。そこでは、さまざまな**自己回帰過程**の適合方法が扱われているが、多くのシミュレーションソフトウェアではこれらをサポートしていない。

これまで「I」（独立性）の仮定に焦点を当ててきたので、ここからは「ID」（同一分布）の仮定を考察する。この仮定から外れるのは、異なる訓練レベルやスキル水準を持つ複数のオペレータから得られたサービス時間のような不均質なデータセットが原因となる。このようなケースでは、すべてのサービス時間を一つにまとめたヒストグラムに、複数の明確なモード（ピーク）が現れることがある。対処法としては、どのオペレータからのサービス時間かを追跡し、異なる確率分布を指定した別々のサーバリソースとしてシミュレーションモデルを構築することが挙げられる。そして、訓練やスキルに応じたサービス時間に対して、適切な確率分布を個別に適用する。別の例として、病院の救急部門のシミュレーションがある。ここでは患者がそれぞれ異なる症状の**緊急度**（重症度）で到着し、サービス時間は緊急度に依存するため、異なる緊急度カテゴリには別々のサービス時間分布が必要となる。また、確率分布が時間とともに変化する場合も「ID」の仮定から外れる。この場合は、**変化点**を（システムの物理的特性やデータ自体から）特定し、異なるデータ期間に対して個別の確率分布を用いるようシミュレーションモデルを修正する。

6.1.6.2　適合する分布がない場合

6.1.4 項で述べたプロセスは最良のシナリオの一つであり、物事はいつもスムーズにいくわけではない。データに適合する標準的な分布を見つけるために最善の努力をしたにもかかわらず、残念ながら、すべてのp値が小さく、すべての分布の適合が棄却されることがある。これは、データが受け入れられない、または何らかの理由で失敗したことを意味するのであろうか。いや、それは逆に、「標準的な」分布リストがデータを適切に表現できなかったことを意味する（結局、サービス時間のようなデータは観察される現象についての「真実」である。その責任は、データに合わせるべき分布の側にある）。

このような場合、Stat::Fit は経験分布を生成することができる。これらの分布は基本的にはヒストグラムの一種であり、Simio シミュレーションではこれから変数を生成できる。Stat::Fit のメニューで、File→Export→Export Empirical を選択し、Simio との互換性を保つために（デフォルトの Density ボタンではなく）Cumulative ラジオボタンを選択する。これにより、Windows のクリップボードにv_i以下の変量を生成する一連のv_iとc_iの組がコピーされる。v_iはデータ値のi番目に小さいものであり、c_iは対応するv_i**以下**の変量を生成する**累積**確率である。Simio にこの情報をコピーした後の処理は、離散的または連続的な分布かによって異なる。Simio Reference Guide（F1 キーまたは Simio の?アイコン）では、Contents タブの Modeling in Simio→Expression Editor, Functions and Distributions→Distributions を選択してから、必要に応じて Discrete（離散）または Continuous（連続）を選ぶ。

- 離散の場合、適切な Simio 式は、Random.Discrete(v1, c1, v2, c2,…)となる。データセットに必要な数のペアが続き、累積確率c_iで各v_iの値が生成される（たとえば、確率c_4はv_4**以下**の値を生成する確率であるため、確率$c_4 - c_3$でv_4に**等しい**値が生成される）
- 連続の場合、Simio 式は Random.Continuous(v1, c1, v2, c2,…)となる。生成される CDF は各(v_i, c_i)を通過し、直線で結ばれる。つまり、v_1の最小値 0 からv_iの最大値 1 まで上昇する、区分的に線形な「点を接続した」CDF となる。

離散と連続の両方のケースにおいて、この分布は下限（最小データポイント）と上限（最大データポイント）を持つ有限の分布であることに注意されたい。つまり、どちらの方向にも無限の裾を持たない。

「何も適合しない」問題へのもう一つのアプローチは、データテーブルから直接サンプルを抽出できる Simio の機能を活用することである。基本的に、サンプルデータの全セットを Simio のデータテーブルにインポートし、入力パラメータを使用してテーブルからランダムサンプルを抽出する。テーブル値に基づく入力パラメータの使用についてさらに詳しく知るには、6.5 節を参照されたい。

6.1.6.3　データがない場合

明らかにこれは望ましい状況ではない。一般的な対処法は、そのシステム（または類似システム）に詳しい「専門家」に、合理的に起こりうる最小値と最大値を尋

ね、一様分布を指定することである。この一様分布が両極端に過度の重みを与えていると感じる場合は、三角分布を代替として使用できる。三角分布では状況に応じて、範囲の中央付近に最頻値（PDF のピークであり、必ずしも平均ではない）を設定する。このような対処法でモデルを動かせるなら、実際にはモデルを感度分析ツールとして活用すべきであり（後述の「データの適切な量」参照）、どの入力が出力にもっとも影響を与えるかを確認するためでもある。そして、少なくともそれらの重要な入力については今後いくつかのデータを収集し、分布への適合を試みるべきである。6.5 節で説明するように、Simio の Input Parameters 機能は、主要な出力性能指標にもっとも適した入力に関する目安を提供する。

6.1.6.4　データが「多すぎる」場合

もう一つの極端な状況は、観測した実データのサンプルサイズが何千にも及ぶ場合である。一般に多くのデータを持つことは有利である。しかし、非常に大きなサンプルサイズでは、適合度検定が高い統計的**検出力**を持つことを認識する必要がある（この検出力とは帰無仮説を棄却する確率である）。そのため、グラフによる視覚的確認では完全に合理的に見える大きなサンプルでも、適合度検定ではすべての分布の適合が棄却される可能性が高い。このような場合、すべての仮説検定と同様に適合度検定も完璧ではないことを忘れてはならない。サンプルサイズが大きいために適合度検定で棄却された分布でも、視覚的に適切と判断できるものであれば、その分布を採用すべきだろう。

6.1.6.5　データの適切な量

分布適合に必要な実データのサンプルサイズについての疑問は当然生じる。残念ながら、この質問に対する普遍的な回答は存在しない。しかし、対処法の一つとして、重要な出力が入力分布の変化にどの程度敏感かを測定するために、シミュレーションモデル自体を感度分析ツールとして活用することができる。「入力分布がわからなければモデル構築すらできない」と考えるかもしれないが、厳密には正しいものの、実データ収集前でも、まずは合理的な仮定に基づく入力分布（現実離れしていないもの）を用いてモデルを構築できる。多くの場合、簡単な一様分布や三角分布で合理的な推測が可能であり、システムに精通した人物からパラメータ情報を得ることができる。これらの入力分布を変化させて、どれが出力にもっとも影響を与えるかを確認する。そして、出力への影響が小さい入力よりも、重要な入力分布についてのデータ収集に集中すべきである。「データがない場合」で述べたように、6.5 節で扱う Simio の Input Parameters 機能は、モデルの各入力について必要なデータ収集量の目安を提供する。

6.1.6.6　正しい答えは何か？

最後に指摘しておきたいのは、分布の指定は厳密な科学ではないということである。同じデータセットを用いても、2 人の分析者が異なる分布を選択することがある（両方とも完全に合理的な場合がある）。ここでいう「合理的」とは、データにじゅうぶん適合しているが異なる分布となるケースを指す。このような場合、第 2 の基準（たとえば、分布の平均を変更するパラメータ操作の容易さなど）を

考慮するとよい。ある分布が他の分布より操作が簡単であれば、異なる入力分布の平均を試す場合（たとえば、平均して 20%高速の窓口を設置した影響を検討する場合など）に備えて、より単純なほうを選択するかもしれない。

6.2　入力のタイプ

6.1 節で単変量の分布の適合を説明したので、ここではシミュレーションモデルへの数値入力の種類について、より一般的に考える。これらの入力は、確定的と確率的という区分、およびスカラー・多変量・確率過程という 3 区分を組み合わせた 2×3 の二次元フレームワークで分類できる。

6.2.1　確定的と確率的

確定的な入力は、ある航空会社が特定の空港に持つ自動チェックインキオスクの数のような単なる定数である。これはシミュレーション実行中に変更されない。ただし、キオスクがランダムな時点で故障し、その後一定期間修理を受ける場合は例外である。

確定的入力のもう一つの例としては、歯科医院における患者の予約時間が挙げられる。しかし、歯科医院の予約に遅れたり早く到着したりした経験はないだろうか。より現実的なモデルでは、到着時間は確定的な予約時間として設定し、そこに到着の遅れ（正の値）や早着（負の値）を表す**偏差** RV（確率変数）を加えることができる（患者が平均して時間通りに来ると仮定すれば、期待値は 0 になる）。これは RV を用いた**確率的入力**の例である。このような状況では、Model 3-3 のスプレッドシートによる待ち行列シミュレーションのように、到着時間間隔の分布を特定するのが一般的である。Simio では、Source オブジェクトの Arrival Type に `Arrival Table` を設定することでこの機能を実装している。

多くの場合、シミュレーションへの入力は確定的または確率的のどちらかに分類される：

- 空港でのチェックインキオスクからセキュリティまでの旅客の歩行時間。距離はすべての人にとって同じであるが、歩行速度は明らかに個人差がある。
- （確定的な）注文数と対比される、実際の出荷における商品数。
- 打ち抜き加工工場での機械による材料のプレスとカット時間。シートメタルが一定の速度で引き込まれ、プレス金型が一定の速度で動作する場合、これはほぼ確定的と言える。このプロセスに小さな変動を RV で表現するかどうかは、モデル構築の詳細レベルに関わる問題である（詳細なモデルが必ずしも優れたモデルとは限らない）。
- 車両の「定期的な」メンテナンス時間。基本計画は確定的であるが、予期せぬ問題が発見された場合には、必要となる追加（ランダムな）時間をモデル化することもある。

確定的または確率的のどちらでモデル化するかは、モデル設計上の判断である。実際のシステムに合致する方法を選ぶべきことは明らかであるが、シミュレーション出力への影響も考慮する必要がある。12 章では、特定の入力について計画フ

ェーズでは確定的に扱い、リスク分析フェーズでは確率的に扱う方法を検討する。

6.2.2　スカラーと多変量と確率過程

入力が一つの数値であり、確定的または確率的である場合、それは**スカラー**値である。特に、このスカラーが確率的に扱われるとき、**単変量**と呼ばれる。これは、シミュレーションへの入力をモデル化するもっとも一般的な方法である。一度に一つのスカラー数（または RV）であり、たいていはモデル全体でそのような入力がいくつか存在する。これらは、一般に統計的に互いに独立していると仮定される。6.1 節での考察では、暗黙のうちに、モデルはすべての確率的な入力がモデル全体にわたって単変量および互いに独立であると仮定した。製造業のモデルでは、このようなスカラー単変量の入力は、部品の処理時間やその後の検査時間を表すだろう。救命救急では、患者の連続した到着時間間隔、年齢、性別、保険状況、診察室での所要時間、診断コードと処置コード（自家用車で病院へ行くか救急車を呼ぶかなど）が含まれるかもしれない。

しかし、シミュレーションモデル内の異なる入力間に関係が存在するケースがある。その場合には、独立したものとしてではなく、入力**ベクトル**の成分（または座標）として扱わなければならない。これらの座標の一部がランダムならば、これは**確率ベクトル**または多変量分布と呼ばれる。重要なのは、入力確率ベクトルの座標間に従属関係や相関関係を認めることで、座標が独立であると仮定するよりも現実的な入力とすることができる点である。そして、これはシミュレーションの出力にも影響を及ぼす。前項の製造業の例では、処理時間と検査時間の間に正の相関を認めることで、ある部品が他の部品よりも長い時間がかかる（つまり問題のある部品である）という現実を表現できる。また、関節炎を患う幼児や妊娠による合併症を示す初老男性のような、救命救急ではありえない状況が生じるのを防ぐことができる。もしそれらの入力がすべて独立に生成されるなら、そのような非現実的な組み合わせが発生する可能性がある。別の例として、消防署のシミュレーションを考えよう。そこでは、特定の出動で派遣される消防車と救急車の数に正の相関がある（大規模な火災ではそれぞれ数台が必要だが、小規模な火災ではおそらくそれぞれ 1 台のみでじゅうぶんだろう）。これをどのようにモデル化し実行するかについては、Mueller（2009）を参照してほしい。このような状況のいくつかは論理的にモデル化できる（たとえば、医療診療所のモデルでは、最初に性別を生成した後、妊娠合併症の診断を許可する前にチェックを行う）が、他の状況では、相関関係や同時確率分布を利用して統計的に関係をモデル化できる。このような非独立性が現実世界のシステムに存在する場合、それを無視してすべての入力を独立に生成すると、シミュレーションの出力結果に誤差が生じる可能性がある。

入力確率ベクトルを特定する一つの方法は、6.1 節のように、まず座標の各単変量確率変数を一度に一つずつ分布に適合させることである。これらは入力確率ベクトルの**周辺分布**と呼ばれる（2 次元の離散確率ベクトルの場合は、同時分布表の周辺として表示されることがある）。次に、統計書で述べられる標本相関推定量によって相互相関を推定するためにデータが使用される。周辺単変量分布と相

関行列を特定することは、必ずしも確率ベクトルの同時確率分布を完全に規定するものではないことに注意が必要である。ただし、同時分布するものが正規確率分布である場合は例外である。

入力確率ベクトルの次元が無限になると、シミュレーションモデルを駆動する入力として**確率過程**（の実現）を考えることになる。通信システムのモデルでは、パケットのストリームを入力確率過程として考える（それぞれが特定の到着時間と特定のサイズを持つ）。シミュレーションへの入力ストリームとして使うための非常に一般的な時系列モデルに適合させるロバストな方法については、Biller and Nelson（2005）を参照してほしい。

このような「非標準的」なシミュレーションの入力モデリングの詳細については、Kelton（2006、2009）などを参照してほしい。

6.2.3　時変到着率

多くの待ち行列システムでは、外部からの到着率は時間の経過とともに著しく変化する。たとえば、1 日のファストフードレストランの客数、年間の救急室の患者数、1 日の都市高速道路の交通量などが挙げられる。入力データ間の相関関係を無視することで出力結果にエラーが生じる可能性があるのと同様に、到着における非定常性を無視することでもエラーが発生するかもしれない。24 時間にわたる「平均」到着率を一定として高速道路の通行を（誤って）モデリングすることを想像してみよう。それは午前 3 時の過大評価とラッシュアワーの混雑の過小評価につながる（Harrod and Kelton（2006）で、この方法により生成される誤差についての数値例が示されている）。

時変到着率を表すもっとも一般的な方法は、**非定常ポアソン過程**によるものである（非同次ポアソン過程とも呼ばれる）。ここでは、到着率を一定の均一割合λとする代わりに、シミュレーション時刻tの関数$\lambda(t)$とする。任意の時間間隔$[a, b]$の間に到着する数は、平均$\int_a^b \lambda(t)dt$の（離散）ポアソン分布に従う。したがって、速度関数$\lambda(t)$が高くなると、時間間隔内の到着の平均数も高くなる（もちろん、等しい持続時間間隔を仮定している）。到着率が**実際に**λで一定であれば、これは割合λの定常ポアソン過程となる。これは平均$= 1/\lambda$で IID の指数関数の RV を到着時間間隔とする到着過程に相当する。

もちろん、シミュレーションで非定常ポアソン過程をモデル化するには、速度関数$\lambda(t)$の推定値を特定するために観測データの使用法を決定する必要がある。これは、Kuhl et al.（2006）とその引用文献で詳細に研究されているトピックである。$\lambda(t)$を推定する一つの簡単な方法は、**区分的定数**関数によるものである。ここでは、一定期間（例として高速道路の交通で 10 分間隔とする）の時間間隔で、到着率が**実際に**一定であると仮定するが、各 10 分間隔の終わりには異なるレベルに変化することがある。各 10 分間の期間内で一定の割合を仮定することが妥当かどうかを判断するには、そのシステムについての知識が必要である。それぞれ 10 分の期間の割合を指定するには、その期間中の到着を正確にカウントし、可能であれば複数の平日にわたって平均をとる（間隔内到着率が整数である必要はない）。この区分定数割合推定方法は比較的簡単だが、Leemis（1991）に示され

ているように、優れた理論的背景に基づいている。

Simio は、エンティティがシステムに到着する **Source オブジェクト**で、前述のプロセスから到着を生成することをサポートしている。Arrival Mode（到着モード）に `Time Varying Arrival Rate`（時変到着率）を指定する。到着率の関数は Simio の Rate Table（レートテーブル）で別途指定する。7.3 節では、病院の救急部門における区分定数の到着率による非定常ポアソン到着過程の実装例を紹介する。Simio では、全レートを対応する Rate Table に入力する前に、**時間**当たりの単位に換算しなければならないことに注意が必要である。たとえば、到着率のデータがそれぞれ 10 分間の期間中の到着数であれば、**時間**当たりの到着率に変換するには、期間ごとのレート推定値を 6 倍する必要がある。

6.3　乱数生成法

すべての確率的シミュレーションは、**乱数**を「生成」する方法から始まる。乱数とは、特にシミュレーションにおいて、0 から 1 の間に一様かつ連続的に分布した観測値を意味する。乱数は互いに独立である必要がある。これは理想論であり、文字通りには実現できない。そこで、独立しており一様に分布しているように**見える** 0 と 1 の間の多くの値を生成する数値計算アルゴリズムが開発されてきた。生成された乱数は、特定の証明可能な理論上の条件（極めて長期間自身を繰り返さないなど）を満たし、一様性と独立性に関する統計的・理論的な検定に合格しているという意味で、乱数に「見える」と表現される。これらのアルゴリズムは、**乱数生成法**（Random-Number Generator；RNG）として知られている。

優れた RNG を構築するために多くの研究がなされているが、これはほとんどの人が想像するよりもはるかに困難な課題である。（現在は過去のものとなったが）古典的な手法の一つに、**線形合同法**（Linear Congruential Generator；LCG）がある。これは次の漸化式に基づいて整数Z_iの数列を生成する：

$$Z_i = (aZ_{i-1} + c)(\mathrm{mod}\, m)$$

ここでa、c、mは慎重に選択すべき非負の整数定数（aとmは 0 より大きい）であり、**シード**値$Z_0 \in \{0,1,2,\cdots,m-1\}$を指定する。$Z_i$は、$(aZ_{i-1}+c)$を$m$で除算した際の**余り**を表すことに注意が必要である。mで割った余りなので、各Z_iは 0 と$m-1$の間の整数となる。0 と 1 の間の乱数U_iが必要なので、$U_i = Z_i/m$とする。あるいは$m-1$で割ることもできるが、実際にはmが非常に大きいため、あまり問題にならない。図 6.19 は、Excel スプレッドシート `Model_06_01.xls`（付録 C で説明するようにダウンロード可能）の一部であり、最初の 100 個の乱数を生成するための LCG 実装を示している。Excel 関数 `MOD` を使用して、列 F に剰余を返している。

LCG のパラメータa、c、mおよびZ_0は、セル B3 から B6 にあり、他の値に変更できるように設定されており、スプレッドシート全体が自動的に更新される。F 列に生成されたU_i値を見ていくと、最初はかなりよく「ばらついて」見えるかもしれない。しかし、少し詳しく調べると、シード$Z_0 = 7$は$Z_{22} = 7$に再び現れ、その

E7　=MOD($B$3*E6+$B$4, $B$5)

線形合同法による乱数生成

a:	17		i	Z_i	U_i
c:	8		0	7	n/a
m:	23		1	12	0.5217
Z_0:	7		2	5	0.2174
			3	1	0.0435
			4	2	0.0870
			5	19	0.8261
			6	9	0.3913
			7	0	0.0000
			8	8	0.3478
			9	6	0.2609
			10	18	0.7826
			11	15	0.6522
			12	10	0.4348
			13	17	0.7391
			14	21	0.9130
			15	20	0.8696
			16	3	0.1304
			17	13	0.5652
			18	22	0.9565
			19	14	0.6087
			20	16	0.6957
			21	4	0.1739
			22	7	0.3043
			23	12	0.5217
			24	5	0.2174
			25	1	0.0435

図 6.19　Excel ファイルで実装された線形合同法による乱数生成

後に生成された数列は、最初からまったく同じ順序で繰り返される。このようなシードの早期繰り返しを防ぐために、Z_0を別の値に変更したくなるかもしれない。しかし、すぐにそれが不可能だと気づくだろう。

なぜなら、mod m演算には整数剰余の可能性に制約が存在し、m番目の乱数を生成した時点から繰り返しが始まるからである（値a、c、mによってはより早く始まる場合もある）。これを RNG の**周期性**と呼び、そのサイクル長を**周期**と呼ぶ。LCG が持つ他のあまり目立たない問題は、長い周期の場合でも、私たちが必要とする一様性や独立性を持つような整数を得るのが難しく、素数や互いに素などのかなり難解な数学（**整数論**）を必要とすることである。

1951 年の Lehmer（1951）による開発後、いくつかの比較的良好な LCG が考案され（つまりa、c、mの許容値が発見され）、長年にわたり成功裏に利用された。しかし、mは明らかに大きな値であった（多くの場合、$m = 2^{31} - 1 = 2{,}147{,}483{,}547$で、約 2.1 億または$10^9$のオーダ）。それにもかかわらず、コンピュ

ータの速度は 1951 年以来大きく発展したため、高品質の RNG として LCG はもはや候補として挙げられなくなった。一つには、10^9の周期の LCG でさえ、現代の一般的な PC ではわずか数分ですべてのサイクルを実行できるためである。そのため、他の異なる方法が開発されてきたが、多くは依然として内部でモジュロ剰余演算を用いており、より長い周期がよりよい統計的性質（独立性と一様性）を示すと考えられている。ここではそれらを詳説しないので、その方法については L'Ecuyer（2006）を、RNG の検定については L'Ecuyer and Simard（2007）を参照してほしい。

Simio の RNG は、**メルセンヌ・ツイスタ**（Mersenne Twister。Matsumoto and Nishimura（1998）を参照）であり、天文学的なサイクル長を持っている（その長さは、10^{6001}である。観測可能な宇宙が約10^{80}の原子を含んでいることと比較するとその大きさがわかる）。同様に重要なことは、優れた統計的性質を持っていることである（最大 623 次元まで、独立性と一様性を持つことが証明されている）。Simio において、少なくとも 2 つの懸念すべき点がある。それは、乱数の不足および低品質の乱数生成である。

Simio に実装される際、メルセンヌ・ツイスタは互いにじゅうぶんに長い膨大な数の**乱数列**に分割されている。これはサイクル全体のサブセグメントであり、任意の 2 つの乱数列が重複することは本質的にありえない。シード（実際にはシード**ベクトル**）には直接アクセスできないが、その必要もない。必要であれば、分布の特性を表す拡張パラメータとして、使用する乱数列を指定できる。たとえば、6.1 節で Stat::Fit によりサービス時間データを適合し、シフトした対数正規分布に対して（デフォルトの乱数列 0 ではなく）乱数列 28 を使用したい場合、`17.4+Random.Lognormal(2.04, 0.672, 28)`と入力する。

なぜ、このようにするのだろうか。その理由の一つは、複数のシナリオ（たとえば、異なる工場のレイアウト）を比較する際、出力の違いがレイアウトの違いによるものであり、乱数による偶然の変動ではないとより確信を得たい場合のためである。モデル内の各入力分布に専用の乱数列を割り当てることで、異なる工場レイアウトのシナリオをシミュレートする際、さまざまな入力に対して乱数を**同期**させることができる。これにより、各レイアウトのシナリオでは同じ時刻に同じジョブが到着し、そのジョブの処理要件なども全シナリオ間で同一となる。この手法によって、代替シナリオ間の結果の違いを説明する際、少なくとも部分的に「乱数列の違いによる影響」を排除することができる。

このように乱数を再利用する手法は、**分散減少法**の一種であり、Banks et al.（2005）や Law（2015）などの一般的なシミュレーション文献で解説されている。この分散減少法は、複数のシナリオで同じ目的に同じ（共通の）乱数を使用するため、**共通乱数法**と呼ばれる。この方法は直感的に魅力的なだけでなく、確率論的な根拠も持っている。Y_AとY_BがそれぞれシナリオAとBに対する同じ性能指標 RV（たとえば、総システム内時間）だとすると、比較においては$Y_A - Y_B$や$Var(Y_A - Y_B) = Var(Y_A) + Var(Y_B) - 2Cov(Y_A, Y_B)$に関心がある（$Cov$は共分散を表す）。共通乱数法を使用することにより、$Cov(Y_A, Y_B) > 0$となり、シナリオを独立して実行す

る場合（その場合、$Cov(Y_A, Y_B)$は 0 となる）と比べて、出力の差$Y_A - Y_B$の分散が減少することにより、Y_AとY_Bに正の相関が見込まれる。さまざまなシナリオで複数回の反復実行をする場合、Simio は各シナリオの反復実行ごとに同じ乱数列の同じ位置から開始するよう調整する。これにより、共通乱数法による同期が最初の反復実行から維持される。また、Simio で複数のシナリオを実行する際も、各シナリオでは使用するそれぞれの乱数列の先頭から開始される。

すべての乱数生成法において重要なことは、予測不可能という観点からは、すべてが真にランダムではないということである。ある生成法に対する決まったアルゴリズムで、同一のシード値（メルセンヌ・ツイスタの場合にはシードベクトル）を用いる場合、必然的に正確に同じ「ランダム」な乱数列が得られる。このような理由から、乱数生成法から得られる値を、技術的により正確な用語として、**擬似乱数**と呼ぶこともある。したがって、Simio のようなシミュレーションソフトウェアでは、モデルを再実行すると、正確に同一の数値結果が得られる。この事実は、「乱数」生成法を用いた場合に起こるはずのないことだと直感的に考えてしまうシミュレーション初心者を驚かせる。しかし、同一の実行内で複数回モデルを**反復実行**させると、反復ごとに乱数列が継続して用いられるため、シミュレーションの出力データの統計分析に必要な、異なる独立した結果が得られる。実際のところ、同じモデルを再実行すると同一の結果が得られるという特性は、（たとえば、デバッグにおいて）非常に有用である。

6.4　確率変量および確率過程の生成

6.1 節および 6.2 節では、モデルへの確率的な入力に対して確率分布を選択する方法を概説し、6.3 節では、（0 と 1 の間に連続一様に分布する）乱数の生成法について説明した。本節では、0 と 1 の間の一様乱数を、モデルで必要な入力分布に変換する方法を説明する。これは一般に、**乱数発生法**と呼ばれている。Simio を使用する場合、これは Simio が内部に備えている機能であり、少なくともおよそ 20 の分布を生成できる。しかし、まれにその他の分布から変量を生成する必要がある場合に備え、基本的原理を理解することは依然として重要である。

実際には、すでに 3 章のいくつかの特殊なケースで乱数発生法を説明してきた。3.2.3 項では、整数$1000,\ 1001, \cdots,\ 5000$上に一様に分布した離散確率変量の生成が必要だった。また、3.2.4 項では、連続一様分布、対数正規分布、およびワイブル分布からの確率変量を扱い、3.3.2 項では、指数分布から連続確率変量を生成した。それぞれのケースで、分布固有の方法を考案し、離散一様の場合には、その方法が正しい理由について合理的な説明を行った（その他のケースは本節で扱う）。

乱数から希望する入力分布の確率変量への変換方法を考案する際の一般的な原則がいくつかある。おそらくもっとも重要な方法は、CDF の**逆関数法**だろう。これは希望する入力分布の CDF であるF_Xの逆関数（代数的な意味の逆関数、つまり逆数ではない）を求める方法である（ここでXは対応する RV である。通常、対応する RV を明確にするため、CDF や PDF、PMF を共に記すことが多い）。CDF は、対応する RV がその引数以下である確率を与えることを思い出してほしい。つま

り、$F_X(x) = P(X \le x)$のとき、RV Xが CDF F_Xを持つ。

まず、連続的な RV Xの CDF F_X（F_Xは連続関数となる）を考える。基本的な考え方は、乱数Uを生成し、$U = F_X(X)$として、その式をXについて解くというものである。その解Xは CDF をF_Xとする確率変量であり、次の段落で解説する。ここで、「試みる」としたのは、分布によって式を簡単に解ける場合とそうでない場合があるからである。解は（単純かどうかに関わらず）$X = F_X^{-1}(U)$で表される。ここで、F_X^{-1}はF_Xの逆関数である。

なぜこのような方法を用いるのか。つまり、解$X = F_X^{-1}(U)$が CDF F_Xの分布を持つのはなぜか。その鍵は、Uが０と１の間に連続一様に分布していることにある。これが真に近いことを保証するために、乱数生成法の質を信頼する必要がある。$[0,1]$の任意の区間、たとえば$[0.2, 0.6]$とすると、乱数Uがその部分区間に該当する確率は部分区間の幅であり、このケースでは$0.6 - 0.2 = 0.4$である。このとき、Uの PDF は以下のとおりである：

$$f_U(x) = \begin{cases} 1 & 0 \le x \le 1 \text{ の場合} \\ 0 & \text{その他の場合} \end{cases}$$

なお、RV がある区間に入る確率は、その区間に対する PDF 下の面積である（図 6.20）。

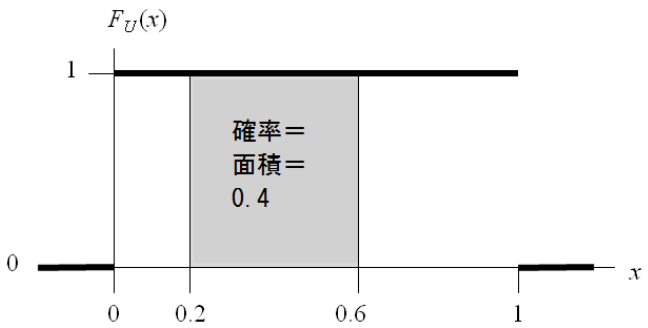


図 6.20　０から１までの連続一様分布の PDF

特に、０と１の間の任意の値wについて、$P(U \le w) = w$である。そのため、「生成された」変量$X = F_X^{-1}(U)$から、Xの範囲内の任意の値x以下となる確率は、次のようになる：

$$P(F_X^{-1}(U) \le x) = P\left(F_X\left(F_X^{-1}(U)\right) \le F_X(x)\right)$$

（両辺に増加関数F_Xを適用）

$$= P\left(U \le F_X(x)\right)$$

（逆関数の定義より）

$$= F_X(x)$$

（Uは０と１の間に一様に分布するため、$F_X(x)$も０と１の間の値をとる）

これは、生成された変量$X = F_X^{-1}(U)$がx以下である確率が$F_X(x)$であることを示している。つまり、生成された変量は期待通り CDF $F_X(x)$を持つ。図 6.21 は、

連続的な場合の逆 CDF 法を示している。図では、縦軸に 2 つの乱数U_1およびU_2がプロットされている（これらの乱数は常に 0 と 1 の間にあるため、任意の CDF のプロット縦軸内に常に「適合」する）。対応する生成された変量X_1およびX_2は、CDF $F_X(x)$の範囲内のx軸上にある（図の CDF では、ガンマや対数正規分布のように、すべての値が正の範囲をとる）。したがって、逆 CDF 法は、縦軸に乱数をプロットし、CDF から対応する値を読み取り（原点より左右どちらにも可能性があるが、図の例では RV Xが常に正のため右側になる）、x軸に垂線を下ろすことで生成された変量を得ることができる。

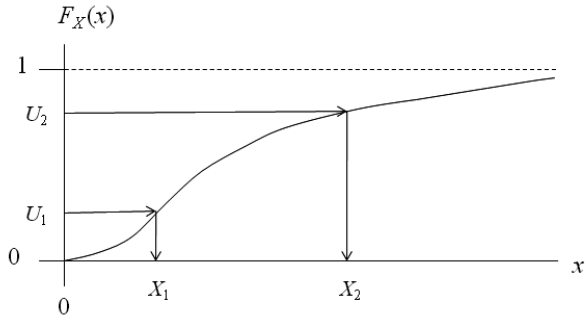


図 6.21　連続型確率変量生成のための逆 CDF 法

ここで、乱数が大きくなると、より大きな変量が得られることに注意してほしい（CDF は非減少関数であり、その逆関数もまた同様であるため）。また、乱数は縦軸において 0 と 1 の間に一様に分布しているため、急勾配の CDF（その導関数である PDF も高い）に「ヒットする」可能性が高くなる。これはまさに我々が望む結果である。すなわち、生成された RV は PDF が高い領域でより高密度になる（PDF が**密度**関数と呼ばれる理由である）。つまり、逆 CDF 法は、乱数の一様分布を希望する RV Xの分布に従って「変形する」方法といえる。

例として、3.3.2 項と同じ分布を利用し、Xが平均$\beta > 0$の**指数分布**を持つと仮定する。多くの確率統計の書籍で述べられているように、指数分布の CDF は次のとおりである：

$$F_X(x) = \begin{cases} 1 - e^{-x/\beta} & x \geq 0 \text{ の場合} \\ 0 & \text{その他の場合} \end{cases}$$

ここで、$U = F_X(X) = 1 - e^{-X/\beta}$とし、$U$について解くと、数行の演算で変量生成の基となる$X = -\beta \ln(1 - U)$が得られる。これは 3.3.2 項で得られたものと同じである。この指数分布の場合、CDF が**閉じた形**の式であったため、すべてがうまくいった。さらに、指数分布の CDF は単純な代数演算により（$U = F_X(X)$をXについて解くことで）簡単に逆関数化できた。他の分布では、（たとえば正規分布のように）CDF が閉じた形の式を持たず、逆関数化できないものもある。また、CDF が閉じた形の式を持つ分布でも、解析的に逆関数を**得られない**場合もある（たとえば、形状パラメータに大きな整数値をとるベータ分布）。したがって、逆 CDF 法は原則的には連続的な場合に常に機能するが、分布によっては求根アルゴリズム

のような数値解析手法を用いる必要がある。

離散のケースでは、CDF が連続的でない点を除いて、逆 CDF 法の**考え方**自体は同じである。すなわち、CDF は階段関数（区分的に定数の関数）であり、離散型 RV Xの対応する値が PMF の値に等しい段の高さによって表現される。図 6.22 では、RV Xの値としてx_1, x_2, x_3, x_4をとる場合の逆 CDF 法を示している。そのため、実施する際には一般に、段に対する適切な「上昇」を見つけるための探索が必要となる。まず、縦軸（もちろん、すべて 0 と 1 の間）方向に段の高さを射影するために、(縦の)間隔$[0,1]$を PMF の値に等しい幅のサブセグメントに分割する。次に乱数Uを生成し、その値を含む縦軸の部分区間を探索する。その後、対応するxが生成された変量Xとして返される（図 6.22 の例では、$X = x_3$）。離散一様の需要を生成するために 3.2.3 項で考案した独自の方法は、実際には、離散型の逆 CDF 法であり、探索ではなく式としてより効率的に実施できる。特定の分布に対するこのような特別な「操作」は一般的であり、場合によっては逆 CDF 法と等価な方法になることがある。

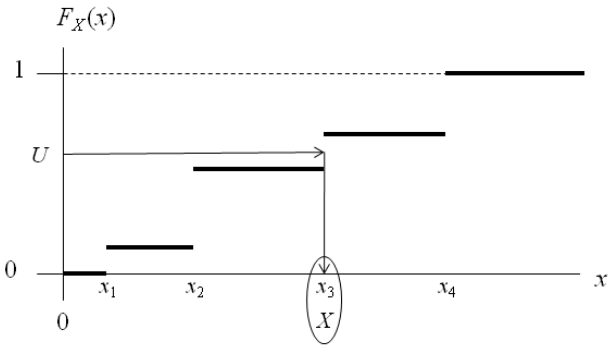


図 6.22　離散型確率変量生成のための逆 CDF 法

逆 CDF 法はある意味で、最高の確率変量生成法であるが、唯一の方法ではない。高速性、高精度、数値の安定性に焦点を当て、確率変量生成に関して多くの研究が行われてきた。この話題についての概要は、Banks et al.（2005）や Law（2015）を参照されたい。また、Devroye（1986）ではより詳細な議論が展開されている。

6.1.6 項で確率ベクトルおよび確率過程の**特定**について簡単に議論したが、それらを**生成**する（あるいは実現する）方法についても考える必要がある。これらの方法を詳しく議論することは本書の範囲を超えるため、前述の参考文献が多くの状況において参考になるだろう。6.2.3 項で述べたように、Simio は割合関数が区分線形となる非定常ポアソン過程を生成するためのメソッドを組み込んでいる。しかし一般に、シミュレーションモデリングソフトウェアは、相関する確率変数や多変量分布の確率ベクトル、あるいは一般確率過程などを生成するための汎用的な方法をサポートしていない。

6.5　Simio Input Parameters の利用

本節では、モデリングを簡便にし、入力解析を改善するために、Simio の **Input**

Parameters を活用する方法を述べる。Input Parameters は、これまで解説してきたような対応するオブジェクトインスタンスのプロパティに直接式を入力するのではなく、入力データを特徴づけて指定する方法を提供する。

Simio Input Parameters を解説するために、図 6.23 に示す 3 つのステーションによる直列の簡単なモデル Model 6-2 を用いる。関心のある 4 つの入力は、エンティティの到着時間間隔と 3 つのサーバの処理時間である。これまでの章で用いた標準的な方法は、オブジェクトインスタンスに直接値を入力するものであった。たとえば、Source1 の Interarrival Time プロパティに `Random.Exponential(2.5)`分、サーバの Processing Time プロパティに `Random.Triangular(1, 2.25, 3)`あるいは `1 + Random.Lognormal(0.0694, 0.5545)`と入力していた。Model 6-2 では代わりに、4 つの入力それぞれに対して個々の Input Parameter を定義し、対応するオブジェクトインスタンスのプロパティで Input Parameter を指定する。このモデル内のすべてのノードは、30 秒（確定的）の Travel Time プロパティを持つタイムパスで接続されている。タイムパスを使用することにより、モデル結果は物理的なパスオブジェクトの長さに影響されなくなる。

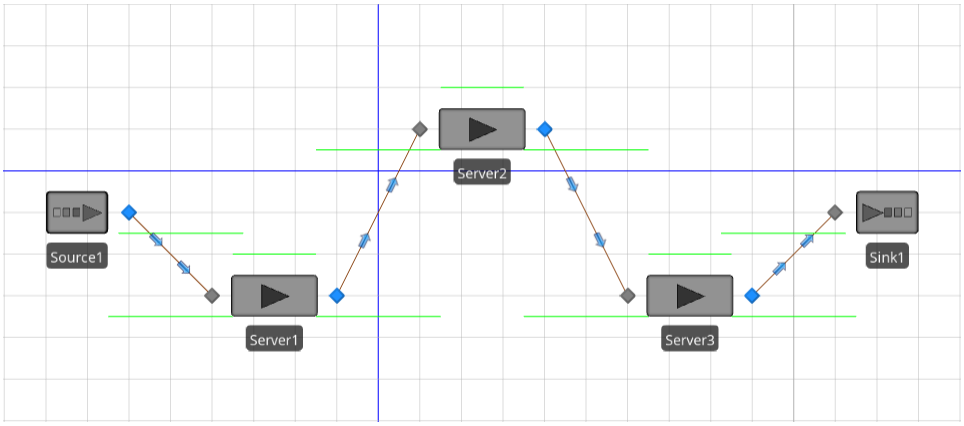


図 6.23　Model 6-2：Input Parameters を用いた 3 つのサーバ

図 6.24 に Model 6-2 の設定済み Input Parameters の定義を示す。Input Parameters 定義ページは、Model オブジェクトの Data タブにある。メインウィンドウは上下に分割されており、上部には 4 つの定義済み Input Parameter が表示され、下部には選択した Input Parameter のサンプルヒストグラムが表示される（なお、Expression タイプまたは Table タイプのパラメータではヒストグラムは表示されない）。Properties ウィンドウには選択した Input Parameter のプロパティが表示される。図 6.24 では、パラメータ PTimes1（Server1 の処理時間パラメータ）が選択されているため、対応するウィンドウにそのヒストグラムとプロパティが表示されている。Model 6-2 では、エンティティの到着時間間隔に `EntityArrivals` を、Server1、Server2、Server3 の処理時間には、それぞれ `PTimes1`、`PTimes2`、`PTimes3` を使用している。

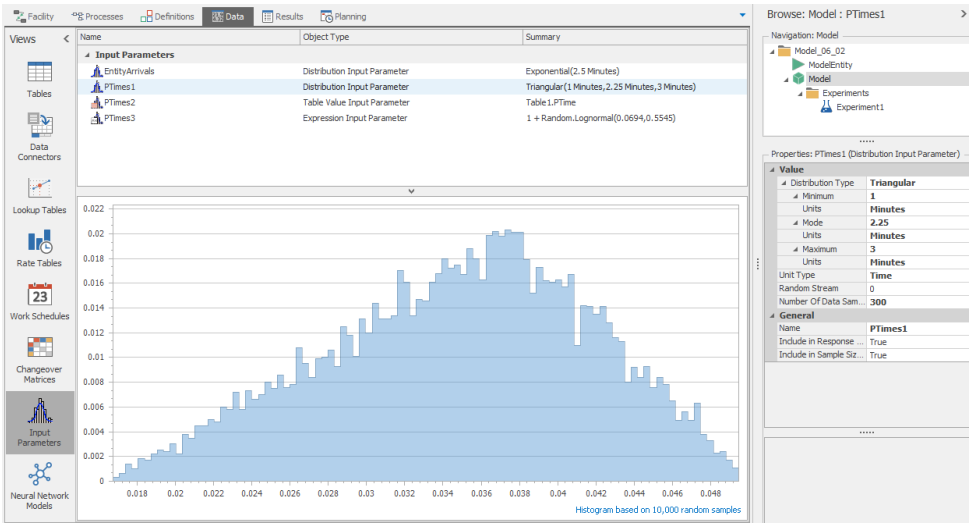


図 6.24　Model 6-2 の Input Parameters 定義

Simio は、現時点で 3 種類の Input Parameters をサポートしている：

- **確率分布**（Distribution）：確率分布と対応するパラメータを指定する。図 6.24 では、Model 6-2 の PTimes1 という名称の Input Parameter が示されている。このパラメータは、(1, 2.25, 3)分の三角分布を定義しており、ウィンドウ下部に表示されているヒストグラムは、この確率分布から抽出された 10,000 サンプルから作成されている。このヒストグラムの目的は、シミュレーション実行中にモデルへの入力として生成される値が、おおよそ意図した範囲と形状を持っているかどうかを、モデル作成者が簡単に確認できる手段を提供することである。プロパティ Number of Data Samples は 300 と指定されており、これはこの分布の当てはめに 300 の実世界の観測値を利用したことを意味する。このプロパティについては 6.5.2 節で詳しく説明する。

  パラメータ EntityArrivals も確率分布のパラメータであり、連続して到着するエンティティ間の到着時間間隔を定義している。ここでは、300 の実世界の観測値に基づき、平均 2.5 分の指数分布に従う。シミュレーション実行時のサンプリングという観点からは、確率分布の Input Parameter を使用することは、オブジェクトインスタンスに直接確率分布の式を用いることと運用上同じであり、本章ですでに議論した。しかし、Simio Input Parameter として定義することで、モデル内の複数の箇所で再利用できるようになる。また、6.5.1 項および 6.5.2 項で述べるように、感度分析やサンプルサイズ誤差の推定においても利点がある。
- **テーブル値**（Table Value）：サンプルとして用いられる値の集合を含む表の列を指定する。プロパティ Table Column は、データテーブルとデータポイントを含む列を指定する（ここでは Table1.PTime）。テーブルデータのヒストグラムは Input Parameters ウィンドウには表示されないが、7.1 節で説明されるデータテーブルの Histogram Tool を使用して表示できる。

  Input Parameter に Table Value タイプを指定すると、Table **列にある値か**

ら無作為にサンプルが抽出される。モデルの実際のシミュレーション実行中も同様に抽出され、適合した分布や経験的に指定した分布からは抽出されない。生成された各変量は、テーブル列にある個別のデータポイントがとりうる値である離散（連続ではない）分布からの観測値のいずれかと常に**等しく**なる。そして、各変量が生成される確率は、テーブル列に現れる回数を、テーブル列の値の総数（この例では 300）で割った値になる。Model 6-2 の Table1 の列 PTime を見ると、たとえば値 2.21 は 17 回現れるため、このヒストグラムおよびシミュレーション中に生成される確率は 17/300＝0.0567 となる。一方、値 2.48 は 1 回しか出現しないため、その生成確率は 1/300＝0.0033 となる。さらに、値 2.48 はこの列（データセット）における最大値であり、2.48 より大きな値は生成されないため、モデルの文脈によっては、分布の右裾の確率が重要かどうかを考慮する必要がある。

つまり、この種の Input Parameter を使用する場合、分布の適合や適合度検定は行われない。この方法の利点は、研究者たちが指摘しているように、特に適切な分布が見つからない場合でも、データの「不十分な適合」を心配せずに済む点がある。テーブル列に含まれる実世界の観測値のサンプルサイズがじゅうぶんに大きければ、適切な範囲の値をカバーできると考えられる（Nelson 2016）。しかし、分布の裾が重要な場合、たとえばサービス時間の分布における右裾が重要である場合には、データセットの最大値で生成される値を制限することがモデルの妥当性に問題を引き起こす可能性がある。つまり、実世界のデータで観測された最大値を超えるような非常に大きなサービス時間が生成されないため、出力尺度に下方バイアスがかかる恐れがある。

・**数式**（Expression）：サンプリングに利用する数式を指定する。図 6.24 で示したように、パラメータ `PTimes3` は式 `1 + Random. Lognormal(0.0694, 0.5545)` を用いる数式タイプのパラメータである。このタイプのパラメータでは、式を評価することによってサンプルが生成される（この例では、シフトされた対数正規分布からサンプリングされる）。このタイプの Input Parameter には、10,000 サンプル値によるヒストグラムは提供されない。

オブジェクトインスタンスで Input Parameter を利用するには、対応するプロパティ値として Input Parameter 名を入力する（図 6.25）。

または、プロパティ名を右クリックし、コンテキストメニューの `Set Input Parameter` のリストから設定することもできる（Referenced Properties の指定方法と同様）。Model 6-2 では、Source1、Server1、Server2 に対応する Input Parameter プロパティを割り当てる。

Input Parameters を用いることで、複数の入力に同じ式を利用できるため、モデリングを簡単にすることができる。しかし、次の 2 項で述べるとおり、より重要な点は入力解析にある。Simio の**応答感度**分析は、それぞれの Input Parameters に対する各実験応答の感度を測定する機能である（6.5.1 項）。この分析は一般に、モデル入力における重要な「実世界」データを収集する前に行われ、データ収集に対する労力を配分する助けとなる（各入力のモデル応答に対す

る重要度を判断できるため)。Simio の**サンプルサイズ誤差推定**分析は、信頼区間に基づく方法を利用し、実験応答の**不確実性**における Input Parameters の包括的な影響を推定する（6.5.2 項)。

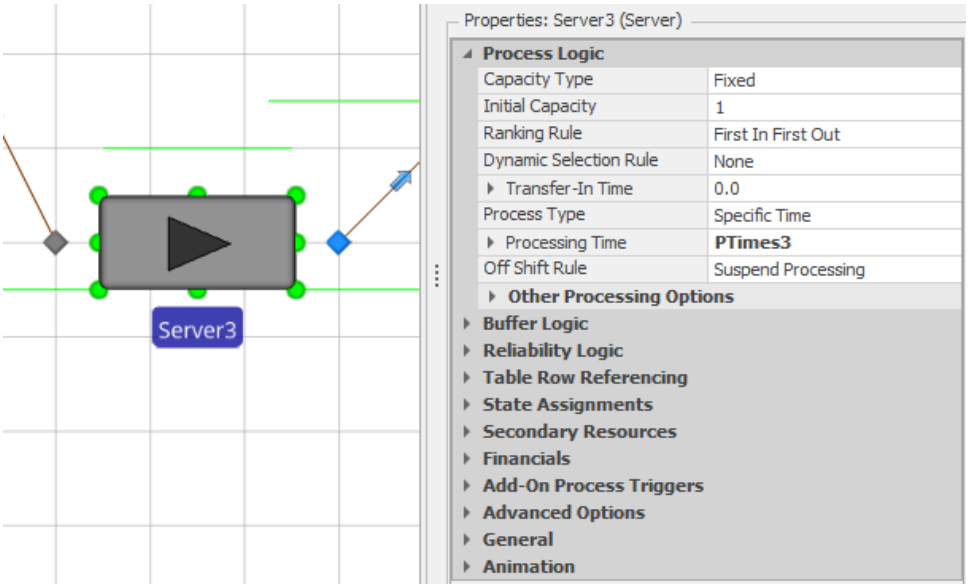


図 6.25　Model 6-2 における Server3 の Processing Time への PTimes3 の設定

6.5.1　応答感度

Simio の**応答感度**分析（Song et al. 2014、Song and Nelson 2015）は、**線形回帰**を用いて、各実験応答を Input Parameters の集合に（近似的に）関連づける。統計学の回帰分析の用語では、応答は回帰式左辺の従属変数Yであり、Input Parameters は右辺の説明変数X_jに相当する。計算される感度は、他の Input Parameters を固定したうえで、ある Input Parameter の 1 単位増加（すなわち、X_jの単位に関わらず+1 の変化）に対する、モデルで選択された実験応答の変化に関する近似的な予測である。したがって、これらの感度は、適合した線形回帰係数$\hat{\beta}_j$あるいは「傾き」の推定値となる。Simio で Response Sensitivity 分析を行うためには、実験の反復実行回数をモデルで用いられる Input Parameters の数よりも厳密に大きくし、回帰モデルに適合するじゅうぶんな自由度を確保する必要がある。ただし、この応答感度分析の処理に必要な反復実行回数（Input Parameters の数を上回る反復実行回数）をあらかじめ実行していれば、Simio は自動的にこの分析を実行する。応答感度分析は、シミュレーションモデルの回帰**メタモデル**近似とも呼ばれる方法と同種である。この分析は、特に「実世界」のデータが利用できない際に、今後の研究やデータ収集の対象となる入力を決定するために有効である。

Model 6-2 には、NIS（システム内数の時間平均）と TIS（平均システム内時間）の 2 つの実験応答がある。`DefaultEntity.Population.NumberInSystem.Average` および `DefaultEntity.Population.TimeInSystem.Average` の式をそ

れぞれ用いている。（各 500 シミュレーション時間の 100 回の反復実行に基づく）Model 6-2 の TIS に対する Response Sensitivity の Tornado Chart ビューを図 6.26 に示す。なお、Tornado Chart のほかに、チャート下部のタブを利用して、Bar Chart、Pie Chart および Raw Data ビューも見ることができる。Tornado Chart では、Input Parameters が各実験応答に対する感度の絶対値の降順に並べられている。青色のバーが原点の左に伸びていれば、負の感度係数であることを示し、右に伸びていれば、正の感度係数であることを示す。それぞれのバーにマウスを合わせるとツールチップが表示され、対応する Input Parameter と実験応答に関して、適合した線形回帰モデルで近似された実際の感度係数の数値が表示される。Bar Chart および Pie Chart では、同様の情報が別の表示形式で提供され、Raw Data ビューではチャートを作成するために用いられたデータを確認できる。

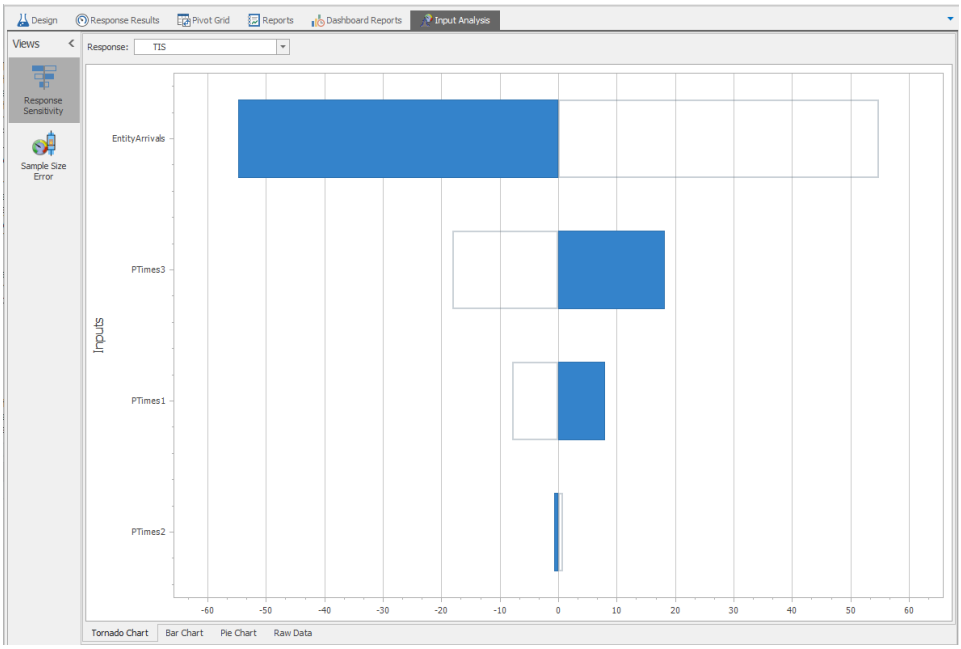


図 6.26　Model 6-2 の TIS に対する Response Sensitivity

Tornado Chart から、TIS は EntityArrivals に対して負の方向に（他の 3 つの Input Parameters よりも）比較的高い感度を示している（つまり、Entity Arrivals が上昇すると、TIS が比較的大きく下降する）。また、TIS は PTimes3（Server3 の処理時間）に対して正の方向に比較的高い感度を示していることがわかる。これは、PTimes3 が増加するにつれて、TIS も著しく増加することを意味する。さらに、PTimes1（Server1 の処理時間）に対しては正の方向に中程度の感度を示している。一方、PTimes2（Server2 の処理時間）に対しては**感度が低い**ことがわかる。したがって、入力確率分布（および出力パフォーマンス尺度の質と妥当性）を改善するために再びデータを収集する場合は、Tornado Chart に基づいて、EntityArrivals と PTimes3 に関してより多くのデータを集め、よりよい推定量を得ることに注力すべきである。次いで PTimes1 についてもデータを集める価値が

あるが、PTimes2 については追加データ収集の必要性は低いだろう。なお、これは TIS に関してのみの見解であり、チャート上部の Response フィールドのプルダウンリストから他の応答を選択する（あるいは Bar Chart や Pie Chart ですべての応答を確認する）ことで、同様の分析を行う必要がある。

前述のとおり、図 6.26 の水平青色感度バーにマウスを合わせると、TIS への回帰として予測される線形回帰係数$\hat{\beta}_j$の推定量について、以下の数値結果が表示される：

・EntityArrivals：-54.843
・PTimes3：18.225
・PTimes1：7.930
・PTimes2：-0.814

これらの数値を解釈する際に、すべての尺度の単位と線形回帰係数の推定量の意味を理解することが重要である。このモデルでは、4 つすべての入力と TIS は時間であり、測定単位はすべて分である。したがって、シミュレーションモデルの回帰（メタ）モデルでは、入力 PTimes1 が **1 分増加**すると、出力 TIS が 7.930 分増加すると予測される。同様に、EntityArrivals が 1 分増加すると、TIS が 54.843 分**減少**すると予測される（係数が負であるため）。TIS の単位を時間に変更する場合、PTimes1 の 1 分の増加により、TIS が 7.930/60＝0.132 **時間**増加する。逆に、入力 PTimes1 の単位を時間に変更すると、回帰の適合（および係数β_jの推定量の単位）も変化するため、PTimes1 の **1 時間**増加により、TIS がおよそ 60×7.930＝475.8 **分**増加すると予測される。このように、**数値** 7.930 は、選択された応答と Input Parameters の単位に基づいて解釈する必要がある。時間以外の尺度（たとえば、到着バッチのバッチサイズ）による Input Parameters がモデル内に存在する場合、その回帰における係数β_jの推定値は、バッチサイズの 1 単位増加（この「単位」は「バッチサイズ」の尺度）に起因する TIS の変化量（単位変更がなければ分単位）として解釈される。Input Parameters が何を表すにせよ（時間、バッチサイズ、その他）、適合した回帰係数は、対応する Input Parameter の **1 単位増加**（入力自体の単位）が応答に与える影響として（応答の単位で）解釈する必要がある。もちろん、回帰分析ではこれは**限界**効果であり、偏微分における**他の変数を一定に保つ**操作と同様である（他の Input Parameters を現在値で固定）。また、線形回帰では非線形応答（シミュレーションではよく**発生する**）に対して正確な線形近似を得られることは少ないため、実世界の観測値範囲を超えて予測を拡張することはできない。さらに、これらの応答感度は、シミュレーションモデルの応答が Input Parameters の+1 単位変化にどのように反応するかの概算を提供するものである。つまり、Input Parameters を変更して異なる結果を得ようとしてシミュレーションを**実行**しても、シミュレーションの**再実行**（多くの場合、大規模で複雑な反復実行）が必要となるため、Response Sensitivity で推定された予測値はあくまでも近似値である（初期シミュレーション実行の結果に依存しない）。

なお、図 6.26 の応答感度分析の**傾向**は直感的に理解できる。TIS はエンティティのシステム内時間の平均であり、待ち行列システムの**混雑度**を示す出力尺度の一つである（他にシステム内エンティティ数の時間平均や窓口利用率などがある）。Input Parameter の `EntityArrivals` は、連続するエンティティの到着時間**間隔**を表し（到着**率**ではない）、この間隔が増加すると、エンティティ間の時間間隔が広がることになり、システム内の混雑が**減少**する（負の関係）。一方、他の 3 つの Input Parameters はすべてサービス**時間**（サービス率ではない）を表すため、サービス時間が増加すればシステムの混雑度も増加する（正の関係）と予想される。このように、応答感度分析の正負の傾向はモデルの妥当性確認の簡易チェックとなる（「コード」の正確さは別問題）。ただし、応答感度分析に予想外の傾向が見られても、必ずしもモデルやコードにエラーがあるとは限らず、さらなる調査が必要である（その「違和感」がモデリングやコーディングのエラーでなければ、モデルやシステムに関する新たな知見を示している可能性がある）。応答感度分析の**傾向**は直感に基づく推測的なものだが、その**重要度**はおおむね信頼でき、近似的な定量化を行う価値がある。

6.5.2　サンプルサイズ誤差推定

Simio の Input Parameters がサポートする 2 つ目の入力解析は**サンプルサイズ誤差推定**である。Simio への実装は Song and Nelson（2015）の研究に基づいている。前述のとおり、確率的なシミュレーションは、確定的な固定値の入力だけでなく、確率分布やその他の確率過程からの入力値を用いるため、不確実性を伴う結果をもたらす。シミュレーション研究における重要な側面の一つは、実世界のシステムに対して妥当で信頼性の高い結論を得るため、適切な統計的手法を用いて出力の不確実性を評価し、必要に応じて低減することである。本書では主に、**所与の**モデル構造に固有のシミュレーション実験の不確実性（無限回の反復実行ができないことによるサンプリング誤差）に焦点を当てている。これには、本章で議論した手法を用いて、指定した確率分布から入力確率変量を生成する（**実現する**）方法も含まれる。

たとえば、Model 6-2 では、Server2 の処理（サービス）時間として、実世界の 300 の観測値から適合した分布に基づいて `Triangular(1, 2.25, 3)`分を指定している。統計的な出力解析では、出力パフォーマンス尺度（たとえばシステム内時間の平均 TIS）の標準偏差に注目する。そのばらつきを減少させるには、それぞれ 500 シミュレーション時間の 100 回を超える反復実行が必要となる可能性がある。しかし、これはすべての入力確率分布（たとえばパラメータ（1, 2.25, 3）の三角分布）が実世界のシステムにおいて**本当に正確で正しい**という仮定に基づいている。実際には、すべての入力確率分布およびそのパラメータが正確であるという**条件つきの**不確実性評価を行っている。もちろん、これらの分布やパラメータが出力尺度と実世界のシステムとの**条件つきでない**分散において**常に**正確であるとは限らない。したがって、得られた結果と実世界のシステムとの間の不確実性は、モデルの構造的な不確実性に依存し、さらに初期の入力確率分布（およびそのパラメータ）に誤差が含まれていることにも起因する（この例では、無限で

はなく 300 のデータ点しか存在しない)。Simio の Sample Size Error Estimation はこれを定量化し、総合的な無条件の不確実性を減少させるために、追加のシミュレーション実行や実世界のデータ収集にどの程度の労力を割くべきかを判断する手助けとなる。

図 6.27 に、Model 6-2 の NIS（平均システム内数）に関する Sample Size Error Estimation を示す。ここで注意すべき点は、応答感度分析とは異なり、サンプルサイズ誤差推定には追加のモデル実行が必要であり、実験の一部として自動的には行われないことである。この分析を実行するには、Sample Size Error リボンの Run Analysis を使用する。

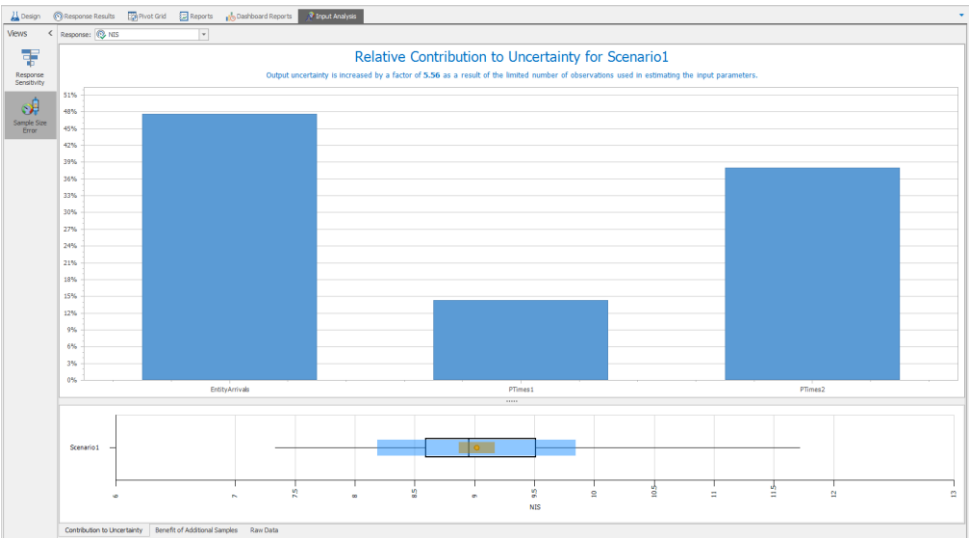


図 6.27　Model 6-2 の NIS に対する Sample Size Error

図 6.27 に示された結果は、次の 3 つの要素から構成されている。

- **半幅拡張係数**（Model 6-2 の NIS では 5.56）：有限回の反復実行を行ったモデルからの不確実性よりも、入力の推定からの不確実性が約 5.56 倍大きいことを意味する。
- 棒グラフ：包括的な無条件の不確実性に対する各 Input Parameter の相対的な寄与率を表す。このケースでは、EntityArrivals に関する不確実性が他の 2 つの入力よりも大きな要因となっている。
- 入力パラメータを（確実に知っているのではなく）推定することによって生じる不確実性に起因する拡張信頼区間（青色）を重ねた SMORE プロット（黄色）

標準 SMORE 信頼区間（黄色のボックス）にマウスを合わせると、信頼区間の半幅が 0.1492 と表示される。一方、拡張 SMORE プロット（青色のボックス）では拡張信頼区間の半幅が 0.8291 と示される。この拡張係数は、入力に関連する不確実性（確率分布を確実なものとして推定することによる不確実性）が、実験の不確実性（有限の反復実行によるサンプリング誤差）よりも 0.8291/0.1492≈5.56 倍大きいことを示している。すなわち、主要な不確実性の要因は、じゅうぶ

んな反復実行回数（またはじゅうぶんに長い反復実行）を持つシミュレーションの実行ではなく、実世界から得られた 300 サンプルサイズというデータ数が、入力確率分布を正確に定義するには不足していることに起因している。そのため、実世界のパフォーマンス NIS をより正確に推定するためには、現在のシミュレーションモデルでさらに長い反復実行を行うの**ではなく**、現場に戻り、入力分布に関するより多くのデータを収集した上で、Input Parameter を再定義することが重要である。

6.6 問題

1. Excel ファイル Problem_Dataset_06_01.xls（付録 C で説明される方法でダウンロード可能）には、コールセンタへの到着時間間隔（分）に関する 42 の観測値が記録されている。Stat::Fit（または他のソフトウェア）を使用して、これらのデータに適合度検定を実施し、確率分布のプロットを作成しなさい。1 つ以上の確率分布を適合させ、シミュレーションモデルでこれらの到着時間間隔を生成する際に推奨する分布を示しなさい。また、すべてのパラメータ設定の影響を考慮し、正しい Simio の式を示しなさい。
2. Excel ファイル Problem_Dataset_06_02.xls（付録 C で説明される方法でダウンロード可能）には、問題 1 のコールセンタにおける通話時間（分）に関する 47 の観測値が記録されている。Stat::Fit（または他のソフトウェア）を使用して、これらのデータに適合度検定を実施し、確率分布のプロットを作成しなさい。1 つ以上の確率分布を適合させ、シミュレーションモデルでこれらの通話時間を生成する際に推奨する分布を示しなさい。また、すべてのパラメータ設定の影響を考慮し、正しい Simio の式を示しなさい。
3. Excel ファイル Problem_Dataset_06_03.xls（付録 C で説明される方法でダウンロード可能）には、問題 1 および 2 のコールセンタにおいて問題解決のために（最初にコールを受けた者以外で）必要なテクニカルサポート人数に関する 45 の観測値が記録されている。Stat::Fit（または他のソフトウェア）を用いて、これらのデータに適合度検定を実施し、確率分布のプロットを作成しなさい。1 つ以上の確率分布を適合させ、シミュレーションモデルでこの通話に必要なテクニカルサポート人数を生成する際に推奨する分布を示しなさい。また、すべてのパラメータ設定の影響を考慮し、正しい Simio の式を示しなさい。
4. 実数aとb（$a < b$）の間における連続一様分布から確率変量を生成するための逆 CDF 法の式を導出しなさい。
5. ワイブル分布から確率変量を生成するための逆 CDF 法の式を導出しなさい。本章で参照した書籍またはオンライン資料を使用して、ワイブル分布の定義（CDF を含む）を調べなさい。式のパラメータが適切に設定されているかどうかを確認するため、Simio のマニュアルでその定義を確認しなさい。
6. モードをmとするaとb（$a < m < b$）間の三角分布の確率変量を生成するための逆 CDF 法の式を導出しなさい。必要に応じて、式をいくつかの部分に分けることに注意してほしい。式のパラメータが適切に設定されているかどうかを確認するため、Simio のマニュアルでその定義を確認しなさい。
7. 6.4 節の議論および図 6.22 で扱った、任意の離散確率変量を生成するための一般的な逆 CDF 法を思い出してほしい。Xを、0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.5, 10.0の値を、それぞれ確率0.05, 0.07, 0.09, 0.11, 0.15, 0.25, 0.10, 0.09, 0.06, 0.03でとる離散確率変数とする。Xの期待値および標準偏差を正確に（小数点以下 4 桁まで）計算しなさい。その後、プログラミング言語でプログラムを書くか、あるいは Excel を使用して、最初の$n = 100$を生成し、さらに別に独立した$n = 1{,}000$の IID 変量を生成しなさい。それぞれのnの値

について**標本**平均および**標本**標準偏差を計算し、誤差および精度の両面から正確な値と比較しなさい。さらに、得られた比較の結果に対してコメントしなさい。この問題に取り組むために必要であれば、組み込みの乱数生成法あるいはより便利な手段のいずれを用いてもよい。Excel を使用する場合は、VLOOKUP 関数の活用を推奨する。

8. 3 章の問題 18 で行った農作物露天商のスプレッドシートシミュレーションを再度行う。今回は、日々の需要量についてより正確な確率質量関数を使用する。ワルサーは需要量に関して良好なデータを記録している。また、顧客はすでにパッケージされた重量でのみ購入できる。オート麦の場合、本章の問題 7 の分布を用い、パッケージの量は $0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.5, 10.0$（単位はポンド）である（0 は顧客が購入しないことを意味する）。えんどう豆は、パッケージの量が$0, 0.5, 1.0, 1.5, 2.0, 3.0$ポンドで、それぞれの需要の確率は$0.1, 0.2, 0.2, 0.3, 0.1, 0.1$である。豆は、パッケージの量が$0, 1.0, 3.0, 4.5$ポンドで、それぞれの需要の確率は$0.2, 0.4, 0.3, 0.1$である。大麦は、パッケージの量は$0, 0.5, 1.0, 3.5$ポンドで、それぞれの需要の確率は$0.2, 0.4, 0.3, 0.1$である。各商品に関して、シミュレートされた日々の需要量を生成する方法については、本章の問題 7 を参照してほしい。

9. 毎月のクレジットカードの請求額に対して、支払利息を表す確率変量を生成するために、入力として乱数だけを用いた計算アルゴリズムを書きなさい。60%の確率で利息が 0 になる（つまりその確率でカード所有者が期日までに全額を完済する）。そうでない場合、支払利息は 20 ドルと 200 ドルの間の一様確率変量となる。次に、適切な Simio の確率変量の式を使用して、これを実現する Simio の式を開発しなさい。

10. 3.2.3 項で、整数$1000, 1001, \ldots, 5000$の間に一様に分布する帽子の需要を生成するために開発した方法が、（検索不要でより効率的に実施できることを除いて）この分布に対する逆 CDF アルゴリズムと同一であることを示しなさい。

第 7 章　モデルのデータ操作

モデルではさまざまな種類のデータが使われている。これまでは、データを入力する際、多くの場合 Standard Library オブジェクトのプロパティに直接入力する方法を採用してきた。たとえば、到着時間間隔の平均値を Source オブジェクトに直接入力したり、処理時間の分布をパラメータとして Server オブジェクトに直接入力したりしていた。一部のデータに対してこの方法で問題ないものの、多くの場合は別のアプローチが必要となる。特に、時間経過とともに到着パターンが変化するような特殊なデータに対しては、特別な表現方法が求められる。データ量が多い場合や、より効率的な形式でデータを表現する必要がある場合は、外部ソースからデータを取り込む必要も生じるだろう。また、モデルを使用するアナリストがモデルの作成者と異なる場合もある。このような状況では、データをモデル内に散在させるのではなく、一元管理することが重要となる。本章では、さまざまな種類のデータについて説明し、これらを最適に表現するための Simio の機能を探究していく。

7.1　データテーブル

Simio の**データテーブル**（Data Table）は、スプレッドシートに似た形式で、プロパティを表す列とデータを表す行で構成される矩形のデータマトリックスである。各列は 1 つのプロパティを表し、Simio が提供する 50 種類以上のデータタイプから選択できる。これには、標準プロパティ（Integer（整数）、Real（実数）、Expression（数式）、Boolean（論理値））、エレメント参照（Tally Statistics、Station、Material）、オブジェクト参照（Entity、Node List）が含まれる。各行は特定のエンティティタイプ、オブジェクト、またはデータの体系的な側面を表現できる。

データテーブルは外部ファイルにインポート、エクスポート、さらにはバインド（7.1.7 項参照）することが可能である。アクセス方法は順次、ランダム、直接、さらには自動アクセスまで対応している。テーブル間に関連づけ（リレーション）を設定すれば、あるテーブルのエントリから別のテーブル内のデータを参照することもできる。基本的なテーブルに加えて、Sequence Table や Arrival Table といった特殊用途向けのテーブルも用意されている。これらについては本節で詳しく説明する。

モデル実行中にファイルを読み書きすることは、実行速度を大幅に低下させる可能性がある。一方、テーブルはデータをメモリ上に保持するため、高速なアクセスが可能である。Simio ではテーブル数に制限がなく、各テーブルの列数も行数も制限されない（システムメモリの制約を除く）。テーブルはデータの整理、表現、使用だけでなく、外部データとの連携にも効果的である。

7.1.1　テーブルの基本

データテーブルは、Data ウィンドウの Table Tools パネルで定義する（図 7.1）。

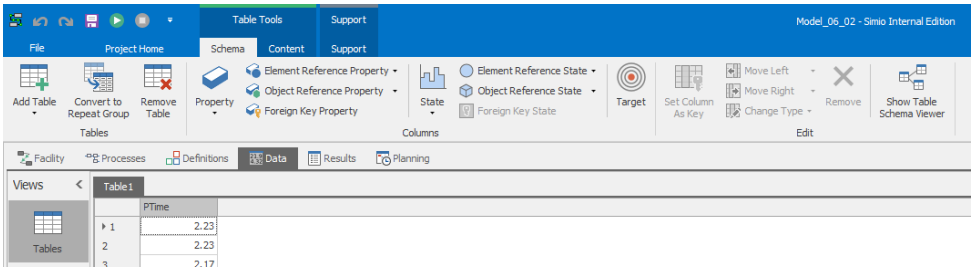


図 7.1　Model 6-2 の Table Tools パネル

このパネルには「Schema」と「Content」という 2 つのリボンがあり、テーブル構造の編集ツールとテーブルコンテンツ関連ツールが分かれている。新規テーブルを追加するには、Schema リボンの Tables セクションの Add Table をクリックする。テーブル追加後、そのタブをクリックすると、Property ウィンドウでテーブル名の変更や説明の追加など、テーブルのプロパティを設定できる。

ヒント：複数のテーブルを追加した場合、それぞれのテーブルが独自のタブを持つ。テーブルが多数ある場合は、タブ行の右端にプルダウンリストが表示され、そこに現在非表示中のタブの一覧がリストされている。4.1.8 項でウィンドウの設置方法を説明したが、その方法はデータテーブルの操作にも適用できる。

テーブルに列を追加するには、まずテーブルを選択し、アクティブにしておく。次に Standard Property、Element Reference、Object Reference、Foreign Key からプロパティのタイプを選択してクリックする。テーブルの列は、一般に表 7.1 の Standard Properties で表現される。

テーブルで Entity、Node、Transporter などのオブジェクトのインスタンスやモデルオブジェクトのリストを参照してデータを取得する場合は、Object Reference を使用する。また、TallyStatistic や Material などの特定のエレメントを参照する場合は、Element Reference を使用する。各列の名前、位置、データ型、その他のプロパティを定義することで、データテーブルの **Schema（スキーマ）**が定義される。Simio では、ユーザのニーズに合わせてスキーマを定義できる。また、アクセスしやすいスキーマを選択したり、外部ファイルのスキーマと一致するものを定義して、データを変換の有無に関わらずインポートして使用することができる。

Simio バージョン 16.256 からは、データテーブル用の**ヒストグラムツール**を導入した（図 7.2）。このツールは Table Tools リボンの Content セクションからアクセスでき、選択したテーブルの数値型列から標準的な度数ヒストグラムと基本的な記述統計を生成できる。ヒストグラムウィンドウは非モーダルなので、複数のウィンドウを同時に開いて比較できる。このツールは基本的な機能のみを提供し、ビンサイズの指定などのカスタマイズはできないが、データは容易にエクスポートでき（7.1.7 項参照）、Minitab などの統計分析ツールを使用してより詳細な分析が可能である。

表 7.1　データテーブルの列で使用可能な Simio Standard Properties

プロパティのタイプ	説明
Boolean	真（1 または非ゼロ）または偽（0）
Color	色の設定値
Changeover Matrix	転換行列への参照
Date Time	特定の日時（例：7:30:00 November 18, 2010）
Day Pattern	スケジュール内の Day Pattern への参照
Enumeration	定義済みリストから選択可能な値のセット
Event	ステップから解放されたトークンを発火するイベント
Expression	実数値として評価される式（例：1.5+MyState）
Integer	整数値（例：5 や-1）
List	文字列リストから選択可能な値のセット
Rate Table	レートテーブルへの参照
Real	実数値（例：2.7 や-1.5）
Schedule	スケジュールへの参照
Selection Rule	選択ルールへの参照
Sequence Destination	エンティティまたはノードへの参照
Sequence Number	整数またはドット区切り整数形式のシーケンス番号
Sequence Table	シーケンステーブルへの参照
State	状態変数への参照
String	文字列（例：Red や Blue）
Table	データテーブルまたはシーケンステーブルへの参照
Task Dependency	タスク間の依存関係の定義

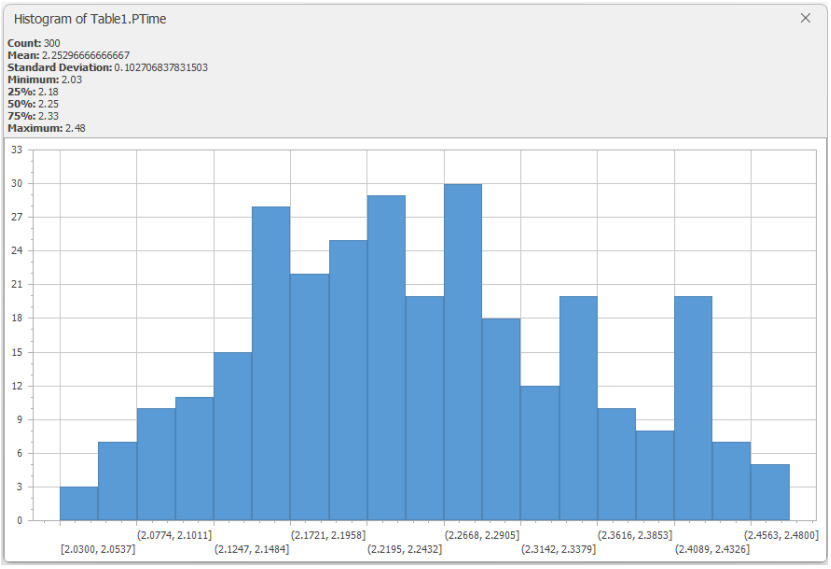


図 7.2　Model 6-2 の PTime1 のヒストグラム

7.1.2　Model 7-1：データテーブルを用いた ED モデル

前述したテーブルの概念を、簡単なヘルスケアの事例を用いて説明する。この事例では、救急診療部門（ED；Emergency Department）において、重症度の異なる患者への処置方法に関する既知のデータを活用する。表 7.2 に示すように、この ED では 4 種類の患者区分があり、それぞれに優先順位と標準的な治療時間が設定されている。

表 7.2　Model 7-1 の救急診療病院における患者基本データ

患者タイプ	優先順位	治療時間（分）
Routine	1	Random.Triangular(3, 5, 10)
Moderate	2	Random.Triangular(4, 8, 25)
Severe	3	Random.Triangular(10, 15, 30)
Urgent	4	Random.Triangular(15, 25, 40)

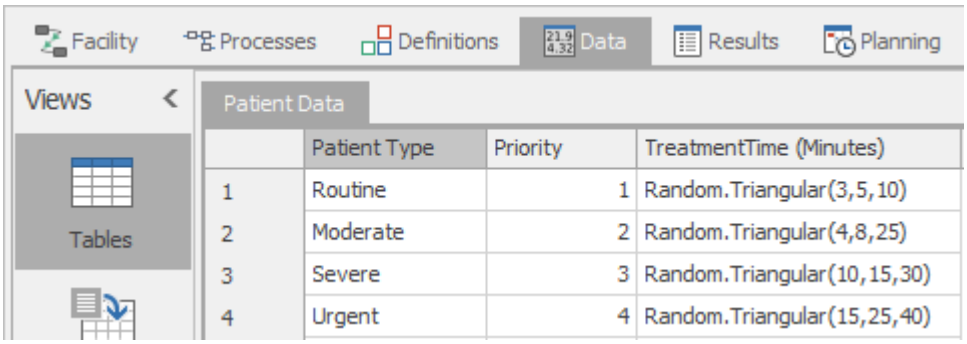
モデル構築のはじめに、エンティティとデータテーブルを定義する：

1. Simio を起動し、新しいモデルを作成する。モデルの Facility ビューで 4 つの ModelEntity インスタンスをドラッグする。各インスタンスを選択し、プロパティから（または F2 キーを押して）名前を Routine、Moderate、Severe、Urgent に変更する。
2. 4.8 節の手順に従って、エンティティを視覚的に区別しやすくする。すべてのエンティティはデフォルトのシンボルを使用するが、各インスタンスに異なる色を設定して区別する。まず、シンボルが見やすいようにズームインする。Moderate エンティティを選択し、Symbols リボンの右側にある Color ボタンの下半分をクリックしてカラーパレットを表示する。黄色を選択し、Moderate エンティティをクリックして適用する。同様に Severe 患者にはライトブルー、Urgent 患者には赤色を適用する。Routine はデフォルトの色のままとする。
3. 次に、データテーブルを作成する。Data タブを選択し、左側の Tables パネルを選択する。アクティブなテーブルがない場合、Table リボンは表示されているが、ほとんどのボタンは使用できない。Add Data Table ボタンをクリックして空のテーブルを追加し、プロパティウィンドウの Name プロパティで名前を PatientData に変更する（名前にスペースは使用できない）。
4. 3 つの列を追加する。最初の列はエンティティタイプを参照するため、リボンで Object Reference をクリックして Entity を選択する。これで Entity Instance 列が作成された。プロパティウィンドウの Name プロパティ（Display Name **ではない**）で PatientType と名前を変更する。2 列目は整数型のため、Standard Property から Integer を選択し、Name プロパティを Priority に変更する。3 列目は Standard Property から Expression を選択し、Name プロパティを TreatmentTime とする。この列は時間を表すため、プロパティウィンドウの Logic カテゴリで Unit Type を Time、Default

Units を Minutes に設定する。

5. テーブルの構造が定義できたので、表 7.2 のデータを 4 行分追加する。データは行方向または列方向のどちらでも入力できるが、ここでは行方向で入力する。左上のセルをクリックすると 4 つのエンティティタイプのリストが表示される（リストが表示されない場合は、前のステップの設定を見直す）。1 行目の Patient Type で Routine を選択し、Priority 列に 1 を入力する。Tab または Enter キーで TreatmentTime 列に移動し、表 7.2 の式 Random.Triangular(3, 5, 10)を入力する。ヒント：データが部分的に隠れる場合は、列の右端をダブルクリックすると必要な幅まで自動的に調整される。同様の手順で、表 7.2 の残りのデータを入力する。

図 7.3 に完成したテーブルを示す。これで、モデルで使用する患者の関連データが定義できた。次節では、テーブルからのデータアクセス方法について説明する。



	Patient Type	Priority	TreatmentTime (Minutes)
1	Routine	1	Random.Triangular(3,5,10)
2	Moderate	2	Random.Triangular(4,8,25)
3	Severe	3	Random.Triangular(10,15,30)
4	Urgent	4	Random.Triangular(15,25,40)

図 7.3　Model 7-1 の救急患者の基本データテーブル

7.1.2.1　データテーブルの参照

テーブルのデータは、TableName[RowNumber].ColumnName または TableName[RowNumber, ColumnNumber]という構文で参照する。この際、テーブル名、行番号、列名（または列番号）を指定する。この構文を使ってデータテーブルを参照しながら、モデルの構築を進めていく。たとえば、PatientData[3].TreatmentTime という構文で、Severe 患者の治療時間を簡単に参照できる。多くの場合、特定のエンティティは**常に**特定のデータ行を参照する。先ほどの例では、患者タイプと対応する行を決定すると、そのエンティティは常にその行からデータを参照するように指定される。このような参照関係の追跡は、独自のプロパティや状態変数で管理することもできるが、Simio にはすでにこの機能が組み込まれている。この機能を使用するもっとも簡単な方法は、Source オブジェクトの Table Row Referencing カテゴリで属性を設定することである。

この例では、4 種類の患者がそれぞれのタイプごとに異なる到着パターンを持つように設定できる。まずは患者タイプを作成するための Source オブジェクトを作成する。図 7.4 は、Severe 患者の Source オブジェクトに対して、テーブル名と行番号を指定する方法を示している。エンティティと特定の行が関連づけられている場合、TableName.ColumnName という簡潔な構文でテーブルのデータを

参照できる。この場合、行番号はすでに設定されているため、たとえば Patient Data.TreatmentTime を使用して**任意**の患者タイプの治療時間を参照できる。

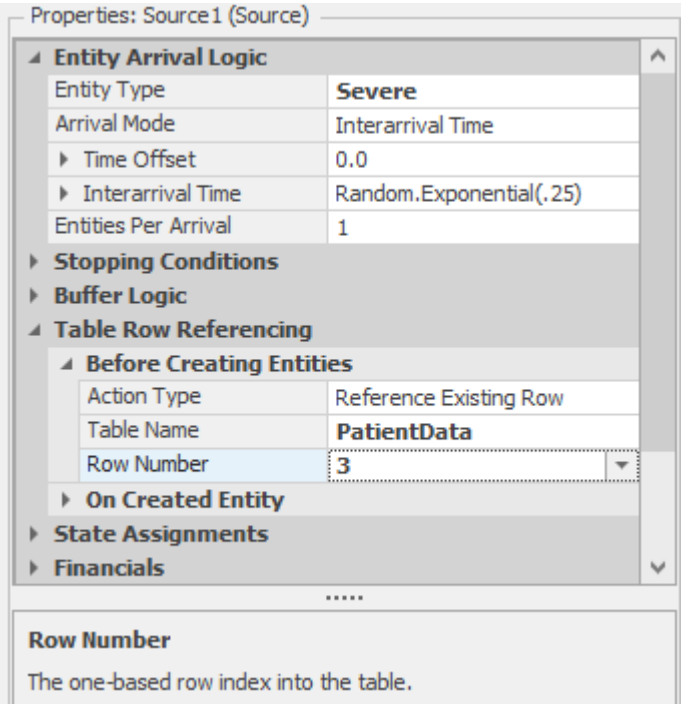


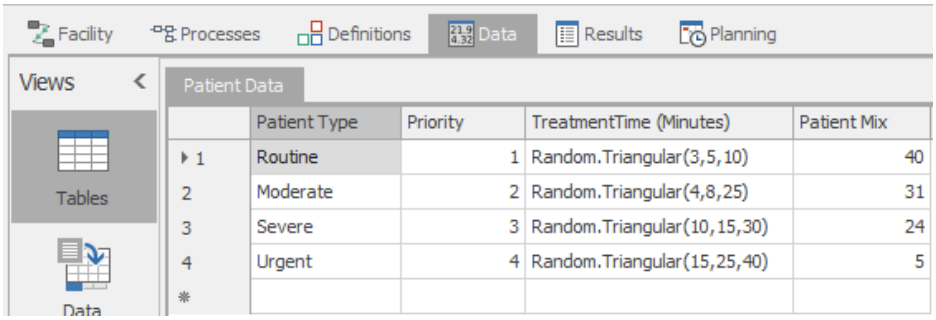
図 7.4　エンティティと特定の行との関連づけ

7.1.2.2　エンティティタイプの選択

このモデルを完成させる前に、テーブルのもう一つの重要な側面について探究する。テーブルでは、これまでのモデルのように 1 つの行が 1 つのエンティティタイプに対応するのが一般的だが、エンティティタイプをランダムに選択することもよくある。Simio では、同じ構成でこの 2 つを同時に実現できる。たとえば、各行（エンティティタイプ）にウェイトを指定する列を追加できる。また、TableName.ColumnName.RandomRow という式を使用すれば、特定の列から行をランダムに選択することができる。

では、以下のステップに従ってモデルを完成させる。

1. 救急診療部門の過去のデータによると、患者の割合は次のとおりである：Routine（40%）、Moderate（31%）、Severe（24%）、Urgent（5%）。この情報をテーブルに追加するため、Data タブの Tables パネルに戻り、リボンから Standard Property の Real を選択する。Name プロパティを PatientMix に変更し、新しい列に上記の患者データを入力する。完成したテーブルは図 7.5 のようになる。5.2 節で説明したように、Simio は Patient Mix 列の値を**相対的**割合として解釈する。ここでは**パーセンテージ**で値を入力したが、**確率**（0.40, 0.31, 0.24, 0.05）や、これらの倍数値（例：4000, 3100, 2400, 500）でも構わない。



	Patient Type	Priority	TreatmentTime (Minutes)	Patient Mix
1	Routine	1	Random.Triangular(3,5,10)	40
2	Moderate	2	Random.Triangular(4,8,25)	31
3	Severe	3	Random.Triangular(10,15,30)	24
4	Urgent	4	Random.Triangular(15,25,40)	5
*				

図 7.5　Model 7-1 の救急患者データの拡張テーブル

2. モデルの構築を続けよう。先の変更により、1 つの Source で特定の分布に従う 4 種類の患者を生成できるようになった。まず、Source をモデルに配置し、Interarrival Time を `Random.Exponential(4)`、Units を `Minutes` と指定する。図 7.4 では特定の行番号で Entity Type プロパティの患者タイプを指定する方法を示したが、ここでは別の方法を採用する。Simio に行番号を選ばせ、その行の PatientType 値に応じて Entity Type を決定する方法である。エンティティ生成前にエンティティタイプを決定する必要があるため、テーブルの行を事前に選択しておく必要がある。そこで、Table Reference Assignment の Before Creating Entities カテゴリで、Table Name を `PatientData` に、Row Number を `PatientData.PatientMix.RandomRow` と指定する。行の選択後、Source は Entity Type プロパティを参照してエンティティタイプを決定する。ここでは Entity Type のプルダウンリストから `PatientData.PatientType` を選択する。図 7.6 にこれらの設定結果を示す。

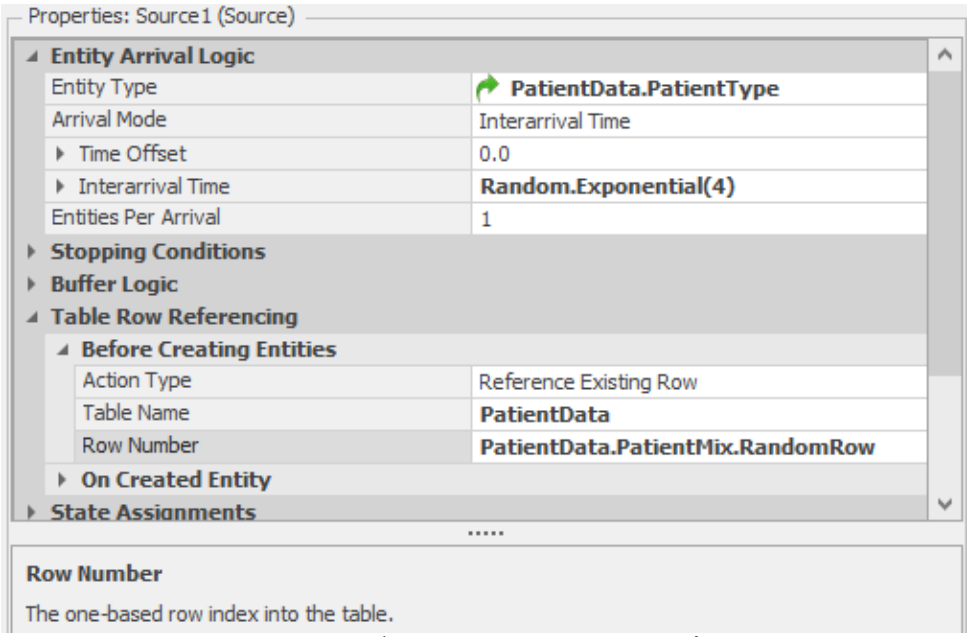


図 7.6　テーブルからのエンティティタイプ選択

3. ここまできたら、モデルの完成まであと少しである。Server を 1 つ追加して、Initial Capacity を 3 とし、Processing Time を PatientData.TreatmentTime で指定する。この例ではテーブルのデータを使用しているが、短縮参照法（Short Reference Method）を使用していることに留意してほしい。ここでは、明確的な行を指定していないため、個々のエンティティに関連づけられている行を Simio に指示することになる。タイプが Routine のエンティティが到着すると、1 行目の Random.Triangular(3, 5, 10)分布からサンプリングされた診療時間を使用する。他方、タイプが Severe のエンティティが到着すると、3 行目の Random.Triangular(10, 15, 30)分布からサンプリングされた診療時間を使用することになる。さらに Sink を 1 つ追加してから、Source と Server、そして Server と Sink の間を Path でつなげる。図 7.7 に完成後のモデルのイメージを示す。

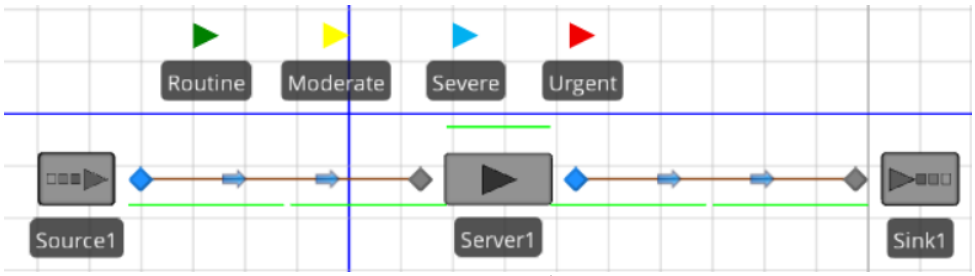


図 7.7　完成した ED モデル Model 7-1

モデルを拡張する前に、データテーブルが正しく実装されているか確認するため、簡単な検証を行う必要がある。患者タイプの割合と各タイプの予想サービス時間から、全体の予想サービス時間（11.96 分）を算出できる。患者の総到着率が時間当たり 15 人であることから、定常状態では 99.64%のサーバ稼働率が予測される。Model 7-1 を 25 日間のウォームアップ期間の後、200 日間を 25 回繰り返して実行したところ、スケジュール稼働率は 99.67%±0.1043（95%信頼区間の半幅）となった。これは重要な検証上の教訓を示している：モデルの妥当性確認は、完成を待つよりも構築過程で行うほうが容易な場合が多い。観測された稼働率は予測値とよく一致しており、患者データのテーブルが正確に実装されたことが確認できた。これでモデルの拡張に進むことができる。

7.1.3　シーケンステーブル

シーケンステーブルは、エンティティの目的地を順序立てて指定するための特殊なデータテーブルである。たとえば、ジョブショップ型の生産工場では、部品の完成に必要な工程（研削、研磨、組立、配送など）の順序を表す。また、交通ネットワークでは、バス路線の停留所（メインストリート、5 丁目、9 丁目、山の手など）の順序を示すために使用される。

シーケンステーブルは、Data ウィンドウの Tables パネルで Add Sequence Table ボタンを使って作成する。作成時に、エンティティのルーティング順序を指定する Sequence 列が自動的に追加される。この必須列が、通常のテーブルと

の主な違いである。設定方法は通常のテーブルとほぼ同じで、値の参照も同様に行える（TableName.PropertyName の形式で）。

シーケンステーブルには、**単純シーケンステーブル**と**リレーショナルデータシーケンステーブル**の 2 種類がある。単純シーケンステーブルは、1 つのシーケンスや 1 種類のエンティティタイプのみを扱う場合に適している。一方、リレーショナルデータシーケンステーブルは、複数のエンティティが同じオブジェクトを異なる順序でたどるような複雑なケースに適している。使用方法は同じだが、設定方法が異なる。まずは単純シーケンステーブルの設定と使用方法から説明する。

7.1.3.1　単純シーケンステーブル

単純シーケンステーブル 1 つにつき、1 つの経路計画を定義する。複数の経路計画（例：複数のバス路線）がある場合は、それぞれに対応するシーケンステーブルを定義する。各行は特定の場所を表し、その場所固有の処理時間や優先度などのプロパティを設定できる。

シーケンステーブルの作成後、エンティティとの関連づけが必要となる。もっとも簡単な方法は、モデル内のエンティティインスタンスで設定する方法である。エンティティインスタンスの Routing Logic カテゴリにある Initial Sequence プロパティで、使用するシーケンステーブルを指定する。エンティティは 1 行目から開始し、指定されたステーションを順に訪問する。この動きは通常自動的だが、アドオンプロセスの SetRow ステップを使用して、現在の行や使用するシーケンスを変更することもできる。

ここで「エンティティはどのように次のステップに進むタイミングを判断するのか」という疑問が生じるかもしれない。シーケンスでは、エンティティに次の目的地を指示する必要がある。この指示は通常、Transfer ノードの Routing Logic カテゴリで行う（Standard Library の全オブジェクトの出力ノードは Transfer ノードである）。Entity Destination Type を By Sequence に設定すると、次の 3 つの処理が実行される：

1. エンティティのシーケンステーブルで次の行が作成される。
2. エンティティの目的地がその行で指定された場所に設定される。
3. 他のテーブルで指定したプロパティ（ProcessingTime など）が、新しい行の値を参照する。

エンティティの移動タイミングを明示的に制御できるため、必要に応じてシーケンス以外のステーションも訪問できる。Entity Destination Type を By Sequence 以外（例：Specific）に設定するだけでよい。これは複数回実行可能で、By Sequence を使用すればその場所からシーケンスを再開できる。

7.1.3.2　リレーショナルシーケンステーブル

前述した多くの概念は、リレーショナルシーケンステーブルにも当てはまる。リレーショナルテーブルの使用方法は基本的に同じだが、構成方法が異なる。主な違いは、1 つのシーケンステーブルに複数の異なるシーケンス（例：特定のエンティティの訪問セット）を組み合わせられることである。エンティティのインス

タンスで Initial Sequence プロパティを使う代わりに、別のテーブルでシーケンス情報を提供できる。この機能は、キーと外部キー（Foreign Key）として定義された特別な列を介して、関連するシーケンステーブルをリンクすることで実現される。リレーショナルテーブルの基本的な使用方法は Model 7-2 で示され、より詳細な説明は 7.1.6 項で扱う。

7.1.4　Model 7-2：シーケンステーブルを用いた拡張 ED モデル

前述の ED モデル（Model 7-1）をより詳細で興味深いものに拡張する。患者の流れは以下のようになる：すべての患者はまず受付を訪問する。Urgent（緊急）タイプの患者を除き、全員が診療受付エリアに向かう。診療受付後、患者はもっとも早く利用可能な診察室へ移動する。診察後、Routine（通常）タイプの患者は退出するが、それ以外の患者は追加治療のために施設内に残る。一方、Urgent タイプの患者は受付後、重症度の高い症状の治療のため直接検査室へ向かう。当該患者は症状が安定するまで検査室で治療を受け、その後治療室に移動して退出する。

Model 7-1 の構築時、最初にエンティティを配置したことを覚えているだろうか。一般的に、モデル構築はテーブルで参照される各オブジェクトの配置から始まる。今回も同様に、まずオブジェクトを配置していく。シーケンステーブルは主に場所（具体的には、オブジェクトの入力ノード）を参照するため、それらの場所を表す Server オブジェクトの配置から始める。その後、新しいテーブルを作成し、モデルにプロパティを追加する作業に移る。

1. Model 7-1 を開く。Server と Sink の間のパスが不要なため削除する。Server 1 のプロパティで、`Processing Time` を右クリックして `Reset` を選択する。
2. Standard Library から `Server` をダブルクリックし、モデルの作業エリアで 4 回クリックして 4 つの Server を追加する。Esc キーを押すか右クリックして配置モードを終了する。追加した 5 つの Server に次の名前を設定する：`SignIn`、`Registration`、`ExamRooms`、`TraumaRooms`、`TreatmentRooms`。Sink を 1 つ追加し、2 つの Sink の名前を `NormalExit` と `TraumaExit` にする。
3. Data ウィンドウの `Tables` パネルに移動する。シーケンステーブルが参照できるよう、既存の PatientType 列を唯一の Primary Key（主キー）として指定する。`PatientType` 列を選択し、リボンの `Set Column as Key` をクリックする。ここで、`TreatmentTime` 列を削除する（後ほど新しい ServiceTime 列に置き換える）。削除は、`TreatmentTime` 列の見出しをクリックし、リボンの `Remove Column` をクリックする。
4. これでシーケンステーブルを作成できる。リボンの `Add Sequence Table` をクリックし、シーケンステーブルのプロパティエリアで名前を `Treatments` に設定する。Sequence 列が自動的に追加されることを確認する。
5. 治療の種類を識別するため、別の列を追加する。リボンから `Foreign Key` をクリックし、特定の治療法を一意に識別する列を作成する。この例では、治療の種類は PatientData テーブル内の PatientType と対応している。新しい列のプロパティで名前を `TreatmentType` とする。これは外部参照のため、

外部から値を取得する。Table Key プロパティで、Treatment Type の値の取得元として PatientData.PatientType を選択する。

6. 最後に、Treatments テーブルに処理時間の列を追加する。リボンの Standard Property から Expression を選択する。これにより、各場所での滞在時間に使用する数値や式を入力するプロパティが作成される。プロパティウィンドウで、Name（Display Name ではない）をより汎用的な ServiceTime に変更する。この列は時間を表すため、ServiceTime のプロパティの Logic カテゴリで、Unit Type を Time に、Default Units を Minutes に設定する。
7. データ入力を効率化するため、もう 1 つ設定を変更する。Sequences 下の最初のセル内のプルダウンリストには、モデル内の全ノードが表示される。このリストから目的地を指定できる。現時点で独立したノードは存在しないが、これを使用して、リストをオブジェクトの入力ノードのみに制限する。このオプションではオブジェクト名が表示され、選択が容易になる。Sequence 列をクリックし、プロパティウィンドウで Accepts Any Node を False に変更する。プルダウンリストを確認すると、より簡潔なリストが表示される。
8. 次に治療データを入力する。1 行目では、Sequence の下のプルダウンメニューから SignIn を選択し、Treatment Type の下のプルダウンリストから Routine を選択して、Service Time 欄に 2 を入力する。図 7.8 に示すように、同様の手順で残りの行を追加してデータを入力する。
9. データの入力が完了したら Facility ウィンドウに戻ってモデルを完成させる。
10. これまでの議論を振り返ると、エンティティの移動がシーケンスに従う場合、各出力ノードでその設定が必要となる。まず、Source の出力側にある青色の TransferNode をクリックし、Routing Logic カテゴリで Entity Destination Type 欄を By Sequence に変更する。この設定は各出力ノードで必要だが、Simio には便利なショートカットがある。この例ではすべてのエンティティの移動がシーケンスに従うため、一括変更が可能である。任意の青いノードをクリックし、キーボードの Ctrl キーを**押しながら**他のすべての青いノード（計 6 カ所）を選択する。これにより、選択した 6 つのノードの Entity Destination Type を一度に変更できる。
11. 次に、使用される可能性のあるすべてのノードをパスで接続する。不要なパスを追加しても、作業が冗長になるだけで問題は生じない。たとえば、Urgent タイプの患者は Registration をスキップするため、Registration から TraumaRooms へのパスが使用されることはない。
12. ここで、患者のサーバ滞在時間（サーバの Process Time プロパティ）について設定する。すべてのサーバで同じ方法を使用し、各患者に紐づけられたシーケンスのアクティブステップの情報を活用する。この式は TableName.PropertyName 構文で表現され、この例では Treatments.ServiceTime となる。この式をサーバの Process Time 値として入力する。エンティティ（患者）の情報は、現在のテーブルと行に自動的に関連づけられる。

Patient Data　Treatments

	Sequence	Treatment Type	ServiceTime (Minutes)
1	SignIn	Routine	2
2	Registration	Routine	Random.Uniform(3,7)
3	ExamRooms	Routine	Random.Triangular(5,10,15)
4	NormalExit	Routine	0.0
5	SignIn	Moderate	2
6	Registration	Moderate	Random.Uniform(3,7)
7	ExamRooms	Moderate	Random.Triangular(10,15,20)
8	TreatmentRooms	Moderate	Random.Triangular(5,8,10)
9	NormalExit	Moderate	0.0
10	SignIn	Severe	1
11	Registration	Severe	2
12	ExamRooms	Severe	Random.Triangular(15,20,25)
13	TreatmentRooms	Severe	Random.Triangular(15,20,25)
14	NormalExit	Severe	0.0
15	SignIn	Urgent	.5
16	TraumaRooms	Urgent	Random.Triangular(15,25,35)
17	TreatmentRooms	Urgent	Random.Triangular(15,45,90)
18	TraumaExit	Urgent	0.0
*			

図 7.8　Model 7-2 における Treatments 定義用のリレーショナルシーケンステーブル

13. 患者の優先順位について考える。これは社会的地位ではなく、症状の重症度に基づく。たとえば、重度の心臓発作の患者がいる場合、軽い咳や風邪の患者の治療を優先すべきではない。患者テーブルですでに優先順位を定義しているが、サーバの選択基準をデフォルトの先入先出（FIFO；First In First Out）から変更する必要がある。各サーバの Ranking Rule 属性を Largest Value First に設定し、テーブルの Ranking Expression 属性を Patient Data.Priority に設定する。これにより、Urgent タイプの緊急患者（優先順位 4）が他の患者（優先順位 1～3）より優先的に治療を受けられる。
14. このモデルのサーバは個別の窓口ではなく、救急診療部門の各エリアを表す。各サーバの Initial Capacity を表 7.3 に示される値に設定する。サーバ上部の緑のラインは現在処理中のエンティティを表し、キャパシティに合わせて長さを調整できる。また、サーバ左側の線は待機中の患者を表し、こちらも必要に応じて長さを調整できる。これらの線はモデル実行中でも自由に調整可能である。

表 7.3　救急診療部門（ED）の各エリアのサーバ数

サービスエリア	初期キャパシティ
SignIn	1
Registration	3
ExamRooms	6
TreatmentRooms	6
TraumaRooms	2

すべての手順に従った場合、モデルは完成している。オブジェクトの配置は多少異なる可能性があるが、おおむね図 7.9 のようになる（Browse ウィンドウは非表示）。

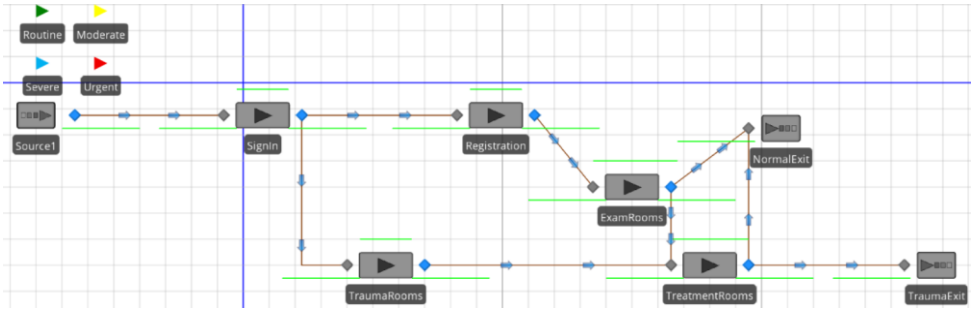


図 7.9　シーケンステーブルを適用後の ED モデル Model 7-2

これまでと同様に、モデルの妥当性を検証する必要がある。検証では、待ち行列ネットワークモデル（図 7.10）を用いて 5 つのサーバの定常状態における期待稼働率を見積もる。表 7.4 は、待ち行列ネットワークモデルから得られた期待稼働率と、Model 7-2 の実行結果を比較している。実行結果は、ウォームアップ期間 25 日間、実行期間 200 日間の設定で、25 回のシミュレーションを繰り返した結果である。表 7.4 が示すように、実験結果は期待値とよく一致している。このモデルの妥当性が確認できたため、さらなる拡張に進むことができる。

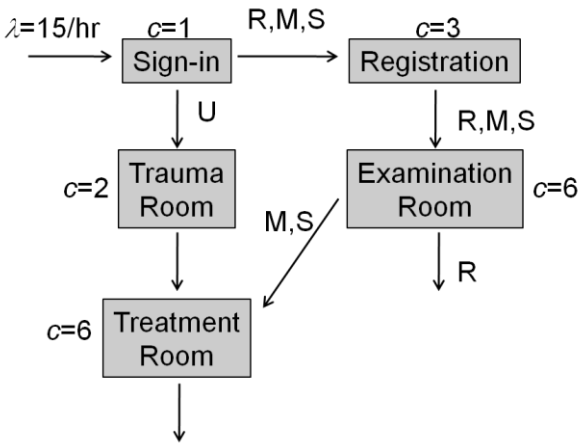


図 7.10　シーケンスを用いた ED モデルの待ち行列ネットワーク

表 7.4　Model 7-2 の検証結果

サーバ	期待稼働率	シミュレーション結果
Sign-in	0.4213	0.4214 ± 0.0797
Registration	0.3358	0.3359 ± 0.0678
Trauma Rooms	0.1563	0.1567 ± 0.1113
Examination Rooms	0.5604	0.5608 ± 0.1090
Treatment Rooms	0.4032	0.4038 ± 0.1132

7.1.5　到着テーブルと Model 7-3

これまで説明していないテーブルの種類として**到着テーブル**がある。これは、Source オブジェクトの Arrival Mode プロパティと組み合わせて使用し、テーブルで指定された特定の到着パターンを生成する。この到着は確率的または確定的のいずれかとすることができる。

一般的な確率論的シミュレーションモデルでは、**確定的到着**（Deterministic Arrivals）はあまり一般的ではないが、スケジューリング（Scheduling）と検証（Validation）という 2 つの重要な用途がある。スケジューリングでは、既定の作業を最適な方法で処理してシステム目標を達成することを目指す。一方、検証では既知のイベント群をモデルで実行し、実際の結果とモデルの結果を比較分析する。どちらの場合も、到着テーブルを使用して入荷作業や資材配達などの活動を表現できる。テーブルの各エントリは生成されるエンティティに対応する。

Source オブジェクトには 2 つの追加プロパティがあり、同じテーブルを**確率的到着**（Stochastic Arrivals）にも利用できる。Arrival Time Deviation（到着時間差異）では指定した到着時刻の前後における確率的なばらつき（Random Deviation）を設定でき、Arrival No-Show Probability では特定の到着が発生しない確率を設定できる。これらの属性により、患者（後述）、入荷原材料、配達トラックなどの到着予定を容易にモデル化できる。なお、Run タブの Advanced Options にある `Disable Randomness` オプションを選択すると、モデルは確定的モードで実行され、他のモデル要素も同様に確率的な機能が無効になる。

到着時刻のリストを含む列があれば、どのテーブルでも到着テーブルとして使用できる。到着テーブルは到着時間の提供だけでなく、期待されるルーティングやパフォーマンス基準を決定するためのプロパティやデータも保持できる。それでは、この ED モデルにおける到着テーブルの使用方法を説明しよう。まず、Model 7-2 に到着時間と患者タイプの列を持つテーブルを追加する。

1. Data ウィンドウの Tables パネルを開き、`Add Data Table` をクリックして、`TestArrivals` という名前をつける。
2. `Standard Property` をクリックして `Date Time` を選択し、属性名を `ArrivalTime` とする。
3. 次に、`Foreign Key` をクリックしてエンティティの到着タイプを決定する属性を追加する。このプロパティ名を `PatientType` とし、Table Key 欄に `PatientData.PatientType`、Default Value 欄に `Routine` と入力する。外

部キー（Foreign Key）によりこの列は既存のデータテーブルにリンクされる。ここまででじゅうぶんだが、必要に応じて以前設定した Source オブジェクトの Table Reference Assignments を削除することもできる。

4. 次に、サンプルデータ（例：前日の実際の患者到着データ）を入力する。図 7.11 のデータを入力する。シミュレーション開始時刻を深夜からにしたい場合はそのように設定できる。また、Date Time プロパティの代わりに Real タイプのプロパティを作成し、時間単位（例：Minutes）を自由に指定することも可能である。時間レコードは作成前に自動的にソートされるため、エンティティを厳密な到着順に入力する必要はない。

Patient Data　Treatments　Test Arrivals

	Arrival Time	Patient Type
1	6/12/2024 12:08:00 AM	Urgent
2	6/12/2024 12:10:00 AM	Routine
3	6/12/2024 12:11:00 AM	Moderate
4	6/12/2024 12:14:00 AM	Routine
5	6/12/2024 12:28:00 AM	Moderate
6	6/12/2024 12:32:00 AM	Severe
7	6/12/2024 12:35:00 AM	Routine
8	6/12/2024 12:35:00 AM	Moderate

図 7.11　到着テーブルのサンプル

5. このテーブルを使用するには、Source 側での設定が必要となる。Arrival Mode プロパティで Arrival Table を選択し、Arrival Time Property で到着を決定するテーブルとプロパティを指定する。ここでは図 7.12 に示すように、TestArrivals.ArrivalTime を選択する。

Properties: Source1 (Source)

Entity Arrival Logic	
Entity Type	**PatientData.PatientType**
Arrival Mode	**Arrival Table**
Arrival Time Property	**TestArrivals.ArrivalTime**
Entities Per Arrival	1
Repeating Arrival Pattern	False
Other Arrival Stream Options	
Arrival Events Per Time Slot	1
Arrival Time Deviation	**Random.Triangular(-0.2,0,0.2)**
Arrival No-Show Probability	0.0
Stopping Conditions	
Buffer Logic	

図 7.12　Source オブジェクトでの到着テーブルの設定

6. さらに、Source では、ランダムな到着時刻を指定するために、Arrival Time Deviation を `Random.Triangular(-0.2, 0, 0.2)` と設定する。これにより、各患者は予定時刻の前後 0.2 時間（12 分）に到着する可能性がある。

7.1.6 リレーショナルテーブル

名前が示すとおり、**リレーショナルテーブル**（Relational Tables）は独立して存在するのではなく、相互に定義された関係を持つテーブルである。これらの関係は、**テーブルキー**（Table Key）と**外部キー**（Foreign Key）の機能によって形成される。`Set Column As Key` ボタンを使用すると、選択した列が他のデータテーブルから参照可能であることを示すことができる。このボタンはその列をテーブルの主キーとし、テーブル間のリンクを確立する。各キー値は一意である必要があり、重複は許されない。ただし、複数の列を組み合わせて一意の値となる場合は、それらをまとめて 1 つのテーブルキーとして使用できる（複合キーという）。

他のテーブル（単数または複数）は、メインテーブルのテーブルキーを指定する外部キー列を含めることで、メインテーブルとリンクできる。リンクが確立されると、明示的な操作なしにリンク先テーブルの任意の列にアクセスできるようになる。たとえば、Model 7-2 では、PatientData テーブルで 4 つの一意の値を持つ PatientType をキーとして指定し、その後作成した Treatments シーケンステーブルで PatientType を外部キーとして設定した。これにより、エンティティが PatientData テーブルの特定の行に関連づけられると、自動的に Treatments テーブルの関連行にもリンクされ、`Treatments.ServiceTime` の値を使用できるようになる。

リレーショナルテーブルには、テーブル間の関係を表示できる**マスター詳細ビュー**（Master-Detail View）が備わっている。キー列を持つテーブルでは、詳細ビューを使用して特定のキーつき行に関連する行のコレクションを確認できる。行の前にある小さな＋記号は、詳細ビューが利用可能であることを示している。この+をクリックすると、関連テーブルの該当部分が展開表示される。Model 7-3 で確認してみると、PatientData テーブルの PatientType 各エントリー（行）に＋記号がついていることがわかる。タイプが Moderate の患者の前にある+記号をクリックすると、図 7.13 の画面が表示される。

Treatments タブの下に、Treatments テーブルの関連する行がすべて表示される。この例では、Moderate タイプの患者に対する治療順序とその場所に関するデータが表示されている。また、TestArrivals タブでは、TestArrivals テーブルに登録された Moderate タイプの患者の到着データをすべて確認できる。詳細ビューは「-」をクリックして閉じることができ、各キーは個別に開閉できる。

テーブルスキーマビューアは、Simio バージョン 17.261 で追加された機能で、モデル内のデータテーブル用の基本的な **ER 図**（Entity Relationship Diagram：**実体関連図**）を生成する（図 7.14 の Model 7-3 の ER 図を参照）。ER 図はデータテーブル構造を表示し、キー列（PatientData.PatientType）と関連列（Treatments.TreatmentType）を明確に示す。1 と*の記号は、PatientData テーブルの各 PatientType 行が Treatments テーブルに 1 つ以上の関連行を持つ

Patient Data　Treatments　Test Arrivals

	Patient Type	Priority	Patient Mix
1	Routine	1	40
2	Moderate	2	31
3	Severe	3	24
4	Urgent	4	5

Treatments　Test Arrivals

	Sequence	Treatment Type	ServiceTime (Minutes)
1	SignIn	Moderate	2
2	Registration	Moderate	Random.Uniform(3,7)
3	ExamRooms	Moderate	Random.Triangular(10,15,20)
4	TreatmentRooms	Moderate	Random.Triangular(5,8,10)
5	NormalExit	Moderate	0.0

図 7.13　リレーショナルテーブルのマスター詳細ビュー

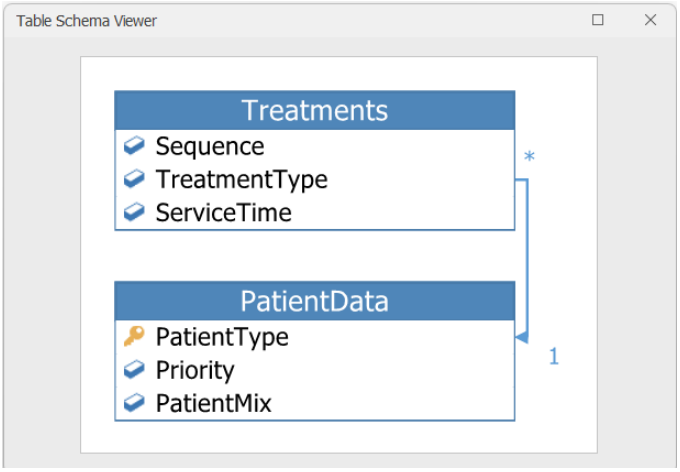


図 7.14　Model 7-3 のテーブルスキーマビューア

ことを表す（**一対多**の関係）。スキーマビューアは Table Tools リボンの Schema セクションから利用できる。

リレーショナルテーブルは強力な機能を備えており、データセットを効率的に表現し、簡単に参照することができる。この機能は外部データとのリンクにも特に有用である。多くの場合、シミュレーションに必要な外部データはリレーショナルデータベースで提供されており、独自に作成するデータテーブルでも同様の関係を表現できる。

7.1.7　テーブルのインポートとエクスポート

これまでの議論を通じて、テーブルの価値と有用性について理解を深めてきた。少量のデータであれば手動入力が適しているが、データ量が増えた場合は、ファイルの**インポート**と**エクスポート**機能の活用を勧める。

Content リボンメニューの Save to CSV ボタンを使用すると、テーブルを CSV（コンマ区切り値）ファイルにエクスポートできる。この機能は、フォーマット済みファイルの作成、既存テーブルの外部編集（Excel など）、またはバックアップの作成に便利である。図 7.15 では、Model 7-3 を例にこのインタフェースを示しており、リボンの右端に Save to CSV ボタンが配置されている。

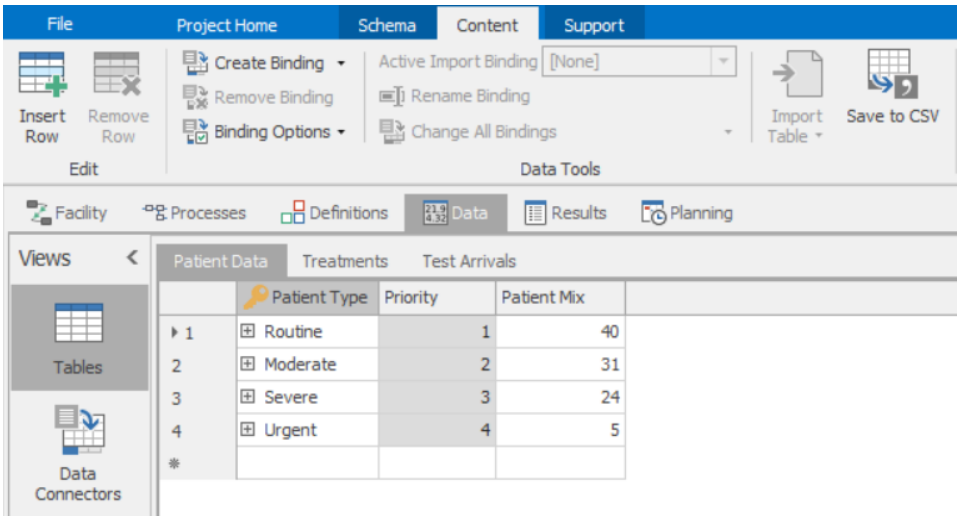


図 7.15　データ接続とバインディングのインタフェース

また、データ接続およびバインディング機能を使用すると、CSV、Excel、その他の一般的なデータベース形式からデータをテーブルにインポート・エクスポートできる。**データ接続**（Data Connectors）でファイルの読み書きに必要なパラメータを設定し、**バインディング**（Binding）機能（特に Create Binding ボタン）でテーブルを 1 つ以上のデータ接続にリンクできる。両方のボタンは図 7.15 に示されている。利用可能なインポートバインディングの種類は図 7.16 のとおりである。

インポート・エクスポートの前にテーブルを外部ファイルにバインドする必要があるが、バインドとデータ接続は任意の順序で設定できる。たとえば、最初に各テーブルのバインドを作成し、その後でデータ接続に移動してファイル名とパラメータを指定することができる。図 7.17 は、CSV コネクタのデータ接続パラメータを示している。

テーブルを特定のファイルにバインドした後、Binding Options を設定できる（図 7.18）。ここでは、モデル実行開始時にデータを手動でインポートするか（Import ボタンクリック時）、自動的にインポートするかを選択できる。データが頻繁に更新される場合は、自動インポートが適している。その他の高度なオプションも利用可能である。

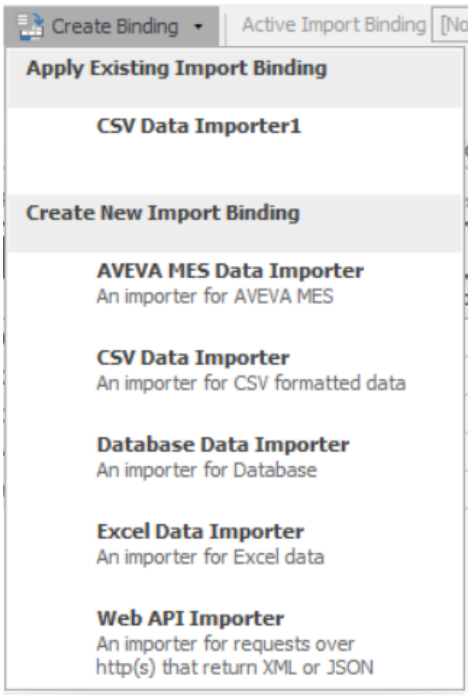


図 7.16　バインディングの種類

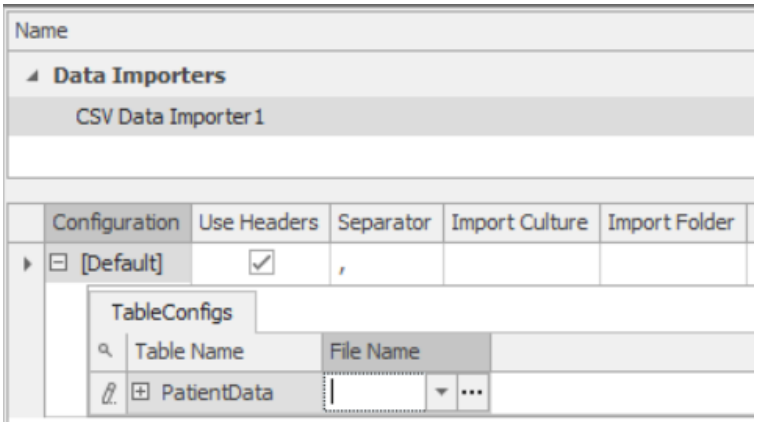


図 7.17　CSV データ接続のパラメータ設定

同様の手順とオプションはエクスポート時にも使用できる。これは外部でのデータ編集（エクスポート、編集、再インポート）、データテーブルの永続保存、実行結果のエクスポートに特に役立つ。バインディングの詳細な設定方法については、Simio ヘルプの Importing and Binding to Tables を参照するとよい。

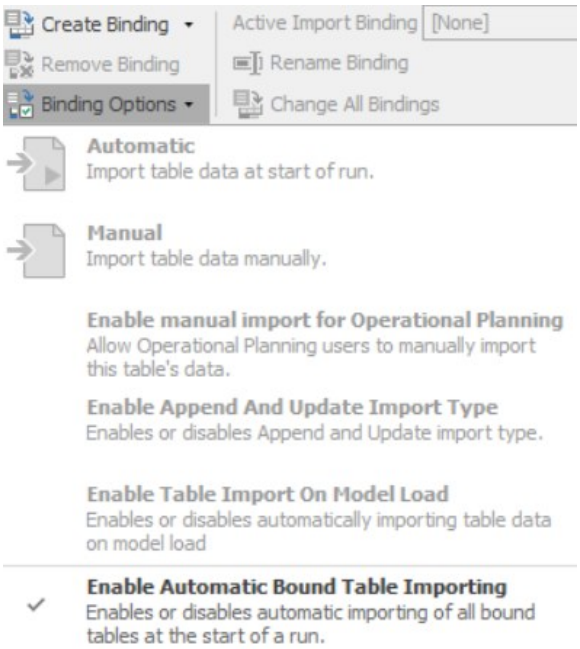


図 7.18　インポートとエクスポートのバインドオプション設定

7.2　スケジュール

5.1.5 節で述べたように、任意のオブジェクトはリソースとして用いることができる。リソースには、時間の経過とともに変化する能力（たとえば、使用可能なユニット数）がある。5.3.3 節で説明したように、オブジェクトの能力が時間とともに変化するモデルをシミュレーションする方法の 1 つが**スケジュール**の使用である。多くのオブジェクトは**作業スケジュール**に対応しており、これにより能力を時間の経過とともに自動的に変更できる。能力は数値で表され、リソースがオンシフト状態（能力が 0 より大きい）かオフシフト状態（能力が 0）かを示す。Worker 等の一部のオブジェクトでは、能力は 0（オフシフト）か 1（オンシフト）の二択となる。一方、多くのオブジェクトでは、スケジュールで可変的な能力を設定できる（たとえば、8 時間は 5、2 時間は 4、14 時間は 0 など）。

7.2.1　カレンダ作業スケジュール

カレンダ作業スケジュールには、Pattern-Based と Table-Based の 2 種類がある。Pattern-Based スケジュールは、Day Patterns（デイパターン）、Work Schedules（作業スケジュール）、Exceptions（例外）という 3 つの主要要素で構成される。

Day Pattern は 1 日の作業期間を定義する。作業期間は必要な数だけ設定でき、指定のない期間は非作業期間として扱われる。Simio には Standard Day という名前のサンプル Day Pattern が用意されている（図 7.19 の下部）。これは 4 時間の作業、1 時間の休憩、さらに 4 時間の作業という構成である。このサンプルは必要に応じて変更や置き換えが可能である。Day Pattern の各作業期間では、

Start Time（開始時刻）と、Duration（継続期間）または End Time（終了時刻）の指定が必要である。リソース能力が初期値の 1 と異なる場合は、Value 列でその値を設定する。Cost Multiplier 列は、通常のリソース使用レートとは異なるコストが発生する期間に使用する。たとえば、通常のスケジュールに 2 時間の残業を追加する場合、この 2 時間のコスト乗数を 1.5 倍に設定できる。

Simio には、図 7.19 の上部に示されている Standard Week という名前のサンプル作業スケジュールが含まれている。**作業スケジュール**には、繰り返しの作業期間を構成するデイパターンの組み合わせが含まれている。繰り返し期間は 1 日から 28 日の間で設定でき、一般的には 7 日間である。作業期間は、Start Date（開始日）としてカレンダ上のどの日からでも開始できる。作業スケジュールが 7 日間で、毎週同じ曜日（例：月曜日）から開始する場合、図 7.19 のように列ラベルは週の実際の曜日となる。スケジュールが毎週異なる曜日から始まる場合や、期間が 7 日間でない場合は、列ラベルは単に Day 1, Day 2, ..., Day *n* となる。この場合、スケジュールされていない日は、データ入力を防ぐため灰色で表示される。

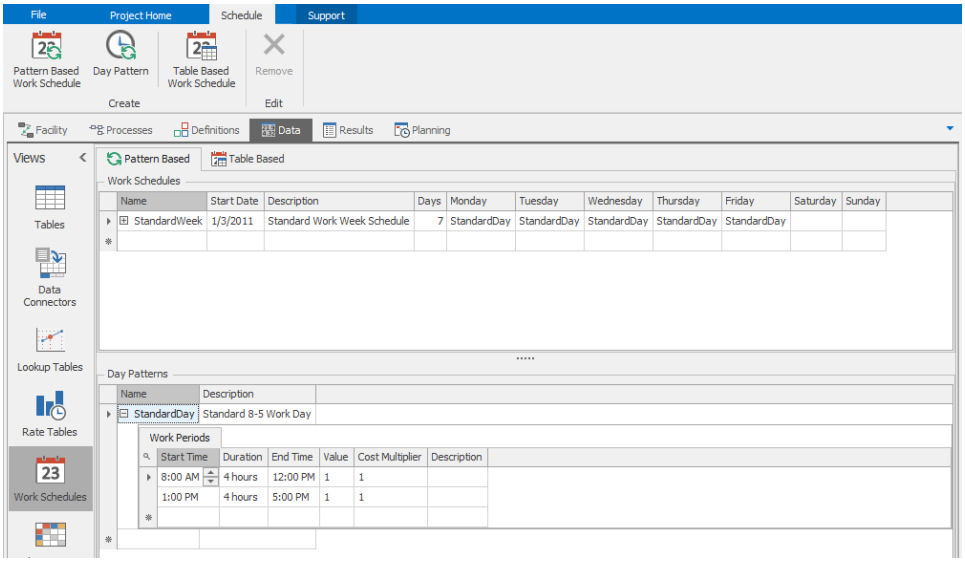


図 7.19　Pattern-Based 作業スケジュールのサンプル

スケジュールの 3 つ目の構成要素は**例外**（Exception）である。例外は、特定期間の基本スケジュールを上書きするもので、残業、計画的なメンテナンス、休暇期間などを定義できる。例外には、作業スケジュール名の左にある+をクリックしてアクセスする。例外は 2 種類ある。1 つ目は**ワークデイの例外**（Work Day Exception）で、特定の日に通常と異なる作業パターンを使用する場合に適用する。たとえば、早退日用に AugustFridays というデイパターンを定義し、8 月の全金曜日にこれを適用できる。2 つ目は**作業期間の例外**（Work Period Exception）で、特定の日付と時間帯に対して、通常のスケジュールとは異なる値を設定する。これは主に長期休暇期間や作業時間延長期間などに使用する。

大量のスケジュールがある場合、特にデータがすでにデータベースに存在する

場合は、パターンベースの入力が煩雑になることがある。そこで Simio は**テーブルベースの作業スケジュール**(Table-Based Work Schedules)を提供している。これらのスケジュールはデイパターンと同じ情報を必要とするが、各期間の開始時刻、終了時刻、値、コスト乗数の情報を、参照するテーブルと列を指定するだけで設定できる（図 7.20 参照)。

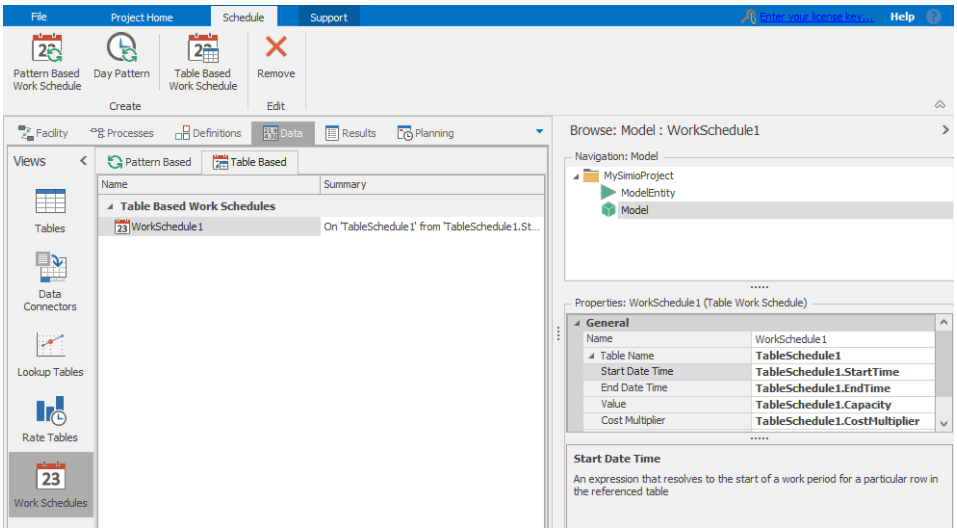


図 7.20　Table-Based 作業スケジュールのサンプル

カレンダ作業スケジュールの設定や使用方法について、詳しくは Simio ヘルプまたは Simio Reference Guide で「Schedule」を検索すると確認できる。

7.2.2　手動スケジュール

カレンダによるスケジュールは便利だが、ときには追加的な制御や、逆に単純な手動制御が必要になることがある。手動スケジュール（Manual Schedule）はこのために用意されている。もっとも基本的なレベルの手動スケジュールはシンプルで、能力に希望する値を入力し、その値を維持する期間を設定して、必要に応じて繰り返すだけである。たとえば、Server1 という名前のサーバに対して、1 日 8 時間の繰り返しスケジュールを作成する手順は以下のとおりである：

1. Add-on Process Triggers で `Run Initialized` をダブルクリックする。これにより、Server オブジェクトの初期化時に、このプロセスが自動的に 1 回実行される。
2. Assign と Delay ステップを 1 組追加し、さらにもう 1 組追加する。これらを用いて、Server1.CurrentCapacity を 1 に設定して 8 時間待機し、次に Server1.CurrentCapacity を 0 に設定して 16 時間待機するよう設定する。
3. 図 7.21 に示すように、End コネクタをドラッグして、最初の Assign ステップに接続する。これにより 4 つのステップが無限に繰り返される。

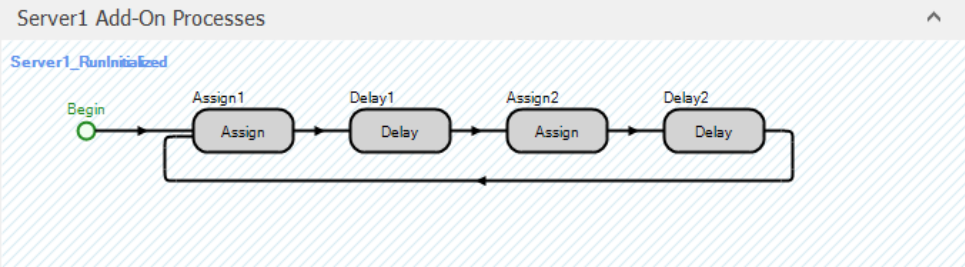


図 7.21　1 日 8 時間の手動スケジュール作成例

手動スケジュールは高い柔軟性を持つ。スケジュールは必要に応じて詳細に定義でき、遷移ロジックも追加できる。たとえば、昼食時間になったときに、リソースを即座に停止させるか、進行中の作業を一時停止して昼食後に再開するロジックを実装できる。また、異なるロジックの実装も可能である。たとえば、後片付けのために、シフト終了 30 分前から新規作業の受け入れを停止したり、シフト終了時に作業が残っている場合は完了まで継続したりするロジックを組むことができる。これらは一例に過ぎないが、プロセスをじゅうぶんに活用することで、高度なスケジュール動作をモデル化できる。このように、モデルの要件に応じて知的なオブジェクトを作成することが可能である。

7.3　レートテーブルと Model 7-4

レートテーブル（Rate Table）は、これまで説明してきたテーブルとは大きく異なる。汎用性がなく列の追加もできないが、その代わり明確な目的がある。それは期間ごとのエンティティの時変到着率を指定することだ。これまでの例では、実行期間中の到着率は常に一定だった。しかし現実世界、特にサービス業では、到着率は時間とともに変動するのが一般的である。たとえば、銀行では特定の時間帯に顧客の入店頻度が高くなる。同様に、コールセンタへの問い合わせ電話も、特定の時間帯に集中する傾向がある。

これまでのモデルでは、**定常ポアソン過程**（Stationary Poisson Process）を仮定し、個々の到着は独立して、一定の平均を持つ指数分布に従って発生するとしていた。時間変動する到着率を実装するには、**非定常ポアソン過程**（Nonstationary Poisson Process）が必要である。この到着プロセスの詳細については 6.2.3 項を参照してほしい。また、モデルの妥当性を確保するには、現実世界での到着率の「ピーク」と「谷」をモデルに反映することが重要である。

到着時間間隔の分布が指数関数的であることを前提に、単に平均到着時間を状態として指定し、その値を離散的に変更すればよいと考えるかもしれない。しかし、このアプローチは誤りである。期間間の移行が適切に考慮されないため、不正確な結果となってしまう。レートテーブルを使用することで、時間変動する到着率を正確に表現できる。

レートテーブルは、Source オブジェクトが時変到着率を持つ非定常ポアソン過程に従ってエンティティを生成する際に使用する。時間間隔（Interval）の数と各間隔の継続時間（Interval Time Duration）を指定できる。レートテーブルは、等

間隔の時間区分と各区分における平均到着率のペアで構成される。各時間間隔内の平均到着率は一定だが、次の時間間隔開始時に変更できる。到着率の時間単位は常に**時間**（Hour）単位であり、レートテーブルでの時間間隔の単位設定とは**独立**である。たとえば、10 分間隔を指定する場合でも、その到着率は **1 時間当たりの**値として換算する必要がある。最後の時間間隔が終了すると、レートテーブルは自動的に最初から繰り返される。このように到着率を区分的に一定と仮定することは、一見単純に見えるかもしれないが、実際には現実の到着率関数をよく近似できることが示されている（Leemis（1991）参照）。

標準の Source オブジェクトでレートテーブルを使用するには、Source の Arrival Mode 属性を Time Varying Arrival Rate に設定し、Rate Table プロパティで適切なレートテーブルを選択する。

この概念を ED の Model 7-2 の例で具体的に説明しよう。最初のモデルでは、エンティティの到着を固定の平均 4 分間隔で近似していたが、実際の到着率は 1 日を通じて変動することがわかっていた。表 7.5 は、1 日を 4 時間ごとに区切った場合の患者の平均到着数を示している。

表 7.5　Model 7-4 の実績到着データ

時間帯	当該時間帯に到着した患者の平均数
0:00 ～ 4:00	49
4:00 ～ 8:00	31
8:00 ～ 12:00	38
12:00 ～ 16:00	36
16:00 ～ 20:00	60
20:00 ～ 24:00	70

それでは、このデータをモデルに追加する。

1. Model 7-2 を立ち上げ、Data ウィンドウを開いて Rate Tables パネルを選択する。
2. Rate Table ボタンをクリックして新しいレートテーブルを追加し、名前を ED_Arrivals に変更する。
3. 到着率は 1 時間単位で 24 回分の時間間隔を設定できるが、今回は 4 時間単位で 6 つの時間帯の到着情報のみ把握しているため、テーブルのプロパティで Interval Size を 4 Hours、Number of Intervals を 6 に設定する。
4. 実績情報は **4 時間**ごとの患者の平均到着数であるため、これを **1 時間**当たりの到着率に変換する必要がある。表 7.5 の到着率を 4 で割った値をレートテーブルに入力する。図 7.22 に設定完了後の画面を示す。
5. 最後に、Facility ウィンドウに戻り、レートテーブルを使用するよう Source を設定する。Arrival Mode プロパティで Time Varying Arrival Rate を選択し、Rate Table プロパティで ED_Arrivals を選択する（図 7.23）。

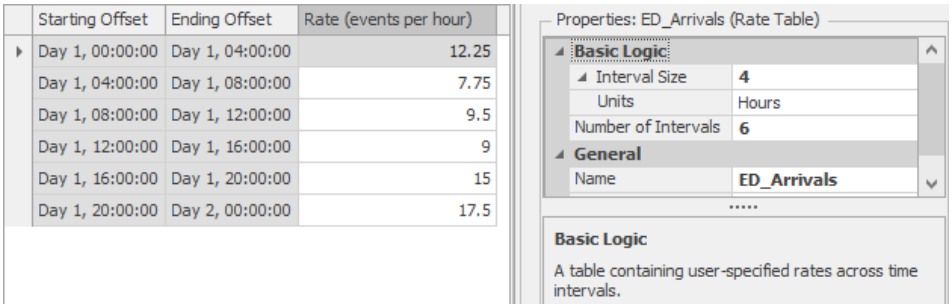


図 7.22　患者の時間帯別到着率を表すレートテーブル

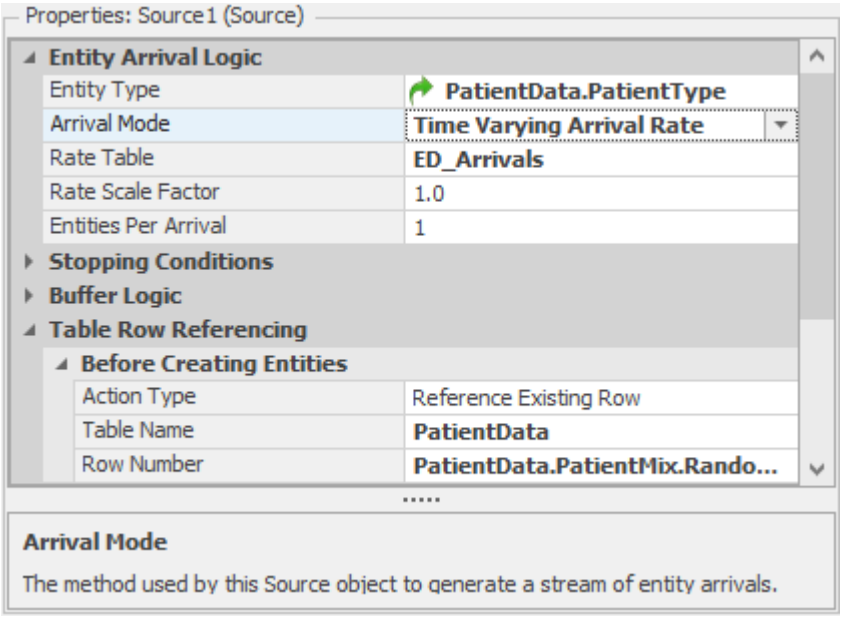


図 7.23　時変到着率の Source オブジェクトへの設定

修正後のモデルを実行すると、大きく異なる結果が表示される。全体的な患者数が減少し、さらにピーク時の多数の到着患者への対応が必要となるためである。

このモデルでは使用していないが、Source オブジェクトには Rate Scale Factor というプロパティがある。このプロパティを使用すると、個別の値を変更する代わりに、係数でテーブル全体の値を一括して変更できる。たとえば、レートテーブルの値全体を 50%増加させたい場合は、Rate Scale Factor を 1.5 に設定する。これにより、サービス負荷の変化に対する実験（例：平均患者負荷が 50%増加した場合のシステムの反応）が容易になる。

7.4　Lookup テーブルと Model 7-5

モデルを構築する際に、他の値（完了したサイクル数など）に依存する値（処理時間など）が必要になることがある。これは、簡単な式で表現できる場合（例：`Server1.CycleCount*3.5` または `Math.Sqrt(Server1.CycleCount)`）や、直接テーブル参照で対応できる場合（例：`MyTable[Server1.CycleCount].`

ProcessTime）がある。また、非線形の関係を扱う場合には、ルックアップテーブルが有用である。**Lookup テーブル**は、このような用途のために設計された特殊な目的のテーブルである。Lookup テーブルは、時間、カウント数、その他の独立変数や数式に基づいて値を参照する機能を提供する。典型的な使用例として、作業者の熟練度に応じて作業完了時間が変化する学習曲線のモデル化がある。また、部品のサイズや重量に基づいて処理時間やバッテリー放電率を決定する場合にも活用できる。

Lookup テーブルは、Data ウィンドウの Lookup Tables パネルで作成する。リボン上の Lookup Table をクリックしてテーブルを追加すると、x（独立）の値と $f(x)$（従属）値の列を持つ表が作成される。テーブルは TableName [X_Expression]という構文で参照でき、TableName は Lookup テーブルの名前、X_Expression は独立変数の値を表す。たとえば、ProcessingTime[Server1.CycleCount]は、Server1.CycleCount の現在値に基づいて ProcessingTime テーブルから値を取得する。Lookup テーブルは定義された値をそのまま返すか、値が中間にある場合は線形補間で計算する。指定された値が定義範囲外の場合は、もっとも近い端点の値が返される。

では、Model 7-4 を拡張して、Registration Time を調整するための Lookup テーブルを追加してみよう。ここでは診療受付処理にコンピュータシステムを使用することを想定する。このシステムは早朝と夕方には処理が速いが、正午に向かって徐々に遅くなる特性がある（図 7.24）。

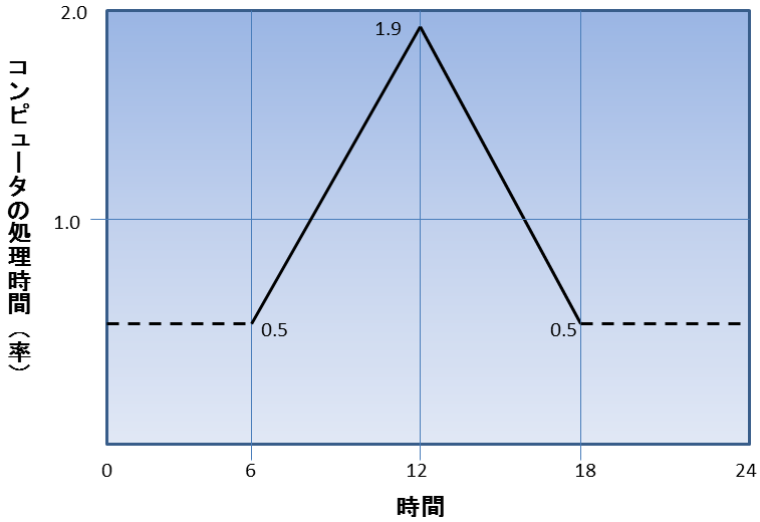


図 7.24　1 日の時間帯別コンピュータ応答性（速度係数）

このコンピュータの処理速度を表現するために、処理時間の調整係数として Lookup テーブルを作成する。

1. Model 7.4 を開き、Data ウィンドウの Lookup Tables パネルを選択する。
2. Lookup Table をクリックして新しい Lookup テーブルを追加し、名前を ComputerAdjustment に変更する。

3. 図 7.24 のデータを追加する。データポイントは 3 つあり、6 時の値が 0.5、12 時の値が 1.9、18 時の値が 0.5 である。6 時以前または 18 時以降は、端点の値（この場合は 0.5）が適用される。12 時では正確に 1.9 の値が生成され、その他の時間帯では直線的に補間される。たとえば、8 時は 6 時と 12 時の間の 1/3 の位置にあるため、0.5＋0.466＝0.966 という値が返される。
4. 最後に、Facility ウィンドウの Registration サーバに戻り、Processing Time を修正する。まず、Process Time プロパティを右クリックして Reset を選択する。Treatments テーブルで指定した処理時間に、コンピュータの応答係数（Lookup テーブルから取得）を乗算する必要がある。Lookup テーブルには当日の経過時間を渡すため、Simio の組み込み関数 DateTime を使用する。その結果、Registration の Process Time プロパティの式は Treatments.ServiceTime*ComputerAdjustment[DateTime.Hour(TimeNow)]となる。

この修正により、モデルは夜間の受付での処理時間を短縮し、コンピュータの応答が遅くなる昼間は診療受付での処理時間を延長できるようになった。

7.5　リストとチェンジオーバー

5.4 節では、リストの概念を紹介した。リストは、文字列、オブジェクト、ノード、トランスポータなどの集合を定義するために使用される。また、転換行列（Changeover Matrix）の転換可能な状態（色やサイズなど）を定義したり、選択（リソースの専有やトランスポータの選択など）リストを提供するためにも使用される。リストは Definitions ウィンドウの Lists パネルから追加できる。

リストのメンバーは、0 から始まる数値のインデックス値を持つ。リスト値は、List.ListName.Value の構文で参照できる。たとえば、Color という名前のリストが Red、Green、Blue、Yellow というメンバーを持つ場合、式 List.Color.Yellow では 3 の値が返される。BikeColor という名前の ListProperty（リストメンバーを値として取るプロパティ）を持つ場合は、BikeColor == List.Color.Yellow で条件をテストできる。

リストの他の応用については 8.2.7.3 で後述するが、ここでは**文字列型リスト**（String List）のみを扱う。名前が示すとおり、文字列型リストとは文字列の集合である。このリストは、数値ではなく、人が読みやすい文字列で項目を識別したい場合に使用する。

文字列型リストの作成は簡単である。Definitions ウィンドウの Lists パネルで String ボタンをクリックすると、新しい文字列型リストを作成できる。図 7.25 で示すように、ウィンドウの下部に各項目を表す文字を入力する。

チェンジオーバーとは、異なるタイプのエンティティ間で必要な変換を記述するための一般的な用語である。製造業では、製品の小から大へのサイズ変更や色の変更などがこれに該当する。チェンジオーバーは、サービス分野ではあまり一般的ではないが、ときとして使用される。たとえば、患者の入れ替え時に、Routine（通常）タイプの患者間の清掃時間や遷移時間は、Serious（重症）や Urgent（緊急）タイプの患者の場合と比べて大幅に短い。チェンジオーバーは、Server の

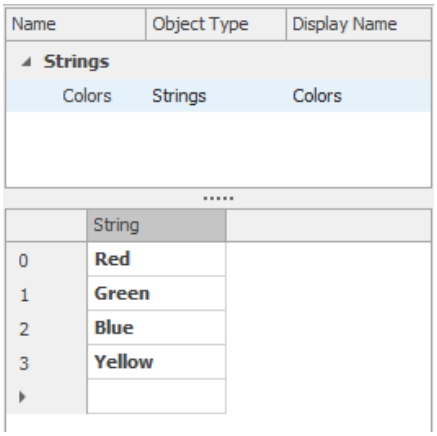


図 7.25　文字列型リストによる色の定義例

Task Sequences（10.4 節で説明）と関連オブジェクトに組み込まれているが、アドオンプロセスを使用して他のオブジェクトにも追加できる。

チェンジオーバーには、Specific、Change-Dependent、Sequence-Dependent の 3 種類がある。**Specific** はもっとも単純なチェンジオーバーで、全エンティティが同じ式を使用する。これは単純な定数（5.2 分など）でも、エンティティの状態変数や Lookup テーブルを含む複雑な式（Entity.MySetuptime や MyDataTable.ThisSetupTime など）でも可能である。他のタイプのチェンジオーバーでは、最後に処理されたエンティティの情報を追跡し、次に処理するエンティティの情報と比較する。**Change-Dependent** では、値の具体的な内容よりも、値が変更されたかどうかに着目する。たとえば、色が変更された場合は特定のチェンジオーバー時間が必要だが、変更がない場合は異なる時間（0.0 など）が必要となる。**Sequence-Dependent** では、離散的な値の集合内での変化を追跡する。値の組は Simio リストで定義され（Small、Medium、Large など）、各「From」から「To」への組み合わせごとに固有のチェンジオーバー時間を設定できる。これには転換行列という別の Data ウィンドウ機能が必要となる。

転換行列（Changeover Matrix）はリストに基づく行列である。多くの場合、リストは特性を名前で識別できる文字列のリストである（Red、Green、Blue、Yellow を含む Colors リストなど）。エンティティは通常、リストから値を受け取るプロパティまたは状態変数（例：Color）を持つ。転換行列は、特定のリスト内の値を From/To 行列として表示する。各セルには、ある From の値を持つエンティティから別の To の値を持つエンティティへの移行に必要な時間を格納する。転換行列が適用されると、行は前のエンティティの値に基づいて選択され、列は現在のエンティティの値に基づいて選択される。

Data ウィンドウの Changeovers Matrices パネルで転換行列を定義できる。新しいチェンジオーバーを追加するには、リボンの Changeover Matrix ボタンをクリックする。List Name プロパティに、作成済みのリスト名を入力する。すると下のウィンドウが拡張され、リストの全メンバーによる行列が表示される。各交点の値を変更して、ペア間の移行時間を設定する。図 7.26 では、Yellow へ

の移行時間が 10 分、他の色への移行時間が 2 分となっている。色の変更がない場合は、移行時間は不要である。

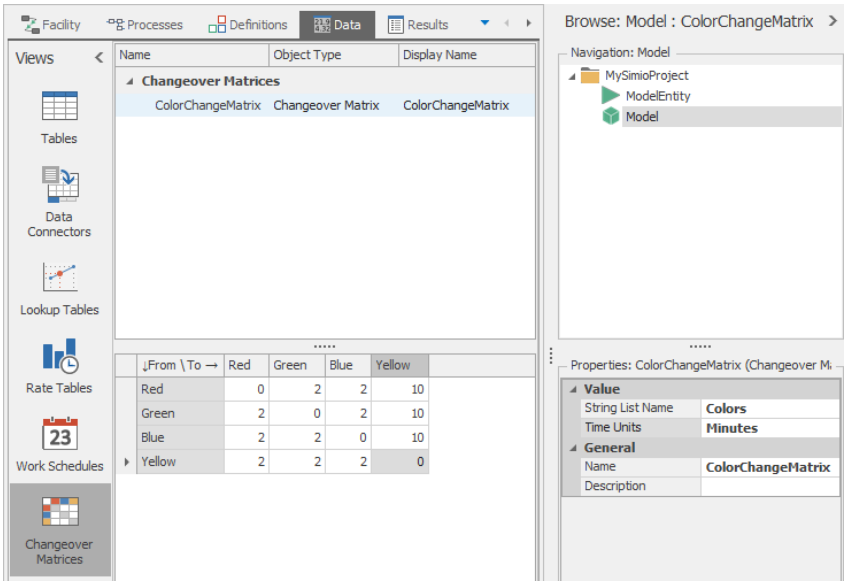


図 7.26　色の変更に基づく転換行列の例

7.6　状態変数配列

5.1.2.2 では、状態変数という概念を紹介したが、配列と初期化という 2 つの重要な話題は後回しにした。**状態変数配列**（State Arrays）とは、関連する状態変数の集合である。初期設定では、状態変数はスカラーであり、次元（Dimension、あるいはセットサイズ）は 0 となっている。

状態変数は、Dimension プロパティの値を設定することで配列にできる。Dimension プロパティを Vector に設定する（または 1 を入力する）と、状態変数は 1 次元となり、Rows プロパティが配列内の行数を決定する。同様に、Matrix に設定する（または 2 を入力する）と、Rows と Columns プロパティが行と列の数を決定する。2 次元以上の配列が必要な場合は、Dimension プロパティに最大 10 までの整数を入力できる。Simio は最大 10 次元までの配列をサポートする。

状態変数配列の参照には、StateName[row]、StateName[row, column]、または StateName[row, column, dimension 3, ...,dimension 10]という構文を使用する。たとえば、Weight 配列の 7 行 5 列を参照する場合は、Weight[7, 5]と記述する。重要な点として、状態変数配列のインデックスは 1 から始まる。たとえば、5 要素のベクトルを指定する場合、0～4 ではなく 1～5 を使用する。各状態変数には Initial Value プロパティがあり、これで配列内のすべての項目を同じ値に初期化できる。個別の値で初期化したい場合は、プロセス（通常はオブジェクトの OnInitialized プロセス）で Assign ステップを使用するか、Read ステップを使用して外部データソースから初期値を読み込むことができる。

配列の次元設定と初期化をより簡単に行う方法がある。状態変数の Dimension プロパティを[Table]に設定すると、テーブルの内容に基づいて状態変数配列の

次元が自動的に設定される。テーブルの各行が配列の行となり、テーブル内の各**数値**列（整数、式、日時など。ただし Element や Object Reference は**除外**）が列となる。初期化時には、各数値フィールドの値が評価され、適切な配列要素にコピーされる。この方法は、テーブルの外部ファイルバインド機能と組み合わせることで、外部データソースと完全に一致する配列を簡単に作成・設定できる。

7.7　データ駆動型モデル

再利用可能なコンポーネントを持つモデルを作成したいケースは多い。たとえば、複雑なワークステーションを構築してデバッグを完了した後、そのワークステーションを同じ施設や他の施設の類似のワークステーションに再利用したいと考えるだろう。オブジェクトの利用は解決策として重要だが、データテーブルからデータを取得するオブジェクトを活用することで、さらなる柔軟性を得ることができる。

さらに、このコンセプトを拡張して、**モデル全体**を再利用可能にしたい場合もある。たとえば、あるシステムに類似の施設（港、工場、病院など）が多数存在し、各施設が基本的な構造は同じでも、主要な構成データや運用データが異なる場合、このアプローチが有効である。

これらの目的は、汎用コンポーネントの使用か汎用モデルの構築によって達成できる。データ駆動型モデリングまたは**マクロモデリング**（Macro Modeling）は、オブジェクトを明示的に指定する代わりに、「渡された」主要データで汎用モデルを作成する手法である。各エンティティは、自身の処理方法を決定するためのデータをオブジェクトに提供できる。オブジェクトは、エンティティまたは関連データテーブルの指示に従ってエンティティを処理する。このアプローチにより、カスタマイズの責任がオブジェクトからエンティティと設定データに移り、オブジェクトをより単純化し、種類を削減できる。これにより、データがオブジェクトから分離され、分散していたデータを一箇所に集約できる。これが**データ駆動型モデル**である。

Simio は、データ駆動型モデリングをサポートする複数の機能を提供している。すでに説明したように、5.1.2 項では、プロパティ参照を Controls として使用することで、主要パラメータの検索と変更が容易になることを示した。これらのコントロールにより、モデルの属性や実験のコントロール値を通じて主要データを供給できるため、データ駆動モデルの作成が簡単になる。

Model 7-2 では、患者のタイプ、構成、治療場所、治療時間などのデータをテーブルから取得することで、モデルのオブジェクトにデータを直接埋め込まずにモデルを構築した。この包括的なアプローチによりデータ駆動型モデリングを実現した。モデルオブジェクトは、特定のアクションではなく、「テーブルで指定された種類の患者を、指定された割合で生成し、指定された場所で、指定された時間で治療する」という指示に従う。これにより、モデリングの専門知識が少ない人でも、データテーブルの値を調整するだけでモデルを使った実験が可能になる。

オブジェクト参照とエレメント参照の状態変数・属性により、このコンセプト

をさらに発展させることができる。オブジェクト参照は、その名のとおり、オブジェクトまたはオブジェクトリストを指定する。これにより、患者の種類ごとに必要なスタッフや機器（リソース）のリストを指定して、前述のモデルを拡張できる。同様に、エレメント参照は材料や統計量などの要素を指定する。これを使用して、患者に必要な特殊材料（手術用パックなど）や、患者データを記録するための特別な統計量を指定できる。

7.7.1　テーブルとリピートグループ

データ駆動型モデリングの説明には、Simio のテーブルと**リピートグループ**（Repeat Group）の特殊な関係について触れる必要がある。リピートグループは、複数の行にわたって繰り返すことができるプロパティのセットである。その例として、多くのオブジェクト（Source など）で利用可能な Assignments がある。リピートグループはテーブルに似ていると感じるかもしれないが、そのとおりである。リピートグループはオブジェクトに埋め込まれたテーブルと考えることができ、Simio ではテーブルとリピートグループを相互に変換できる。

簡単な例で説明しよう。状態変数の割り当てリストを作成したいが、そのデータをオブジェクトではなくテーブルで指定したい場合を考えてみる。

1. 新しいモデルから始め、Source を 1 つ追加する。
2. Data タブに移動し、新しいテーブルを追加する。Add Data Table ボタンの下矢印をクリックして、Add Data Table From Schema を選択する。これにより、モデルに定義されている任意のスキーマ（テーブルレイアウト）に一致する新しいデータテーブルを作成できる。
3. このモデルでは、リストを下にスクロールして、Source スキーマを選択する。すると、図 7.27 のようなダイアログが表示される。

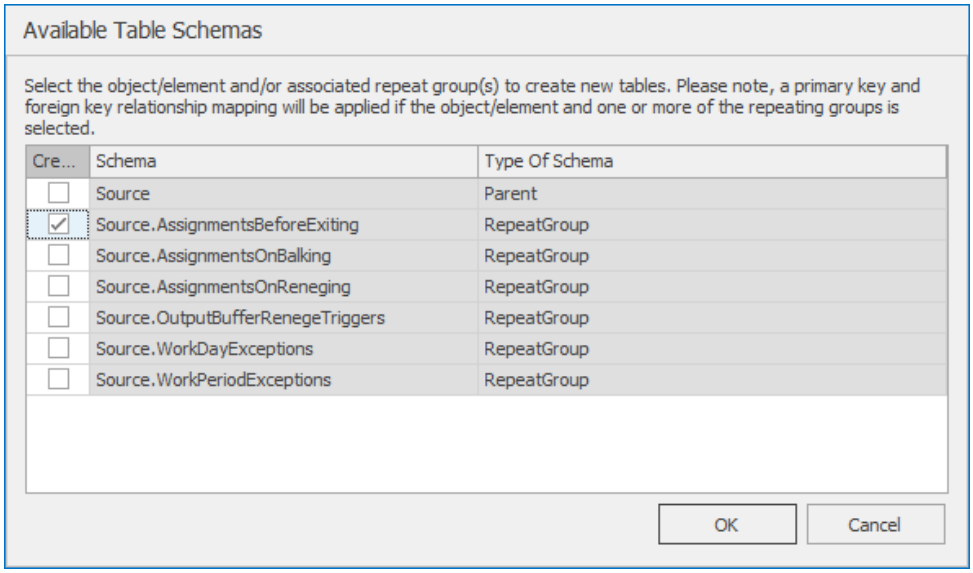


図 7.27　Source オブジェクトにより定義されたデータスキーマ

4. Source.AssignmentsBeforeExiting スキーマに対応する Create 列のボックスをチェックし、OK を押す。これにより新しいテーブルが作成され、最初の列が State Variable、2 番目の列にその状態に割り当てる値や式を入力できる。このテーブルに StartingAssignments という名前をつける。
5. 図 7.28 のように、テーブルに 3 つの行を追加する。最初の行では、モデルの Cost の状態を 100 ずつ増やし、2 行目と 3 行目ではモデルエンティティの Priority と Picture の状態をそれぞれ 3 および 1 に設定する。

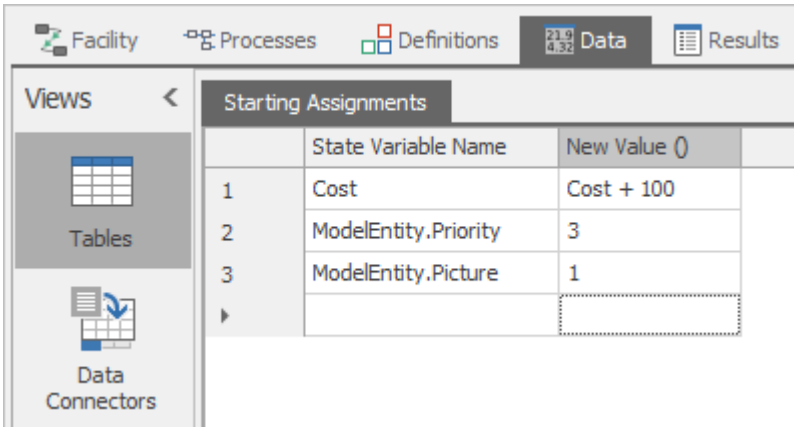


図 7.28　テーブルで定義した割り当てのサンプル

6. テーブルの作成と入力が完了したら、これまで使用してきたプロパティ参照の機能を使用する。Facility ビューの Source オブジェクトで、State Assignments の下の Before Exiting プロパティ名を**右クリック**する。新しいプロパティ参照を作成する代わりに、先ほど定義したテーブルがリストに表示され選択できる（図 7.29）。これを選択すると、プロパティ参照と同じ緑色の矢印が表示され、データの取得場所が示される。今回は、テーブルの 1 つ以上の行にあるデータの集合を参照している。

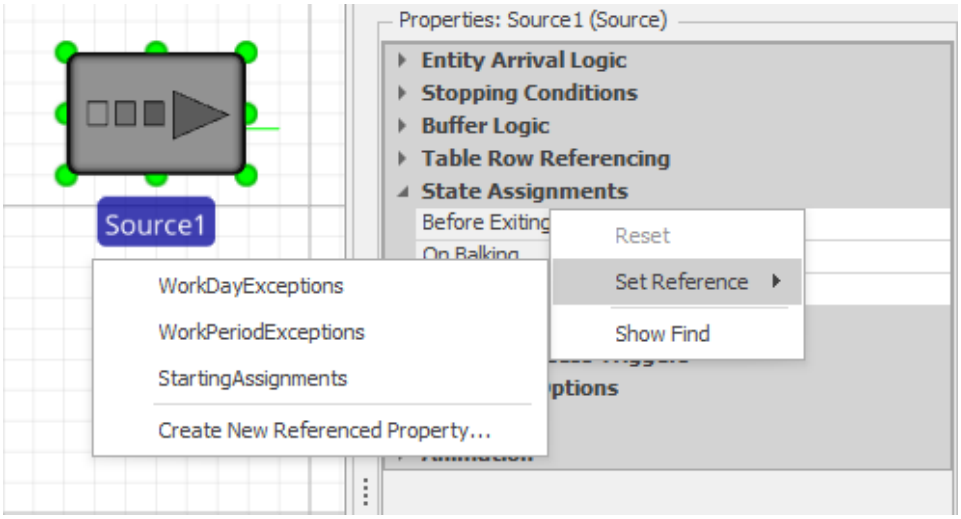


図 7.29　リピートグループのテーブルデータによるプロパティ参照の設定

この例では、テーブルを使ったリピートグループの作成手法を説明した。この手法をリレーショナルデータテーブルと組み合わせることで、オブジェクトとデータテーブルを結びつけ、より本格的なデータ駆動型モデリングを実現できる。12.11 節と 12.12 節では、モデル優先アプローチとデータ優先アプローチを用いた、簡単なデータ駆動型スケジューリングモデルの構築方法を紹介する。

マクロモデリングの主な目的は、多様な要素を扱える抽象的なオブジェクトを作成することである。たとえば、多数の類似したサーバを表すオブジェクトがこれにあたる。一方、データ駆動型モデリングの主な目的は、データを抽象化して一箇所にまとめ、モデルの実験とメンテナンスを容易にすることである。両者は目標は異なるものの、オブジェクトの値を直接指定せず、外部からデータを取得することで汎用的なオブジェクトを作成するという手法を共有している。これらの手法により、モデルの理解、使用、保守がより簡単になる。

7.8　データ生成モデル

本章の終わりに、データに関するもう 1 つの重要な概念として、既存のデータからモデルを生成する方法を紹介しよう。**データ生成モデル**（Data Generated Models）（または**データ優先**（Data First）モデルとも呼ばれる）は、外部ファイルからインポートしたデータに基づいて汎用モデルを自動生成する手法である。これはシミュレーションの価値を劇的に高める可能性を秘めた真の「ゲームチェンジャー」といえる。モデリングの専門知識が少なくても迅速にモデルを構築できる点が特に魅力的だ。この章では詳しく触れないが、12.5.1 項で紹介する**デジタルツイン**（Digital Twin）は、頻繁に更新される大規模モデルを扱う際に特に有効である。データ生成モデルは、データベースとの直接連携、スプレッドシートや CSV の連携、XML 変換など、さまざまなデータソースと接続するための柔軟な解決策を提供する。この手法で作成したモデルは、設計、計画、スケジューリングをサポートするデジタルツインとして効果的に活用できる。

既存のデータから直接モデルを構築することが有効な場面は数多くある。たとえば、多くの組織では Microsoft Visio などのフローチャート作成ツールでビジネスプロセスモデルを作成している。標準的な Visio ステンシルには完全なシミュレーションモデルの作成に必要な情報が不足していることが多いが、カスタムや拡張ステンシルにはじゅうぶんな情報が含まれていることがある。このモデル（データとモデル構造の両方）を完全なシミュレーションパッケージにインポートすれば、確率的分析や時間ベースの分析が可能になる。さらにシミュレーション機能を追加することで、フローチャートモデルを拡張できる。これは既存データからモデルを生成する、分野に依存しない（domain-neutral）基本的な応用例である。Simio Shared Items フォルダには、このアプローチを示す例が用意されている。

多くの組織がデータを表現するために採用している国際的なデータ標準がいくつか存在する。**B2MML**（Business to Manufacturing Markup Language）は、そうした一般的なデータ標準の 1 つである。

> B2MML は、国際標準 IEC/ISO62264 として知られる ANSI/ISA-95 規格群（ISA-95）の XML 実装である。B2MML は、ISA-95 規格のデータモデルを実装する XML スキーマ群（中略）で構成されている。企業は、（中略）B2MML を使用して、ERP やサプライチェーンマネジメントシステムなどのビジネスシステムと、制御システムや製造実行システムなどの製造システムを統合することができる（ISA-95.com 2016）。

7.1 節で説明したように、データの構成やインポート・エクスポートをモデル化できるが、Simio はさらに別のアプローチも提供している。企業が B2MML などのデータ規格を採用する場合、通常は単一のテーブルを超えて、一連のデータとモデリング技法に拡張される点に注目してほしい。これをサポートするため、Simio は**テンプレート**機能を実装している。Simio のテンプレートは、他の製品のテンプレート（Word の履歴書テンプレートなど）と同様に、プロジェクトを標準的な方法で実装するためのスケルトンを提供する。これにはデータテーブルスキーマ、カスタムオブジェクト、それらの間の関係などが含まれる。

新規プロジェクトの開始時にテンプレートを適用できる `File→New From Template` を選択すると、図 7.30 のようなオプションが表示される。データが注文基準か品目基準かに応じて、2 つの ISA95（B2MML）データスキーマが用意されている。これらのスキーマで新規プロジェクトを開始すると、リレーショナルデータテーブルスキーマ（ただしデータなし）とカスタムオブジェクトが事前設定され、これらのオブジェクトがデータテーブルから直接構成されるよう接続されている。ISA95 規格に準拠した既存データは、これらのテーブルにインポートできる。なお、ISA95 はガイドラインであるため、企業による実装は標準と多少異なることが多いが、Simio のデータテーブルはカスタマイズや変換が可能である。MESA（Manufacturing Enterprise Solutions Association）は、B2MML 規格に関する優れた情報源である（International 2018）。12.12 節では、B2MML データファイルに基づくモデル構築について解説する。

製造実行システム（MES：Manufacturing Execution Systems）は、生産の追跡・制御を行うソフトウェアベースのシステムで、主に物理的操作を管理する。効果的に機能させるには「モデル」に基づく必要がある。上記のように、Simio は MES から直接データを抽出し、そのデータから Simio モデルを構築・設定できる。

図 7.30 の下 3 つのテンプレートを使用すると、Wonderware（AVEVA 社の主要 MES 製品）のデータをサポートするスキーマでデータテーブルを作成できる。このプロセスは、SAP などの製品から ERP データを直接抽出する際にも使用できる。さらに、独自のテンプレートを作成することも可能だ。独自のデータスキーマ（データなし）と対応するオブジェクトを作成し、Templates フォルダに保存すれば、テンプレートのリストに表示され選択できるようになる。

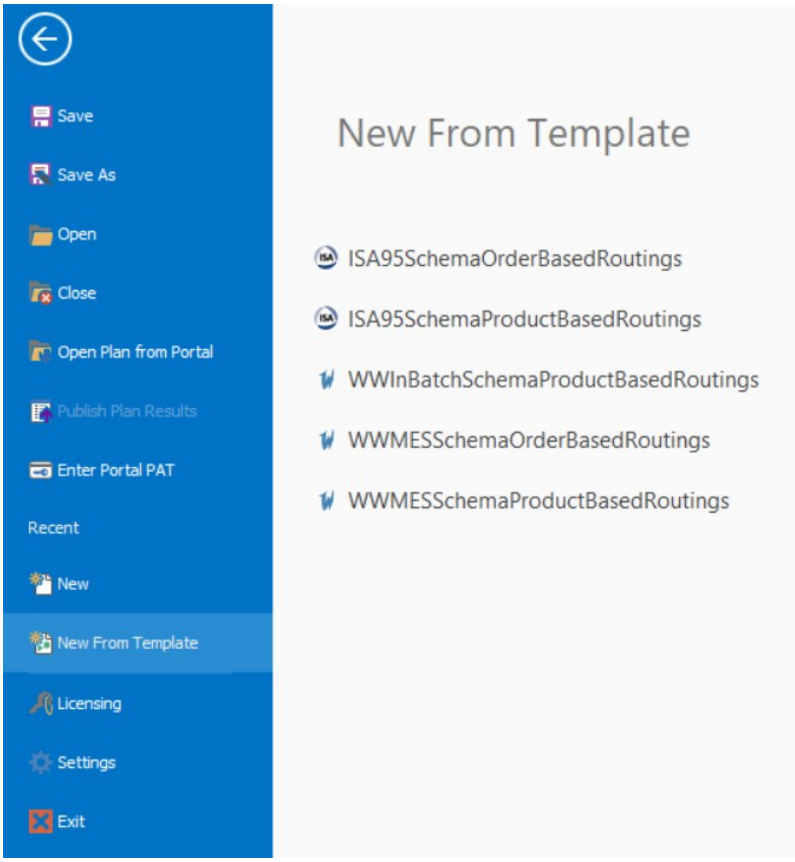


図 7.30　新規プロジェクト作成時に選択可能なテンプレート

Simio には、インポートしたテーブルとリレーショナルテーブルの内容に基づき、モデルコンポーネントとそのプロパティを自動作成する機能がある。詳細な手順は Simio のヘルプトピック Table-Based Elements (Auto-Create)を参照してほしい。この機能は既存のテンプレートに組み込まれており、たとえば Resources テーブルをインポートすると、リソースが自動的に Facility ウィンドウに配置される。

このデータ生成アプローチで完全なモデルを構築し、意味のある結果を得られるケースは限られている。より一般的なのは、既存のデータからまず基本モデルを生成し、それに段階的な改善を加えて意味のある結果を得る方法である。完全なモデルの生成を目指す場合は、既存のデータソースを拡張して必要なデータを追加できる。このアプローチは依然として非常に有効な手法である。

7.9　まとめ

データのインポート、保存、アクセスは、ほとんどのモデルにおいて重要な要素である。本章では、データの種類と Simio でのデータ格納方法について説明した。テーブル構造、特にリレーショナルテーブルは、データ処理に柔軟な手法を提供する。また、オブジェクトのデータを抽象化することで、モデルの使用とメンテナ

ンスが容易になる。モデルは大量のデータを必要とするため、可能な限りデータのインポート、場合によってはモデル全体のインポートを検討する価値がある。

7.10　問題

1. Model 7-1 においてデータテーブルで各エンティティタイプの優先順位を指定したが、まだモデルでは使用していない。重症度の高い患者が軽症の患者より先に治療されるよう、その優先順位を実装しなさい。実行前に予想される挙動を考察し、実行後、各エンティティタイプのシステム内滞在時間の変化を分析・報告しなさい。
2. (問題 1 で修正する前の)Model 7-1 の Server の能力を 4 に設定する。利用率(Server1.Capacity.ScheduledUtilization) と滞在時間 (Sink1.TimeInSystem.Average) を分析するため、以下の実験を行いなさい：10 日間のウォームアップ期間を設け、100 日間のシミュレーションを 25 回実行する。次に、季節変動により患者構成 (Patient Mix) が変化し、Urgent 患者が 10%に増加、Moderate 患者が 24%、Routine 患者が 36%に減少した場合の待ち時間を、変更前のモデルと比較しなさい。
3. 問題 2 で修正した Model 7-1 に、小規模救急病院の「再診 (Returns)」カテゴリを追加する。これは包帯交換、装具調整、ギプス除去、抜糸などで再来院する患者である。新しい患者構成は Returns 8%、Routine 30%、Moderate 26%、Urgent 10%となる。Returns 患者の処置時間は 3 分～30 分の範囲とする。この変更後の結果を問題 2 と比較しなさい。
4. より実態に即して、問題 3 の患者の約 3 分の 2 が日中（午前 8:00－午後 8:00）に来院すると想定する。1 日の総来院数を維持したまま、この来院パターンを反映するようモデルを修正しなさい。修正後のモデルのパフォーマンスを問題 3 と比較し、日中・夜間それぞれの滞在時間（Length of Stay；LOS）を測定する。両時間帯の平均 LOS を 0.5 時間以下に保ちながら、最小限の人員配置を実現する方策を検討しなさい。
5. 詳細分析により、問題 3 の再診患者の治療時間分布が判明した：10%が 20～30 分、残り 90%が 3～20 分を要する。この場合の新しい LOS を算出し、LOS を 0.5 時間以下に維持しつつスタッフの稼働効率を向上させる具体的な提案を経営陣向けにまとめなさい。
6. Routine および Return 患者は、待ち時間に対する忍耐力が比較的低く（30～90 分）、治療を受けずに帰ることがある（Leave Without Being Seen；LWBS）。一方、他の分類の患者は、診療を受けるまで待ち続ける。問題 3 のモデルを修正し、待ち時間の限界を超えた患者がシステムから離脱できるようにしなさい。さらに、平均待ち時間と、治療を受けずに帰った患者の割合を報告しなさい。
7. ある小さな無料クリニックでは、午前 8 時から正午まで 1 人の医師が患者を診察している。医師は 1 人の患者の診療に 6～14 分（平均 10 分）を要するため、理論上は 1 時間に 6 人、1 日に合計 24 人の患者を診ることができる。現在、このクリニックは 10 分間隔で患者の予約を入れているが、一部の患者は 15 分早く来院したり、30 分遅れて来院したりして、他の患者に迷惑をかけている。さらに、約 10%の患者は予約をキャンセルせずに来院しないため、医師の時間と他の患者が使える予約枠が無駄になっている。そこで、医師の稼働率を最大化するため、クリニックは予約方針の変更を検討している。たとえば、20 分おきに 2～3 人の患者を予約する方式に変更し、その効果を評価したい（目標は 1 日 24 人の診療を維持)。この新方式では、予定された全患者の診療が終了するまで医師が診療を続けることを想定するが、12:30 を超過すると医師の不満が高まる。システムの評価基準は以下の 3 点とする：実際に診療した患者数、患者の平均待ち時間、全診療完了までの医師の勤務時間。
8. 救急部門スタッフの大きな懸念は、来院時に重症 (Severe) と分類された患者が、治療

前に容態が緊急（Urgent）レベルまで悪化し、検査室（Trauma Rooms）での即時の応急処置が必要となるケースである。診察中の Severe 患者の 10%にこのような容態悪化が見られると仮定し、Model 7-2 を修正しなさい。修正前後のモデルで、待ち時間とスループットの変化を比較しなさい。特に、「即時の応急処置」をどのようにモデリングするかが重要となる。

9. 救急部門スタッフは 8.5 時間シフトで勤務し、各シフトには 1 時間の食事休憩（できればシフト中間）を含む「理想的な」スケジュールに従っている。食事休憩中は他のスタッフが業務を引き継ぐ。たとえば、「診療受付（Registration）」担当者は、「受付（Sign In）」担当者の食事休憩中にその業務をカバーする。各部門の最小要員は表 7.6 に示すとおりである。シフト交代時には、新旧スタッフ間の情報共有と引き継ぎのため 30 分のブリーフィング時間を設けている。Model 7-2 を更新し、合理的な勤務スケジュールを作成しなさい。スタッフの稼働率を再評価し、休憩中の「引き継ぎ」のモデリング方法を検討しなさい。

表 7.6 各サービスエリアにおける休憩時の最小必要人員（問題 9）

サービスエリア	最小要員
受付	1
診療受付	2
診察	4
治療	4
検査	1

10. 緊急時には、検査室（Trauma Rooms）のスタッフは、患者の処置中に食事休憩の時間が来ても業務を継続せざるを得ない。彼らは業務の合間に軽い食事を取るようにしている。この状況を反映するために問題 9 のモデルを更新し、「シフトにおける実質勤務時間」を見積もりなさい。

11.宝石加工システムの Simio モデルを設計・構築する必要がある。このシステムでは、鉱山から届いた石が研磨（polishing）、鑑定（grading）、仕上げ（finishing）の工程を経て完成する。一部の石は再研磨（repolishing）と再鑑定（regrading）が必要となる場合がある（フロー図は図 7.31 参照）。施設は 24 時間 365 日、3 シフト体制で稼働しており、鉱山からの石の到着は不規則である。表 7.7 には各シフトの時間帯と石の到着率を示している（注：このモデルでは休憩時間は考慮せず、シフト交代による生産性低下もないものとする）。到着プロセスは非定常ポアソン過程（NSPP）と仮定する。作業特性は表 7.8 に示すとおりである。全工程に作業者が 1 名配置されているが、再研磨と再鑑定の作業者は第 1 シフトのみ稼働する。不規則な到着とリソースのスケジュールに加え、以下の機能が必要である：

a) 石を表すエンティティは、プロセス開始時は緑色である。再研磨（repolishing）が必要な石は赤色に変わり、仕上げ（Finishing）工程で再び緑色に戻る。
b) 研磨（polishing）と再研磨（repolishing）工程の待ち行列内のエンティティ数、およびシステム内の石の平均数を表示するステータスプロットを作成する（3 項目を 1 つのプロットに表示し、時間軸は 24 時間とする）。
c) 再研磨と再鑑定の両工程内の石の数を時系列で追跡するユーザ定義統計量を作成する。
d) 各石の再研磨回数（未実施も含む）を追跡するユーザ定義統計量を作成する。
e) 再研磨が必要な石のシステム内滞在時間を追跡するユーザ定義統計量を作成する。
f) 200 日間の実験を 15 回反復実行し、以下の実験応答を求める：
    i. 再研磨・再鑑定工程内の石の平均数

ii.　再研磨と再鑑定工程内の石の最大数
iii.　1 個の石当たりの平均再研磨回数
iv.　再研磨が必要な石の平均システム内滞在時間
v.　研磨プロセスの稼働率

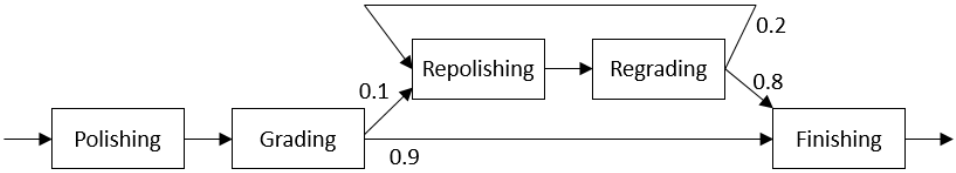


図 7.31　問題 11 の宝石処理システムのフロー図

表 7.7　問題 11 のシフト時間と石の到着率（1 時間当たりの石数）

シフト	時間帯			石の到着率
1	7:00 a.m.	–	3:00 p.m.	30
2	3:00 p.m.	–	11:00 p.m.	15
3	11:00 p.m.	–	7:00 a.m.	5

表 7.8　問題 11 の宝石加工に関わる作業プロセス

工程	作業時間（分）	シフト 1	シフト 2	シフト 3
研磨	Exponential(3)	Y	Y	Y
鑑定	Triangular(3, 3.33, 3.67)	Y	Y	Y
仕上げ	Uniform(2, 4)	Y	Y	Y
再研磨	Exponential(7.5)	Y	N	N
再鑑定	Triangular(6, 7.5, 9)	Y	N	N

第 8 章　アニメーションとエンティティ移動

本章では、これまでに学んだ概念を基に、アニメーションとエンティティ移動に関する知識を深めていく。まず、2D と 3D アニメーションの実用的な価値について、典型的なプロジェクトを例に検討する。次に、利用可能なアニメーションツールを紹介する。さらに、エンティティ移動については、フリースペース（ネットワーク外）移動を概説し、Standard Library オブジェクトが提供するエンティティ移動機能を詳しく解説する。

8.1　アニメーション

4.8 節で基本的なアニメーション概念を紹介したが、本節ではそれらをより詳しく解説する。最初に、アニメーションの重要性とその適切な活用について説明し、後半では、アニメーション作成に必要な追加ツールとテクニックを紹介する。

8.1.1　なぜアニメートするのか

これまでのモデルにも、さまざまなアニメーションを取り入れてきた。テキストベースの商用シミュレーション製品も一部残っているが、この数十年で、ほとんどの製品に標準的なアニメーション機能が搭載されるようになった。これは、アニメーションがモデルの構築と理解を大幅に容易にするためである。

8.1.1.1　システムの理解

シミュレーションモデルは大規模で複雑になりがちである。モデル製作者にとって、この複雑さを管理し、詳細と全体像の両方を把握することは容易ではない。しかし、簡単なアニメーションでも、可視化によって理解が深まる。理想的なシミュレーションソフトウェアとは、基本的なアニメーション作成が極めて容易なものである。現在の多くの製品がそうであり、実際、Simio でも 2D アニメーションはデフォルトで簡単に作成でき、モデル製作者の負担は最小限で済む。

有効なモデル**検証**には多くの要素があるが、アニメーションの綿密な観察は特に重要である。モデルの進行を段階的に確認することで、詳細な検討が可能になる。また、イベント周辺でのモデルの反応を観察することで、一般的な状況か特殊な状況かを即座に判断できる。さらに、アニメーションを早送りや一時停止しながら詳細を観察するだけでも、多くの問題点や改善の機会を見出すことができる。

注意：アニメーションは検証において重要だが、それは検証の唯一の方法**ではなく**、また主要な手段で**あるべきではない**。有効な検証には、さまざまな技法を組み合わせる必要がある。

8.1.1.2　コミュニケーション

シミュレーションの主要な目的は、システムの理解と利害関係者への価値提供である。アニメーションはこの両方に貢献する。利害関係者にモデルロジックの数値やダイアグラムを提示しても、多くの場合、一目見て無視されるか混乱を招くだけである。しかし、親しみやすいアニメーションを見せると、たちまち興味を

示す。彼らは自身の施設に関する知識と、アニメーションで示される要素、動き、状況を即座に比較し始める。まるで実システムを窓越しに観察するかのように、設計についての議論を始めることもある。このように、アニメーションは前述の二つの目的達成への確実な第一歩となる。最新技術では、利害関係者が外部とのコミュニケーションに使用できるじゅうぶんな品質のアニメーションが可能であり、プロジェクトの推進や組織の発展にも活用されている。

もちろん、この機能には通常コストがかかる。**利害関係者**向けの適切なアニメーション構築は、使用するモデリングソフトウェアに大きく依存する。特に、アニメーションがモデルと別個に構築される製品では、その依存度は高い。ときには、アニメーションが完全に別のソフトウェアで作成され、モデル実行後の**後工程**として実行されることもある。このような後処理型の分析では、モデル実験の範囲が著しく制限される。

幸いなことに、Simio のデフォルトアニメーションは、多くの場合「利害関係者の期待」に近い。Simio ではアニメーションが組み込まれており、モデル構築と**同時**に作成される。そして、アニメーションを確認しながら、完全に**対話的**にモデルを実行できる。

8.1.1.3　3D アニメーションの重要性

ここまで 3D アニメーションについては触れていなかった。その必要性や用途に疑問を持つかもしれない。15 年前なら、3D アニメーションはほとんどのプロジェクトで不要で、労力に見合わないと判断されただろう。しかし、状況は大きく変化した。高品質なグラフィックスは一般的となり、日常生活に浸透している。今では多くの人が 3D アニメーションを容易に扱え、その利点を理解している。この新しい基準により、2D アニメーションは時代遅れに見え、利害関係者の関心を引きにくい。3D と同等の理解や信頼を得ることも難しい。さらに、3D アニメーション作成に必要な労力は大幅に減少した。モデル構築者は高度な描画技術や大規模なカスタムシンボルライブラリ、複雑なグラフィックソフトの操作技術を必要としない。現代のソフトウェアでは、基本的な芸術的センスがあれば、特殊な技能やツール、カスタムライブラリなしでも、説得力のあるアニメーションを作成できる。

アニメーションは重要で、多くの人にとって魅力的であり、その価値は広く認識されている。しかし、どんなよいものでも、行き過ぎは本末転倒となる。アニメーションを「完璧に」作り込もうとすると、モデル構築、妥当性確認、分析など、プロジェクトの他の重要な部分にじゅうぶんな時間を割けなくなる。アニメーションは実システムの近似的な表現に過ぎないことを認識しておく必要がある。プロジェクトの目的達成に必要じゅうぶんなアニメーションを開発することが重要である。利害関係者とともにアニメーションの適切なレベルを決定し、その計画に忠実に従うべきである。プロジェクトの他の重要な側面をすべて完了させた後で、アニメーション構築の時間を確保することを推奨する。

8.1.2　ナビゲーションと View のオプション

Facility ウィンドウの上部にグレーの領域が表示されている（まだ無効にしていない場合）。表示をオン・オフするには、Facility ウィンドウで H キーを押す。以下で、これらのナビゲーションコントロールについて詳しく説明する。

・2D と 3D の表示を切り換えるには、2 キーまたは 3 キーを押すか、View リボンの 2D/3D ボタンをクリックする（図 8.1）。

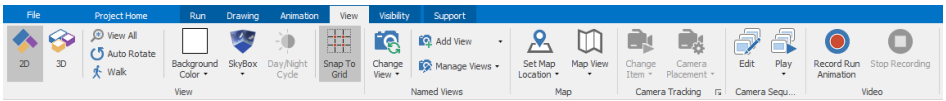


図 8.1　Simio View リボン

・空いているスペースで左クリックして Facility ウィンドウをドラッグすると、ビューをパンできる。**注意**：オブジェクト（特に大きな背景オブジェクト）をクリックすると、パンではなくオブジェクトが移動してしまう。これを防ぐには、マウスの中ボタン（スクロールホイール）でクリック&ドラッグするか、背景オブジェクトを右クリックして Lock Edits を選択する。
・マウスホイールをスクロールすると、視点を拡大・縮小できる。マウスホイールがない場合は、Control キーを押しながら右クリックし、上下に動かすことで同様の操作が可能である。
・オブジェクトを選択すると表示される緑色のドット（ハンドル）を使って、サイズ変更や回転ができる。ハンドルをクリック&ドラッグでサイズを変更し、Control キーを押しながら左クリックで回転できる。
・View リボンの上の View All ボタンをクリックすると、モデル全体が表示されるようにウィンドウが調整される。ズームやパンでオブジェクトを見失った場合に、元の視点に戻すのに便利である。
・View リボンの Background Color ボタンで 2D・3D 両方の背景色を変更できる。以前のモデルでは 3D で黒背景が使われていたが、現在は 2D と 3D で異なる背景色は設定できない。
・SkyBox と Day/Night Cycle ボタンでアニメーションの背景表示や変化を制御できる。これは港や輸送ネットワークなど、屋外のモデルで特に有用である。

3D ビューのときだけ適用できる操作もある：

・右クリックしたまま左右にドラッグすると、3D ビューの視点が回転する。
・右クリックしたまま上下にドラッグすると、ズームイン・アウトができる。
・Shift キーを押しながらオブジェクトを移動すると、**床面**に対して垂直方向に動かせる。
・Control キーを押しながらオブジェクトを移動すると、他のオブジェクトの上に**積み重ねる**ことができる。
・3D ビューで R キーを押すか、View リボンの Auto Rotate ボタンをクリックすると、3D ビューが**回転し始める**。Escape キーで回転を停止できる。

図 8.2 の Visibility リボンではアニメーションの表示方法を細かく調整できる：

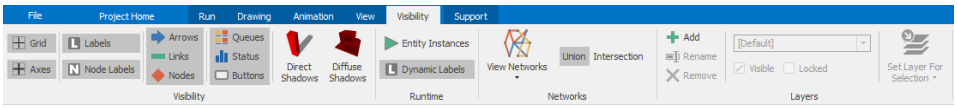


図 8.2　Visibility リボン

・Visibility グループには、特定の構成要素の表示をオン・オフできるボタンがある。より現実的なアニメーションを実現するため、多くの場合はほとんどの表示を無効にする。ただし、モデルを確認する際にはボタンを有効にする必要がある。たとえば、Nodes アニメーションが無効の場合、ノードを選択できなくなる。
・Visibility グループの `Direct Shadows` と `Diffuse Shadows` ボタンで、オブジェクト周囲の影を詳細に調整できる。
・Networks グループのボタンで、移動経路のネットワークを個別またはグループで表示できる。
・Layers サブセクションでは、オブジェクト、アニメーション、グラフィックをモデル内の定義済みレイヤに配置できる。

これらの機能の詳細な使用方法は、Simio のヘルプを参照してほしい。
モデルの準備が整ったら、View リボンで以下の機能も活用できる：

・**View**（**ビュー**）では、2D・3D 表示、ズーム比、回転などのモデル表示方法を設定できる。Named View グループの `Add View` ボタンで現在の視点に名前をつけられる。`Manage Views` で既存の視点の編集や削除ができる。`Change View` ボタンは 2 段構成で、上部は命名された視点を順に切り替え、下部はリストから視点を選択できる。
・Camera Tracking グループには、最適なカメラポイントを設定するツールがある。特定のオブジェクトにカメラを固定したり、エンティティや乗り物などの**動くオブジェクト**の前後からの視点を設定したりできる。
・Camera Sequence グループでは、プレゼンテーションに向けて「ストーリー性のある」カメラワークを作成できる。複数のカメラの動きとタイミングを設定可能である。
・最後に、Video グループでは、アニメーションと関連アクティビティをビデオ（avi）ファイルとして**録画**できる。

View と Visibility リボンに加えて、Project Home リボンにも表示オプションがある。`Render to Oculus` ボタンで、接続された Oculus HMD デバイスへの 3D ビューのレンダリングを制御できる。この技術は比較的新しく高価だが、魅力的なアニメーション効果を生み出せる。

8.1.3　Drawing リボンによる背景アニメーション

次に、モデルに表示される静的なアニメーション（動きのない要素）について説明する。Simio は描画ツールを主目的としていないが（Google SketchUp などの

無料の高機能ツールが利用可能)、基本的な描画・ラベリング機能を提供している。また、シンボルのインポートとその外観の調整も可能である。これらのツールを活用することで、モデルの現実性と信憑性を簡単かつ迅速に高めることができる。図 8.3 の Facility ウィンドウの Drawing リボンを見てみよう。

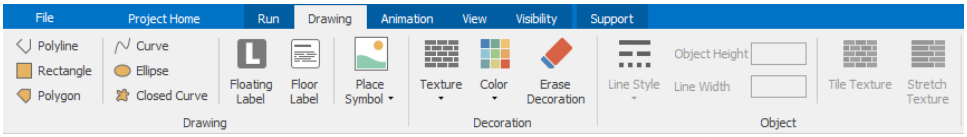


図 8.3　Facility ウィンドウの Drawing リボン

図 8.3 の左側の Drawing グループには、6 個の基本的な描画ツールがある：**直線**（Polyline)、**長方形**（Rectangle)、**多角形**（Polygon)、**曲線**（Curve)、**楕円**（Ellipse)、および**閉曲線**（Closed Curve)。これらの形状に慣れていない場合は、実際に試してみるとよい。図形の描画時は、最後のポイントをクリックした後に右クリックで描画を終了する。Escape キーを押すと描画を中止し、配置したポイントが消去される。図形には回転ポイントという別のポイントが表示されることがある。この回転ポイントは自由に移動でき、他のポイントを Ctrl キーを押しながらクリック&ドラッグすると、オブジェクト全体がこの回転ポイントを中心に回転する。

Drawing リボンの Decoration グループと Object グループには、オブジェクトの外観を変更するツールが用意されている。色、テクスチャ（パターン)、テクスチャのタイル設定や伸縮、線のスタイル、幅、オブジェクトの高さなどを調整できる。すべての機能の詳細は省略するが、創造的な活用例をいくつか紹介する：

- 壁を作る：まず壁の長さの線を引く。Line Width を 0.1 メートル、Object Height を 2 メートルに設定し、3D ビューに切り替える。Texture ボタンをクリックし、テクスチャ（レンガパターンが適している）を選択して壁オブジェクトに適用する。
- 会社のロゴを作成する：ロゴの JPG ファイルを共有フォルダ¥Simio¥Skins フォルダに保存する(パスは OS により異なる)。すると、そのファイルが Texture ボタンの選択肢として表示される。ロゴの形に合わせて長方形を描き、上面か側面にロゴテクスチャを適用する。または、長方形を縦長に設定してサイドにロゴやテクスチャを適用すると、サイン、ビルボード、テレビモニタのような表示になる。
- 簡単な建物を作成する：長方形を作成して高さを設定し、側面にテクスチャを適用する。または、実際の建物の表面写真を撮影して適用することもできる（Trimble 3D Warehouse や Google Maps から、より質の高い建物の写真を入手できる）。

マップ機能については、Simio は View リボンで GIS（グラフィック情報システム）マップをサポートしている。まず、マップビューを選択し、住所または緯度/経度で地図の位置を設定する。インタラクティブな背景が表示され、その上にノードを配置できる。2 つのノードを選択すると、表示された道路システムを使用

して GIS システムがノードを接続する。この機能にはインターネット接続が必要で、オフライン時にモデルを読み込むと背景は表示されない。

Drawing リボンには他に 3 つの重要なボタンがある。**FloatingLabel** は空間に浮かぶラベルを作成でき、視点や表示倍率に関係なく常に正面を向き、一定のサイズで表示される。**Floor Label** は床面に描かれたような表示になり、複数行のテキストや色、サイズ、書式の設定が可能である。また、動的な情報表示のためにテキストに式を組み込むこともできる。

Place Symbol はもっとも汎用性の高いボタンで、モデル背景へのシンボル配置に多くのオプションを提供している。ボタン上部では最後に選択したシンボルを簡単に配置でき、下部をクリックすると 3 つの主要な選択肢を含むダイアログが表示される：

- 4.8 節で説明したように、内蔵の Simio シンボルライブラリからシンボルを選択できる。
- `Import Symbol` では、ローカルファイルを取り込める。Sketchup で作成したファイル、CAD プログラムからエクスポートした DXF ファイル、JPG、BMP、PNG などの画像ファイルに対応している。ただし、DXF ファイルは詳細すぎる場合がある（建築構造のナットやボルトの細部まで含むなど）。DXF ファイルを使用する場合は、エクスポート前に不要な要素を削除することを勧める。JPG や PDF の 2D 建物レイアウトは、アニメーションのよい基礎となる。これにより寸法と背景を効率的に配置でき、その後 3D オブジェクト（壁など）を追加して奥行きを表現できる。
- `Download Symbol` では、4.8 節で説明したように Google 3D ギャラリーから検索できる。可能な限り、ポリゴン数の少ない「シンプル」なシンボルを選択することを勧める。Google 3D ギャラリーのシンボルも DXF ファイルと同様の問題があり、過度に詳細な表現はシミュレーションアニメーションには不向きである。たとえば、完全なフォークリフトモデルを選んだ場合、マフラーブラケットのボルトまで詳細に作り込まれているため、モデルサイズが 10MB にも膨れ上がることがある。このような場合は、シンボルを Sketchup に読み込んで不要な詳細を削除できる。

8.1.4　Animation リボンの状態変数アニメーション

オブジェクトの動きを画面で確認できるだけでなく、モデルのパフォーマンスを把握するための視覚的フィードバックや対話機能も必要だ。Simio はこれらの機能を提供するツールセットに加えて、表示方法（Console ウィンドウまたは Facility ウィンドウ）も選択できる。

8.1.4.1　Console ウィンドウ

すべてのモデルには **Console ウィンドウ**があり、選択した**状態変数**をグラフィカルに表示したり、対話型ボタンを配置したりできる。Console ウィンドウは Definitions タブで定義する。このタブは、ライブラリオブジェクトを配置できない点を除いて、Facility ウィンドウと同様の外観である。Console ウィンドウに

は、本節で説明する項目のみを配置できる。

Console ウィンドウに状態変数項目を配置する利点は 2 つある：

・Facility アニメーションとは別に**ビジネスグラフィックス**を管理できる。必要に応じてグラフィックスを非表示にしたり、表示用の画面領域を確保したりできる。
・モデルを後でオブジェクトとして使用する場合、このコンソールはオブジェクトの右クリックメニューから利用できる。オブジェクトのユーザは、設計者が作成した状態変数の表示を確認できる。埋め込みオブジェクトのコンソールには追加の利点があり、利用可能な場合は 2 台目のモニタなど、Simio ウィンドウの外部に表示できる。**ヒント**：コンソールはオブジェクトの**ドキュメンテーション**の保存に最適な場所であり、フロアラベルとして追加するだけでよい。

8.1.4.2　Facility ウィンドウ

状態変数アニメーションは Facility ウィンドウにも配置できる。この方法の利点は、関連するオブジェクトの近くに配置できることと、モデル内の異なる場所に配置して名前つきビューと関連づけられることだ。主な欠点は、これらが主に 2D のビジネスグラフィックスであり、3D Facility ウィンドウでは必ずしも見栄えがよくないことである。

8.1.4.3　状態変数アニメーションツール

状態変数オブジェクトの表示ウィンドウに関係なく、選択用の同じようなボタンが用意されているが、これらは 8.1.3 項で説明した Facility ウィンドウの Drawing リボンとはまったく異なる。Facility または Console ウィンドウに状態変数オブジェクトを追加するには、Animation リボン（図 8.4）をクリックする必要がある。

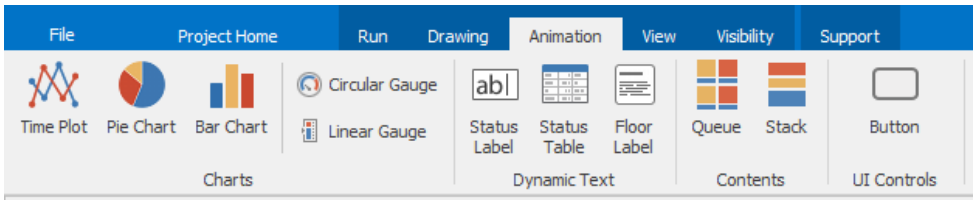


図 8.4　Facility ウィンドウの Animation リボン

Animation リボンに含まれるツールには、次のものがある：

・**Time Plot**（折れ線グラフ）：時間経過に伴う 1 つ以上の値の変化を表示する。
・**Pie Chart**（円グラフ）：複数の値を合計に対する割合として比較する。
・**Bar Chart**（棒グラフ）：式の値の時間変化を表示する。
・**Circular Gauge** と **Linear Gauge**（円形・直線ゲージ）：モデルの値を視覚的にわかりやすく表示する。
・**Status Label**（状態変数ラベル）：静的テキストや任意の式の値を表示する。
・**Status Table**（状態変数テーブル）：別のソーステーブルで定義された式の値を表示する。

・**Floor Label**（フロアラベル）：8.1.3 項で説明した Floor Label と同様。
・**Queue**（待ち行列）：待ち行列内の待機中エンティティを表示する。
・**Stack**（スタック）：コンテナの内容を状態変数スタックバーとして表示する。
・**Button**（ボタン）：ユーザとモデルの対話機能を提供する。クリック時に（プロセスにリンクされた）イベントが発生する。

8.1.5　Symbols リボンによるシンボルの編集

Facility ウィンドウでオブジェクトをクリックすると、アクティブなリボンは自動的に Symbols リボン（図 8.5）に切り替わる。このリボンでシンボルの変更やアニメーション機能の追加を行い、オブジェクトのアニメーションをカスタマイズできる。一見多くの新しいオプションがあるように見えるが、実際には多くの機能が既出のものだ。たとえば、4.8 節で新しい ATM Customer を選択する際に使用した Project Symbols カテゴリや、`Import Symbol` と `Go To 3D Warehouse` リボンボタンは、Drawing リボンで説明した `Place Symbol` ボタンと同じオプションである。また、Decoration グループの項目も Drawing リボンの Decoration グループと同一である。

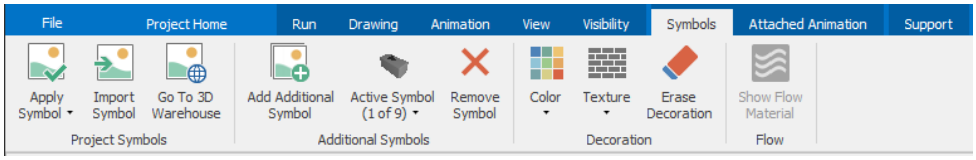


図 8.5　シンボルの編集のための Symbols リボン

8.1.5.1　Attached Animation（付属アニメーション）

Attached Animation グループは図 8.4 の Animation リボンと外見は同じだが、「Attached」という言葉が示す重要な違いがある。このリボンはオブジェクトを選択すると開かれ、Attached Animation グループからアイテムを配置すると、それらは選択したオブジェクトに**付属**する。Server などの固定オブジェクトの場合、画面上でオブジェクトを移動させると、付属アイテムも一緒に動く。さらに、Entity、Vehicle、Worker などの動的オブジェクトに付属させた場合は興味深い動作をする。モデル内を移動する際、付属アイテムはそれぞれの動的オブジェクトとともに移動する。以下に実用的な例を見てみよう：

・デフォルトの Vehicle シンボルにある緑色の水平な線は、乗り物が運ぶエンティティを表示するための `RideStation.Contents` をアニメートする付属待ち行列である。
・同様に、Combiner オブジェクトでエンティティをグループ化する場合、親エンティティオブジェクトに付属待ち行列を追加してグループメンバーを表示できる。`BatchMembers` 待ち行列をアニメートすると、現在の親バッチのメンバーを確認できる。この例は SimBits の `CombineThenSeparate` と `Regenerating Combiner` で確認できる。
・エンティティに関連するテキストや数値情報も表示できる。SimBit の `OverflowWIP` では、各エンティティの生成時間を確認するため、エンティティ

の式 TimeCreated を使用して付属の状態変数ラベルを追加した。また、待ち行列内でも読めるようラベルを回転させた。

付属情報の追加は、視覚的な確認とモデルの検証に非常に役立つ。付属アニメーションはオブジェクトの一部となるため、その範囲はオブジェクト自体に限定される。これにより、オブジェクトの視点からあらゆる式に参照を設定できるという利点がある。そのため、上記の例では、Vehicle1.RideStation.Contents ではなく、RideStation.Contents という参照を使用した。前者はオブジェクトではなくモデル自体の範囲にあるため機能しない。

8.1.5.2　Additional Symbols（追加シンボル）

Symbols リボンの最後のグループは Additional Symbols グループだ。この機能では、オブジェクトに関連づけられたシンボルセットに対して、シンボルの追加、編集、削除が可能だ。デフォルトでは各オブジェクトには 1 つのシンボルしかないが、多くの場合、複数のシンボルが必要になる。以下に有用な例を示す：

1. 1 つのエンティティタイプで多様性を表現したい場合（例：Person エンティティで 5～10 種類の異なる人物シンボルを表示したい場合）。
2. モデル内の進行状況に応じて変化するエンティティ（例：処理の進行や検査不合格時に見た目が変わる部品）。
3. 各状態（Busy、Idle、OffShift など）に応じて異なる見た目を表示したい Server（または他のオブジェクト）。

ほとんどのオブジェクトの Properties ウィンドウには、Current Symbol Index や Random Symbol などのプロパティを持つ Animation カテゴリ（図 8.6）がある。これらのプロパティは、少なくとも 1 つの追加シンボルをオブジェクトに加えるまで無効となることに注意が必要だ。これらのプロパティを使用して、指定したシンボルから選択できる。

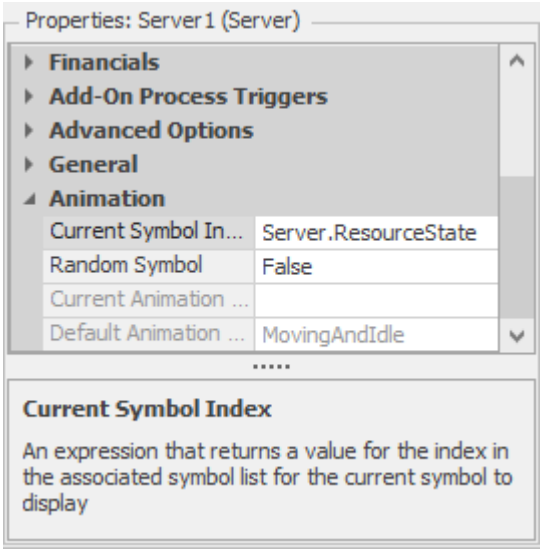


図 8.6　サーバに対するデフォルトのアニメーションプロパティ

オブジェクトが作成されると、Random Symbol プロパティが表示される。このプロパティを True に設定すると、定義済みシンボルのセットから１つのシンボルが自動的に選択される。たとえば、上のケース１で３種類のシンボル（男性、女性、子供）を定義した場合、Random Symbol を True に設定すると、各エンティティは３分の１の確率で各シンボルが割り当てられる。

Current Symbol Index プロパティは、オブジェクトのシンボル番号の参照先をSimioに指定するために使用する。シンボルには０から順に番号がつけられる。たとえば、５つのシンボルがある場合、０から４までの番号が割り当てられる。上のケース２をアニメートするには、ModelEntity.Picture に Current Symbol Index を設定する。その後、エンティティがモデル内で変化する際、表示したいピクチャのインデックスにエンティティ状態変数を割り当てる。

Current Symbol Index プロパティは、オブジェクト状態変数だけでなく、式も使用できる。たとえば、SimBit の OverflowWIP では、部品の作成時間を視覚的に表現するため、エンティティの Current Symbol Index に式 Math.Floor (ModelEntity.TimeCreated)を使用した。時間が数時間にわたるため、１日の各時間帯に新しいシンボルを割り当てている。

Server のデフォルト Current Symbol Index プロパティは_ServerName_.ResourceState である。これは、Starved、Processing、Blocked、Failed、OffShift、FailedProcessing、OffShiftProcessing、Setup、OffShiftSetup の標準リソース状態が０から８である９つのシンボルを想定している。**ヒント**：ヘルプにおいて「List States」（リスト状態変数）トピックを見て、他のオブジェクトの状態変数の定義を見てほしい。

8.1.6　よりリアルなエンティティアニメーション

4.8 節では、Simio のシンボルライブラリとアニメーション人物を含むフォルダについて説明した。図 4.34 の下段に示したフィルタ（Domain、Type、Action）を使用すると、最適なシンボルをより効率的に見つけることができる。特に Action フィルタには、Animated と Static の２つの選択肢がある。「Animated」とは、シンボル自体に動きを持たせたものを指す。Simio では、上級者向けに、**あらゆる**タイプのシンボル（火花を散らす機械や車輪の回転する乗り物など）でこの機能を提供している。付属のシンボルライブラリでは人物のアニメーションのみが提供されているが、これを活用して説得力のあるアニメーションを作成できる（図 8.7）。

多彩な人物、服装、色が収録されている。男性、女性、女の子、男の子、お年寄り、アニメのキャラクタなど、幅広い選択肢がある。ビジネスウェア、ヘルスケアウェア、カバーオール、インフォーマルウェアを着用したモデルが用意されており、さまざまな人種・民族の背景も考慮されている。これらのシンボルの最大の特長は、リアルな「動き」を表現できることだ。特に歩行時のアニメーションは、脚や身体が自然に動くため、非常に説得力がある。図 8.8 では、標準の Simio people で利用できる 22 種類の定義済みアクションを紹介している。**ヒント**：このメニューは、アクティブシンボルが適用されているとき（例：エンティティに「アニメー

ション」シンボルを適用したとき）に限り、List Animations of Active Symbol 下に表示される。

図 8.7　ATM を使用する人物のアニメーション

Available Animations

These are the available animations for this object. You can set the Current Animation Index expression to either an integer corresponding to the Index value, or to a string corresponding to the animation Name.

Index	Name
1	Stand
2	Stand Shifting Weight
3	Stand Talking on Phone
4	Stand Texting
5	Stand Picking Up
6	Stand Moving Hands
7	Stand Carrying Front
8	Stand Carrying Side
9	Walk
10	Walk Carrying Front
11	Walk Carrying Right
12	Walk Carrying Left
13	Walk Pushing
14	Run
15	Sit
16	Sit Legs Crossed
17	Sit Talking
18	Sit Moving Hands
19	Sit Driving
20	Sit to Lie Down
21	Sleep
22	Dance

図 8.8　人物アニメーションで利用可能なアクション

リストを見るとわかるように、非常に自由度が高いアニメーションが可能だ。たとえば、芝刈り機や車椅子を押す人のアニメーションを作成したい場合はアクション 13「Walk Pushing」を選択する。また、病院のベッドで寝ている人をアニメートしたい場合はアクション 21「Sleep」が適している。これらのアクションの多くは、アニメートされた人物のシンボルを選択した時点で自動的に組み込まれる。たとえば、アニメーションの人物が待ち行列で待機している際、よりリアルな表現のためにさまざまなスタンディングアクション（1～8）がランダムに繰り返される。

アニメーションの動作を制御する一つの方法は、図 8.9 に示されたプロパティを調整することだ。特定のアクションを指定する場合、ModelEntity.Animation にそのアクションを割り当てる。たとえば、待ち行列に入る直前に ModelEntity.Animation を Sit に設定すれば、待ち時間中は座った状態になる。その後、待ち行列から出る際に ModelEntity.Animation に**空白**を設定することで、デフォルトの動作に戻る。

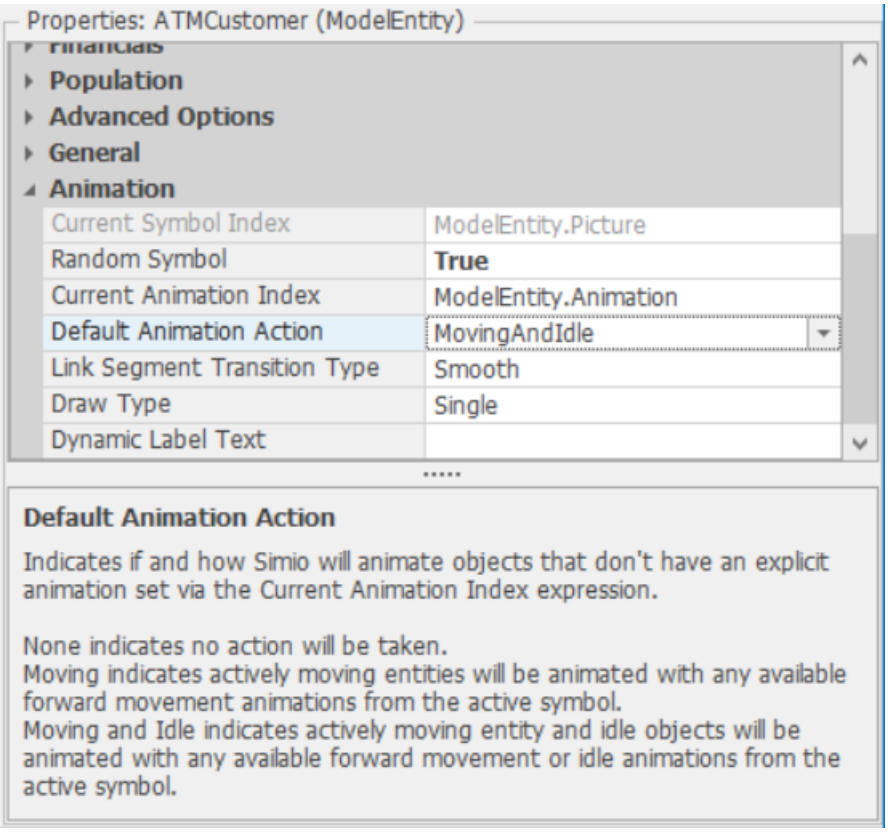


図 8.9　エンティティの Animation プロパティ

8.1.7　Model 8-1：PCB 組立のアニメート

上述の新しい知見を活用して、PCB モデルの作業を進めていこう。ただし、開始前に以前の注意事項を念頭に置き、**プロジェクトの目的に必要じゅうぶんな詳細さでアニメーションを開発する**ことにする。適切なシンボルが見つからない場

合や、理想的な表示ができない場合もあるだろうが、それでじゅうぶんなのだ。合理的な近似でじゅうぶんな場合が多い。**時間と指示が与えられてから、必要に応じてアニメーションを改良すればよい。**

まずは 5.4 節で説明した PCB モデル Model 5-3 を読み込み、Model 8-1 として保存しよう。これは仮想のシステムなので、アニメーション作成にある程度の自由度がある。最初に、PCB オブジェクト用のより適切なシンボルを探そう。

- PCB オブジェクトをクリックし、Symbols ライブラリを開く。ライブラリ内で PCB（コンピュータの小さい基板）として使えるシンボルを探す。残念ながら、適当なものが見つからないので、別の方法を試してみよう。
- PCB オブジェクトを再度クリックし、Download Symbol ボタンを選択する。オンライン接続時は、Trimble 3D Warehouse 検索画面が表示される。PCB を検索すると、多くの候補が見つかった中から、Gumstix Basix R1161 を選んで Download Model を実行する。**ヒント**：検索結果の最初に表示される Nat's PCB は適切に見えるかもしれないが、ダウンロード後にモデルサイズが数メガバイト増加することに注意してほしい。これは、シンボルが複雑で非常にポリゴンが多いためである。大きなシンボルを誤ってダウンロードした場合は、より小さいものに置き換え、Project→Symbols ウィンドウ（Navigation ウィンドウのフォルダアイコンから左パネルの Symbols View を選択）から削除できる。ただし、削除前にそのシンボルがモデル内で使用されていないことを確認する。使用中の場合、シンボルインスタンスはモデル内に残る。
- シンボルのダウンロード後、Simio Import ウィンドウが開く。ここでシンボルの回転、サイズ変更、保存ができる。短辺が右になるよう Rotate ボタンで回転させ、最長寸法を 0.3 メートルにするため Width を 0.3 に設定する（他の寸法は比例して変化する）。最後に Name を PCB に、Documentation を Gumstix Basix に変更する。
- シンボルは小さく表示されるが、これは後で調整するので問題ない。

次に、同様の手順で部品配置機械を変更しよう。

- 部品配置機械は Simio シンボルライブラリにないため、そのステップは省略する。Placement オブジェクトをクリックし、Download Symbol ボタンから 3D Warehouse 検索画面を開き、placement machine を検索する。
- Fuji QP351 を選択してダウンロードする。方向とサイズが適切なため、OK をクリックするだけで Placement オブジェクトに適用できる。

3 つの並列ステーションも部品配置機械だが、より高性能な機械を表現するため、異なるシンボルを選ぼう。上側の機械から作業を始め、先ほどと同じ手順を繰り返す。より高性能に見える Fuji QP 242e を選択してダウンロードし、180 度回転させて幅を 1.5 メートルに調整する。

シンボルをダウンロードするたびに、プロジェクトライブラリに追加される。残りの 2 つの並列機械にも、先ほどダウンロードした同じ部品配置機械を適用できる。検査と再加工は手作業なので、適切な机のシンボルでアニメートする。新た

なシンボルをダウンロードすることもできるが、Equipment カテゴリ内の TableSaw が作業台としてじゅうぶんなので、両方の手作業ステーションにこれを使用する。

ここまでの作業で違和感を覚えるかもしれない。シンボルは 2D 表示の白地では見づらく、かなり小さく見える。これはモデルの詳細度の問題だ。以前のモデルではステーション間の移動を瞬時と仮定し、部品の詳細や機械のサイズ、その近似に注意を払わなかった。このような単純化はプロジェクト初期では有効だが、やがて重要な課題となる。まさに今がその時期である。

多くの場合、背景画像として使用できるシステムレイアウトの JPG などのファイルがあり、それを基に設備シンボルのサイズと配置を調整できる。この例ではそれがない。元のモデルでは機械を約 10 メートル間隔で配置したが、実システムでは 1～2 メートルしか離れていない。機械を移動する前に、各サーバに関連するオブジェクトを設定しよう：

・3 つの並列ステーションが画面いっぱいになるようズームする。
・画面下側（もっとも見やすい位置）から作業を始める。入力ノード（Input@FinepitchSlowStation）をクリックし、部品配置機械シンボルの入力側に隣接するようドラッグする。出力ノードも同様に配置する。2D と 3D 表示を切り替えながら両方の視点で確認して調整するとよい。これでアニメーションのノードとパスは図 8.10 のような配置になる。

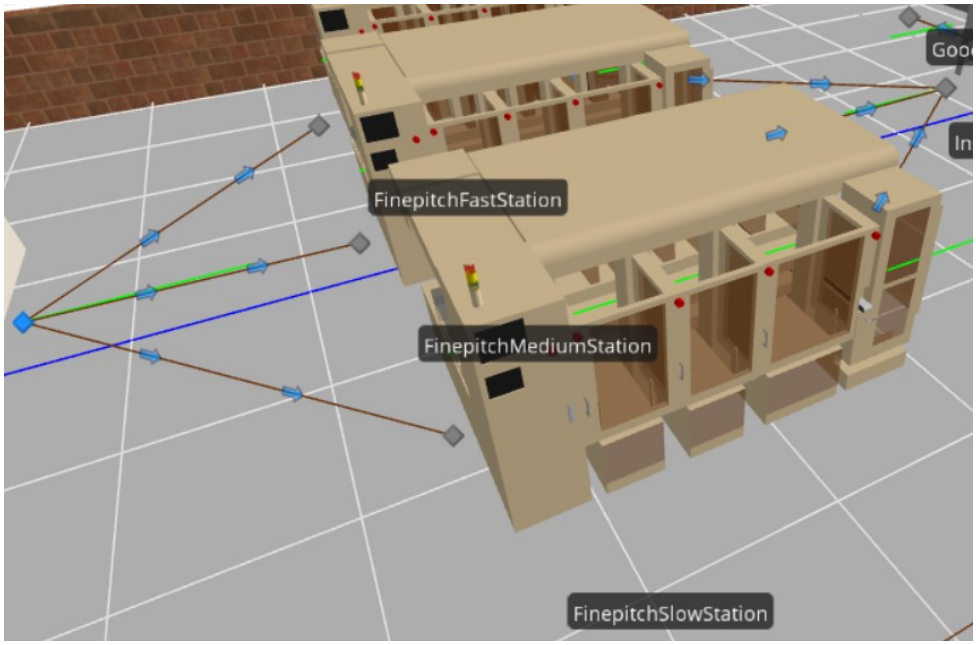


図 8.10　FinepitchSlowStation のアニメーション編集

・Processing.Contents という待ち行列（2D 表示では機械上の緑色のライン）は、現在処理中のものを表示する。これを機械上（実際には内部）に配置するとより見やすくなる。2D 表示で待ち行列を部品配置機械の土台部分にドラッグす

る。この時点では床面にあるので、3D 表示に切り替えて Shift キーを押しながらドラッグし、土台部分の上に持ち上げる。図 8.10 のように、土台部分の中心に待ち行列が来るよう両端を調整する。**ヒント**：大きなオブジェクトの内部では小さなシンボルを見失いやすいため、位置が確認できるまでは表示オブジェクトより長い待ち行列のままにしておき、最後に長さを調整するとよい。

- Output Buffer 待ち行列を短くし、出力側に隣接するよう移動する。入力バッファの容量はゼロのため、入力側の待ち行列は不要で、混乱を避けるために削除する。
- 残りの 2 台の部品配置機械でも同じ 3 ステップを繰り返す。
- Placement、Inspection および Rework も同じ手順でアニメーションを修正する。ただし、入力バッファ待ち行列は使用する可能性があるため、削除せずに残しておく。

上述のすべてを完了しても、モデルはまだ完成度が低く見える。実行すると、移動する PCB シンボルは小さな点のように見える。シンボルと関連オブジェクトのスケールは設定したが、レイアウト全体のスケールをまだ実寸に合わせていない。ここでは、その調整を行う。

- 2D ビューで、FinepitchSlowStation を選択し、FinepitchMediumStation から約 1 メートル上方に平行に配置する（グリッドを使うと便利である）。次に、FinepitchFastStation を選択し、FinepitchMediumStation から約 2 メートル下方に平行に配置する。この時、待ち行列とノードは各オブジェクトと一緒に移動することに注意する
- 残りのオブジェクトも、それぞれの間隔が約 2 メートルになるように移動する。
- 標準ライブラリのシンボルを使用しているオブジェクトは現在かなり大きく表示されている。必要に応じて他のシンボルに置き換えることもできるが、ここでは角をドラッグして適切なサイズに縮小するだけにする。

最後の仕上げとして、床と壁を追加する。

- 2D Facility ビューで、Drawing リボンの Rectangle ボタンを使用して設備の床面積全体を覆う長方形を描く。Color ボタンで床をグレーに設定する。グリッドを透過させたい場合は、3D ビューに切り替えて Shift キーを押しながらドラッグし、床を少し下げる。さらに、床を右クリックし、Lock Edits を選択して不用意な移動を防ぐ。
- 2D ビューに戻り、床の左側、後方、右側に Polyline で壁を描く。Drawing パネルで壁の幅を 0.2、高さを 1 メートルに設定する。3D ビューで Texture ボタンから適切なテクスチャを選んで壁に適用し、右クリックで Lock Edits を選択して不用意な移動を防ぐ。

本節では、前節で学んだ基本的な手法をいくつか使用して、アニメーションを改良し、おおよその実寸表示を実現した（図 8.11）。

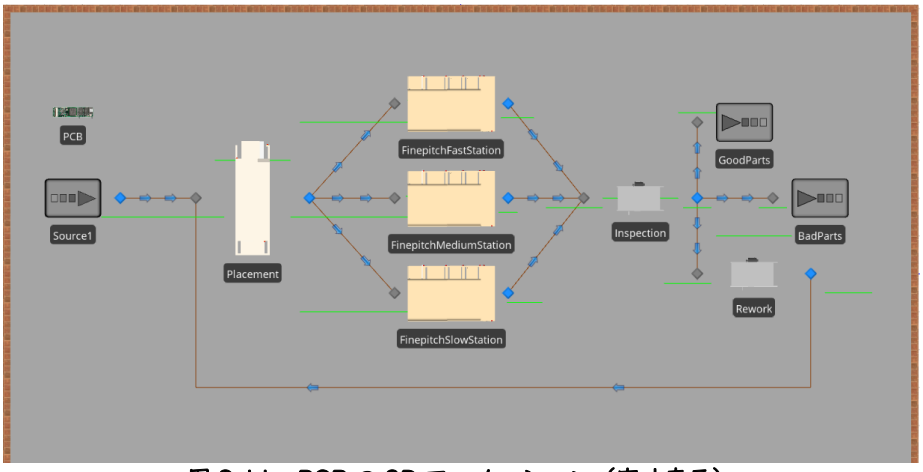


図 8.11　PCB の 2D アニメーション（実寸表示）

このように後から実寸表示に修正できることは便利だが、一般的には最初から少なくともおおよその実寸でモデルを構築するほうが簡単である（理想的には、縮尺図面をもとに作業を始めるとよい）。そして、比較的シンプルなエンジニアリングツールであっても、わずかな労力で図 8.12 のような、より現実的な表示に変換することができる。

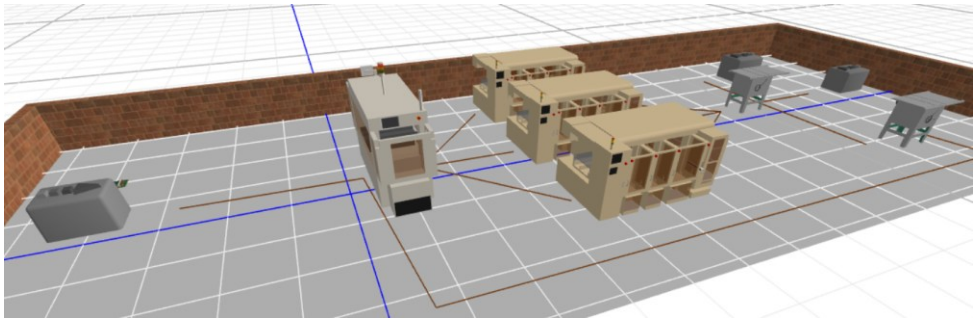


図 8.12　PCB の 3D アニメーション（実寸表示）

アニメーションの改良（作業台のアップグレードなど）をさらに続けることも可能だが、残りは課題としておく。ただし、後ほどいくつかの拡張機能を追加する際に再度取り上げる。

8.2　エンティティ移動

これまでのモデルの多くで、エンティティがある位置から別の位置へ移動する動きを実装してきた。モデルによっては実際の動きを表現していたが、瞬時の移動として扱ったものもある。Simio ではエンティティ（または「流れる」エンティティ）の移動方法が複数用意されている。表 8.1 にその主な移動手段をまとめている。もっとも簡単で直感的な方法は、Standard ライブラリに組み込まれたネットワーク機能を使用することだ。各移動手段の詳細は以下の項で説明するが、まずはネットワークに依存しない移動方法について簡単に紹介する。

表 8.1　Simio のエンティティ移動

フリースペース	Assign 移動、Transfer ステップ、Travel ステップ
Standard ライブラリ	Connector、TimePath、Path、Conveyor、Vehicle、Worker
Flow ライブラリ	Container エンティティ、Flow コネクタ、Pipe
Extras ライブラリ	Crane、Robot、Lift トラック、Elevator

8.2.1　フリースペースを通るエンティティ移動

通常は Standard ライブラリのリンク移動機能を使用することが多いものの、より柔軟な移動方法が必要になる場合がある。**フリースペース**とは、ネットワーク上に存在しないモデル領域（オブジェクト間の「空間」など）を表す用語である。エンティティが「物理的な位置」（図 8.13 のステーション、ノード、またはリンク）上にない場合、そのエンティティはフリースペースに存在することになる。エンティティはフリースペース内に存在し、そこを移動でき、その移動をアニメーション表示できる。表 8.1 に示すフリースペース移動の 3 つの選択肢は、主に物理的な場所とネットワークの関与度によって区別される。

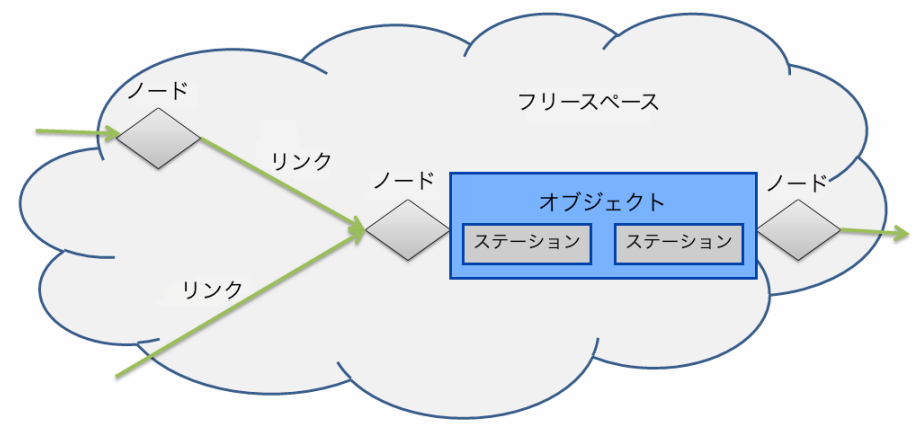


図 8.13　エンティティはノード、リンク、ステーション、フリースペースに存在可能

ネットワーク外で**瞬時に**移動させたい場合は、プロセスで **Transfer ステップ**を使用できる。Transfer ステップは、ステーション、リンク、ノード、またはフリースペース**から**、別のステーション、リンク、ノード、またはフリースペース**まで**の移動（多くの場合、瞬時の移動）を開始する。一般的な使用例として、Create ステップとの組み合わせがある。エンティティは生成時にフリースペースに出現し、特に指定がなければデフォルトの速度でフリースペースを移動し始める。そのため、通常は Create ステップの直後に、目的の位置への移動を指示する Transfer ステップを配置する。また、物理的な位置を持たないフリースペースへの瞬時移動が必要な場合もある。これは特に**エージェントベースモデリング**（ABM；Agent-Based Modeling）でよく見られる手法である。エンティティの位置、方向、速度、加速度は、以下の移動状態変数で設定できる：

・位置パラメータ：Movement.X、Movement.Y、Movement.Z
・方向パラメータ：Movement.Pitch、Movement.Heading、Movement.Roll

・運動パラメータ：Movement.Rate、Movement.Acceleration、Movement.AccelerationDuration

これらのパラメータのほとんどはフリースペースでのみ機能し、リンク上では使用できないことに注意する（たとえば、リンク上での加速度の設定にはこれらを使用しない）。ただし、より簡便な方法として、Simio は **Travel ステップ**を提供している。これは上述の機能をより使いやすい形で実装したものだ。エンティティはフリースペースに留まり、ネットワーク構成要素の**上に**配置されることはないが、Travel ステップを使用してネットワークに**沿って**移動するよう指定できる。これにより、ネットワークの制約を受けることなく、あたかもネットワーク上にいるかのように位置を制御できる。たとえば、ネットワーク**上**では加速機能はサポートされていないが、フリースペースでネットワークに**沿って**移動する場合は加速が可能だ。さらに、絶対座標や相対座標、指定したオブジェクトへフリースペースを通じて直接移動させることもできる。Travel ステップには、Direct To Destination と Follow Network Path という２つの **Steering Behavior** プロパティがある。後者は廊下を歩く人々の自然な動きを再現するのに適しており、経路の幅全体に分散したり、互いを避けるような動きを作り出せる。

設定を容易にするため、エンティティ（ModelEntity、Vehicle、Worker、ContainerEntity など）の Initial Travel Mode プロパティには、次の３つの選択肢がある。

・Network Only は、エンティティがネットワーク上のみを移動でき、フリースペースでの移動を行わない設定である。
・Free Space Only は、エンティティがフリースペースを移動でき、ネットワーク上での移動を行わない設定である。名前が示すとおり、ネットワークの制約から解放され、フリースペースを介してオブジェクト間を直接移動できる。これは多数の位置が存在する場合（全国規模の流通ネットワークなど）や、エンティティが特定の経路に従う必要がない場合に特に有用である。
・Network If Possible は、上記２つを組み合わせたデフォルトの設定だ。エンティティはまずネットワーク上での目的地への移動を試み、それが不可能な場合は自動的にフリースペースを使用する。

Network If Possible はデフォルトの設定であり、たとえばリンクで接続されていない Source と Sink を配置した場合、エンティティはフリースペースを使って両者間を移動できる。この設定は非常に便利だが、慎重に使用する必要がある。エンティティはほぼ確実に目的地に到達できるが、これは必ずしも望ましくない場合がある。特に、エンティティをネットワークに制限したい場合や、必要なリンクの設定を忘れた場合には注意が必要だ。このような場合、モデルは動作するが、その挙動は誤っており、このエラーの発見は困難になる可能性がある。

5.4 節で学んだエンティティの目的地選択の概念は、フリースペースの移動にも適用される。デフォルトでは、エンティティは常にもっとも近い目的地に移動するが、Entity Destination Type にはさまざまなオプション（By Sequence、

Continue、Select From List、Specific など）がある。

具体例については、SimBit の FreeSpaceMovement を参照してほしい。この SimBit には、フリースペース移動の概念を説明する 3 つのモデルが含まれている。また、SimBit の TravelWithSteeringBehavior ではステアリング動作についても解説している。

8.2.2　コネクタ、タイムパスおよびパスの利用

これまでの説明では、Standard ライブラリの機能を用いて、ネットワークを介してエンティティを移動させてきた。最初に考慮すべき点は、エンティティが自律的に動くか（例：人間や自律型オブジェクト）、それとも移動に補助（例：人、車両、コンベア、その他の装置）が必要かということである。表 8.1 の Standard ライブラリのオプションについては次項で詳しく説明するが、ここではまず、リンクの基本概念と共通の特徴について説明しよう。

8.2.2.1　General Link プロパティ

Simio において**リンク**は、エンティティが移動するノード間の経路を表す固定オブジェクトとして定義される。この定義には以下の重要な概念が含まれる：

・リンクはオブジェクトであり、実行中は移動しない固定的な要素である。
・リンクはノードを接続し、単独または単一のノードでは存在できない。
・エンティティのみがリンク上を移動できる。Vehicles と Workers もエンティティから派生した特別な機能を持つエンティティである。

また、すべてのリンクには以下の共通の特徴がある：

・リンクには必ず開始ノードと終了ノードが必要である。
・リンクは 1 つ以上のネットワークに属し、ネットワークはリンクの集合である。すべてのリンクは「Global」という特別なネットワークに属している。
・リンクオブジェクトは等間隔の位置（セル）で定義される長さを持つ。
・リンクの制御と位置はエンティティの先頭によって決定される。エンティティの先頭がリンクに進入すると、そのリンクに組み込まれたロジックによって移動が制御される。
・Simio は各エンティティの先頭と末尾を追跡し、衝突と追い越しを検知する。これにより、プロセスにおけるリンクの動作を柔軟にカスタマイズできる。

Standard ライブラリには、コネクタ（Connector）、パス（Path）、タイムパス（TimePath）、コンベヤ（Conveyor）というリンクオブジェクトがある。以下では、最初の 3 つについて説明する。

8.2.2.2　コネクタ(Connector)

コネクタはもっとも単純なリンクである。2 つのオブジェクトを時間遅延なし（ゼロ時間）で直接接続する。移動が瞬時に行われるため、エンティティ間の干渉は発生しない。実質的に、一度に 1 つのエンティティのみがリンクを通過でき、エンティティは開始ノードから終了ノード（またはその関連オブジェクト）まで

直接移動するように見える。

コネクタには Selection Weight プロパティがあり、ノードに複数の出力リンクが接続されている場合の選択に使用される。この機能については 5.2 節で説明し、Model 5-1 で実装した。

8.2.2.3　タイムパス（TimePath）

タイムパスは、前述の Selection Weight を含むすべての基本的なリンク機能を備えている。コネクタと異なり、Travel Time プロパティでエンティティの移動時間を指定できる。また、Traveler Capacity で同時に移動できるエンティティ数を制限でき、Entry Ranking Rule で待機中のエンティティの中から次の移動許可を与えるものを選択できる。

タイムパスには進行方向を決定する **Type** があり、`Unidirectional` または `Bidirectional` を選択できる。Unidirectional は開始ノードから終了ノードへの一方向の移動のみを許可する。Bidirectional は双方向の移動を許可するが、**同時に一方向の移動のみ可能**である（つまり、異なる方向への同時移動は不可）。完全な双方向経路が必要な場合は、個別の単方向経路として実装する必要がある。双方向パスでは、パス状態変数の DesiredDirection に以下のいずれかを設定して**方向**を制御できる：

・`Enum.TrafficDirection.Forward`
・`Enum.TrafficDirection.Reverse`
・`Enum.TrafficDirection.Either`
・`Enum.TrafficDirection.None`

タイムパスは `On Entering` と `Before Exiting` での State Assignments をサポートしている。さらに、重要な場面でロジックを実行できる Add-on Process Triggers も備えており、これらによってリンクの動作と関連オブジェクトとの相互作用をカスタマイズできる。

8.2.2.4　パス（Path）

Standard ライブラリの**パス**オブジェクトには、TravelTime プロパティ以外は、タイムパスと同じ機能がある。代わりに、パスは各エンティティの DesiredSpeed とパスの長さから移動時間を計算する（例：毎分 2 メートルの Desired Speed を持つエンティティが 6 メートルのパスを移動するには、**少なくとも** 3 分必要となる）。移動時間には他のパラメータも影響を与える。Allow Passing パラメータを `False` に設定すると、速いエンティティが遅いエンティティに追いついた場合、速いエンティティは自動的に減速して後続する（詳細は SimBit の `Vehicles PassingOnRoadway` を参照）。なお、Allow Passing を許可すると、エンティティ同士が同じ空間を占めることができる。複数のエンティティが同時にパスに進入した場合、デフォルト設定の True では、すべてのエンティティが同じ空間を共有し、一つのエンティティが移動しているように見える。また、パスには Speed Limit を設定でき、エンティティの Desired Speed を制限することができる。

パスには Drawn To Scale プロパティがある。これを `False` に設定すると、表

示上の長さとは異なる論理的な長さを指定できる。この機能は、正確な距離が必要な場合や、アニメーション表示のために長さを圧縮したい場合（例：部署間の移動をモデル化する際、実際の距離が長い場合）に有用である。

8.2.3　コンベアの利用

8.2.3.1　コンベアの基本概念

コンベア（Conveyor）は、エンティティを一定の経路に沿って搬送する固定式の装置である。コンベアは（パスと同様に）リンクであるが、他の Standard ライブラリのリンクと異なり装置としての性質を持つため、別に説明する。コンベアにはさまざまな種類がある。箱を扱う動力式または重力式の**ローラコンベア**、食品や石炭を運搬する**ベルトコンベア**や**バケットコンベア**、器具や車体を搬送する**オーバーヘッドパワーコンベア**や**フリーコンベア**などである。重工業では長年コンベアが活用されてきたが、近年は軽工業やその他の産業でも革新的な応用例が増えている。食品・消費財産業では、瓶や缶の充填、包装作業に高性能な高速コンベアを使用している。通販やドラッグストアでは、処方薬の調剤工程の自動化に軽負荷コンベアを採用し、多くの大規模食料品店でもレジにベルトコンベアを導入している。

Standard ライブラリの Conveyor オブジェクトは、前述の Path オブジェクトと多くの機能を共有しているが、重要な独自機能も備えている。現行の Simio コンベアは反転機能を持たないため、Type プロパティはない。つまり、すべてのコンベアは開始ノードから終了ノードまで一方向にのみ動作する。また、Speed Limit プロパティの代わりに、稼働時の初期速度を指定する Desired Speed プロパティがある。さらに、Allow Passing プロパティはない。すべてのエンティティがコンベアと同じ速度で移動するため、追い越しが発生しないためである。

エンティティはコンベアの速度で移動するか、停止しているかのいずれかの状態となる。エンティティの速度がコンベアの速度と完全に一致している場合（移動中でも停止中でも）、そのエンティティは**連動**状態にある。一方、コンベアが動いているにもかかわらずエンティティが停止している場合（コンベアがエンティティの下でスリップしているような状況）、エンティティは**非連動**状態となる。この仕組みについて、以下で詳しく説明する。

Entity Alignment は、エンティティがコンベア上で取り得る有効な位置を決定するプロパティである。Entity Alignment が `Any Location` の場合、エンティティはリンク上のどの位置でも連動可能である。`Cell Location` の場合は、コンベアの Number of Cells（セル数）を指定する必要があり、エンティティは各セルの境界に先頭が揃った位置でのみ連動できる。このプロパティは、バケットコンベア上のバケットやパワー&フリーシステム上のキャリアなど、コンベア上に離散的な構成要素がある場合に使用される。

Accumulating プロパティは、エンティティのコンベアからの非連動を許可するかどうかを決定する。Accumulating が `False` の場合、コンベア上のすべてのエンティティは常に連動状態を維持し、一つのエンティティが停止する必要があ

る場合（終端到達時など）、コンベアとその上のすべてのエンティティも停止する。True の場合、コンベア終端で停止したエンティティは非連動状態となるが、コンベアと他のエンティティは動作を継続する。後続のエンティティが停止中のエンティティに追いつく（「衝突」する）と、それも非連動となって停止する。この現象を**蓄積**と呼ぶ。

停止していたエンティティがコンベアを離れると、逆のプロセスが始まる。移動の障害が解消されると、2 番目のエンティティは再連動を試みる。Entity Alignment が Any Location の場合、この再連動は即座に成功する。Cell Location の場合、エンティティは先頭がセル境界に揃うまで待機し、その後に連動して移動を再開する。

コンベアとパスの最後の相違点として、コンベアは装置としての性質を持つため、（5.3.4 項で説明した Server と同様の）Reliability プロパティと、信頼性に関連する Add-on Process Triggers を備えている。

8.2.4　Model 8-2：コンベアがある PCB 組立

PCB モデルの改良を続け、これまでに学んだ知見を応用してみよう。3 つの Finepitch 部品配置はすべてコンベアで供給・配送され、インバウンド側もアウトバウンド側もバッファ領域を持たない。つまり、部品はコンベアから直接供給され、移動できない場合は**ブロック**される。まず、3 つの Finepitch ステーションを選択し、Output Buffer の Buffer Capacity を 0 に設定する（Input Buffer はすでに 0）。6 個のコネクタすべてを選択するには、Ctrl キーを押しながら各コネクタをクリックする。これにより、グループとしてプロパティを一括変更できる。

- グループ内のいずれかのオブジェクト上で右クリックし、Convert to Type→Conveyor を選択して、6 個のコネクタをコンベアに変換する。
- Desired Speed を 1 Meters per Minute に設定する。
- コンベアらしい外観にするため、Simio 内蔵のコンベアパスデコレータを使用する。ただし、デフォルトサイズが大きすぎるため、より現実的なコンベアパスを表現する線を作成する。
  - General Category→Physical Characteristics→Size→Width で、0.2 メートルに設定する。
  - 選択状態を保ったまま、Link Tools リボンを開く。Path Decorators セレクタのボタンをクリックし、Roller を選択する。他のパス装飾も試して、効果を確認するとよい。
- コンベヤの外観を改善したら、エンドポイントの高さを調整して、パーツが機械の適切な位置に接続されるようにする。3D ウィンドウ内で Shift キーを押しながらドラッグし、各ノードの高さを調整する。これは機械の入力・出力ノードの両方で必要である。曲がり点を追加する場合は、既存のコンベアを削除せずに、Link Tools リボンの Add Vertex ボタンを使用して色々な構成を試せる。

検査工程では、テーブルに 3 つの部品までしか蓄積できない。それ以外の部品は入ってくるコンベア上に待機させる必要がある。そのため、Inspection の

InputBuffer を 3 に設定する。図 8.14 のように、検査の入力バッファ待ち行列（InputBuffer.Contents）を調整し、検査員のテーブル上に 3 つの部品を表示できる適切な短い長さの線にする。

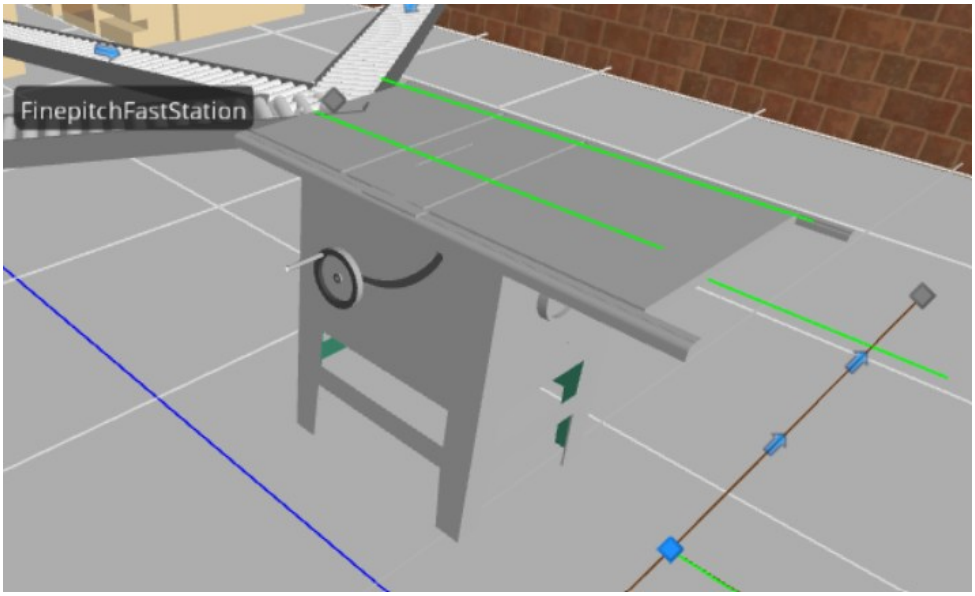


図 8.14　ノードと待ち行列の 3D ビュー配置図

モデルを実行し、詳細に挙動をチェックする。蓄積型コンベアであるため、運搬の最後で部品が停止することに注意する。ただし、他の部品はコンベア上で移動を続ける。特にオフシフト時には、Inspector に向かうコンベア上にエンティティが蓄積し始める。また、FinepitchFastStation は生産率が高いため、より早く部品が滞留する傾向にある。しかし、これはブロックを避けたい機械である。そこで、コンベアに入る際のエンティティの優先順位を変更することで、より速い機械での加工を優先させることにする。

- 一番上のアウトバウンドコンベアを選択し、State Assignments 下の OnEntering プロパティに進む。エントリ右側の点をクリックすると、Repeating Property Editor が表示される。State Variable プロパティを ModelEntity.Priority とし、New Value プロパティを 3 に設定する。
- 中央のアウトバウンドコンベアで同じ手順を繰り返し、New Value プロパティを 2 に設定する。
- Inspection のエントリーノード（Input@Inspection）を選択し、Initial Capacity プロパティを 3(各コンベアから 1 つずつ)に設定する。これにより、3 つのエンティティ（各コンベアから 1 つ）がノードに入り、Inspection から選択される状態になる。エンティティはまだコンベアを離れていないが、先頭がノードに到達している点に注意する。
- Inspection を選択し、Ranking Rule を Largest Value First へ、Ranking Expression を Entity.Priority に変更する。これにより、待機時間の長いエ

ンティティより、優先度の高いエンティティが先に選択されるようになる。

実際には、検査員がオフシフトの際は優先順位に関係なく部品は処理されないため、この変更がモデルの挙動に与える影響は限定的である。ただし、一番上のコンベアを空の状態に保ち、もっとも速い機械のブロックを最小限に抑えて、できるだけ早く空き状態に戻すのに役立つ。

これまでは、その他のコンベア機能についてほとんど触れていない。さまざまなプロパティを組み合わせることで、多様なタイプのコンベアをモデル化できる。また、複雑な合流や分岐がある場合は、コンベアや周辺オブジェクトのアドオンプロセスを活用して、複雑なシステムをモデル化できる。さらに、コンベアは変更可能な Process ロジックで完全に制御できるため、独自の動作をする Conveyor オブジェクトを作成することも可能である。

8.2.5　Worker の利用

これまでの節では、エンティティの移動方法について説明してきた。エンティティの移動を補助する手段は多岐にわたる。具体的には、エンティティを目的地まで運ぶための治具、カート、作業員、バスなどの制約つきリソースが挙げられる。もっとも単純なケースである固定リソースから見ていく。リソースの現在位置からの移動時間やアニメーションが重要でない場合は、必要時にリソースを占有し、完了時に解放するだけでじゅうぶんである。このアプローチは、移動の許可が本質的な要件である場合（例：交通整理員による通行許可）に適している。

より一般的なケースとして、運搬者の現在位置や各種プロパティ、状態変数が重要となる場合がある。Simio の Standard ライブラリには、この目的のために Worker と Vehicle という２つのオブジェクトが用意されている。これらは Entity から派生したオブジェクトで、エンティティの全機能を備えつつ、さらに追加機能も持っている。ノードでの挙動に若干の違いはあるが、エンティティと同様にネットワーク上を移動できる。

8.2.5.1　Worker

Worker は他のエンティティを運搬する機能を持つエンティティである。Worker は動的で、実行中にランタイムオブジェクト（RunSpace）内に生成される。Initial Number in System の設定により、利用可能な Worker の数を指定できる。各 Worker のコピーは個別のオブジェクトとして個人を表現するため、Initial Capacity プロパティは 1 以上に設定できない。Worker は主に人間を表現することを目的としているため、Reliability などの機械特有のプロパティは持たないが、作業員の Work Schedules に従って動作する。Worker の主要なプロパティについては、図 8.15 を参照してほしい。

エンティティと同様に、Worker は指定されたネットワーク上またはフリースペースを経由して移動することができる。デフォルトでは、ネットワークをたどり、可能な場合はフリースペースを使用する。Routing Logic Category では、Worker に `Default Node (Home)` の値を設定し、Worker の開始位置と「Home」ロケーションへの帰還位置を指定する必要がある。

Properties: Worker1 (Worker)

Resource Logic	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Park While Busy	False
Travel Logic	
Initial Desired Speed	2.0
Initial Travel Mode	Network If Possible
Initial Network	Global
Network Turnaround Method	Exit & Re-enter
Free Space Steering Behavior	Direct To Destination
Routing Logic	
Initial Priority	1.0
Initial Node (Home)	
Idle Action	Park At Node
Off Shift Action	Park At Node
Transport Logic	
Financials	
Add-On Process Triggers	
Population	
Initial Number In System	1
Maximum Number In System Limit	2500
Advanced Options	
Log Resource Usage	False
Display Name	
Can Transfer In & Out Of Objects	False
Gantt Visibility	True
General	

図 8.15　Worker オブジェクトのデフォルトプロパティ

設定が必要なアクションは 2 つある。Idle Action は、Worker が作業や作業要求がないときの動作を指定する。OffShift Action は、Worker がオフシフト（容量がゼロ）になったときの動作を指定する。これらのアクションにはそれぞれ 2 つの選択肢がある。1 つ目は、現在のノード位置に留まるか、指定されたホームノード位置に戻るかの選択である。2 つ目は、Park（ネットワークから離れて駐車場に移動）するか、ネットワーク上に留まるかの選択である。ネットワーク上に留まる場合、他の通行の妨げとなる可能性がある。

Transport Logic カテゴリには、Worker が同時に運搬できるエンティティ数を指定する Ride Capacity がある。また、Load Time と Unload Time で、エンティティごとの積み降ろしに必要な時間を設定できる。

エンティティと同様に、Worker にも Population カテゴリに Maximum Number in System Limit がある（検証ツールとして機能する）。このプロパティで予想される Worker の最大数を設定でき、この制限を超えて Worker が生成された場合、Simio はエラーを表示する。また、Advanced Options には Can Transfer In &

Out Of Objects プロパティがあり、これによって Worker が外部ノードを介してオブジェクト（例：サーバ）に入ることができるかどうかを制御できる。Worker は通常、エンティティやその他のオブジェクトに対してサービスを提供する役割を担うため、このプロパティはデフォルトで False に設定されている。これは、Worker がサーバ内でエンティティが実行するようなプロセスロジックを実行する必要がないためである。

また、Worker には 2 つの動作モードがある。1 つは**可動リソース**として機能するモードで、この場合 Worker は占有・解放され、モデル内の各位置間を移動する。もう 1 つは**トランスポータ**としての動作で、エンティティの積み降ろしや運搬を行うことができる。

8.2.5.2　リソースとしての Worker の利用

Worker は他のリソースオブジェクトと同様に、占有・解放が可能である。Server、Combiner、Separator オブジェクトでは、Secondary Resources プロパティで直接設定できる。また、すべての Standard ライブラリオブジェクトにおいて、アドオンプロセスの Seize ステップと Release ステップを使用してリソースの占有・解放を行うことができる。どちらの方法でも、図 8.16 に示す Repeating Property Editor が表示される。

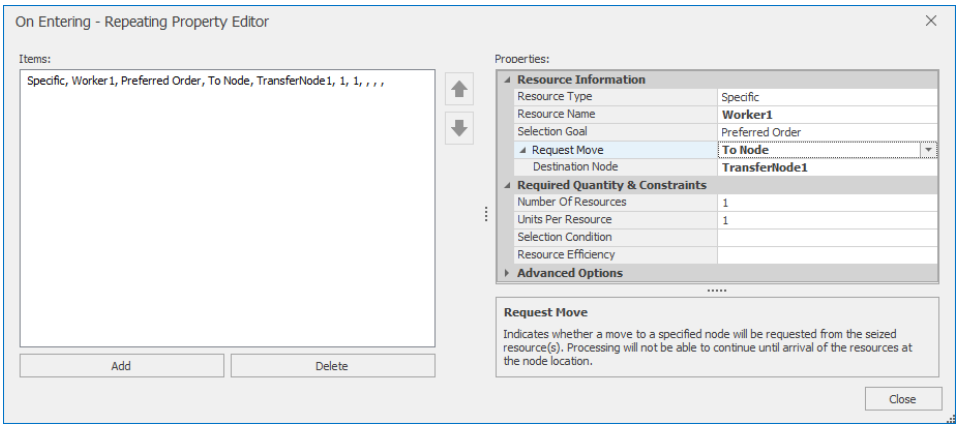


図 8.16　移動可能オブジェクトの占有に関するプロパティ

もっとも簡単な利用法は、必要なリソースが既知の場合に Resource Type を Specific に設定する。その後、Resource Name プロパティに必要なオブジェクトの名前(例:Worker1)を入力する。ただし、トランスポータ(Worker や Vehicle)の場合、Specific リソースを指定しても同じ名前の複数のユニットが存在する可能性があることに注意する。リソースのセットから順番に選択したい場合は、Resource Type を From List に設定し、選択したいリソースリストを指定する。

複数のリソースが選択可能な場合、Advanced Options カテゴリで選択ルール（Selection Goal と Selection Condition）を設定できる。同カテゴリでは、必要数やその他のオプションも指定可能である。

リソースを特定の位置に移動させる必要がある場合は、Request Move プロパティを To Node に設定し、Destination Node プロパティで目的地を指定する（図

8.16)。任意のノードを目的地として使用できるため、固定オブジェクトの隣にノードを配置して可動リソースの目的地として設定することが効率的である。

Request Move は、**可動リソース**（Worker や Vehicle などのエンティティから派生したオブジェクト、または Entity）でのみ機能する。Request Move 設定時、リソースが占有され指定ノードに移動するまで、トークンは Seize ステップに留まる。リソース解放時は、自動的に次のタスクに再割り当てされるか、Idle または OffShift 状態となる。

8.2.5.3　トランスポータとしての Worker の利用

Worker は、シンプルな Vehicle として機能する。Worker には、複数のエンティティを異なる位置間で運搬できる基本的な輸送機能が備わっている。Worker の設定方法は、Vehicle やその他の Transporter と同じである：

- TransferNode を選択する（Standard ライブラリオブジェクトのほとんどには、出力側にトランスファーノードが設置されている）。
- Transport Logic カテゴリで、Ride On Transporter プロパティを Always に設定する。これにより追加プロパティが表示される。使用するトランスポータを指定するまで、モデルはエラー状態となり実行できない。
- Transporter Type プロパティは、前述のリソースの Object Type プロパティと同様に機能する。特定のトランスポータを名前で指定するか、Selection Goal と Selection Condition を使用してリストから選択できる。
- Reservation Method では、トランスポータの選択方法と予約方法を指定する。以下の 3 つの選択肢がある：
  - Reserve Closest：現在もっとも近いトランスポータを選択する。同距離の場合は Selection Goal で決定する。選択されたトランスポータは Reserve（予約済み）状態となる。**予約**は双方向の約束となり、トランスポータは指定場所への移動を、エンティティは予約したトランスポータの待機を約束する。
  - Reserve Best：Selection Goal を使用して最適なトランスポータを選択する。同条件の場合はもっとも近いものが選ばれる。選択されたトランスポータは Reserve 状態となる。
  - First Available at Location：予約は行わず、適切なトランスポータが到着するまで待機する。トランスポータが定期運行スケジュールに従うか、カスタムロジックで制御されていない場合、トランスポータが到着しない可能性があるため注意が必要である。

8.2.6　Vehicle の利用

Vehicle は Worker とほぼ同様の利用方法である。Vehicle は占有・解放が可能で、位置間を移動できる。また、トランスポータとしてエンティティの積み降ろしや運搬を行うことができる。多くの機能が Worker と共通しているが、主な違いは、人ではなく装置をモデル化することを目的としている点である。さらに、いくつかの追加機能がある。

Vehicle には Routing Type があり、これによってルートの挙動を指定できる。

On Demand に設定すると、Vehicle は要求に応じて知的に対応し、乗車と降車を適切に計画する。Fixed Route に設定すると、Vehicle は定められたノード訪問順序（7.1.3 項）に従う。都市交通に例えると、On Demand はタクシーのような運行方式、Fixed Route はバスや地下鉄のような定期運行方式となる。

Vehicle には Reliability Logic（信頼性ロジック）が備わっている（装置は故障する可能性があるという前提に基づく）。故障の設定は、Calendar Time Based（時間ベース）または Event Count Based（イベント数ベース）から選択できる。特に後者は柔軟性が高く、外部イベントへの反応だけでなく、移動距離、充電状態、その他のパフォーマンス基準に基づいた内部イベントをトリガーとすることができる。

これらの違いはあるが、Vehicle と Worker は基本的に似た動作と用途を持つ。

8.2.7　Model 8-3：病院の職員に関する ED の拡張

救急診療のモデルを発展させるため、これまでに学んだ内容を活用する。ED モデルの Model 7-5 を基に（Model 8-3 として）、Worker を使用してスタッフ配置を拡張する。ただし、ED の機能を向上させる前に、アニメーションの一部を修正する必要がある。Trauma（検査室）、Treatment（治療室）、Exam（診察室）のオブジェクトは、個々のベッドではなく、複数の患者を収容できる区画を表現している。各ベッドを個別にモデル化することも可能だが、現時点でそこまでの詳細さは不要である。その代わりに、複数のベッドを備えた部屋を表現するため、改良版のアニメーションシンボルを作成していく。

8.2.7.1　新しいシンボルの作成

Project 名は初期設定でファイル名（例：Model_08_03）と同一である。ナビゲーションウィンドウで Project 名を選択すると、プロジェクトのすべての構成要素を含むウィンドウが表示される。このウィンドウで構成要素の追加、削除、編集が可能である。アクティブなシンボルを確認するには、左側の Symbols アイコンをクリックする。ここでシンボルの編集や、未使用シンボルの削除ができる。今回は新しいシンボルを追加する：

- Create New Symbol をクリックする。すると、Facility ウィンドウに似た描画ウィンドウが開き、2D・3D ビューと使い慣れたリボンが表示される。
- 10 メートル×3 メートルの部屋を作成する。Polyline ツールを使用して、四方の壁を 1 本の線で 4 セグメントとして描く。必要に応じて、ドアを表現するために線に切れ目を入れてもよい。Drawing リボンで、Object Height を 1、Line Width を 0.05 に設定し、Texture を選択する（この例では Banco Nafin を使用）。
- 患者間の仕切りを設置する。Polyline で最初の仕切りを作成し、Drawing リボンで、Object Height を 1、Line Width を 0.04 に設定する。次に、Texture を選択する（Dark Green Marble を使用）。
- 最初の仕切りをコピー&ペーストし、1 メートル間隔で合計 9 つの仕切りを作成する。

・Properties ウィンドウで Name プロパティを HospitalSuite に設定する。
・プロパティパネルを折りたたむと、図 8.17 のようなシンボルが表示される。
・作業完了後、Navigation ウィンドウで Project ウィンドウまたは Model ウィンドウに戻る。シンボルは自動的に保存される。

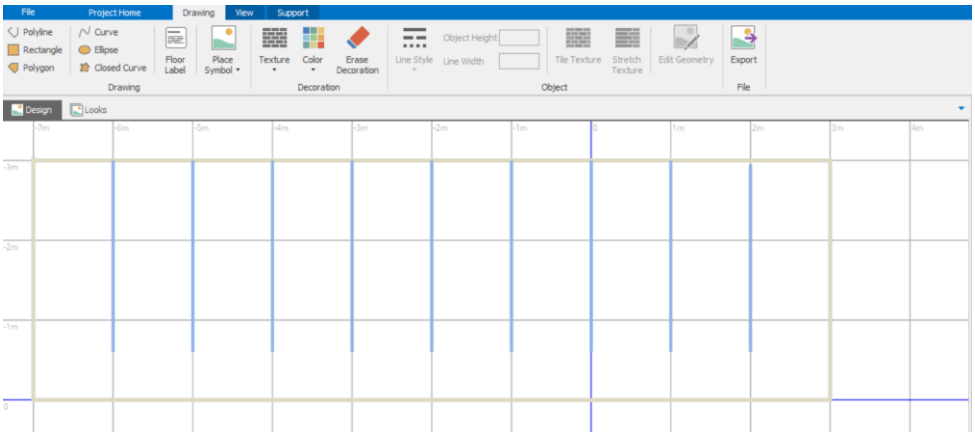


図 8.17　治療室のシンプルな描画

作成したシンボルはプロジェクトライブラリの一部となった。このシンボルを、ExamRooms、TreatmentRooms、TraumaRooms に適用する。これらは元のシンボルとはサイズが大きく異なるため、モデルの実行を一時停止して調整を行う必要がある（調整例は図 8.18 を参照）：

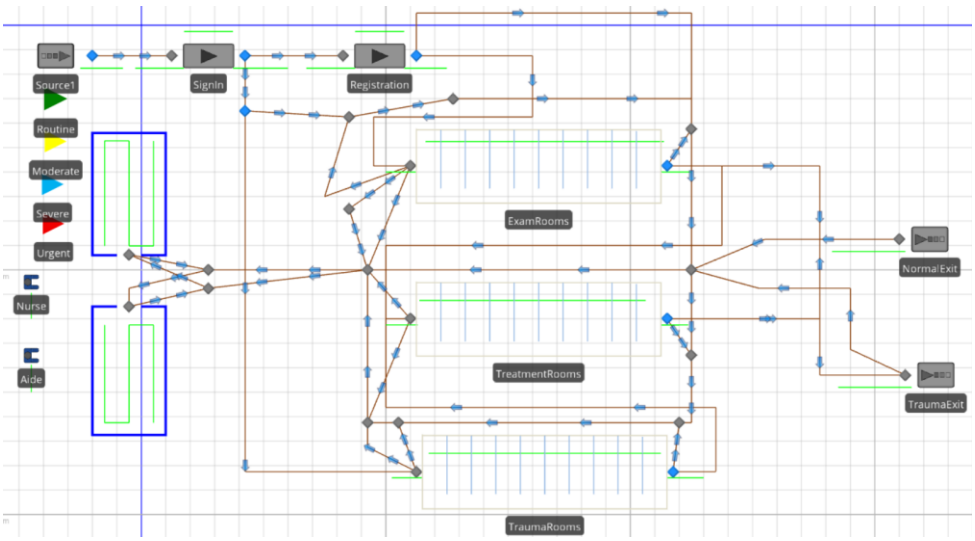


図 8.18　Model 8-3 における複数ベッドを備えた部屋の新シンボルによる表現

・レイアウトが自然に見えるようにオブジェクトの配置を調整する。
・ノードの位置をオブジェクトの外側に調整する。特にドアを作成した場合は調整が必要となる。
・オブジェクト間の既存のパスを削除し、適切な折れ線パスに置き換える。
・各部屋のアニメーション待ち行列の位置を調整する。InputBuffer.Contents

を入力ノードの近くに、OutputBuffer.Contents を出力ノードの近くに配置する。また、Processing.Contents を部屋の長さに合わせて調整する。これにより、各部屋の「占有者」（患者）を効果的に表示できる。

8.2.7.2　モデルの更新

ED のレイアウトを更新したので、次にスタッフ配置を追加する。スタッフは看護師（Nurse）と看護助手（Aide）の 2 種類をモデル化する。表 8.2 に示すように、診察室（Exam）、治療室（Treatment）、および検査室（Trauma）では、すべての患者に介助者をつける必要がある。定期検診（Routine）の患者は Aide が空くまで待機する。中程度（Moderate）および重度（Severe）の患者は、可能な場合は Aide が介助し、Aide が不在の場合は最初に空いた Aide または Nurse が付き添う。緊急（Urgent）の患者は、可能な場合は Nurse が、それが難しい場合は Aide が介助する。出口までの付き添いは、すべての患者に対して可能な場合は Aide が、不在の場合は Nurse が行う。

表 8.2　患者の目的地と必要な介助者の種類

患者のタイプ	Exam（診察室）	Treatment（治療室）	Trauma（検査室）	Exit（出口）
Routine（定期健診）	Aide	−	−	Aide 選好
Moderate（中程度）	Aide 選好	Aide 選好	−	Aide 選好
Severe（重度）	Aide 選好	Aide 選好	−	Aide 選好
Urgent（緊急）	−	Nurse 選好	Nurse 選好	Aide 選好

それでは、Worker のための待機場所（ホーム）を作成することから始める。

- これまでと同様に、折れ線を用いて 3×5 メートルの Nurse ステーションを描く。壁の高さを 1 メートルに設定し、入口を表す BasicNode を追加して NurseStation と命名する。
- ナースステーション内部に独自の待機待ち行列を設置するため、まずノードの自動待機待ち行列機能をオフにする。具体的には、NurseStation を選択し、Appearance リボンの Parking Queue ボタンで機能を解除する。次に、Draw Queue ボタンで ParkingStation.Contents 待ち行列を選択し、部屋の中に待ち行列を描く。必要に応じて、複数の看護師を効率的に配置できるよう、待ち行列を部屋の周囲に沿って設計できる。
- 図 8.19 に示すようにナースステーションを作成する。
- 上記の 3 ステップを繰り返して、Aide 用の待機場所となる AideStation を作成する。

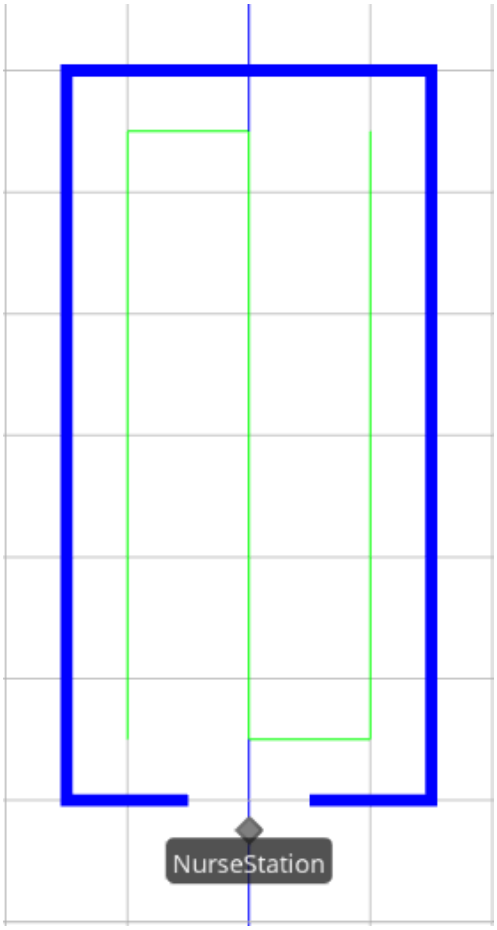


図 8.19　ノードと駐車待ち行列を備えた Nurse ステーション

次に、Worker オブジェクトを作成し、設定を行う。

・まず、Worker をモデルに配置して Nurse と命名する。Initial Desired Speed を 0.5（メートル/秒）に設定する。Routing Logic カテゴリで、Initial Node (Home)を NurseStation に設定し、Idle Action と OffShift Action を Park At Home に設定する。Population カテゴリでは、Initial Number in System を 4 に設定し、これにより 4 人の看護師（Nurse）がモデルの初期状態として配置される。

・次に、もう 1 つの Worker をモデルに配置して Aide と命名する。Initial Desired Speed を 0.5（メートル/秒）に設定する。Routing Logic カテゴリで、Initial Node (Home)を AideStation に設定し、Idle Action と Off Shift Action を Park At Home に設定する。Population カテゴリでは、Initial Number in System を 6 に設定し、これにより 6 人の看護助手（Aide）がモデルの初期状態として配置される。

8.2.7.3　リストの定義と利用

7.5 節で説明したように、Simio の**リスト**は、選択時に参照されるオブジェクト

の集合である。特に**目的地（ノード）やリソース（オブジェクト）の選択**で頻繁に使用する。リストからの選択では、Random や Smallest Value など、複数の選択メカニズムがサポートされているが、デフォルトは Preferred Order である。Preferred Order は、リストの先頭から利用可能な最初の項目を選択する方式である。

表 8.2 を確認すると、特定のタイプの Worker（例：Aide）が必要な場合と、Preferred Order リストから選択する場合（例：可能なら Aide を、不可能なら Nurse を使用）があることがわかる。すでに移動用のシーケンス表を使用しているため、この情報を新しいプロパティ（列）として追加するのは簡単だ。Worker はトランスポータの一種であるため、プロパティタイプはトランスポータまたはトランスポータリストのいずれかになる。各プロパティは特定の 1 つのデータ型である必要があるため、すべてをトランスポータリストとし、選択肢が 1 つしかない場合はリストに単一のメンバーを追加する。それでは、表 8.2 を確認しながらリストを作成していこう。

- リストは Definitions ウィンドウの Lists パネルで定義する。新しいリストを追加するには、List Data リボンの Create セクションで作成したいリストのタイプをクリックする。表 8.2 によると、介助者として選択する 3 つのリストが必要だ：Aide のみ、Aide 選好、および Nurse 選好である。Transporter ボタンを 3 回クリックして 3 つのリストを作成し、AideOnly、AidePreferred、Nurse Preferred と命名する。
- AideOnly リストを選択すると、画面下部にそのリスト内のトランスポーター一覧が表示される（初期状態では空）。最初のセルの右端をクリックし、トランスポータのプルダウンリストから Aide を選択する。AideOnly リストは 1 つの選択肢でじゅうぶんなため、これで定義完了である。
- AidePreferred リストも同様のプロセスで作成する。1 行目に Aide、2 行目に Nurse を設定する。このリスト順序は、Preferred Order ルールによって、利用可能なら Aide を、そうでない場合は利用可能な Nurse を選択することを意味する（どちらも利用できない場合は、最初に利用可能になったものを選択）。
- NursePreferred リストも同様に作成する。1 行目に Nurse、2 行目に Aide を設定する。

これでリストが作成できたので、Treatments テーブルにデータを追加できる。Data ウィンドウの Tables パネルで、Treatments タブを選択する。このテーブルは、患者が次の訪問場所を選択する際や、移動中、到着時のプロパティを決定する際に参照される。このテーブルには、診察室（Exam）と検査室（Trauma）だけでなく、すべての訪問場所が含まれている。ここで、移動手段のプロパティを追加する。

- Object Reference ボタンをクリックし、Transporter List タイプを選択して、プロパティ名を EscortType とする。これは各地点での付き添いに必要な Worker グループを指定するために使用する。すべての移動に付き添いが必要と

は限らないため、この列の Required Value プロパティを False に設定して null 値を許可する。

・新しい列のデータを表 8.2 のデータを用いて完成させる。完成すると、図 8.20 のようになる。

Patient Data　Treatments

	Sequence	Treatment Type	ServiceTime (Minutes)	Escort Type
1	SignIn	Routine	2	null
2	Registration	Routine	Random.Uniform(3,7)	null
3	ExamRooms	Routine	Random.Triangular(5,10,15)	AideOnly
4	NormalExit	Routine	0.0	AidePreferred
5	SignIn	Moderate	2	null
6	Registration	Moderate	Random.Uniform(3,7)	null
7	ExamRooms	Moderate	Random.Triangular(10,15,20)	AidePreferred
8	TreatmentRooms	Moderate	Random.Triangular(5,8,10)	AidePreferred
9	NormalExit	Moderate	0.0	AidePreferred
10	SignIn	Severe	1	null
11	Registration	Severe	2	null
12	ExamRooms	Severe	Random.Triangular(15,20,25)	AidePreferred
13	TreatmentRooms	Severe	Random.Triangular(15,20,25)	AidePreferred
14	NormalExit	Severe	0.0	AidePreferred
15	SignIn	Urgent	.5	null
16	TraumaRooms	Urgent	Random.Triangular(15,25,35)	NursePreferred
17	TreatmentRooms	Urgent	Random.Triangular(15,45,90)	NursePreferred
18	TraumaExit	Urgent	0.0	AidePreferred
*				

図 8.20　介助者を含む Treatments テーブル

8.2.7.4　ネットワークとロジックのアップデート

スタッフを利用するためのネットワークとロジックを更新して、このモデルの拡張を完了させる。ネットワークへの追加を開始する前に、1 つの修正が必要である。SignIn（受付）を離れる大多数の患者は、付き添いなしで直接 Registration（診療受付）に向かう。これらの患者には Ride On Transporter プロパティを Never に設定する必要がある。一方、Urgent の患者は TraumaRooms への付き添いが必要なため、Ride on Transport プロパティを Always に設定しなければならない。このジレンマを解決する簡単な方法として、SignIn の隣に新しい TransferNode を設置し、Urgent の患者をそこに誘導する。これにより、すべての患者は付き添いなしで SignIn を出発し、Urgent の患者のみが介助を待つための新しいノードに移動する。この変更を実装するには、SignIn から TraumaRooms までの既存のパスを削除し、SignIn の隣に新しいノード（WaitUrgentEscort）を追加して、SignIn から WaitUrgentEscort まで、さらに WaitUrgentEscort から TraumaRooms までのパスを設置する。

これでネットワークの拡張準備が整った。スタッフは既存のネットワーク上で患者に付き添うが、スタッフ自身も独自のパス上を移動する。すべての入力・出力ノード間の自由な移動を可能にするため、スタッフの円滑な移動用のパスを追加する必要がある。

このネットワークの構築には複数のアプローチが考えられる。もっとも網羅的なアプローチは、すべてのノードの組み合わせ間に双方向のパスを設置することである。これは完全な移動経路を実現できるが、非常に手間のかかる作業となる。もう一つの極端なアプローチは、パスを一方向（時計回りなど）の巡回路として設定することである。これはパス数を最小限に抑えられるが、非効率で非現実的な移動を強いることになる。そこで、中間的なアプローチを採用する。基本的には一方向の巡回路としながら、効率的な移動を実現するため、特定の目的地ノード間を結ぶ近道となるパスを追加的なノードとともに設置する。

このような中間的アプローチに基づき、図 8.21 のハイライト部分が示すノードとパスを追加した。

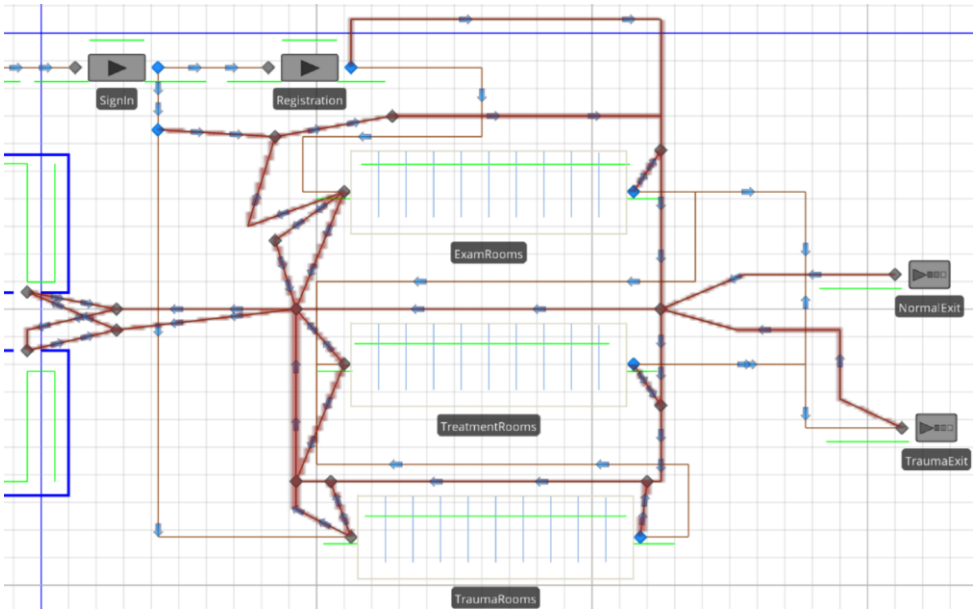


図 8.21　スタッフ移動用のネットワークのパス（ハイライト部分）

図を参考に、パスとノードをモデルに追加する。新しいパスを選択してハイライトし、右クリックしてネットワーク StaffNetwork に追加する。ネットワークは特定のエンティティのアクセスを制限するために使用できるが、この場合はその機能は不要だ。ただし、ネットワークが複雑になった場合、可視化とデバッグのために命名されたネットワークにパスを割りつけておくと便利である。この設定を行うには、パスをハイライトし、Visibility リボンで View Networks ボタンを選択する。

これまでに、スタッフの生成、選択用リストの作成、そしてテーブルへの参照データの追加を行った。また、介助用のパスも設定した。これらの基盤が整ったの

で、次はエンティティがいつ付き添いを待つかを設定する必要がある。この設定は**トランスファーノード**で行う。

まず、新しいノード WaitUrgentEscort から始める。このノードを選択し、Ride On Transporter を Always に設定する。Transporter Type は From List に設定する。患者と場所のリストをテーブルから取得するため、Transporter List に Treatments.EscortType の値を選択する。最後に、Reservation Method を Reserve Best に設定し、Selection Goal（最適な選択の定義方法）は Preferred Order のままとする。

次に、介助が必要な他の 3 つの出力ノード（Registration、ExamRooms、TraumaRooms）を選択する。これらのプロパティも同様に設定する。個別に設定することも、グループ選択して一括で行うこともできる。

まだモデルを保存していない場合は、ここで保存して実行してみる。すべての設定が正しければ、以前のバージョンと同様に患者の移動を確認できるだけでなく、今回の設定により特定の場面でスタッフが患者に付き添って移動する様子も観察できる。患者がスタッフの正面ではなく横に寄り添って移動するように、Worker の RideStation 待ち行列の位置を調整するとよい。

ここで「実際の病院と比べてどうだろう。患者が自分で治療をするのだろうか。このモデルでは、スタッフは患者の移動に付き添うだけで、治療中はいなくなってしまう」と疑問に思うかもしれない。この機能については 9 章のモデルで追加する。また、アニメーションと患者の移動にはさまざまな拡張方法があるため、本章で説明した以外の方法も自由に試してみるとよい。

8.3　まとめ

本章では、2D・3D アニメーションの価値と、検証およびコミュニケーションにおける活用について説明した。また、プロジェクトの優先業務や重要な分析技術を犠牲にしてまで、アニメーションに過度な労力を費やすことの危険性についても述べた。

モデルのアニメーション化に必要な基礎知識として、基本的なアニメーションと表示オプションの概要を解説した。さらに、エンティティなどの動的オブジェクトが、フリースペース、リンク、コンベア上を移動する方法や、Standard ライブラリの Worker や Vehicle といった他の動的オブジェクトと連携して移動する方法について詳しく説明した。

これらに加えて、その他の多様なモデリング状況とアプローチについては、10 章で取り上げる。

8.4　問題

最初の 4 つの問題は、有用な分析よりもアニメーションの理解に重点を置く。特に指定がない限り、問題を解くために適切なモデルパラメータを自身で設定すること。

1. Model 4-3 と同様の新しいモデルを構築する。このモデルではアニメーションが主な焦点となるため、本章で学んだ技法を活用して現実的な表現を目指す。顧客として、少なく

とも男性２名、女性２名、子供１名を含む複数の歩行アニメーションつきの人物を設定する。より現実感を高めるため、Trimble 3D Warehouse のシンボルを使用する。

2. 問題１の銀行に、ATM 待ち用のベンチシートを追加する。問題１の解答にベンチを追加し、向きを調整可能な待ち行列点とアニメーションインデックスを使用して、ベンチに座って待つ顧客を表現する。待ち行列で Keep In Place オプションを選択した場合の影響を確認する。また、ATM 利用時に顧客が ATM の方を向くよう待ち行列を修正し、適切なアニメーションインデックスを追加してアニメーションを改善する。

3. Model 7-2 を拡張し、各エンティティインスタンスにフロアラベルを追加する。このラベルでサービス水準（システム内時間、分単位）を表示し、患者の種類を背景色で区別する。さらに、１つ以上のフロアラベルをモデルに追加して、各患者タイプの以下３項目を表形式で表示する：処置中人数（Number In-Process）、システム離脱人数（Number Processed）、離脱患者の平均サービス水準（Average Level of Service）。

4. ３組の Source-Path-Sink を設置し、Source から Sink へのエンティティ移動に Vehicle を使用するモデルを構築する。既存ライブラリから乗り物のシンボルを選択し、各組に異なる Network Turnaround Methods（Vehicle プロパティ）を設定して、それぞれのアプローチを比較検討する。

5. Speedee Car Rentals は 24 時間営業で、２種類の顧客が来店する。25%がプレミアム顧客で、入店から５分以内の待ち時間が保証されており、超過時はレンタル料金から 15 ドルが割引かれる。残りはレギュラー顧客で、先着順となる。顧客は２分間隔で指数分布に従って到着し、プレミアム窓口（係員１名）またはレギュラー窓口（係員２名）まで、時速１マイルで距離４メートルを移動する。サービス時間は２～７分の一様分布に従う。係員の費用は１時間当たり 55 ドル（間接費込み）で、顧客１人当たり８ドルの収入が発生する（ただし割引分は差し引き）。リンクを使用せずにシステムをモデル化し、10 日間の反復実行を行う。各シナリオについて、係員タイプ別の稼働率と顧客タイプ別の待ち時間を報告しなさい。また、画面上と出力結果に、割引適用されたプレミアム顧客数、総割引額、システムの純利益（割引・費用考慮後）を表示しなさい。

・シナリオ a）各係員は、指定された種類の顧客のみを担当する。
・シナリオ b）上の条件を緩和し、係員の新しい就業規則（顧客担当の条件と方法）を設定する。各シナリオを評価指標に基づいて評価する。
・シナリオ c）システム改善のための別のソリューションをモデル化する。リンクの活用も検討する。

第 9 章　Simio の高度なモデリング

本章では、前章までに扱わなかった Simio の高度なモデリング概念について説明する。3 つのモデルを通じて、それぞれの発展的な概念を示していく。共通のテーマは、包括的な実験と OptQuest アドインを用いたシミュレーション最適化の活用である。

9.1 節では、第 8 章の救急診療モデルを再考し、意思決定の側面に焦点を当てる。意思決定機能として、**総費用**（total cost）という評価基準を設定し、これを用いて代替システム構成を評価する。また、OptQuest のシミュレーションベース最適化アドインの使用方法も解説する。9.2 節では、ピザのテイクアウトレストランモデルを取り上げ、Worker オブジェクトを使ってプールされたリソースシステムを実証する。手動で定義した実験と OptQuest を使用して、最適なリソースレベルを探索する。9.3 節では、ステーション間に有限容量バッファを持つ組立ラインモデルを開発し、異なるバッファ配置がラインスループットに与える影響を分析する方法を説明する。

本章で扱えない高度な機能や概念も数多くあるが、これをさらなる研究の出発点としてほしい。Simio の SimBits や、年次開催の Winter Simulation Conference（www.wintersim.org）、その他のシミュレーション関連会議の論文集で、より多くのシミュレーションや Simio 特有の概念を学ぶことができる。

9.1　Model 9-1：ED モデルの再考

8.2.7 節では、救急診療（ED）モデルに病院のスタッフを追加し、エンティティ移動とアニメーション（Model 8-3）を組み込んだ。しかし、実際の患者の治療については説明せず、モデルでの実験も実施しなかった。この節では、主に**意思決定**支援のために Model 8-3 をどのように変更するかを示す。特に、**リソース配置**と**スタッフ配置**の決定に役立つモデルの活用方法に焦点を当てる。

患者の治療と評価基準の要素を追加して Model 8-3 を拡張できるが、代わりに抽象的なバージョンを作成することにした。具体的には、Worker オブジェクトと移動経路を省き、エンティティ移動には Connector オブジェクトを使用する。この選択には 2 つの理由がある。第一に、抽象的なモデルはよりシンプルで説明しやすい。第二に、「最適な」シミュレーションモデルとは、プロジェクトの要件を満たすもっともシンプルなモデルだという考えを強調できる。Model 8-3 の目的は、Standard ライブラリの Worker オブジェクトを使った移動とアニメーションの実証だった。一方、Model 9-1 では、意思決定に活用できるモデルを示し、**シミュレーションベースの最適化**を実証することを目指す。

図 9.1 は Model 9-1 の Facility ビューを示している。このモデルには、SignIn、Registration、ExamRooms、TraumaRooms、TreatmentRooms の Server オブジェクトと、4 種類の患者 Enitity オブジェクト（Routine、Moderate、Severe、Urgent）がある。Nurse と Aide の Worker オブジェクトは、Doctor と Nurse の Resource オブジェクトに置き換えた。また、移動ネットワークと Path オブジェ

クトは、より単純な Connector オブジェクトのネットワークに置き換えた。物理的な患者の動きをモデル化する必要がないため、患者エンティティと各部屋のアニメーションシンボルも省略し、モデルを簡素化した。

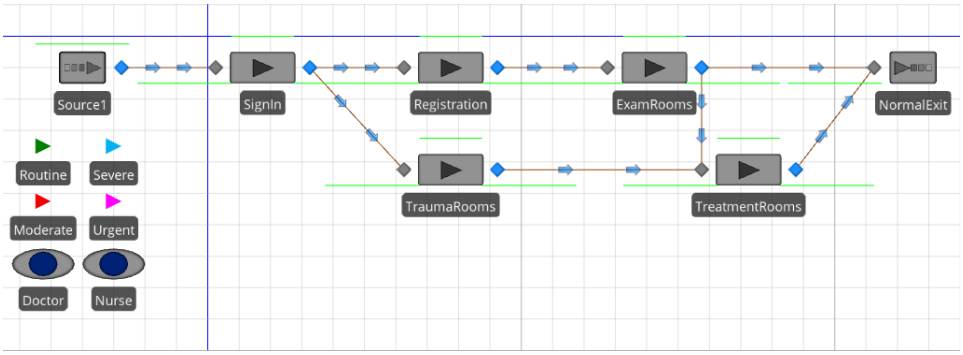


図 9.1　Model 9-1 の Facility ビュー

Model 9-1 は、4 つの基本型の患者が到着し、シーケンス表を使って指定されたルートでシステムの異なる経路に従うという、同じ基本的な問題構造を維持している。患者ルーティングの唯一の違いは、Urgent の患者が Model 8-3 では TraumaExit オブジェクトを通って終了したのに対し、このモデルでは NormalExit オブジェクトを通って終了することである。図 9.2 に Model 9-1 用に更新された PatientData データテーブルと Treatments シーケンステーブルを示す。PatientData テーブルには WaitingTime 列が追加され、Treatments テーブルからは EscortType 列が削除されている。

Patient Data　Treatments

	Patient Type	Priority	Patient Mix	Waiting Time
1	Routine	1	40	RoutingWaiting
2	Moderate	2	31	ModerateWaiting
3	Severe	3	24	SevereWaiting
4	Urgent	4	5	UrgentWaiting

Patient Data　Treatments

	Sequence	Treatment Type	ServiceTime (Minutes)
1	SignIn	Routine	2
2	Registration	Routine	Random.Uniform(3,7)
3	ExamRooms	Routine	Random.Triangular(5,10,15)
4	NormalExit	Routine	0.0
5	SignIn	Moderate	2
6	Registration	Moderate	Random.Uniform(3,7)
7	ExamRooms	Moderate	Random.Triangular(10,15,20)
8	TreatmentRooms	Moderate	Random.Triangular(5,8,10)
9	NormalExit	Moderate	0.0
10	SignIn	Severe	1
11	Registration	Severe	2
12	ExamRooms	Severe	Random.Triangular(15,20,25)
13	TreatmentRooms	Severe	Random.Triangular(15,20,25)
14	NormalExit	Severe	0.0
15	SignIn	Urgent	.5
16	TraumaRooms	Urgent	Random.Triangular(15,25,35)
17	TreatmentRooms	Urgent	Random.Triangular(15,45,90)
18	NormalExit	Urgent	0.0

図 9.2　Model 9-1 の PatientData と Treatments データテーブル

このモデルでは患者の待ち時間を追跡する。具体的には、患者が最初に到着してから医師や看護師の診察を受けるまでの待ち時間を測定する。この待ち時間は、患者の**サービスレベル**とリソースおよびスタッフ配置費用の最小化のバランスを取るために使用する。患者タイプごとの待ち時間は、PatientData テーブルの

WaitingTime エントリで、Tally 統計量を用いて追跡する。患者とスタッフの物理的な移動はモデル化していないため、EscortType 列は不要となった。Model 9-1 では、Model 7-5 と 8-3 の受付プロセスで使用した Computer Adjustment は使用せず、他の Server と同様に、Treatments.ServiceTime 列を Server Processing Time プロパティとして使用する。

また、ExamRooms、TreatmentRooms、TraumaRooms のリソースに加えて、Model 9-1 には患者の診察と処置に必要なリソース要件も組み込んでいる（Model 8-3 では、Nurse と Aide の Worker オブジェクトを患者の付き添いに使用したが、各部屋での患者の**処置**は明確にモデル化していなかった）。表 9.1 に診察・処置の要件を示す。

表 9.1　診察・処置に必要なリソース要件

部屋	診察・処置に必要なリソース
Exam Room	Doctor もしくは Nurse（Doctor 優先）
Treatment Room	Doctor および Nurse
Trauma Room	Doctor および Nurse もしくは Doctor（Nurse 優先）

（シーケンス表で指定されたように）患者が特定の部屋に到着すると、患者は Doctor および/または Nurse を処置に必要なリソースとして占有する。リソースが利用可能でない場合は、（実際の救急治療室と同様に）リソースが利用可能になるまで患者はそれぞれの部屋で待機する。システム構成要素のさまざまな組み合わせを評価できるように、（Resource オブジェクトとしてモデル化された）診察・処置リソースと（Server オブジェクトを用いてモデル化された）物理的な部屋のリソースは区別する。これは、医師の人数より多くの診察・処置室を持つ一般的な病院の状況を想定しているためだ。

救急治療室では、（Model 8-3 と同様に）PatientData テーブルで指定された優先順位に基づいて、Doctor と Nurse リソースが割り当てられる。たとえば、医師を待っている Routine 患者と Moderate 患者がいる状況で Urgent 患者が到着した場合、Urgent 患者は待ち行列の最前列に移動する。また、待機中の患者が Routine 患者のみの場合、Severe 患者や Moderate 患者が到着すると、それらの患者が待ち行列の前方に移動する。この優先順位づけは、Doctor と Nurse リソースの Ranking Rule プロパティで実現できる。図 9.3 に示すように、Doctor リソースでは、Ranking Rule プロパティを Largest Value First、Ranking Expression プロパティを PatientData.Priority に設定する。これにより、エンティティは PatientData.Priority フィールドの値に基づいて待ち行列で**順位づけ**される（なお、リソースが異なる場所から使用される場合でも、Simio には待機エンティティのための単一の内部待ち行列がある。この仕組みにより、本モデルで必要なグローバルな優先順位づけが簡単に実装できる）。Urgent 患者が Routine 患者から医師や看護師を**先取り**（preempt）する機能の実装については、読者への課題として残しておく。

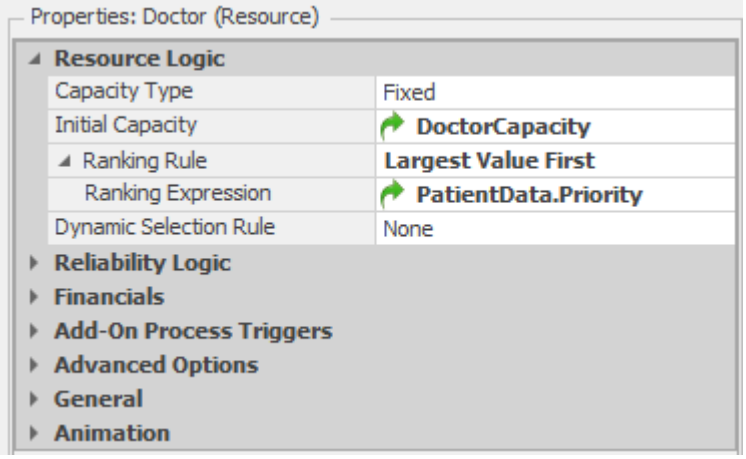


図 9.3　Doctor リソースのプロパティ設定

診察・処置リソースの患者への配置をモデル化するため、各部屋のリソースに関連づけられた**アドオンプロセス**を使用する（5.1.4 節を参照）。アドオンプロセスは、既存の Simio オブジェクト内のロジックを補完するメカニズムを提供する。現行モデルでは、患者が部屋（診察室、治療室、または検査室）に到着したとき、処置を施す医師や看護師を「呼び出し」、その「処置を施すリソース」が到着するまで患者が部屋で待機するためのロジックを追加する必要がある。これまでの Server オブジェクトのロジックは、処置を施すリソースではなく、部屋の容量の割当のみを扱っている。そのため、部屋が利用可能になったときに処置を施すリソースを呼び出すプロセスをアドオンする必要がある。

図 9.4 は、ExamRooms オブジェクトの 2 つのアドオンプロセスを示している。サーバ容量（ExamRooms サーバ）をエンティティに割り当てたとき、ExamRooms_Processing プロセスが実行され、処理が始まる。そして、ExamRooms_AfterProcessing のアドオンプロセスは、エンティティが処理を完了したとき実行され、サーバ容量を解放する。

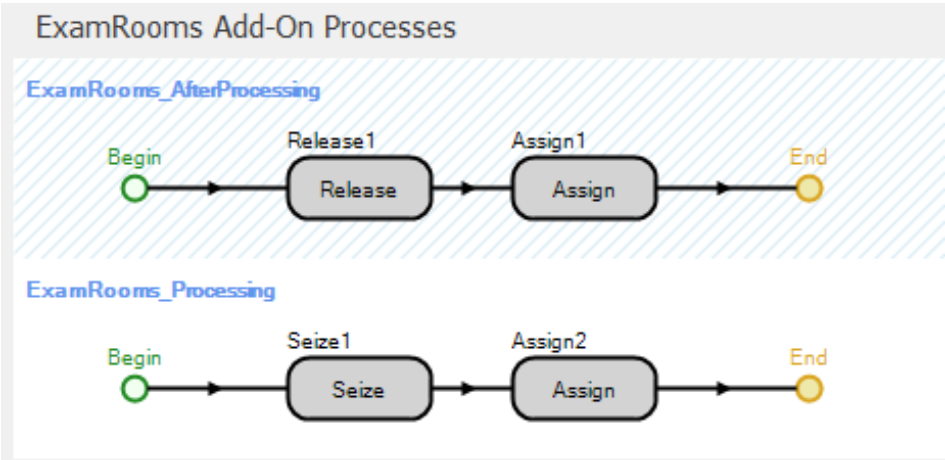


図 9.4　ExamRooms オブジェクトのアドオンプロセス

患者が ExamRooms サーバに入る際に行う 2 つの処理は、診察・治療リソースの占有と、リソース割当時の待ち時間の記録である。表 9.1 に示すように、診察室の患者は医師か看護師のいずれかを必要とし、両方が利用可能な場合は医師が優先される。8.2.7.3 で説明したように、エンティティが複数の選択肢から優先順位に基づいて選択する際は、Simio のリストと PreferredOrder オプションを使用する。この仕組みをここでも採用する。具体的には、Doctor と Nurse オブジェクトを含むリスト（DoctorNurse、最初の要素は Doctor オブジェクト）を定義し、アドオンプロセスの Seize ステップでこのリストからリソースを占有する。図 9.5 に DoctorNurse リストと Seize ステップの詳細を示す。医師を優先的に選択するため、Selection Goal プロパティには PreferredOrder を指定している。なお、Server オブジェクトの Secondary Resources プロパティでも同様の機能を実現できるが、ここではセカンダリリソース割当の内部動作を明確に説明するため、この方法を選択した。

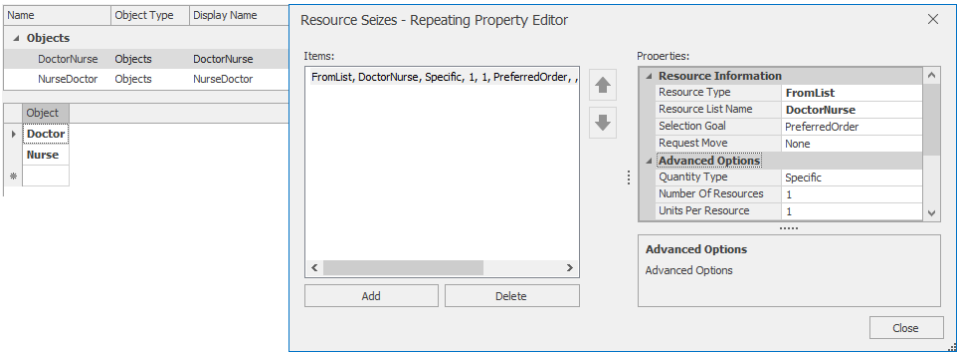


図 9.5　ExamRooms のアドオンプロセスにおけるリストと Seize の詳細

Assign ステップでは、Assign2 が患者の待ち時間の初期要素を記録し、Assign1 が患者の診察室退室時間を**記録する**（詳細は後述）。Assign2 ステップは、医師または看護師リソースがエンティティに割り当てられた時点で実行される。各患者について、医師や看護師の待ち時間をすべて追跡する必要があることに注意してほしい。患者は複数の場所で待機する可能性があるため、エンティティ状態変数（ModelEntity.WaitingTime）を使用して、各患者の総待ち時間を累積する。WaitingTime と tmp（後述）のエンティティ状態は、いずれも Real 型で、シミュレーション時間（TimeNow のデフォルト単位）における小数点つきの時間を記録する。図 9.6 に、両 Assign ステップのプロパティを示す。

Assign2 ステップでは、TimeNow - ModelEntity.TimeCreated の値を状態変数 ModelEntity.WaitingTime に割り当てている。このステップは、現在のシミュレーション時刻（モデル状態変数 TimeNow で記録）から患者の到着時刻（エンティティ生成時刻）を差し引き、その値をエンティティ状態変数 ModelEntity.WaitingTime に代入する。Routine の患者の場合、これが総待ち時間となる。その他の患者タイプについては、TreatmentRooms 到着後も待ち時間の追跡が続く。この追加待ち時間を追跡するため、Assign1 では現在のシミュレーション時刻をエンティティ状態変数 ModelEntity.tmp に記録する。この記録時刻は、エンティ

ティが治療室に到着して必要なリソースが割り当てられる際に使用される。ExamRooms_Afterprocessing プロセスの Release ステップでは、それまでのプロセスで占有していた医師または看護師リソースを解放する。

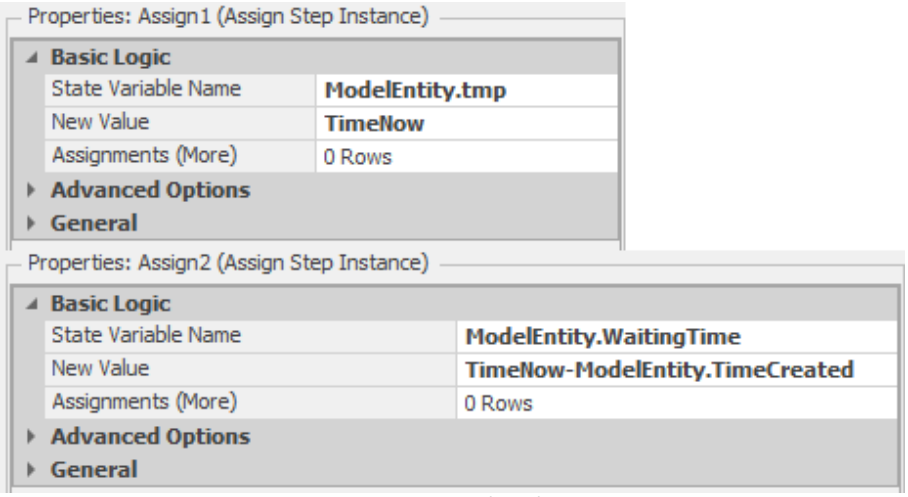


図 9.6　Assign ステップのプロパティ設定

図 9.7 は、TraumaRooms オブジェクトのアドオンプロセスを示している。このプロセスは基本的に ExamRooms オブジェクトと同じだが、TraumaRooms_Processing プロセスに 2 つの Seize ステップがある点が異なる。1 つ目のステップでは Doctor オブジェクトを、2 つ目のステップでは Nurse オブジェクトまたは Doctor オブジェクトを占有する。この場合、医師 2 名ではなく医師 1 名と看護師 1 名の組み合わせが必要となるため、NurseDoctor リストを使用する。このリストでは Nurse オブジェクトが Doctor オブジェクトより前に配置されており、ExamRooms オブジェクトで使用した DoctorNurse リスト（図 9.5）とは逆の順序となっている。Release ステップ（Release2）では、処理完了後に解放すべき**両方の**リソースを指定する。

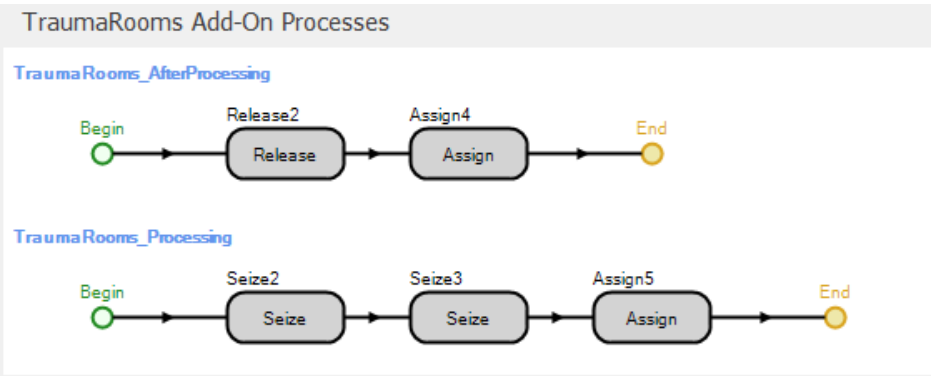


図 9.7　TraumaRooms オブジェクトのアドオンプロセス

図 9.8 は、TreatmentRooms オブジェクトのアドオンプロセスを示している。患者は検査室（Urgent 患者）または診察室（Moderate 患者および Severe 患者）

から治療室へ移動するため、エンティティ生成時刻を用いた待ち時間の追跡はできない。その代わりに、前に記録した時間（ModelEntity.tmp）を使用する。患者の総待ち時間に治療室での待ち時間を加算するため、Seize 直後に Assign を使用する（リソースが利用可能になるまで、エンティティ（より正確には**トークン**だが、この場合はエンティティを表す）が Seize ステップで待機することを覚えておいてほしい。この待ち行列時間を加算する）。

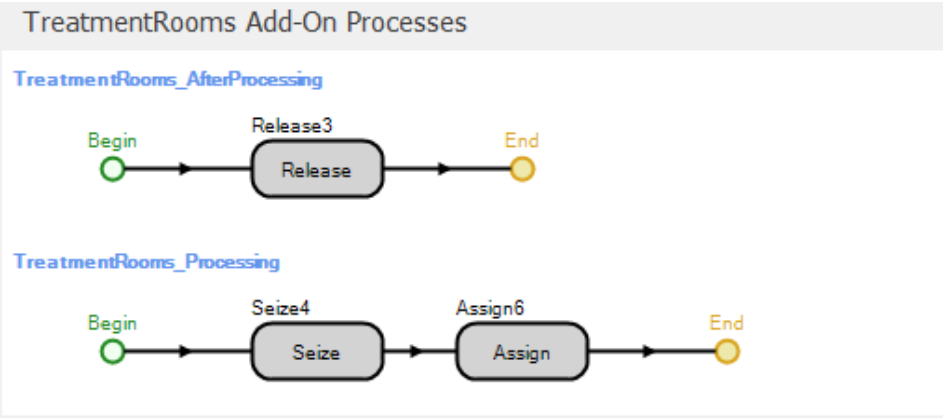


図 9.8　TreatmentRooms オブジェクトのアドオンプロセス

このプロセスでは、前の TraumaRooms_Processing プロセスとは異なり、別々の Seize ステップを使用せず、1 つの Seize ステップで必要な 2 つのリソースを同時に占有する。図 9.9 に Seize ステップのプロパティとリピートグループを示す。なお、これら 2 つの方法を使い分けているのは機能説明のためであり、どちらのプロセスもステップが 1 つでも 2 つでも同様に実装できる。

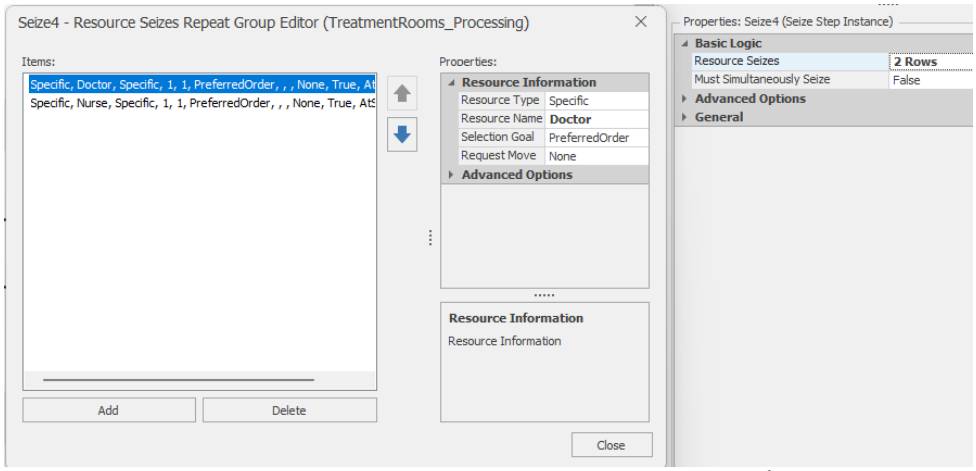


図 9.9　2 つのリソースを占有する Seize4 ステップ

図 9.10 は、TreatmentRooms_Processing アドオンプロセスの Assign ステップのプロパティを示している。エンティティを任意の時点で記録し、その記録時刻と後の時刻との時間間隔をタリー統計量で記録するという基本概念は、**パフォーマンス測定基準**として時間間隔を記録する場合と同じ考え方である。Simio な

どのシミュレーションパッケージでは、エンティティの生成から破棄までの間隔など、一部の時間間隔は簡単に追跡できるが、個々の問題で重要となる時間間隔を自動的に判断することはできない。そのため、これらの**ユーザ定義統計量**を追跡するための明確な設定方法を理解することが重要である。

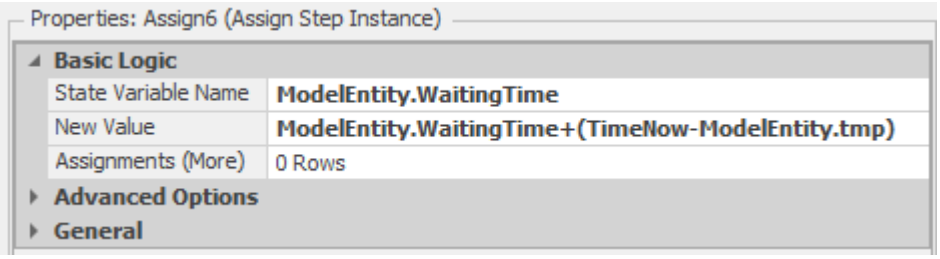


図 9.10　エンティティの待ち時間を追加する Assign ステップのプロパティ設定

図 9.11 は、NormalExit オブジェクトのアドオンプロセスを示している。患者がシステムを離れるとき、リソース配置分析で用いる待ち時間の統計を更新する。Tally ステップ（プロパティについては図 9.12 参照）は、WaitingTime エンティティ状態変数に蓄積された患者の待ち時間を記録する。

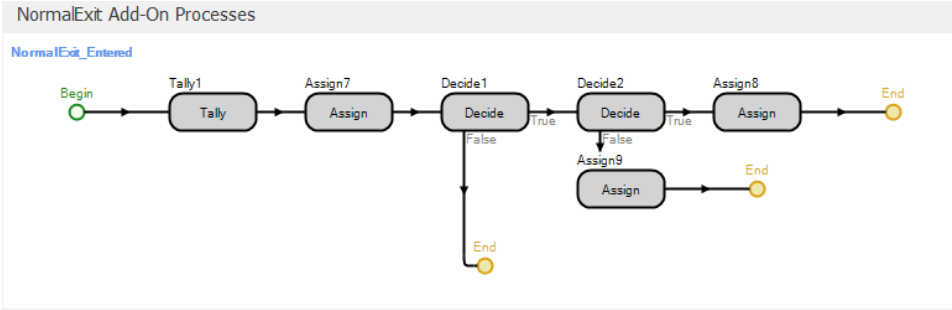


図 9.11　NormalExit オブジェクトのアドオンプロセス

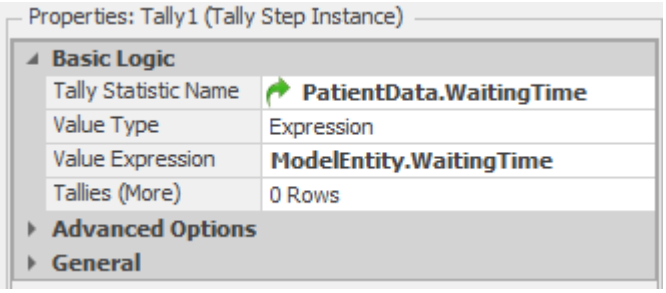


図 9.12　Tally ステップのプロパティ

Tally Statistic Name プロパティの `PatientData.WaitingTime` の値は、観測値が PatientData テーブルの WaitingTime 列で指定されたタリー統計量オブジェクトに記録されることを示している（データテーブルは図 9.2、タリーと出力統計は図 9.13 を参照）。これにより、患者タイプごとに待ち時間が記録される。これは Simio **データテーブル**機能の活用例の一つである。Simio はテーブルからタリー統計量を検索し、将来新しい患者タイプを追加しても参照を更新する必要

はない。Assign ステップでは、エンティティ状態変数 WaitingTime で蓄積される待ち時間にモデル状態変数 TotalWait を加えることで、全エンティティの総待ち時間を簡単に追跡できる。

Name	Object Type
Output Statistic Elements	
TotalWaitingTime	Output Statistic Element
TotalCost	Output Statistic Element
Satisfaction	Output Statistic Element
Tally Statistic Elements	
RoutingWaiting	Tally Statistic Element
ModerateWaiting	Tally Statistic Element
SevereWaiting	Tally Statistic Element
UrgentWaiting	Tally Statistic Element

図 9.13　Model 9-1 の Model エレメント

このプロセスの最後のステップは、患者**サービスレベル**の追跡に使用される。ED の例では、**サービスレベル**を「総待ち時間が 30 分以下の Routine 患者の割合」と定義している（30 分はデフォルト値として任意に設定）。まず最初の Decide ステップで Routine 患者を特定する（ステップのプロパティは図 9.14 を参照）。続いて、患者の待ち時間に基づいて「満足」しているかどうかを判定する。これは 2 つ目の Decide ステップで行う（ステップのプロパティは図 9.15 を参照）。

Properties: Decide1 (Decide Step Instance)

Basic Logic	
Decide Type	ConditionBased
Condition Or Pro...	**Is.Routine**
Advanced Options	
General	

図 9.14　Routine タイプの患者を識別する Decide ステップのプロパティ

Properties: Decide2 (Decide Step Instance)

Basic Logic	
Decide Type	ConditionBased
Condition Or Probability	**ModelEntity.WaitingTime <= SatisfactionThreshold**
Advanced Options	
General	

図 9.15　患者の待ち時間の満足度を判定する Decide ステップのプロパティ

実験の簡素化のため、固定値の 0.5（30 分間）ではなく、プロパティ参照（SatisfactionThreshold）を使用していることに注目してほしい。患者が満足している場合は、モデル状態変数を増加させて満足した患者（SatisfiedRoutine）をカウントする。不満足の場合は、同様にモデル状態変数を増加させて不満足な患者（UnsatisfiedRoutine）をカウントする。これらの計測は 2 つの Assign ス

テップで行う。シミュレーション実行全体のサービスレベルは出力統計量で計算する。図 9.13 にモデル出力統計量を、図 9.16 には満足している患者の割合を計算する Satisfaction 出力統計量のプロパティを示す。

Properties: Satisfaction (Output Statistic Element)

Basic Logic	
Unit Type	Unspecified
Expression	**SatisfiedRoutine/(SatisfiedRoutine+UnsatisfiedRoutine)**
Reporting & Logging	
General	

図 9.16　Satisfaction 出力統計量のプロパティ

最終的に、リソース配置分析のために複数のプロパティ参照を使用する。救急診療において必要となる診察室・治療室・検査室の数と、医師・看護師の数を決定することが重要だったため、それぞれの項目にプロパティ参照を定義した。図 9.17 ではモデルプロパティを示している。また、図 9.3 で DoctorCapacity プロパティが Doctor リソースの初期容量を指定する例を示したのと同様に、ExamRooms、TreatmentRooms、TraumaRooms の各 Server オブジェクトの容量で Nurse オブジェクトの容量を指定した。

図 9.18 に Model 9-1 の実験例を示している。プロパティ参照は実験において **Controls** として表示される点に注目してほしい。これにより、実験の各シナリオでこれらの値を変更できるため、モデル自体を修正する必要がなく、異なる**実験シナリオ**を用いてさまざまな配置構成を簡単に比較できる。

Properties

DoctorCapacity	Expression Property	DoctorCapacity
ExamCapacity	Expression Property	ExamCapacity
TreatmentCapacity	Expression Property	TreatmentCapacity
TraumaCapacity	Expression Property	TraumaCapacity
NurseCapacity	Expression Property	NurseCapacity
SatisfactionThreshold	Real Property	SatisfactionThreshold

図 9.17　Model 9-1 の Model プロパティ

Design　Response Results　Pivot Grid　Reports　Dashboard Reports　Input Analysis

Scenario			Replications		Controls					
	Name	Status	Required	Completed	DoctorCapacity	ExamCapacity	TreatmentCapacity	TraumaCapacity	NurseCapacity	SatisfactionThreshold
	Scenario1	Idle	10	0 of 10	5	6	6	3	1	0.5

図 9.18　Model 9-1 の実験例

リソース配置問題をシミュレートするために、各リソースユニットに費用を割り当てる必要がある。Model 9-1 では、診察室、治療室、検査室、医師、看護師の数に焦点を当てている。表 9.2 に、各リソースの週当たりの費用を示す。また、患者の待ち時間 1 時間につき$250 の「待ち時間の費用」を設定し、リソースとスタッフ配置の総費用を調整する。これにより、リソース費用の最小化と患者満

足度の最大化という、相反する目標のモデル化が可能となる。

表 9.2　Model 9-1 におけるリソースの各ユニットの週当たり費用

リソース	週当たり費用
Exam Room	$2,000
Treatment Room	$2,500
Trauma Room	$4,000
Doctor	$12,000
Nurse	$4,000

モデルではすでに総待ち時間を集計しているため、反復実行を 1 週間に設定し、「総費用」を出力統計量として計算できる。TotalCost 出力統計量の計算式は以下のとおりである：

2000 * ExamCapacity
+ 2500 * TreatmentCapacity
+ 4000 * TraumaCapacity
+ 4000 * NurseCapacity
+ 12000 * DoctorCapacity
+ 250 * TotalWaitingTime.Value

これにより、システム構成を総費用とサービスレベルという 2 つの指標で評価できる。目標は、総費用を可能な限り抑えつつ、サービスレベルを利害関係者が定めた最小値以上に保つことだ。図 9.19 は、4 つのシナリオを持つ初期実験を示している。実験の Responses には 2 つの出力統計（総費用の TC とサービスレベルの Satisfaction）が定義されている。比較を容易にするため、TC 応答では TotalCost の値を 10,000 で除している（この変換は単位の変更のみで、シナリオ間の比較には影響しない）。このサンプル実験では、モデルを 7 日間実行した。

Scenario		Replications		Controls						Responses	
Name	Status	Required	Completed	DoctorCapacity	ExamCapacity	TreatmentCapacity	TraumaCapacity	NurseCapacity	SatisfactionThreshold	TC	Satisfaction
Scenario1	Comple...	10	10 of 10	3	6	6	2	8	0.5	135.274	0.51335
Scenario2	Comple...	10	10 of 10	5	6	6	2	10	0.5	24.1954	0.970648
Scenario3	Comple...	10	10 of 10	4	7	7	3	9	0.5	39.8756	0.855517
Scenario4	Comple...	10	10 of 10	2	5	1	1	6	0.5	471.506	0.951079

図 9.19　Model 9-1 の初期実験

ここでは、モデルの分析対象として設定した費用がすべて仮定的な値であることに注意が必要だ。実際の現場では、これらの値を正確に決定・特定することは容易ではない。教育環境において学生が直面するもっとも重要な課題の一つが、正確なモデルデータの発見と収集の難しさだろう。

TotalCost 指標は患者の待ち時間を使用するため、システムを定常状態で評価できるようウォームアップ期間を設定する必要がある。Simio 実験の Warm-up Period プロパティを使用すると、モデルの統計はウォームアップ期間終了時に自動的にリセットされる。ただし、患者の総待ち時間を蓄積するモデル状態変数（TotalWait）は、統計要素とは異なり、Simio が自動的にリセットしないため、ウォームアップ期間終了時に手動でリセットする必要がある。これは組み込みの

OnRunWarmUpEnding プロセスで Assign ステップを使用することで簡単に実現できる（図 9.20）。このプロセスへは、Process リボンの Select Process ボタンからアクセスできる。

図 9.21 は、ウォームアップ期間を組み込んだ実験結果を示している。この実験では、実行期間を 10 日間、ウォームアップ期間を 3 日間に設定している。

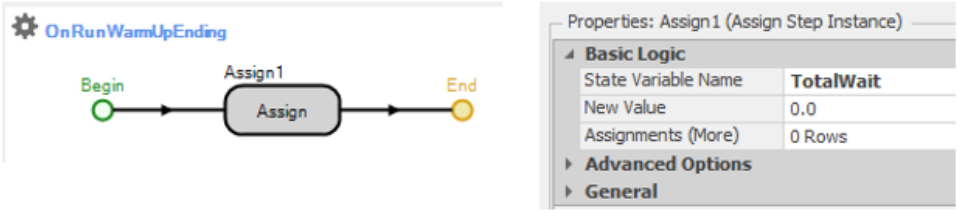


図 9.20　ウォームアップ期間終了時の TotalWait のリセット

Replications		Controls						Responses	
Required	Completed	DoctorCapacity	ExamCapacity	TreatmentCapacity	TraumaCapacity	NurseCapacity	SatisfactionThreshold	TC	Satisfaction
10	10 of 10	3	6	6	2	8	0.5	135.695	0.440659
10	10 of 10	5	6	6	2	10	0.5	20.7596	0.97985
10	10 of 10	4	7	7	3	9	0.5	32.4475	0.832076
10	10 of 10	2	5	1	1	6	0.5	439.226	0.950482

図 9.21　ウォームアップ期間を持つ Model 9-1 の実験結果

9.1.1　OptQuest による Model 9-1 の最適リソースレベルの探索

病院の経営者は、5 つの能力レベル（医師、診察室、治療室、検査室、看護師）の最適な組み合わせを見つけ出したいと考えている。目標は総費用を最小化しながら、適切なサービスレベル（Routine 患者の待ち時間が 30 分以下である割合）を維持することだ。経営者は、このサービスレベルが 0.8（80%）以上であることを重視している。つまり、Routine 患者の 20%未満しか 30 分以上待たせてはいけない。なお、この基準値は主観的なものであり、待ち時間の閾値や満足度の目標値は、利害関係者が決定すべきものである。能力レベルには以下の制約がある：

・各能力レベルは整数値のみ
・検査室（1 以上）を除き、すべて 2 以上
・医師（Doctor）：最大 10 名
・看護師（Nurse）：最大 15 名
・診察室（Exam）・治療室（Treatment）：各最大 10 室
・検査室（Trauma）：最大 5 室

この条件について詳しく考えてみよう。目標は、Simio モデルのアウトプットである総費用の**最小化**だ。一見すると、各能力レベルを最小値（検査室は 1、他は 2）に設定するのが最適解に思えるかもしれない。しかし、総費用には患者の待ち時間に対するペナルティ（1 時間当たり$250）が含まれている。そのため、最小の能力レベルでは待ち行列が長くなり、かえって総費用が増加する可能性が高い。また、サービスレベルを 0.8 以上に保つ必要があり、これは最小値以上の能力レベルが必要となることを示唆している。

この問題は、出力が確率的で（サンプリング誤差を含む）、従来の**数理計画法**（線

形計画法、非線形計画法、整数計画法など）では解くことが難しい。しかし、最適化問題として定式化することで、状況の理解が深まる。数理計画的な定式化は以下のとおりである：

min　TotalCost
Subject to:
$2 \leq \text{DoctorCapacity} \leq 10$
$2 \leq \text{ExamCapacity} \leq 10$
$2 \leq \text{TreatmentCapacity} \leq 10$
$1 \leq \text{TraumaCapacity} \leq 5$
$2 \leq \text{NurseCapacity} \leq 15$
上記 5 つの能力は整数
$\text{Satisfaction} \geq 0.8$

ここで最小化の対象となるのは、DoctorCapacity、ExamCapacity、Treatment Capacity、TraumaCapacity、NurseCapacity の 5 つの変数だ。**目標関数**は TotalCost で、「Subject to」以下が**制約条件**となる（最後のものは非目標関数の出力）。最初の 5 つの制約と整数条件は入力に関するもので、実験設定時に確認できる。一方、Satisfaction は出力応答であり、0.8 以上という条件の充足はシミュレーション実行後でないとわからない（そのため、これは制約ではなく**必要条件**と呼ばれる）。

では、この最適化問題を具体的にどう**解く**べきだろうか？まず考えられるのは、図 9.19 のサンプル実験を拡張し、より多くのシナリオで容量レベルを探索することだ。TC（総費用）を昇順にソートし、サービスレベル要件（$\text{Satisfaction} \geq 0.8$）を満たす最初のシナリオを見つければよい。しかし、能力レベルの制約から計算すると、シナリオ数は 9×9×9×5×14=51,030 となり、これは実験を設定するには多すぎる。また、各シナリオで信頼できる Total Cost と Satisfaction の推定値を得るには、約 120 回の反復実行（各 10 日間）が必要だ。クアッドコア 3.0GHz のノート PC で 1 回の実行に約 1.7 秒かかるとすると、51,030×1.7×120 秒、つまり約 120 日（4 か月）を要する。これは単なる例示だが、実際の問題でこのような総当たり的なアプローチは現実的ではない。

このアプローチでは、明らかに劣位なシナリオや必要条件を満たさないシナリオの実行に 4 か月もの時間を費やすことになる。より効率的な方法が必要だ。最適化問題を 5 つの入力変数の観点から見ると、5 次元空間の整数格子点を探索する問題といえる。ある格子点から開始し、近接点をシミュレートしながら、総費用を下げつつサービスレベル要件を満たす方向へ探索を進めることができる。この種の**ヒューリスティック探索法**に関する研究は多いが、その多くは確定的な関数の最適化に焦点を当てている。これらの研究に基づいて開発されたソフトウェアの一つが、OptQuest（OptTek Systems, Inc.（www.opttek.com））だ。Simio などのシミュレーションパッケージとの親和性が高く、今回の病院の能力問題のような課題に対して、最適に近い実行可能解を探索できる。詳しい情報は、Winter Simulation Conference のオープンアクセスチュートリアル（Glover et al. 1999）から始めるとよい。より詳細はハンドブック（Glover and Kochenberger 2003）

を参照してほしい。

この研究成果を活用し、OptQuest は 5 次元空間内の特定の点（ユーザが初期値として 5 つの容量を設定）から探索を開始する。総費用を削減する方向を決定しながら、入力パラメータの制約と出力要件（$\text{Satisfaction} \geq 0.8$）を満たすよう探索を進める。なお、この例では使用していないが、OptQuest では容量変数間の線形制約も次のように指定できる：

$$5 \leq \text{DoctorCapacity} + \text{NurseCapacity} \leq 20$$

このような制約は、それぞれ不等式として指定する。OptQuest は 5 次元空間の特定点から最適点（あるいは少なくとも**最良**の点。これらの手法は最適解の保証はできない）へ効率的に探索を進めるため、複数のヒューリスティック探索法（**散布検索法**、**タブー探索**、**ニューラルネットワーク**など）を組み合わせている。Glover et al.（1999）は次のように述べている：

> …最適化とシミュレーションの分野では、質の高い解を生み出すシミュレーション手順の確立が長年の課題だった。…最適化手順は、入力に対するシミュレーションモデルの出力を評価に用いる。この評価と、過去の評価結果の統合分析に基づいて、新しい入力値を決定する。…最適化手順は特殊な「非単調探索」を行うよう設計されており、生成される入力値は必ずしも常に改善するわけではないが、時間とともに最良解への効率的な経路を提供する。このプロセスは、適切な終了基準（通常はユーザが指定する探索時間）を満たすまで続く。

散布検索法、タブー探索、ニューラルネットワークの OptQuest での具体的な活用については、Glover et al.（1999）と Glover and Kochenberger（2003）を参照してほしい。

Simio で OptQuest を使用するには、まず実験で最適化問題を設定する。最小化（または最大化）したい項目（この例では総費用の最小化）、入力制約（容量の範囲と整数制約）、その他の要件（$\text{Satisfaction} \geq 0.8$）を指定する。その後、Simio で実験することを止め OptQuest を起動する。OptQuest は実行するシナリオとその順序を効率的に決定し、4 か月の総当たり計算よりもはるかに短時間で（完全な最適性は保証できないものの）最良解を見つけ出す。時間短縮の一例として、あるシナリオが既存の解より明らかに劣ることが判明した時点で反復実行を打ち切る方法がとられる。

ここからは、Simio で OptQuest を具体的に設定する手順を説明する。まず、OptQuest の使用・不使用両方で実験できるよう、新しい実験を（Project Home リボンの `New Experiment` ボタンから）作成する。この実験は `OptQuest1` と名づけた。次に、Simio の**アドイン**として OptQuest を追加する。実験アドインは Simio の拡張機能で、実験の設計・実行能力を強化する（アドインはユーザ自身や他者が作成可能で、詳細は Simio のマニュアルを参照）。アドインを使用して、新規シナリオの設定・実行・結果の解析が可能だ。これら 3 つの機能を組み合わせることで、OptQuest のような強力なツールを実現できる。アドインの追加は、

Experiment の Design リボンの Select Add-In ボタンから行う。リストから OptQuest for Simio を選択する（図 9.22）。

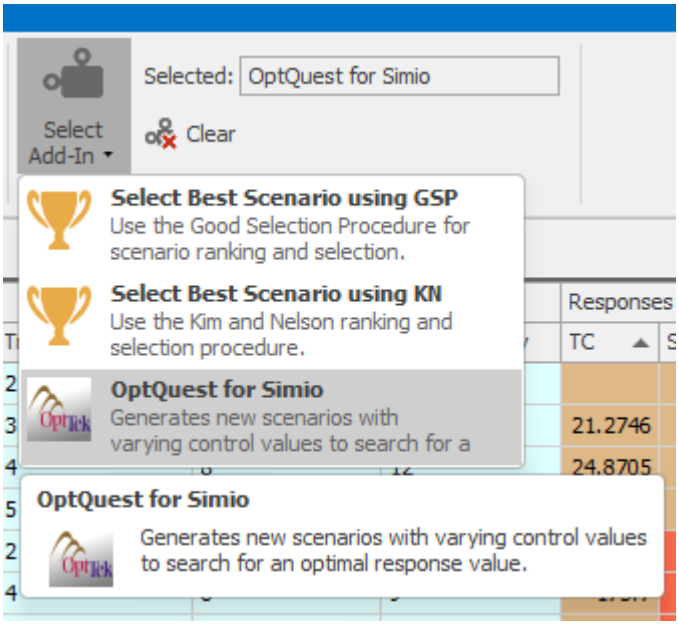


図 9.22　OptQuest アドインの選択

OptQuest を選択すると、”Setting an add-in on an experiment cannot be undone. If you set the add-in, the current undo history will be cleared. Do you want to continue?”（実験にアドインを設定すると元に戻せません。アドインを設定すると現在のアンドゥ履歴が消去されます。続行しますか？）という警告が表示される。アンドゥ履歴を保持したい場合は「**いいえ**」を選択し、モデルをコピーしてからそのコピーに OptQuest を設定するとよい。必要なければ「**はい**」を選択すると、Simio がアドインを読み込む（このときアンドゥ履歴は消去される）。OptQuest アドインが追加されると、実験に OptQuest の動作を制御するための複数のプロパティが追加される（図 9.23）。

OptQuest 関連のプロパティは以下のとおりである：

- Min Replications：OptQuest が定義した各シナリオについて、Simio が実行する最小の反復回数。
- Max Replications：OptQuest が定義した各シナリオについて、Simio が実行する最大の反復回数。
- Max Scenarios：Simio が実行する最大シナリオ数。OptQuest は、目標関数の最適値（この場合は Total Cost の最小値）を探索し、必要条件（Satisfaction が 0.8 以上）を満たす**決定変数**（Doctors、Exam Rooms、Treatment Rooms、TraumaRooms、Nurse の数）の組み合わせを体系的に探索する。このプロパティは探索するシナリオ数に上限を設定する。必ずしもこの上限まで実行するわけではないが、特に大規模な解空間を持つ問題では、この制限が有効な終了条件となる。ただし、この値を小さく設定しすぎると、OptQuest が見つけられる

Properties: OptQuest1 (Experiment)

Analysis	
Warm-up Period	**3**
Units	**Days**
Default Replications	10
Confidence Level	95%
Upper Percentile	75%
Lower Percentile	25%
Primary Response	**TC**
Advanced Options	
OptQuest for Simio - Parameters	
Min Replications	**10**
Max Replications	**30**
Max Scenarios	**100**
Confidence Level	**95%**
Relative Error	**0.1**
Objective Type	**Single Objective**
General	
Name	OptQuest1

図 9.23　OptQuest アドイン追加後の「OptQuest1」の Experiment プロパティ

解の質が低下する可能性がある。

・Confidence Level：シナリオ間の統計的比較を行う際の信頼水準。

・Relateive Error：信頼区間の半幅が標本平均に対して満たすべき割合。たとえば、標本平均が 50 で、Relateive Error が 0.1（信頼区間の半幅を平均の 10% 以内に抑える）の場合、信頼区間は 50 ± 5（5 ＝ 0.1 × 50）以内となる。OptQuest は初期の反復実行後にこの条件を確認し、未達成の場合は追加の反復実行を行う（Max Replications の上限まで）。

また、OptQuest アドインを加えると、実験の Control にプロパティが追加される。OptQuest アドインでは、最適化の制約を Control または Constraint のいずれかで指定できる。本モデルでは、各制約が**単一の**決定変数（モデル内のプロパティ参照）の値を制限する必要があるため、Control を使用する。一方、医師と看護師の総数を制限するような線形の式の組み合わせによる制約を使用する場合は、Constraint を使用する。

図 9.24 は、OptQuest アドイン追加後の DoctorCapacity 決定変数のプロパティを示している（なお、アドイン追加前は Control に編集可能なユーザプロパティが存在しなかった）。

「新しい」Control プロパティは、OptQuest に対して以下の設定を行う：探索可能範囲（Include in Optimization）、決定変数の取り得る値（Minimum Value と Maximum Value プロパティ値の間）、そして探索時の増分（Increment プロパティ値）。図 9.24 の Control プロパティは、DoctorCapacity の制約を 2 から 10 までの整数として指定している。ExamCapacity、TreatmentCapacity、Trauma Capacity および NurseCapacity 決定変数についても、同様のプロパティ設定を

Properties: DoctorCapacity (Control)

DoctorCapacity	
Description	
OptQuest for Simio - Parameters	
Include in Optimization	**Yes**
Minimum Value	**2**
Maximum Value	**10**
Increment	**1**

図 9.24　OptQuest アドイン追加後の DoctorCapacity のプロパティ

行った。なお、SatisfactionThreshold 決定変数は今回の最適化では使用しないため（プロパティ参照でしきい値を指定したため）、Include in Optimization プロパティを No に設定する。最適化に使用しない決定変数については、Simio がすべてのシナリオでデフォルト値（この場合は 0.5）を使用する。

この最適化で残る唯一の「制約」は、Satisfaction レベルに関する必要条件だ（Satisfaction は入力ではなく出力であるため、制約条件ではなく必要条件と呼んでいる）。図 9.25 に示す Satisfaction レスポンスのプロパティは、Lower Bound を 0.8 に設定した点を除き、前回の実験と同じ設定である。

Properties: Satisfaction (Response)

OptQuest for Simio - Parameters	
Weight	**1**
General	
Name	**Satisfaction**
Display Name	Satisfaction
Expression	**Satisfaction.Value**
Unit Type	Unspecified
Objective	None
Lower Bound	**0.8**
Upper Bound	

図 9.25　Satisfaction レスポンスのプロパティ

OptQuest を設定する最終段階は、**目標関数**の定義だ。これは、最適化の評価基準を OptQuest に設定することを意味する。総費用を最小化したいため、（前述のとおり）TC レスポンスを定義し、Objective プロパティを Minimize に設定する（図 9.26）。

前述のように、TC レスポンスの定義において TotalCost の値を 10,000 で割っていることに注意してほしい（実験テーブルで結果を目視で比較しやすくするため）。最後に、TC レスポンスを最適化の目的として設定するため（この場合は TC の最小化）、TC に実験プロパティ Primary Response を設定する。

Run ボタンをクリックすると、OptQuest の実行過程を観察できる。図 9.27 は、OptQuest の実行結果から得られた上位シナリオを示している（Total Cost の昇順で並べているため、最上位のシナリオが最小総費用となっている）。また、図にはリストの下方に位置するシナリオも表示されている（シナリオ 071 以降）。注：使用する Simio のバージョンによって表示されるシナリオが異なる場合があるが、

Properties: TC (Response)

OptQuest for Simio - Parameters	
Weight	**1**
General	
Name	**TC**
Display Name	TC
Expression	**TotalCost.Value / 10000**
Unit Type	Unspecified
Objective	**Minimize**
Lower Bound	
Upper Bound	

図 9.26　TC レスポンスのプロパティ

Scenario			Replications		Controls						Responses	
	Name	Status	Required	Completed	DoctorCapa...	ExamCa...	Treatment...	Trauma...	NurseCap...	SatisfactionThreshold	TC	Satisfaction
☑	067	Completed	10	10 of 10	5	6	5	2	11	0.5	20.1005	0.996647
☑	091	Completed	10	10 of 10	5	7	5	2	10	0.5	20.1274	0.999096
☑	093	Completed	10	10 of 10	5	6	5	2	10	0.5	20.1534	0.992866
☑	055	Completed	10	10 of 10	5	7	5	2	11	0.5	20.2379	0.99973
☑	050	Completed	10	10 of 10	5	8	5	2	11	0.5	20.3981	0.998606
☑	056	Completed	10	10 of 10	5	8	5	2	10	0.5	20.5039	0.994439
☑	069	Completed	10	10 of 10	5	8	6	2	10	0.5	20.5165	0.992598
☑	053	Completed	10	10 of 10	5	9	5	2	11	0.5	20.6096	0.999656
☑	042	Completed	10	10 of 10	6	6	5	3	9	0.5	20.6344	0.992048
☑	083	Completed	10	10 of 10	6	7	6	2	10	0.5	20.6951	0.9973
☑	070	Completed	10	10 of 10	6	8	6	2	10	0.5	20.8013	0.997629
☑	084	Completed	10	10 of 10	6	6	6	2	11	0.5	20.9669	0.995014
☑	031	Completed	10	10 of 10	6	6	6	3	10	0.5	21.0706	0.994012
☑	034	Completed	10	10 of 10	6	6	4	3	9	0.5	21.2076	0.993285
☑	086	Completed	10	10 of 10	5	7	6	3	11	0.5	21.2746	0.995151
☑	001	Completed	10	10 of 10	6	6	6	3	9	0.5	21.2746	0.980892
☑	089	Completed	10	10 of 10	6	7	6	3	11	0.5	21.3287	0.99928
☑	011	Completed	10	10 of 10	6	6	6	3	8	0.5	21.3391	0.959988
☑	081	Completed	10	10 of 10	5	5	5	2	11	0.5	21.4052	0.95962
☑	090	Completed	10	10 of 10	5	5	5	2	12	0.5	21.5276	0.965561
☐	071	Completed	11	11 of 11	4	4	6	2	10	0.5	29.311	0.781381
☑	003	Completed	10	10 of 10	10	10	10	5	15	0.5	29.6702	0.999913
☑	051	Completed	10	10 of 10	10	7	3	5	5	0.5	29.957	0.999906
☑	058	Completed	10	10 of 10	8	4	6	5	13	0.5	30.9873	0.816184
☑	028	Completed	23	23 of 23	5	10	7	3	7	0.5	31.4307	0.840042
☐	046	Completed	10	10 of 10	10	4	4	3	10	0.5	32.3188	0.781007

図 9.27　初回の OptQuest 実行結果（実験テーブルから抜粋した 2 つの部分）

同じシナリオ構成であれば、類似のレスポンス値が得られるだろう。なお、これらの違いはプロセッサやコアの数、処理速度の違いによっても生じることがある。

リストの一番上のシナリオ 067 の構成は、医師 5 名、診察室 6 室、処置室 5 室、救急処置室 2 室、看護師 11 名で、総費用は 20.1005 万、満足度は 0.997 であった。注目すべき点は、すべての満足度の値が非常に高いことである（実行不可能な解、つまり満足度 80%の必要条件を満たさないものを見るには、リストをシナリオ 071 より下にスクロールする必要がある。これらは赤い背景で表示され、チェックされていない）。

当然、「これが本当に最適解なのか」という疑問が生じる。残念ながら、「確信できない」としか答えられない。前述のとおり、OptQuest はヒューリスティックな

探索手法を用いるため、発見した解が真の最適解かどうかを判断する方法は一般的には存在しない。このシステムには 51,030 のシナリオ構成があり、OptQuest ではその中から 100 シナリオのみをテストするという制限がある。これが混合最適化問題におけるヒューリスティック解の本質である。よい解は得られたと確信しているが、唯一の最適解を保証する方法がないため、これ以上の改善は困難だ。

OptQuest で最適解を確実に発見できるとはいえないが、**ユーザ**が手作業で探索範囲を網羅的に調べようとするよりも、OptQuest の方が効率的である（37 番目のシナリオを確認するころには、きっとコンピュータ画面を見るのに飽きているだろう）。OptQuest が特に威力を発揮するのは、探索範囲を適切に制限できる場合である。たとえば、この問題では Trauma Rooms の数を 1 から 5 に制約したが、Urgent 患者（検査室を必要とする唯一の患者タイプ）の予想到着率は**かなり**低く（1 時間当たり約 0.6 人）、3 室以上の検査室が必要となる可能性は低い。そのため、Trauma Rooms 決定変数の上限を 5 から 2 に減らすと、探索範囲が 51,030 から 20,412 に縮小され、より少ないシナリオで OptQuest の性能を向上できる。これは最適解を求めるプロセスにおけるユーザの知見の重要性を示している。OptQuest は実行中のモデルや実システムの特性を理解していないが、**ユーザ**はそれらを理解しており、その知識を活用することでコンピュータの負荷を大幅に軽減し、最終的な「解」の質を改善できる。では、OptQuest が特定したシナリオの評価と、可能であれば「最良の」シナリオの特定について検討しよう。

9.1.2　サブセット選択と KN の活用

図 9.27 に示されているレスポンス値は、各シナリオを 10 回反復実行して得られた標本平均であるため、**サンプリング誤差**の影響を受けている。そのため、OptQuest 実験から得られた標本平均に基づく順序が、真に最適な解（最小総費用の観点から見た最適なリソース構成）であるかは確実ではない。図 9.28 では、OptQuest 実行による上位 5 つのシナリオを SMORE プロットで示している。

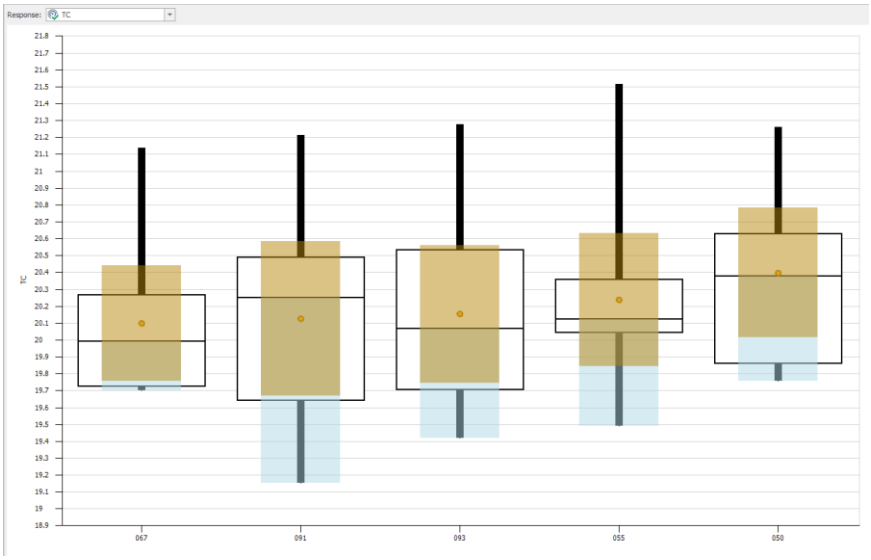


図 9.28　トップ 5 シナリオに対する Model 9-1 の OptQuest 実行の SMORE プロット

シナリオ 067 の総費用（TC）の標本平均は他のシナリオと比べてもっとも低いものの、SMORE プロットで信頼区間を考慮すると、その優位性は不明確となる。実際、図 9.28 において 5 つの標本平均間に**統計的な有意な差異**があるかどうかの判断は困難だ（SMORE プロットの目視検査では正確な統計的有意差を判断できないが、さらなる分析の余地はある）。この不確実性はサンプリングの不確実性に起因しており、ここで Subset Selection Analysis が活用できる。この分析手法については、5.5 節で小規模な例を用いて解説済みである。

図 9.29 は、2 つ目の OptQuest 実行の結果を示している。このときは Trauma Room の上限を 2 に変更し、Max Scenarios を 25 に設定して、Subset Selection Analysis 機能を実行した（Design リボンの上の Subset Selection Analysis アイコンをクリック）。

Scenario			Replications		Controls						Responses	
■	Name	Status	Required	Completed	DoctorCapa...	ExamCa...	Treatment...	Trauma...	NurseCap...	SatisfactionThreshold	TC ▲	Satisfaction
☑	001	Completed	10	10 of 10	6	6	6	2	9	0.5	20.7976	0.981578
☑	011	Completed	10	10 of 10	6	6	6	2	8	0.5	20.9204	0.956662
☑	016	Completed	10	10 of 10	6	7	7	2	10	0.5	21.1561	0.986999
☑	015	Completed	10	10 of 10	6	6	6	2	12	0.5	21.4892	0.995973
☑	023	Completed	10	10 of 10	5	6	8	2	12	0.5	21.6921	0.991178
☑	012	Completed	10	10 of 10	5	5	5	2	8	0.5	21.7886	0.915834
☑	019	Completed	10	10 of 10	6	6	6	2	15	0.5	22.6495	0.997254
☑	025	Completed	10	10 of 10	6	5	4	1	13	0.5	22.8364	0.960614
☑	014	Completed	10	10 of 10	8	8	8	2	9	0.5	23.1533	0.995992
☑	024	Completed	10	10 of 10	9	5	6	2	6	0.5	23.3159	0.954005
☑	013	Completed	10	10 of 10	8	8	8	2	11	0.5	23.6759	0.999915
☑	005	Completed	10	10 of 10	8	8	8	2	12	0.5	24.109	1
☑	017	Completed	10	10 of 10	8	9	8	2	12	0.5	24.3447	0.999475
☑	022	Completed	10	10 of 10	8	7	9	1	5	0.5	24.89	0.923485
☑	003	Completed	10	10 of 10	10	10	10	2	15	0.5	28.4906	1
☑	021	Completed	14	14 of 14	3	8	3	2	11	0.5	40.1616	0.999702
☐	010	Completed	19	19 of 19	6	9	5	2	4	0.5	54.9788	0.644875
☐	004	Completed	30	30 of 30	4	4	4	1	5	0.5	120.11	0.351441
☑	007	Completed	30	30 of 30	8	4	2	2	9	0.5	205.41	0.800683
☐	008	Completed	30	30 of 30	4	3	9	2	3	0.5	357.652	0.24529
☐	006	Completed	25	25 of 25	3	5	4	1	2	0.5	382.092	0.590327
☐	002	Completed	11	11 of 11	2	2	2	1	2	0.5	436.931	0.486869
☐	009	Completed	30	30 of 30	9	2	7	1	10	0.5	461.918	0.0234235
☐	020	Completed	30	30 of 30	10	2	10	1	2	0.5	480.455	0.148875
☐	018	Completed	30	30 of 30	2	2	2	2	15	0.5	978.908	0.0277586

図 9.29　Subset Selection Analysis 実行後の 2 回目の OptQuest 実行結果

この機能は、すべてのシナリオの反復実行結果を考慮して、シナリオを 2 つのグループに分割する：

- 「おそらく最善」グループ：レスポンスの背景色が濃い茶色で表示され、そのサンプル平均が最善のサンプル平均と統計的な差がないシナリオ
- 「棄却」グループ：レスポンスのセルが薄黄色の背景色で表示され、統計的に劣るシナリオ

シナリオごとに 10 回の反復実行による初期実験に基づき、「おそらく最善」グループのシナリオはさらなる検討に値するが、「棄却」グループは統計的に劣っているため、（特定のレスポンスについて）これ以上の検討は不要と判断される。ただし、「おそらく最善」グループのシナリオは、それらが「よい」と「実証」されたわけではなく、単に棄却されていないだけである。

この非対称性は古典仮説検定に似ている。つまり、帰無仮説を棄却することは真実の「証拠」を示すものではない。棄却する証拠がないだけで、一方で帰無仮説を棄却することは、その仮説が偽であるという確かな証拠となる。

注意点として、実験で一つもシナリオが「棄却」されない場合、Subset Selection Analysis が統計的に異なるグループを特定するには反復回数が不十分である可能性がある。この場合、（シナリオ間に実際に**統計的な差異が存在しない**場合を除き）追加の反復実行で解決できる。

この例では、濃い茶色の「おそらく最善」グループには 5 つのシナリオが含まれる（図 9.29）。

ここで「さて、どうしようか」という疑問が生じるだろう。選択肢は 2 つある：5 つの「よい」構成をそのまま採用するか、追加実験を行うかだ。唯一最善の構成（または可能な限り少数の最善シナリオ）を特定したい場合は、現在の「おそらく最善」グループの 5 つのシナリオで追加の反復実行が必要となる。そして、より少ない「おそらく最善」グループ（または理想的には唯一の「最善」シナリオ）を特定するため、Subset Selection Analysis を再実行することになる。

実験では「棄却」グループのシナリオを除外し、残りの 5 シナリオについて各 40 回の反復実行を追加する（合計 50 回の反復実行）。その結果、新しい「おそらく最善」グループには 4 つのシナリオが含まれることになる（結果は省略）。なお、追加の反復実行を行う前に、リボンの Add-ins にある Clear アイコンで OptQuest アドインを「消去」する必要がある。ここでも、4 つの「よい」構成を採用するか、さらなる実験を行うかの選択肢がある。「最善」の構成を追求する場合は、上記のプロセスを繰り返す（新しい「棄却」グループのシナリオを除外し、残りのシナリオで追加の反復実行を行う）。ここでは別のアプローチとして、このプロセスを支援する別の Simio アドインを使用しよう。

Kim and Nelson（2001）を理論的背景とする Select Best Scenario using KN（KN を用いた最善シナリオの選択）アドインは、5.5 節で説明したように、与えられたシナリオの集合から「最善」のシナリオを特定するように設計されている。このアドインは、「最善」のシナリオを判定するか、ユーザが指定した最大数のシナリオに達するまで、1 反復ずつ候補シナリオの追加反復を実行し続ける。以前の OptQuest 使用時と同様に（図 9.22）、この KN アドインを実行するには、Design リボンの Select Add-In アイコンから Select Best Scenario using KN を選択する。すると、アドインに関連する新しい実験プロパティが表示される（図 9.30）。

Confidence Level プロパティ（信頼区間に用いる水準を指定する）に加えて、Indifference Zone と Replication Limit を指定する。Indifference Zone（**無差別領域**）は検出される最小の「重要な」差を表す。たとえば、21.65 と 21.68 は数値的には異なるが、特定の問題では 0.03 の差は重要でないかもしれない（無差別領域の適切な値は、レスポンスの測定対象や研究の文脈によって決まる）。0.10 の無差別領域を指定した場合、実験においてアドインはこれらの値を区別しない。完璧主義的に非常に小さな領域を設定しようとすると、わずかな準最適性を検出

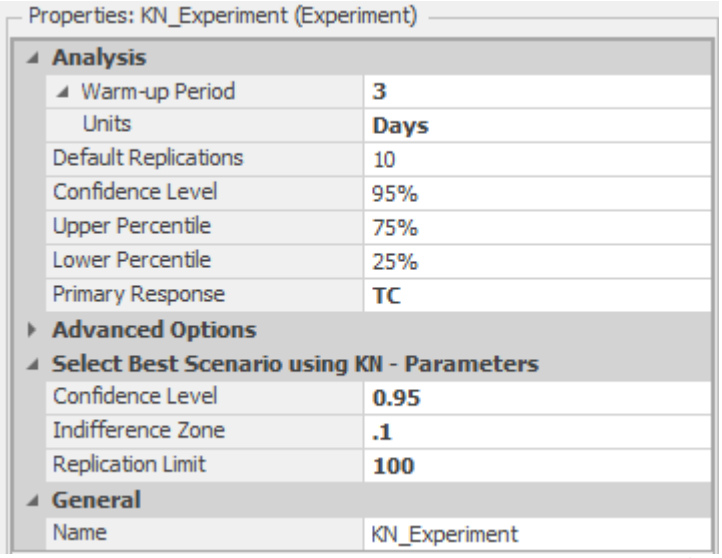


図 9.30　Select Best Scenario using KN アドインの Experiment プロパティ

するために膨大な計算時間が必要となる（計算量は無差別領域が狭くなるにつれて超線形的に増加する）。Replication Limit（反復の限界値）は、最善のシナリオの探索を終了する（そして残りのシナリオを「統計的に差異がない」とする）前に実行する反復回数の上限をアドインに設定する。

図 9.31 は、アドイン実行後の実験結果を示している。最左端のチェックボックスにチェックがついたシナリオ 001、016、011 が、100 回の反復実行後に「最善」の構成として特定された（反復回数はユーザではなく KN の手続きが決定）。複数のシナリオがチェックされた状態で終了した場合、それは現在の反復実行回数（この例では 100 回）と無差別領域に基づいて、それらのシナリオ間に有意な差が見られないことを意味する。シナリオ 001 は、OptQuest の 2 回目の実行（図 9.29）で平均総費用が最小だったものと同じだが、追加の反復実行後では総費用が上昇している（21.045（追加実行後）対 20.798（追加実行前））。SMORE プロット（図省略）では、標本平均の周りに非常に小さい信頼区間が示され、OptQuest 実験での 10 回の初期反復では標本平均が低めに出ている可能性が高いことを示している。このことから、実験結果に基づく意思決定の前には、信頼区間を慎重に確認し、必要に応じて反復実行を増やすべきことがわかる。

Scenario			Replications		Controls						Responses	
■	Name	Status	Required	Completed	DoctorCapa...	ExamCa...	Treatment...	Trauma...	NurseCap...	SatisfactionThreshold	TC	Satisfaction
☑	001	Completed	101	101 of 101	6	6	6	2	9	0.5	21.0447	0.976888
☑	016	Completed	101	101 of 101	6	7	7	2	10	0.5	21.3346	0.986766
☐	015	Completed	77	77 of 77	6	6	6	2	12	0.5	21.5211	0.994522
☑	011	Completed	101	101 of 101	6	6	6	2	8	0.5	21.6968	0.948813
☐	023	Completed	50	50 of 50	5	6	8	2	12	0.5	21.9548	0.985824
☐	019	Completed	10	10 of 10	6	6	6	2	15	0.5	22.6495	0.997254
☐	025	Completed	10	10 of 10	6	5	4	1	13	0.5	22.8364	0.960614
☐	012	Completed	50	50 of 50	5	5	5	2	8	0.5	23.0305	0.908186
☐	014	Completed	10	10 of 10	8	8	8	2	9	0.5	23.1533	0.995992
☐	024	Completed	10	10 of 10	9	5	6	2	6	0.5	23.3159	0.954005

図 9.31　Select Best Scenario using KN アドインによる実行後の実験結果

KN 選択手順の完全な説明と理論的根拠はここでの範囲を超えているが、簡単にいえば、以下のように動作する。まず、すべての競合シナリオに対して初期の反復実行が行われる（最初はすべてのシナリオが競合中とされる）。次に、シナリオの可能なすべてのペアについて、平均値の差の標本分散が計算される。これらの分散に基づいて、特定のシナリオが最善であるか、少なくとも最善の無差別領域内にあるというじゅうぶんな確信が得られるまで、生存シナリオを 1 つずつ繰り返し実行する。この過程で、最善候補でないと判断されたシナリオは除外され、それ以上の反復は行われない。これにより競合シナリオの数が減少し、計算効率が向上する。ただし、ユーザが指定した反復回数の制限（この例では 100 回）に達した時点で、まだ競合中の生存シナリオはすべて最善とみなされる。

この KN による最善シナリオの選択手法は、サブセット選択とは根本的に異なるアプローチを取る。サブセット選択では**固定数の反復実行**を行い、そのサンプルサイズに基づいてシナリオを「おそらく最善」と「棄却」に分類する。一方、KN の手法では、単一の最善シナリオを特定するために、各シナリオに**必要な数の反復**を実行する。

従来の方法と比較して、KN には少なくとも 2 つの重要な利点がある。第一に、**完全に逐次的**な処理が可能で、2 段階サンプリングではなく 1 回ずつ追加の反復実行を行える。2 段階サンプリングでは、初期の反復実行後に 2 段階目として別の複数の反復が行われるため、必要以上の反復回数となる可能性がある（大規模で複雑なモデルでは、1 回の反復に数時間かかることもある）。第二に、シナリオ間で**共通乱数**を同期させることで、標本平均間の差異の真の分散を低減できる。KN の詳細については、Kim and Nelson（2001）を参照されたい。

もう一つの関連手法として、**Select Best Scenario using GSP**（GSP を用いた最善シナリオの選択）アドインがある。この GSP アドインは、大規模モデルに特化したシナリオ選択アルゴリズムである。KN アドインと類似しているが、並列処理環境で多数のシナリオ評価が必要な大規模モデルにおいて、より高いパフォーマンスを発揮する。このアドインは、コーネル大学での博士課程研究の一環として、Sijia Ma によって開発された。両方のシナリオ選択アルゴリズムの詳細は、Simio ヘルプの「Select Best Scenario」で確認できる。

ここで手順を整理すると、最適化問題に対して、まず OptQuest アドインを使用して有望なシナリオ（特定の値を持つプロパティ構成）を特定した。次に、`Subset Selection Analysis` 機能で、シナリオを「おそらく最善」と「棄却」グループに分類した。その後、「おそらく最善」グループのシナリオに対して追加実験を実施し、反復回数を増やしてグループの規模を縮小した。最後に、`Select Best Scenario using KN` アドインで「最善」の構成を特定した。ヒューリスティックな最適化を用いているため、実験空間のすべての構成を評価できるわけではなく、この手順で必ずしも最善の構成（あるいはじゅうぶんよい構成）を発見できるとは限らない。しかし、実験結果は、この体系的なアプローチが試行錯誤的な実験よりも効果的であることを示している。

9.2　Model 9-2：ピザのテイクアウトモデル

Model 9-2 もリソース配置に関する問題を扱う。対象となるのは、電話注文を受け付け、顧客が店頭で受け取るテイクアウト方式のピザレストランである。顧客からの電話は 1 時間当たり 12 本の割合で着信し、**着信時間間隔**は平均 5 分の指数分布に従う。店舗には 4 回線の電話があり、すべての回線が使用中の場合、新規の着信は話し中となる。注文内容は 1 枚（50%）、2 枚（30%）、3 枚（15%）、4 枚（5%）の比率で分布している。電話での注文受付時間は 1.5 分から 2.5 分の一様分布に従う。注文を受けると、システムは「顧客チケット」を印刷し、ピザが完成するまでレジ付近で保管される。このシステムでは、注文ごとにピザの作成、調理、箱詰めが行われる。従業員が作成と箱詰めを担当し、半自動オーブンで調理を行う。

作成ステーションでは、最大 3 枚のピザを同時に作成できるスペースがあるが、従業員は 1 度に 1 枚しか作れない。作成時間は三角分布（最小 2 分、最頻 3 分、最大 4 分）に従う。作成後のピザはオーブンで調理され、その時間も三角分布（最小 6 分、最頻 8 分、最大 10 分）に従う。オーブンは同時に 8 枚まで調理可能で、調理完了後は自動的に取り出されて箱詰め待ちエリアに移動する（これが「半自動」の部分である）。箱詰めステーションでは 3 人まで同時作業が可能だが、従業員は 1 度に 1 枚しか担当できない。箱詰め時間は三角分布（最小 0.5 分、最頻 0.8 分、最大 1 分）に従い、この時間にはカット作業も含まれる。箱詰めが完了すると注文完了となる。なお、顧客の受け取りやデリバリーのプロセスについては、読者への課題として残すことにする。

この店舗では、受注、ピザ作り、箱詰めの作業に従業員を**プール**制で配置している。つまり、全従業員が複数の業務をこなせ、いずれの作業も担当可能である。ただし、作成と箱詰めについては、作業を行うステーションに実際に配置される必要がある。一方、コードレス電話とノートパソコンのおかげで、注文受付は店内のどこからでも応対可能である（専用の受付カウンタは不要）。

図 9.32 は Model 9-2 の完成した Facility ビューを示している。Customer と Pizza のエンティティタイプを定義し、Floor Label でラベルをつけた。Floor Label ではモデルとエンティティの状態変数を表示でき、エンティティの移動に合わせて追従する。

エンティティの Floor Label を作成するには、（ラベルがエンティティに**付属**する性質上）まず Facility ビューでエンティティを選択し、Attached Animation リボンの Floor Label アイコンをクリックする。次に、Facility ビュー上でラベルを配置したい位置をクリックし、ドラッグしてサイズを調整した後、もう一度クリックして確定する。すると、ラベルの書式説明が書かれたデフォルトテキストのラベルが作成される。Appearance リボンの Edit アイコンをクリックすると、Pizza エンティティの Floor Label 編集ダイアログが表示される（図 9.33）。

このとき、カッコ内に OrderNumber エンティティ状態変数を使用することに注意する。ラベルを回転させる場合は、ラベルを選択して Ctrl キーを押しながら、いずれかのコーナー（緑色の円）をクリックする。円が赤色に変わり、Ctrl キー

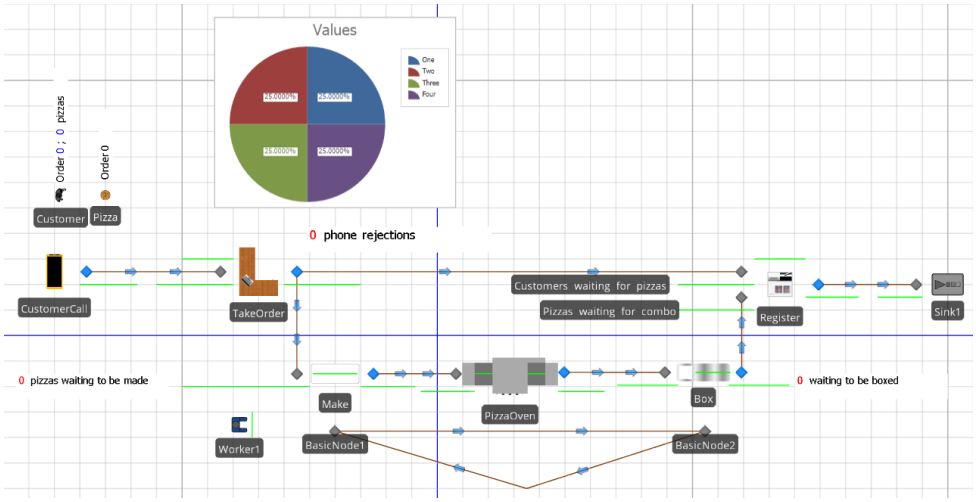


図 9.32　Model 9-2 の Facility ビュー

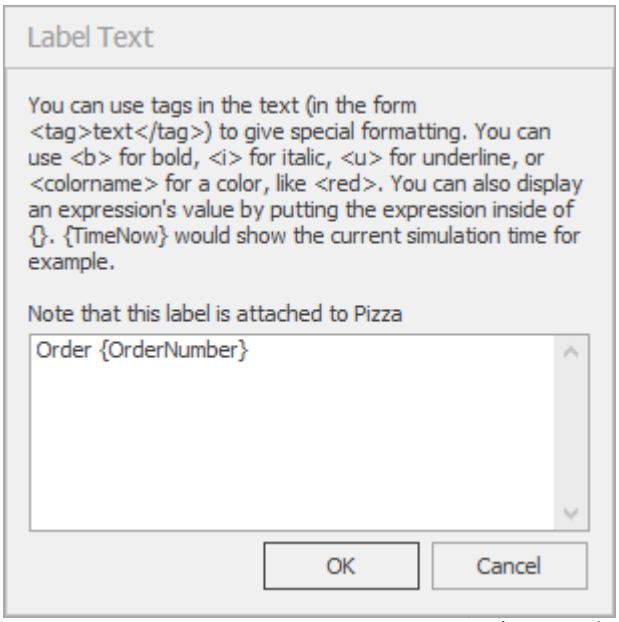


図 9.33　Pizza エンティティの Floor Label の編集ダイアログボックス

を押している間はラベルを回転できる。この回転方法は、Facility ビュー内のすべてのオブジェクトに適用できる。

モデルを実行すると、このプレースホルダの値はエンティティ状態変数値（ピザの注文番号）に変換される。モデルには、Source オブジェクト（CustomerCall）、受注（TakeOrder）、作成（Make）、箱詰め（Box）の各プロセス、そしてオーブン（PizzaOven）を表す Server オブジェクトが含まれている。また、Customer オブジェクトと Pizza オブジェクト（注文されたピザを表す）を結合する Combiner オブジェクト（Register）と、エンティティが離脱する唯一の Sink オブジェクトがある。さらに、Customer と Pizza エンティティの流れを示す Connector オブジェクトと、作成ステーションと箱詰めステーション間の従業員移動を表現す

るため、2 つの Node オブジェクト（BasicNode1 と BasicNode2）間に Path のループを設置している。現在の Facility ビューが図 9.32 とほぼ同じになるよう、オブジェクトの配置を調整しよう。

Source オブジェクトの CustomerCall は、前述の顧客着信過程に基づいて顧客の着信を生成する（Interarrival Time プロパティは Random.Exponential(5)分に設定）。モデルに Customer と Pizza の両エンティティタイプを設定するため、Source オブジェクトの Entity Type プロパティを Customer に指定する。顧客の電話は処理のために Server オブジェクトの TakeOrder へ送られる。ただし、システムですでに 4 人の顧客が通話中の場合、次の顧客は話し中音を聞いて電話を切ることになる。この容量制限はアドオンプロセスでモデル化する。顧客注文の処理前に二択の判断が必要なため、TakeOrder オブジェクトの Input ノードにアドオンプロセスを追加し、電話システムが容量超過の場合は通話を拒否する。このプロセスはエンティティがノードに入る際に実行されるよう、Entered トリガーを使用する。図 9.34 で Input_TakeOrder_Entered アドオンプロセスを確認できる。

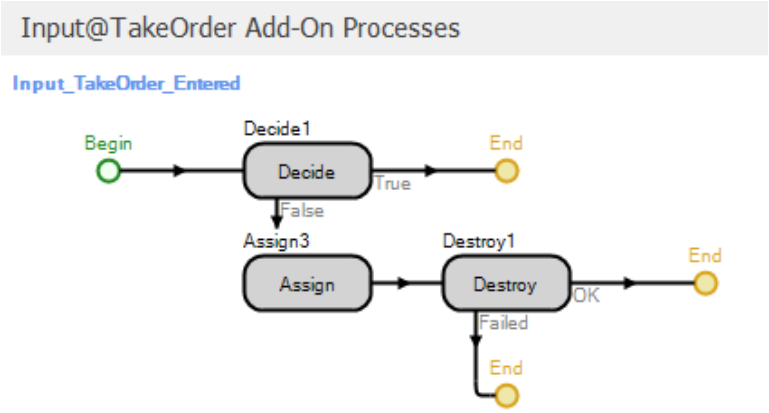


図 9.34　TakeOrder オブジェクトの入力ノードのアドオンプロセス

Decide ステップは、利用可能な電話回線の有無を判定する条件に基づいて判断を行う。電話回線の数は、TakeOrder オブジェクトの容量と対応する InputBuffer 容量を使ってモデル化される。図 9.35 は TakeOrder オブジェクトのプロパティを示している。Initial Capacity プロパティと Input Buffer プロパティはそれぞれ 1 と NumLines（プロパティ参照）に設定されていることに注意する。したがって、通話を受け入れるかどうかの判断には、Decide ステップで次の式を使用する：TakeOrder.InputBuffer.Capacity.Remaining > 0。この条件が真の場合、通話は TakeOrder オブジェクトに接続される。偽の場合は、拒否された通話数を追跡するためにユーザ定義のモデル状態変数 PhoneRejections をインクリメントし、エンティティを破棄する。TakeOrder オブジェクトプロパティと Input_TakeOrder_Entered アドオンプロセスにより、1 人の従業員が 1 件の注文を担当でき、プロパティ参照 NumLines の値によって「保留中」の顧客数が決まる。入

力バッファ容量にプロパティ参照を用いることで、さまざまな電話回線数でのテストが容易になる。このシステムの基本設定では、同時通話数を 4 件に制限するため、プロパティ参照値を 3 に設定する。

Properties: TakeOrder (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	**Random.Uniform(1.5,2.5)**
Off Shift Rule	Suspend Processing
Other Processing Options	
Buffer Logic	
Input Buffer	
Capacity	**NumLines**
Balking & Reneging Options	
Output Buffer	
Reliability Logic	
Table Row Referencing	
State Assignments	
Secondary Resources	
Financials	
Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	
Before Processing	**TakeOrder_BeforeProcessing**
Processing	
After Processing	**TakeOrder_AfterProcessing**
Exited	
Failed	

図 9.35　TakeOrder オブジェクトのプロパティ

顧客がシステムに入ると、従業員は電話に応対して注文処理を行う。従業員のプールをモデル化するため、Worker オブジェクト（Worker1）を使用する。図 9.36 は Worker1 オブジェクトのプロパティを示している。Worker オブジェクトはモデル内で物理的な位置を持ち、移動する従業員をモデル化できる（8 章参照）。以下では、作成ステーションと箱詰めステーションの位置を定義し、各従業員の位置を追跡する。まず、Initial Desired Speed プロパティを 1 メートル/秒、Initial Network を WorkerNetwork、Initial Node (Home) を BasicNode2 に設定する（WorkerNetwork は、箱詰めステーション下の BasicNode2 と作成ステーション下の BasicNode1 を結ぶ Path オブジェクトのループで定義されている）。実験での従業員数の調整を容易にするため、Initial Number In System プロパティにはプロパティ参照（NumWorkers）を使用している。

Properties: Worker1 (Worker)

Resource Logic	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Park While Busy	False
Travel Logic	
Initial Desired Speed	**1**
Initial Travel Mode	**Network Only**
Initial Network	**WorkerNetwork**
Network Turnaround Method	Exit & Re-enter
Free Space Steering Behavior	Direct To Destination
Routing Logic	
Initial Priority	1.0
Initial Node (Home)	**BasicNode2**
Idle Action	Park At Node
Off Shift Action	Park At Node
Transport Logic	
Financials	
Add-On Process Triggers	
Population	
Initial Number In System	**NumWorkers**
Maximum Number In System ...	2500
Advanced Options	
General	
Animation	

図 9.36　Worker1 オブジェクトのプロパティ

Model 9-1 と同様に、TakeOrder オブジェクトの Before Processing アドオンプロセスでは、Seize ステップを使って Worker1 リソースを確保する（図 9.35）。Worker1 リソースが利用できない場合、エンティティは Seize ステップで待機する。従業員は店内のどこでも注文を受けられるため、Worker1 オブジェクトを実際に移動させる必要はなく、単に受注タスクの割当用のリソースとして機能する。これは実際のピザ店で、従業員がコードレス電話とノートパソコンを使って店内のどこからでも注文を受けられる状況を反映している。ただし、Make ステーションと Box ステーションではこの限りではない。

Worker1 リソースが割り当てられると、顧客注文に対応する処理時間の間、エンティティを遅延させる。その後、対応する Pizza エンティティを生成して作成（make）ステーションへ送り、同時に Customer エンティティを送って完成した Pizza エンティティを待機させる。これらの処理は TakeOrder_AfterProcessing アドオンプロセスで実行する（図 9.37）。

通話終了時に実行されるこのプロセスでは、まず Worker1 リソースを解放する。続いて、最初の Assign ステップで以下の状態変数を割り当てる：

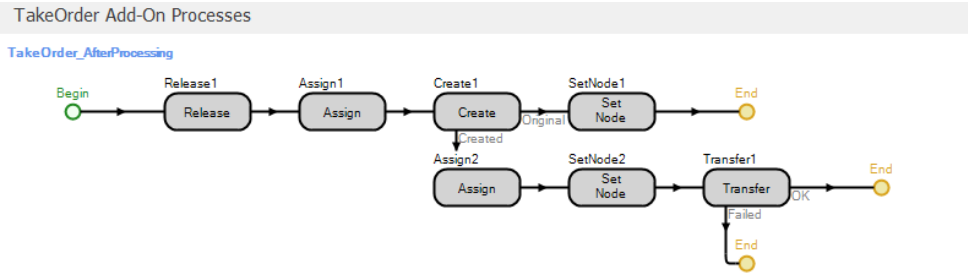


図 9.37　TakeOrder オブジェクトの AfterProcessing アドオンプロセス

- `HighestOrderNumber=HighestOrderNumber+1`：モデル状態変数の `HighestOrderNumber` を増加させ、新規注文の到着を反映する。
- `ModelEntity.OrderNumber=HighestOrderNumber`：Customer エンティティに現在の注文番号を割り当てる。
- `ModelEntity.NumberOrdered=Random.Discrete(1,.5,2,.8,3,.95,4,1)`：注文のピザ枚数を決定し、エンティティ状態変数 `NumberOrdered` に値を保存する。
- `NumberOrdered[ModelEntity.NumberOrdered]=NumberOrdered[ModelEntity.NumberOrdered]+1`：これは、注文のピザ枚数ごとの注文数を追跡するモデル状態変数を増加させる。図 9.38 に示す `NumberOrdered` モデル**状態変数**はサイズ 4 のベクトルで、4 種類の注文サイズそれぞれに対応する状態変数を持つ。シミュレーション中の任意の時点で、各値がそれぞれの注文サイズの総数を表す。図 9.39 は Status Pie チャートと、そのチャートに使用したプロパティおよびリピートグループを示している。

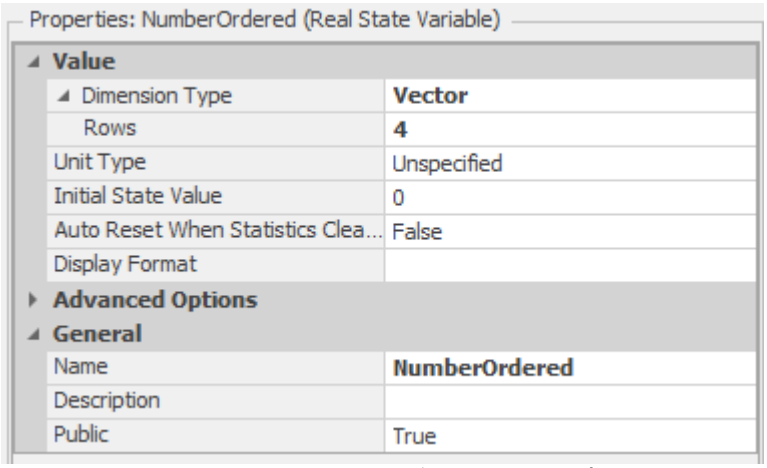


図 9.38　NumberOrdered モデル状態変数のプロパティ

Create ステップでは、NumberOrdered 状態変数を用いて生成する Pizza エンティティの数を決定する（Create ステップのプロパティは図 9.40 参照）。Create ステップの後、オリジナルのエンティティは Set Node ステップへ、新規生成された Pizza オブジェクトは Assign ステップへ移動する。Set Node ステップでは

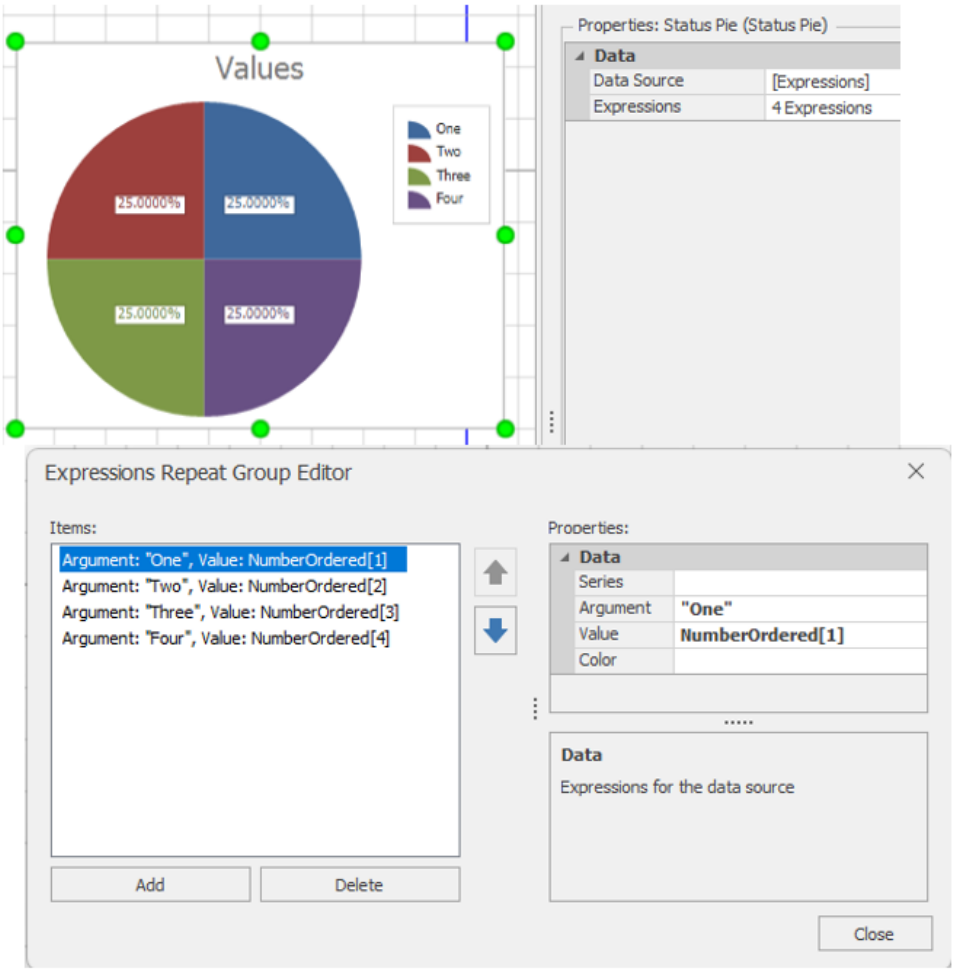


図 9.39　Status Pie チャートとそのプロパティおよびリピートグループ

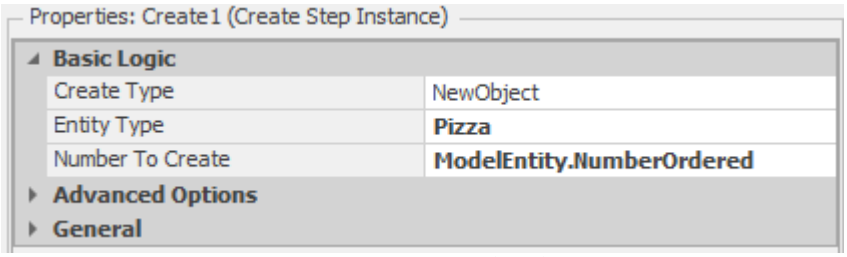


図 9.40　Create ステップのプロパティ

Node Name プロパティに ParentInput@Register を設定する。TakeOrder オブジェクトを出たオリジナルのエンティティは、注文した Pizza エンティティを待つため Register オブジェクトへ移動する（この Customer オブジェクトは前述の「顧客チケット」に相当）。Assign ステップでは HighestOrderNumber の値を ModelEntity.OrderNumber 状態変数に設定する。この OrderNumber 値はすでに Customer オブジェクトの状態変数にも設定されており、これによって Customer オブジェクトと対応する Pizza オブジェクトを関連づけることができ

る。新規生成された Pizza エンティティについては、SetNode ステップで Node Name プロパティを Input@Make に設定し、Make オブジェクトへ移動させる。Create ステップはフリースペースでオブジェクトを生成するため、最後に Transfer ステップで生成されたオブジェクトの配置場所を指定する必要がある。具体的には、Node Name プロパティで指定したノード（Output@TakeOrder）にオブジェクトを移動させる。以上により、Customer オブジェクトをレジスタで Pizza オブジェクトの完成を待たせ、Pizza オブジェクトを作成ステーションへ送り、エンティティ状態変数 OrderNumber を使って各注文に関連するオブジェクトを紐づけることができる。

Combiner オブジェクト Register は、顧客注文を表す Customer オブジェクトと、その注文を構成するピザを表す Pizza オブジェクトを結合する。図 9.41 は Combiner オブジェクトのプロパティを示している。Batch Quantity プロパティは、1 つの「親」に結合される「メンバー」の数を設定する。このケースでは、Customer オブジェクトが親となり、対応する Pizza オブジェクトがメンバーとなる。ModelEntity.NumberOrdered エンティティ状態変数を使って、1 つの注文に含まれるピザの枚数を指定する（同じ状態変数を使って Pizza オブジェクトの生成数も指定したことを思い出してほしい）。**Member Match Expression** と **Parent Match Expression** プロパティは、メンバーと親オブジェクトを正しく対応づけるための式を定義する。

Properties: Register (Combiner)

Batching Logic	
Batch Quantity	**ModelEntity.NumberOrdered**
Matching Rule	**Match Members And Parent**
Member Match Expression	**ModelEntity.OrderNumber**
Parent Match Expression	**ModelEntity.OrderNumber**
Batch Quantities (More)	0 Rows
Other Batching Options	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Parent Transfer-In Time	0.0
Member Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	0.0
Off Shift Rule	Suspend Processing
Other Processing Options	

図 9.41　Combiner オブジェクト Register のプロパティ

図 9.42 には、親待ち行列で待機中の「Order 3; 1 pizzas」というラベルの Customer オブジェクトと、メンバー待ち行列で待機中の「Order 3」というラベルの Pizza オブジェクトが示されている。Floor Label によって状況が一目でわかる点に注目してほしい。この例では、注文番号 3 は 1 枚のピザで構成され、現在箱詰めを待っている状態だ。Pizza オブジェクトが Combiner オブジェクトに

到着すると、Customer オブジェクトと結合され、注文が完了する。結合後、Customer オブジェクトは Combiner オブジェクトを出て Sink オブジェクトへ移動する。

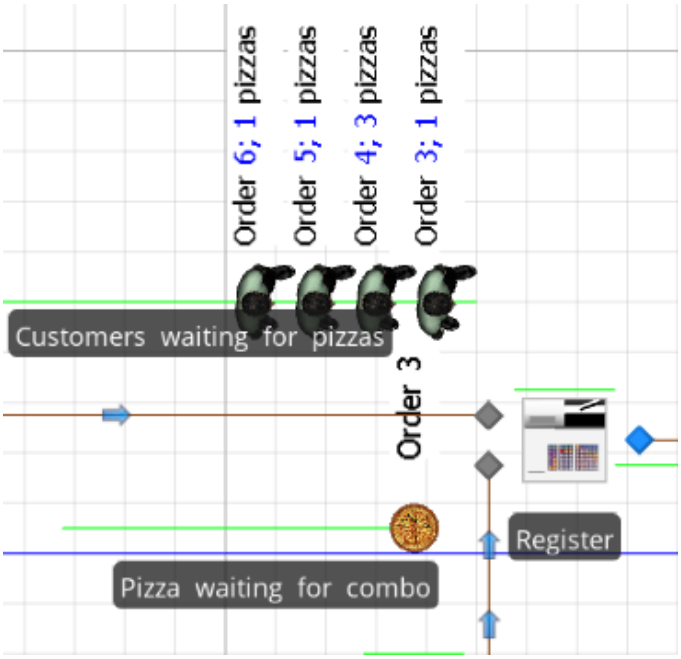


図 9.42　Combiner の待ち行列にある 4 つの親と 1 つのメンバー

Server オブジェクト Make は、ピザを作るステーションをモデル化する。作成ステーションでは 3 人の従業員が同時にピザを作れるため、Initial Capacity プロパティを 3 に設定する。Processing Time プロパティは Random.Triangular(2, 3, 4)分に設定する。作成プロセスには従業員が必要なため、BeforeProcessing アドオンプロセスで Worker オブジェクトを占有し、TakeOrder オブジェクトと同様に、AfterProcessed アドオンプロセスで従業員を解放する。

ここで重要な点は、作成プロセスでは従業員が作成ステーションに実際に存在している必要があることだ。そのため、Worker オブジェクトを BasicNode2（作成ステーションに指定された WorkerNetwork のノード）へ移動させる必要がある。この移動要求は、Worker オブジェクトの占有時に直接指定できる。

図 9.43 は、Make_BeforeProcessing アドオンプロセスの Seize ステップにおける Seizes プロパティエディタを示している。Object Name プロパティ（Worker1）の指定に加え、Request Visit プロパティを ToNode、Node Name プロパティを BasicNode1 に設定することで、オブジェクト占有時に BasicNode1 ノードへの移動を指示する。アドオンプロセスの実行時、Worker オブジェクトが BasicNode に到着するまでトークンは Seize ステップに留まる。

最後に、Server オブジェクト Box も同様に設定する（ただし、Processing Time プロパティは Random.Triangular(0.5, 0.8, 1)とし、Worker オブジェクトは BasicNode2 を訪問するよう指定）。

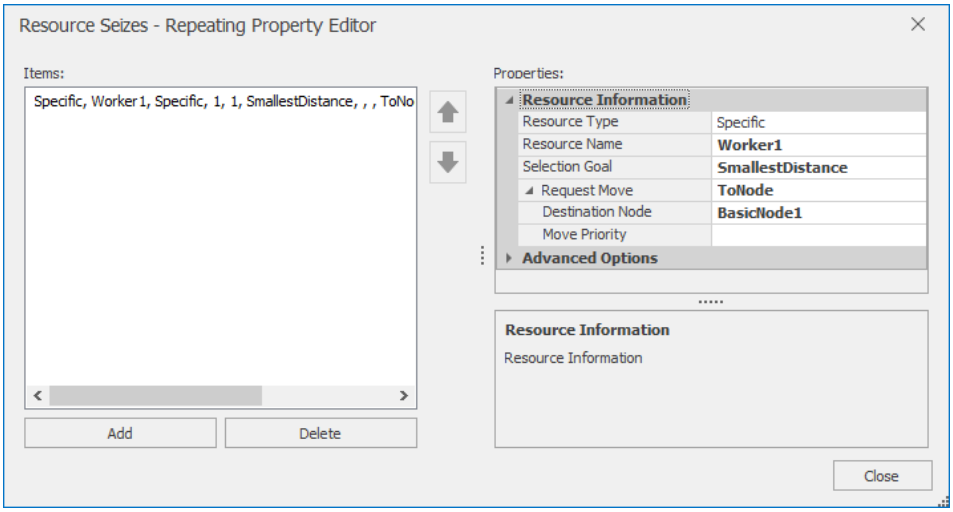


図 9.43　Make_BeforeProcessing アドオンプロセスの Seize ステップのプロパティ

これでモデルは完成だ。顧客からコールが着信したとき、電話回線が利用可能なら、最初に手の空いている従業員が電話に出て注文を受ける。その後、対応する Customer エンティティが注文に関連づけられた Pizza エンティティを生成する。Customer エンティティは注文を構成する Pizza エンティティを待つために送られ、Pizza エンティティは順に Make、Oven、Box ステーションで処理されるために送られる。すべての Pizza エンティティが Box ステーションでの処理を完了すると、注文に対応する Customer エンティティと Pizza エンティティが結合され、注文が完了となる。また、実験をサポートするため、電話回線数（NumLines）と従業員数（NumWorkers）の指定にはプロパティ参照を使用している。

9.2.1　Model 9-2 の実験

図 9.44 は、Model 9-2 で実施した 15 のシナリオ実験の結果を示している（実行期間は 1,000 時間で、ウォームアップ期間なし。なお、この実験条件の選定理由は章末問題 3 で検討する）。

Design　Response Results　Pivot Grid　Reports　Dashboard Reports　Input Analysis

Scenario		Replications		Controls		Responses		
Name	Status	Required	Completed	NumLines	NumWorkers	CustTIS (Minutes)	OvenUtil	RejectionsPerHr
Scenario1	Comple...	10	10 of 10	1	1	16.1354	17.2636	6.0479
Scenario2	Comple...	10	10 of 10	1	2	15.3736	27.5248	2.5825
Scenario3	Comple...	10	10 of 10	1	3	14.6413	31.3347	1.3276
Scenario4	Comple...	10	10 of 10	2	1	25.0372	17.8125	5.987
Scenario5	Comple...	10	10 of 10	2	2	20.2211	30.3132	1.5811
Scenario6	Comple...	10	10 of 10	2	3	16.8346	33.8416	0.4389
Scenario7	Comple...	10	10 of 10	3	1	30.7611	17.8408	5.8981
Scenario8	Comple...	10	10 of 10	3	2	23.564	31.5724	1.2082
Scenario9	Comple...	10	10 of 10	3	3	17.7569	34.4461	0.1712
Scenari...	Comple...	10	10 of 10	4	1	35.8611	17.8675	5.9157
Scenari...	Comple...	10	10 of 10	4	2	26.3616	32.2298	0.9639
Scenari...	Comple...	10	10 of 10	4	3	18.1772	34.8095	0.0782
Scenari...	Comple...	10	10 of 10	5	1	40.9335	17.8344	5.8857
Scenari...	Comple...	10	10 of 10	5	2	28.7535	32.7567	0.8081
Scenari...	Comple...	10	10 of 10	5	3	18.4538	35.0734	0.0365

図 9.44　Model 9-2 の実験結果

実験では、電話回線数（NumLines）と従業員数（NumWorkers）を決定変数とした。また、3 つのレスポンスを定義した：

- CustTIS：顧客注文のシステム内時間（分）。Customer.Population.TimeInSystem.Average で定義され、Unit Type を Time、Display Units を Minutes に設定することで、レスポンス値を分単位で表示する。
- OvenUtil：オーブンのスケジュール稼働率。PizzaOven.Capacity.Scheduled Utilization で定義される。
- RejectionsPerHr：毎時の顧客拒否数（電話回線容量制限による）。ユーザ定義の出力統計量 NumRejections で定義される。図 9.45 に示す NumRejections のプロパティでは、出力統計値を反復終了時に評価し、電話拒否の総数（モデル状態変数の PhoneRejections に蓄積）を現在時刻（TimeNow）で割ることで、この測定基準を算出する。

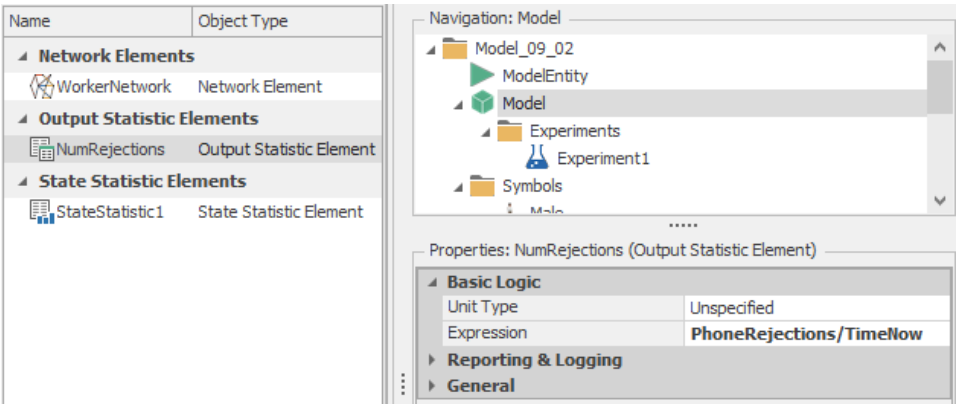


図 9.45　時間当たりの電話拒否数のユーザ定義出力統計量

設定を完了後、1～5 回線の電話回線と 1～3 人の従業員による合計 15 のシナリオを評価実験で検証し、CustTIS の値で昇順にソートした。表の最上位のシステムが、顧客のシステム内時間において「最適」な構成となる。また、（図 9.27 に示したように）手動でシナリオを列挙する代わりに、OptQuest アドインを使用して「最適」な構成を検証することも可能だ（9.1.1 節参照）。

図 9.46 は Model 9-2 での OptQuest の使用結果を示している。この最適化では、2 つの決定変数（Controls）と 2 つのレスポンス（Responses）を実験制御に使用した（より正確には、OptQuest に実験制御方法を設定した）。これらの決定変数とレスポンスは以下のように定義される：

- NumLines（決定変数）：利用可能な電話回線数。1 回線から 5 回線の範囲で指定する。
- NumWorkers（決定変数）：従業員数。1 人から 3 人の範囲で指定する。
- CustTIS（レスポンス）：顧客のシステム内時間。この値を**最小化**することで顧客満足度を向上させる。

Scenario			Replications		Controls		Responses		
	Name	Status	Required	Completed	NumLines	NumWorkers	CustTIS (Minut...	OvenUtil	RejectionsPerHr
☑	010	Comple...	5	5 of 5	1	3	14.6439	31.2271	1.3116
☑	007	Comple...	5	5 of 5	1	2	15.3813	27.5542	2.5664
☐	002	Comple...	5	5 of 5	1	1	16.1585	17.2474	6.0134
☑	013	Comple...	5	5 of 5	2	3	16.8476	33.8048	0.4386
☑	009	Comple...	5	5 of 5	3	3	17.774	34.57	0.1722
☑	005	Comple...	5	5 of 5	4	3	18.1168	34.6433	0.0742
☑	003	Comple...	5	5 of 5	5	3	18.3542	34.9592	0.0316
☑	004	Comple...	5	5 of 5	2	2	20.2666	30.3422	1.583
☑	001	Comple...	5	5 of 5	3	2	23.5946	31.5527	1.1712
☐	006	Comple...	5	5 of 5	2	1	25.1988	17.8297	5.9526
☑	011	Comple...	5	5 of 5	4	2	26.3501	32.1921	0.9188
☑	015	Comple...	5	5 of 5	5	2	28.7638	32.7664	0.796
☐	012	Comple...	5	5 of 5	3	1	30.6716	17.8428	5.9456
☐	008	Comple...	5	5 of 5	4	1	35.8458	17.8591	5.9242
☐	014	Comple...	5	5 of 5	5	1	41.0086	17.8304	5.8758

図 9.46　OptQuest による Model 9-2 の最適化結果

・RejectionsPerHr（レスポンス）：1 時間当たりの顧客電話拒否数。この値を 4 未満に抑える（基準値として設定）。このサービスレベル制約がない場合、システムは電話回線数を減らすことで顧客システム内時間を最小化する可能性がある。その場合、受注可能な顧客数が制限されてしまい、ピザ店の経営者の意図に反することになる。

図 9.47 は Model 9-2 の OptQuest を用いた実験の決定変数とレスポンスのプロパティを示している。決定変数（NumLines と NumWorkers）には最小値と最大値を、CustTIS レスポンスには目標（Minimize）を、RejectionsPerHr レスポンスには上限値（4）を設定している。結果（図 9.46）によると、1 回線・3 人の従業員の構成で最小の顧客システム内時間（14.6439）が達成され、時間当たりの拒否率も許容水準（1.3116）となっている。一方、1 回線・1 人の従業員の構成では、顧客システム内時間がやや長く（16.1585）、時間当たりの拒否数が制約を超過している（6.0134 ＞ 4）。この制約違反は、RejectionsPerHr セルが赤色でハイライトされて示されている。次の分析ステップとしては、Subset Selection Analysis や Select Best Scenario using KN を用いて「最適」な構成を特定することが考えられる。

Properties: NumLines (Control)

NumLines	
Description	
OptQuest for Simio - Parameters	
Include in Optimization	**Yes**
Minimum Value	**1**
Maximum Value	**5**
Increment	**1**

Properties: NumWorkers (Control)

NumWorkers	
Description	
OptQuest for Simio - Parameters	
Include in Optimization	**Yes**
Minimum Value	**1**
Maximum Value	**3**
Increment	**1**

Properties: CustTIS (Response)

OptQuest for Simio - Parameters	
Weight	**1**
General	
Name	**CustTIS**
Display Name	CustTIS
Expression	**Customer.Population.TimeInSystem.Average * 60**
Unit Type	Unspecified
Objective	**Minimize**
Lower Bound	
Upper Bound	

Properties: RejectionsPerHr (Response)

OptQuest for Simio - Parameters	
Weight	**1**
General	
Name	**RejectionsPerHr**
Display Name	RejectionsPerHr
Expression	**NumRejections.Value**
Unit Type	Unspecified
Objective	None
Lower Bound	**0**
Upper Bound	**4**

図 9.47　Model 9-2 における OptQuest 実験の決定変数とレスポンスのプロパティ

9.3　Model 9-3：固定容量バッファ

本章の最後のモデルは、ステーション間に**固定容量バッファ**がある組立ラインをモデル化するものだ。この**組立ライン**では、製品がステーション 1 から最終ステーションnまで連続的に移動する。ここでnはラインのステーション数を表す。ライン内の隣接するステーションは、一定数の製品を保持できる**バッファ**（**バッファ容量**と呼ばれる）によって区切られている。ステーション間にバッファがない場合でも、一貫性を保つためにゼロ容量のバッファとして扱う。ここでは、2 つの隣接するステーションiと$i+1$について考えてみよう（図 9.48）。

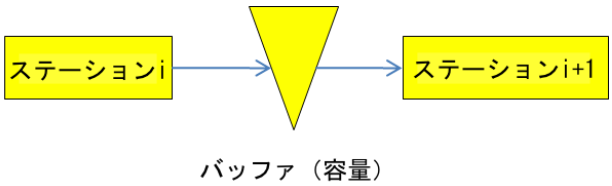


図 9.48　バッファにより分割された 2 つの隣接ステーション

ステーションiで作業が終わったとき、ステーション間のバッファが満杯（またはゼロ容量）の場合、ステーションiは**ブロック**され、次の製品の処理を開始できない。一方、ステーション$i+1$で作業が終わっていても、バッファが空の場合、ステーション$i+1$は**部品がない**（欠品）状態となり、同様に次の製品の処理を開始できない。いずれの場合も、ラインで生産ロスが発生する。処理時間の変動が大きい場合や、ステーションの信頼性が低い場合、バッファ容量を増やすことで、ステーションのブロックや欠品を大幅に減らし、プロセスのスループットを向上させることができる。**バッファ配置問題**とは、バッファをどこに配置し、どの程度の容量にするかを決定する問題である。Model 9-3 ではこのバッファ配置問題を扱う（具体的には、バッファ配置問題を解くためのスループット分析を提供する）。連続生産ラインとその分析についての詳細は Askin and Standridge（1993）に記載されているが、ここまでの説明で本題の理解にはじゅうぶんである。

Model 9-3 の目標は、特定のバッファ構成（ステーション間のバッファサイズの設定）におけるラインの最大スループット量を見積もることである。この文脈での**スループット**とは、単位時間当たりの生産製品数を指す。このモデルでは単純化のため、全ステーションが同一（つまり、処理時間の分布が同じ）で、隣接ステーション間のバッファもすべて同じサイズと仮定している。そのため、バッファ構成は単一の数値で表現できる。Model 9-3 は、Conway et al.（1988）が提示し、Askin and Standridge（1993）が解説した分析を再現するものである。図 9.49 に Model 9-3 の Facility ビューを示す。

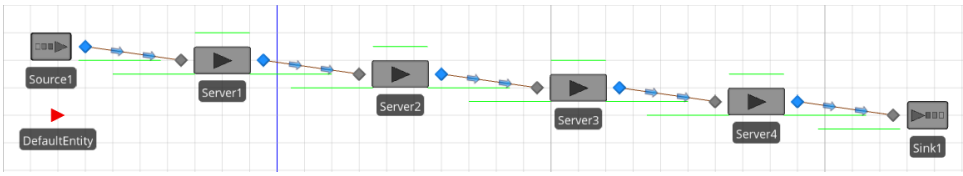


図 9.49　Model 9-3 の Facility ビュー

Facility ビューでは、Model 9-3 は、Model 5-1 の Source と Sink オブジェクト間に 3 つのステーションを追加したように見える。ただし、固定容量バッファを持つ組立ラインをモデル化するには、いくつかの重要な違いがある。以下では、これらの変更点を説明し、さらにプロパティ参照と Simio Response Charts を用いた実験とシナリオ比較の方法について解説する。

デフォルトでは、Connector や Path オブジェクトで 2 つの Server オブジェクトを接続すると、最初のステーションでの処理完了後、エンティティは自動的に 1 番目のステーションの出力ノードから隣のステーションの入力ノードへ移動する。2 番目のステーションが稼働中の場合、エンティティは待ち行列に並ぶ。2 番目のサーバが利用可能になると、待ち行列の先頭のエンティティが処理を開始する。この仕組みは、事実上、2 つのステーション間に無限容量のバッファを設定することになり、有限容量バッファによるブロッキング効果をモデル化できない。

ゼロ容量バッファ（つまりバッファなし）の単純なケースを考えてみよう。この場合、現在のリソースを解放する**前**に、次のリソースに利用可能な容量があることを確認する必要がある。つまり、エンティティの移動開始前に、移動先に空きがあるかを確認する。これは Kelton et al.（2015）では**重複リソース**として説明されている。有限容量バッファの場合も同様の課題があり、次のリソースではなく、バッファに残容量があることを確認する必要がある。

幸いなことに、Simio の Server オブジェクトでは重複リソースを簡単にモデル化できる。図 9.50 は Model 9-3 の Server1 オブジェクトのプロパティを示している。ここで重要なのは、Buffer Logic セクションの中の Input Buffer と Output Buffer プロパティである。Server オブジェクトの Input Buffer は、サーバリソースにじゅうぶんな容量がない場合にエンティティを保持する。同様に、Output Buffer は、処理完了後まだ移動先のないエンティティを保持する。これらのプロパティはデフォルトで `Infinite` に設定されているため、ここでは Input Buffer サイズには `BufferCapacity` というプロパティ参照を使用し、Output Buffer は `0` に設定する。全ステーションをこの構成にすることで、後続ステーションの入力バッファにある内部待ち行列のエンティティ数が BufferCapacity の値未満の場合のみ、エンティティを次の Server へ移動させることができる。また、Processing Time プロパティにはプロパティ参照 `ProcessingTime` を指定している。バッファ容量と処理時間分布にプロパティ参照を使用することで、さまざまなライン構成での実験が容易になる。なお、このモデルでは使用していないが、Simio の標準ライブラリオブジェクトは**退去**と**放棄**の多様なオプションをサポートしている。これらの機能は Balking & Reneging Options プロパティ（図 9.50）で設定でき、10.2 節の Model 10-3 で詳しく扱う。

モデルでラインの最大処理量を見積もるため、エンティティの到着過程を変更する必要がある。標準の Source オブジェクトは、到着時間間隔分布か到着スケジュールでエンティティを生成するが、この最大処理量モデルでは、最初のステーション（Server1）の前に常にエンティティが待機している状態を確保しなければならない。これは、ラインに原料が無限に供給される状態と同じだ。このロジッ

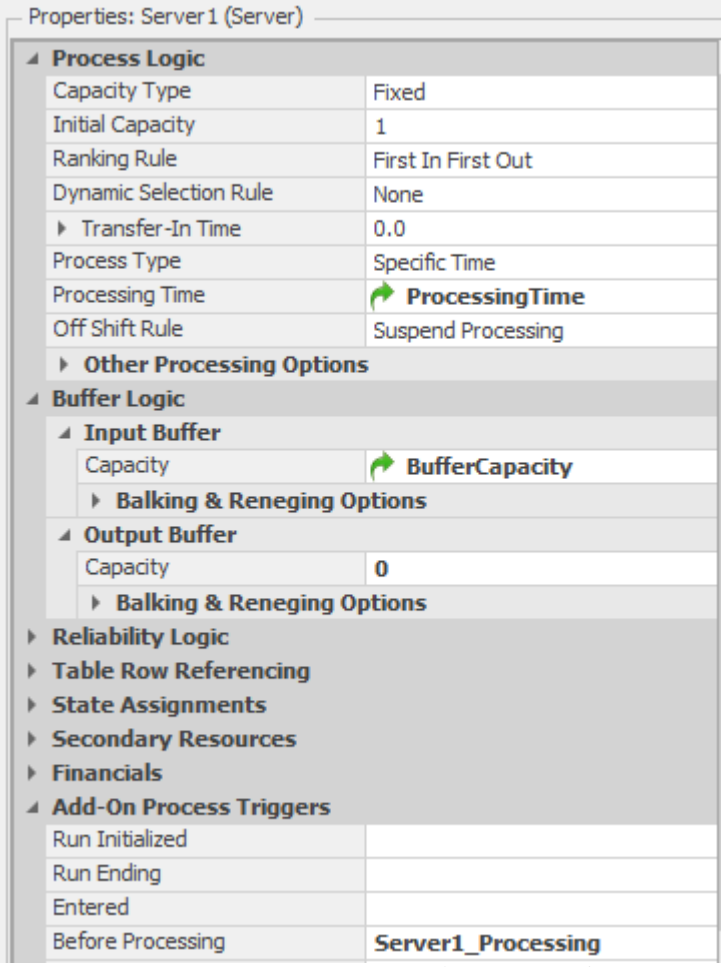


図 9.50　Model 9-3 の Server1 オブジェクトのプロパティ

クを実現するために、Source1 の Maximum Arrivals プロパティを 1 に設定し、時刻 0 で 1 つのエンティティを生成する。その後、Server1 での処理が完了するたびに、エンティティを複製して Server1 の入力に戻す。この複製と経路指定は、Server1 の Server1_Processing アドオンプロセスで実行する（図 9.51）。

Create ステップでは、エンティティを複製する。Create ステップの Created 出力からつながる Transfer ステップは、ノード Input@server1 にエンティティを移動させる。Create ステップの Original 出力がプロセスの End に向かうとき、オリジナルのエンティティは通常の経路を通る。Server2 または Server1 と Server2 の間のバッファに空き容量があれば、2 番目のステーションに移動する。空き容量がない場合は、Output Buffer 容量が 0 に設定されているため、Server1 に留まる。なお、最初のステーションでの無限の原料供給を維持する別の方法については、章末の問題 8 を参照してほしい。

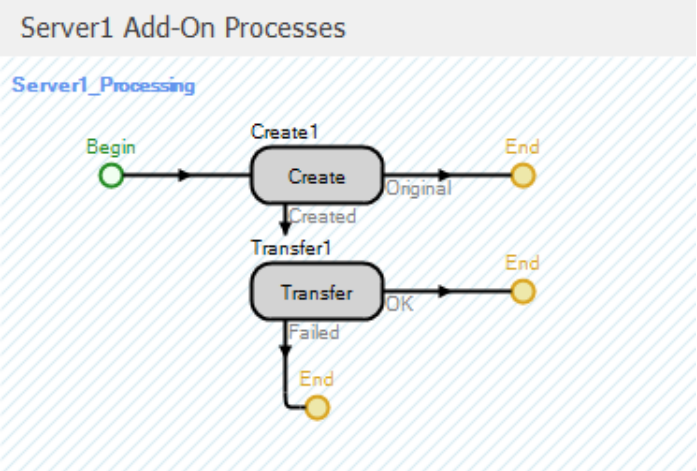


図 9.51　Server1 の Processing アドオンプロセス

初期モデルが完成したので、さまざまな構成での実験が可能になった。図 9.52 は Model 9-3 の実験を示している。ステーション処理時間（ProcessingTime）とバッファ容量（BufferCapacity）にプロパティ参照を定義したため、これらが Controls として実験に表示される。この実験では、処理時間が平均 10 分の指数分布に従い、バッファ容量を 0 から 10 まで 1 ずつ増やした 11 のシナリオを定義した。実行期間は 30,000 分で、ウォームアップ期間は 20,000 分である。各ステーションで平均 10 分の処理時間が必要な場合、10,000 分の実行時間で処理時間に変動がなければ、1,000 個の製品生産が見込める（1 時間当たり 6 個のスループット）。これを測定するため、TotalProduced レスポンスに DefaultEntity.Population.NumberDestroyed 式を使用し、Throughput レスポンスでは(Run.TimeNow-Run.WarmupPeriod)で除算する（なお、生産数の測定には Sink1.InputBuffer.NumberEntered も使用可能）。50 回の反復実行におけるシナリオのテスト結果では、平均スループットはバッファ 0 の 3.105 から、バッファ 10 の 5.231 まで変化した。

Scenario			Replications		Controls		Responses	
☑	Name	Status	Required	Completed	ProcessingTime (Minutes)	BufferCapacity	Throughput	TotalProduced
☑	Scenario1	Completed	50	50 of 50	Random.Exponential(10)	0	3.105	517.5
☑	Scenario2	Completed	50	50 of 50	Random.Exponential(10)	1	3.7962	632.7
☑	Scenario3	Completed	50	50 of 50	Random.Exponential(10)	2	4.19844	699.74
☑	Scenario4	Completed	50	50 of 50	Random.Exponential(10)	3	4.47384	745.64
☑	Scenario5	Completed	50	50 of 50	Random.Exponential(10)	4	4.66608	777.68
☑	Scenario6	Completed	50	50 of 50	Random.Exponential(10)	5	4.8198	803.3
☑	Scenario7	Completed	50	50 of 50	Random.Exponential(10)	6	4.94208	823.68
☑	Scenario8	Completed	50	50 of 50	Random.Exponential(10)	7	5.03688	839.48
☑	Scenario9	Completed	50	50 of 50	Random.Exponential(10)	8	5.10648	851.08
☑	Scenario10	Completed	50	50 of 50	Random.Exponential(10)	9	5.17008	861.68
☑	Scenario11	Completed	50	50 of 50	Random.Exponential(10)	10	5.23068	871.78

図 9.52　Model 9-3 の処理時間に指数分布を使用した実験

図 9.53 は、図 9.52 における実験の Response Chart を示している。図からは、バッファ容量の増加に伴ってスループットが向上する傾向が明確に見られるが、その増加率は次第に低下していく。

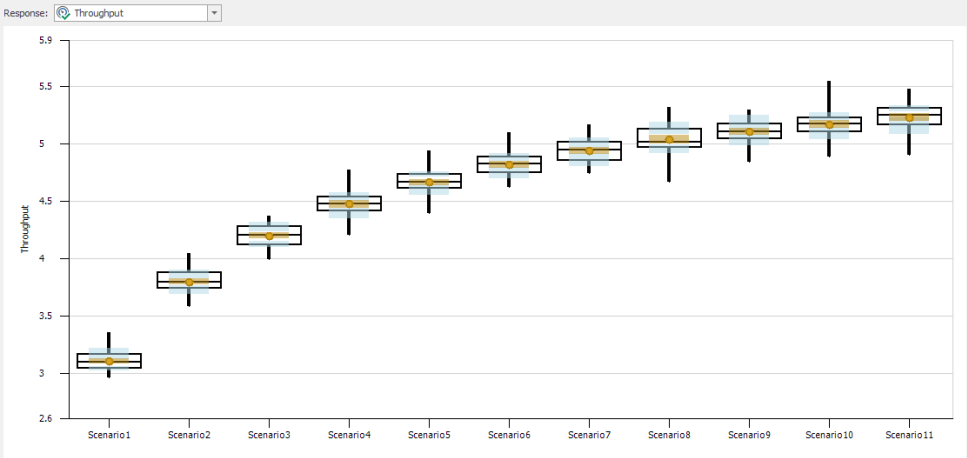


図 9.53　Model 9-3 の処理時間に指数分布を使用した実験のレスポンスチャート

以前に説明したように、Simio はリソースと Server オブジェクトの稼働・遊休状態を自動的に追跡する。具体的に、Simio は以下の状態変数を Server オブジェクトで、デフォルトで監視している：

・Starved（不足）：サーバが遊休状態でエンティティの到着を待機中。
・Processing（処理）：サーバが 1 つ以上のエンティティを処理中。
・Blocked（ブロック）：サーバが処理を完了したが、エンティティを解放できない状態。
・Failed（故障）：サーバが故障中。
・Offshift（オフシフト）：サーバがスケジュール設定中でオフシフト期間。

この連続生産ラインのモデルでは、最初のサーバに原料が無限に供給され、故障やスケジュール制約がないため、サーバは「処理」「不足」「ブロック」のいずれかの状態となる。これらのサーバの状態変数は、Simio の Status Pie 機能で視覚的に追跡できる（図 9.54）。

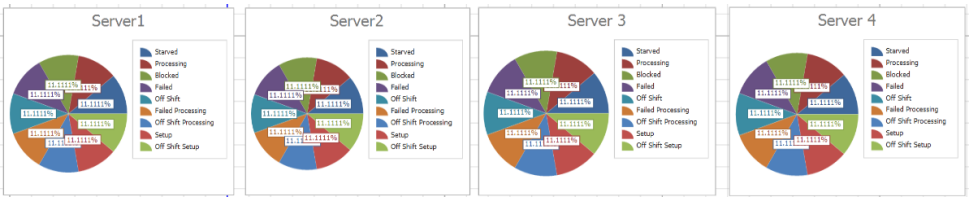


図 9.54　Model 9-3 のサーバ状態変数の円グラフ

Status Pie チャートは、各サーバがそれぞれの状態にあった時間の割合を表示する。図 9.55 は、新規チャート（左）、Server1（右上）、Server3（右下）の Status Pie チャートのプロパティを示している。新規チャートの Data Source プロパティのプルダウンからは、4 つのサーバの ResourceState リスト状態を選択できる。これらのデータソースを選択すると（Server1 と Server3 のチャートの例のように）、Expressions プロパティは自動的に非アクティブになる。

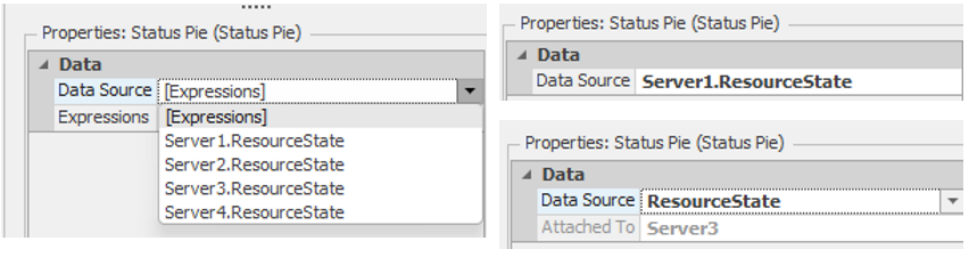


図 9.55　Status Pie チャートのプロパティ：新規チャート、Server1、Server3

2 つのチャートの主な違いは、Server3 のチャートは Server3 に**付属**しているのに対し、Server1 のチャートは付属していない点だ。付属アニメーションオブジェクトを作成するには、対象オブジェクトを選択し、通常の Animation リボンではなく Attached Animation リボンを使用する。この違いにより、Server3 のチャートは Facility ビュー上で Server オブジェクトとともに移動するが、Server1 のチャートは独立して配置されたままとなる。

ラインにおける最初のサーバは常に不足（Starved）状態にならず、最後のサーバは常にブロック（Blocked）状態にならないことに注意が必要だ。これは、最大処理量を算定するように設計されたモデルで期待される挙動である。2 番目と 3 番目のステーションを比較すると、2 番目のステーションではブロッキングが多く発生し、3 番目のステーションでは不足が多く発生する。この傾向は、より多くの工程を持つラインでも同様に続く。つまり、ラインの先頭に近いステーションではより多くの**ブロッキング**が発生し、ラインの末尾に近いステーションではより多くの不足（**スタービング**）が発生する傾向にある。

処理時間分布の指定にプロパティ参照を使用したため、モデルに戻ることなく実験で簡単に値を変更できる。図 9.56 は、実験の Controls セクションのプロパティ参照の値を変更し、処理時間を三角分布(5, 10, 15)分に設定した以外は、図 9.52 と同様の実験の Response Chart を示している。三角分布は指数分布と比べてばらつきが小さいため、三角分布を用いた場合はブロッキングと不足の影響が少なくなると予想される。また、バッファ追加によるスループット向上の効果がより早く現れる（つまり、少ないバッファ数でより高いスループットが得られる）。これらの分析結果は、Askin and Standridge（1993）と Conway et al.（1988）で説明された手法による結果と一致しており、予想通りの結果となっている。

9.4　まとめ

本章では、3 つのシステムを対象に、それぞれの Simio モデルに含まれる高度なシミュレーション概念を概観した。モデリングの側面に加え、各モデルの実験にも焦点を当て、OptQuest によるシミュレーションによる最適化のアドインを紹介した。ユーザが適切な目標、制約条件、必要条件を定義すると、OptQuest はシステムの最適構成に向けて Simio 実験を「駆動」する。OptQuest で特定された複数のシナリオから「最適」解を選択する手順を通じて、ランキングと選択のプ

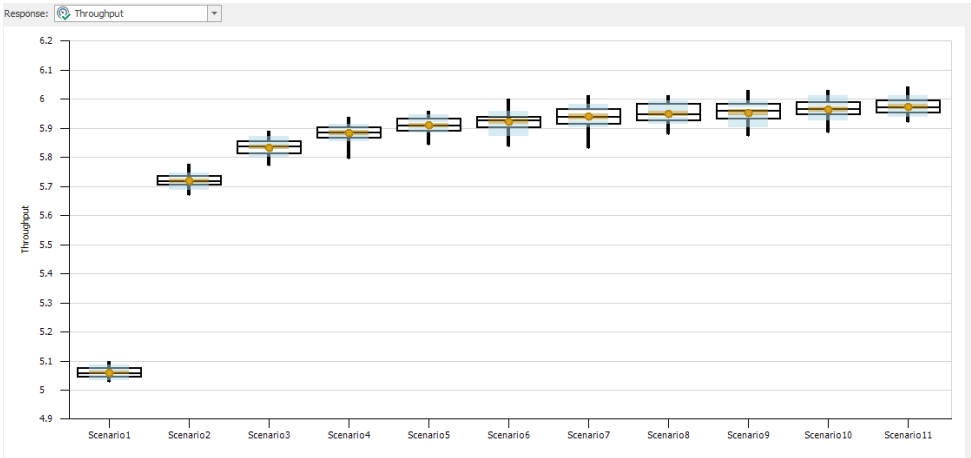


図 9.56　Model 9-3 の処理時間に三角分布を使用した実験のレスポンスチャート

ロセスを解説した。また、Simio の Subset Selection Analysis 機能と Select Best Scenario using KN の活用方法を示した（このランキングと選択のプロセスは、シナリオの特定方法によらず、あらゆるシナリオ集合に適用できる。すなわち、必ずしも OptQuest でシナリオを生成する必要はない）。これらのツールにより、一般的な出力分析のプロセスが大幅に簡素化される。

本書では、シミュレーションの一般的な応用と Simio の機能について表面的な紹介に留まる。これはシミュレーションと Simio の学習における終着点ではなく、むしろ出発点として捉えるべきだ。また、出力分析全般、とりわけシミュレーションによる最適化については入門的な解説に限定しているため、さらなる理解を深めるには、他の文献や実験が大いに役立つだろう。

9.5　問題

1．Model 8-3 を基に、Model 9-1 を修正して患者の物理的な移動と Nurse/Aide による付き添いの機能を追加しなさい。Model 9-1 とリソース要件を比較しなさい。このような詳細を追加することで、結果の妥当性は「向上」するか。その理由は何か。

2．Model 9-1 を修正して、Urgent 患者が Doctor リソースを優先的に使用できるようにしなさい。具体的には、Urgent 患者が到着した時点ですべての医師が診療中の場合、Urgent 患者は非 Urgent 患者から医師を優先的に確保できるようにする。モデルの結果を比較しなさい。この結果から、実際の救急診療システムで同様の変更を推奨できるか。

3．Model 9-2 では、ウォームアップ期間なしで 1,000 時間の実行期間を設定した（これらの設定については説明を省略した）。この設定は適切か。その理由は何か。より短い実行期間で同等の結果を得ることは可能か。実行期間を延長すれば結果は改善されるか。

4．9.2.1 項の Model 9-2 の実験（手動と OptQuest）では、最善なシナリオを選択する際に Subset Selection Analysis 機能や Select Best Scenario using KN アドインを使用しなかった。これらの機能を活用して、「最善」な構成を特定するために必要な反復実行回数を調整しなさい。

5．近年、多くのピザテイクアウト店が配達サービスを提供している。Model 9-2 を修正して顧客注文の配達機能を追加しなさい。顧客の 90%がピザの配達を希望すると仮定する（Model 9-2 と同様に、残り 10%は店頭での受け取り）。ピザ店は配達エリアを 4 つのゾ

ーンに区分けし、同一ゾーンからの注文であれば、配達員は複数の注文を同時に配達できる。表 9.3 は各ゾーンの配達時間分布と顧客の割合を示している。簡略化のため、配達員が複数の注文を配達する場合、各追加注文につき 4 分を加算する。たとえば、配達員が 3 つの注文を配達し、当初の配達時間が 8 分の場合、総配達時間は 8+4+4=12 分となる。顧客到着、処理時間、従業員の特性は Model 9-2 と同一とする。実験を設計し、適切な配達員の人数を決定しなさい。必ず解決策の根拠を示すこと（つまり、どのように配達員の適切な人数を決定したのかを説明すること）。

表 9.3　問題 5 の各ゾーンの顧客割合と配達時間分布（往復配達時間、分単位）

ゾーン	割合	配達時間分布
1	12%	Triangular(5, 12, 20)
2	30%	Triangular(2, 7, 10)
3	18%	Triangular(8, 15, 22)
4	40%	Triangular(6, 10, 15)

6. 問題 5 を修正し、配達を開始する前に、特定のゾーンに配達されるのを待っている注文が**少なくとも** 2 つになるまで配達員が待つようにしなさい。前のモデルと結果を比較しなさい。これは効果的な方針だろうか。また、少なくとも 3 つの注文を必要とする案についてはどうか。

7. Model 9-3 の組立ラインと同様の 6 つのステーションを持つ連続生産ラインを考える。Model 9-3 と同じく、各ステーションペア間には有限容量バッファがある（最初のステーションは常に不足せず、最後のステーションは決してブロックされないと仮定する）。現在のシステムとの違いは、各ステーションの処理時間分布が異なり、ランダムな故障が発生しやすい点である。表 9.4 に処理時間分布と故障データを示す（Station 5 の対数正規（Lognormal）分布パラメータ(4, 3)について、これは**生成された確率変量**の平均と標準偏差を表す。一方、Random.Lognormal()関数の引数は**基になる正規確率変量**の平均と標準偏差である。必要な変換については Simio ヘルプを参照）。連続生産ラインの Simio モデルを作成し、総バッファ容量 30 の最良な配置を見つけるための実験を行いなさい。総バッファ容量の配置例として、各ステーションペア間に 6 つずつ配置する方法がある。目標はラインスループットの最大化である。実験によって最適なバッファ配置を特定できるか。理由とともに説明しなさい。

表 9.4　問題 7 の処理時間と故障・修理分布（分単位、故障はカレンダ基準）

番号	処理時間	故障時間間隔	修理時間分布
1	Triangular(3, 6, 9)	Exponential(360)	Exponential(20)
2	Exponential(3)	Exponential(120)	Triangular(10, 15, 20)
3	Erlang(6, 3)	Exponential(300)	Exponential(5)
4	Triangular(2,7, 12)	N/A	N/A
5	Lognormal(4, 3)	Exponential(170)	Triangular(8, 10, 12)
6	Exponential(6)	N/A	N/A

8. 9.3 節で述べたように、Model 9-3 を変更して、Server1 オブジェクトに対応するアドオンプロセスではなく、Source オブジェクトの On Event Arrival Mode プロパティを使用して、最初のステーションで新しいエンティティを生成するようにしなさい。

9. 9.2.1 項の OptQuest ベースの実験では、サービスレベル制約を必要条件として顧客シ

ステム内時間を最小化したが、スタッフ配置費用は考慮していなかった。代替案として、顧客待ち時間、従業員の人件費、および顧客サービスを統合した単一の「総費用」測定基準を検討しなさい。この総費用測定基準を出力統計に追加し、OptQuest を使用して最適な構成を特定しなさい。

10. Model 9-2 では、受注、ピザ作成、ピザの箱詰めにおいて、**プール**型の従業員施策を採用している。これに対し、専門職型の従業員施策（各従業員が 1 つのタスクに専念する方式）を用いるモデルを開発し、Model 9-2 と結果を比較しなさい。どのような条件下で各施策が適切といえるか、実験結果に基づいて検討しなさい。

第 10 章　その他のモデリングトピック

本章の目的は、読者のモデリングスキルを向上させるモデリング概念と Simio の関連機能を解説することである。本章の各節は互いに独立しており、一貫した「中心的なテーマ」は存在しない。モデリングの必要性に応じて、必要な節を選択して読み進めてほしい。

10.1　Search ステップ

Search ステップは非常に柔軟な処理ステップで、モデル内のコレクションからオブジェクトやテーブル行を検索できる。**コレクション**とは、モデル内のオブジェクトの集合を指す。Entity Populations、Queue States、Table Rows、Object Lists、Transporter Lists、Node Lists などがコレクションの例である（すべての種類についてはヘルプを参照）。本節では、Search ステップの活用方法を 2 つの例題モデルで解説する。さらなる例は SimBit で確認できる（Simio の Support リボンの Sample SimBit Solutions から、キーワード「Search」で検索）。

10.1.1　Model 10-1：エンティティの検索・取り出し

このモデルでは、到着エンティティを「待機ステーション」で待機させ、定期的に取り出して処理を行う。この方法により、エンティティの到着プロセスと処理を効果的に分離できる。図 10.1 に完成したモデルを示す。

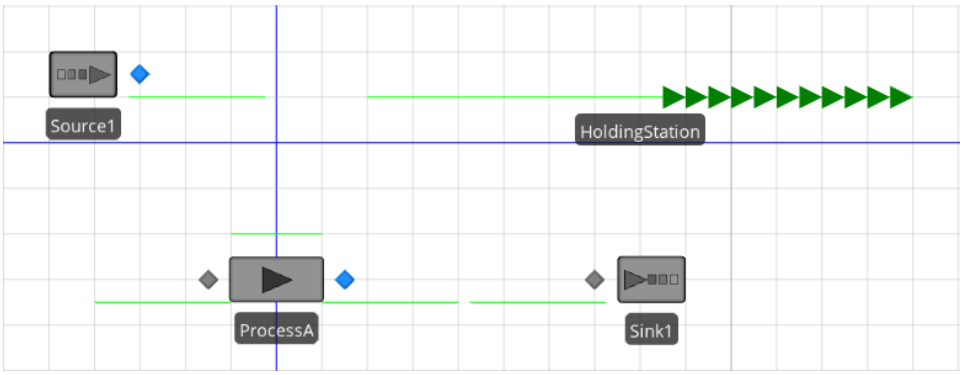


図 10.1　Model 10-1

待ち行列 Holding Station は、モデルで定義された Station 要素と対応している（図 10.2）。Facility View 上の待ち行列は、Animation リボンの Detached Queue ボタンで設置され、その状態名は HoldingStation.Contents となっている。エンティティは Source1 オブジェクトから、CreatedEntity アドオンプロセストリガーを使用してステーションへ搬送される（図 10.3）。

Process1 では、End Transfer ステップによってエンティティの搬送を完了する。このプロセスは、エンティティがステーションに到着したときに自動的に実行される HoldingStation.Entered イベントによってトリガーされる。

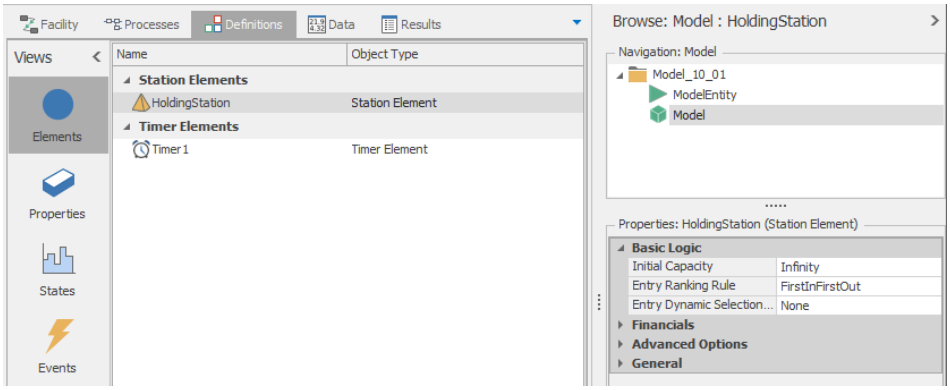


図 10.2　Model 10-1 の定義

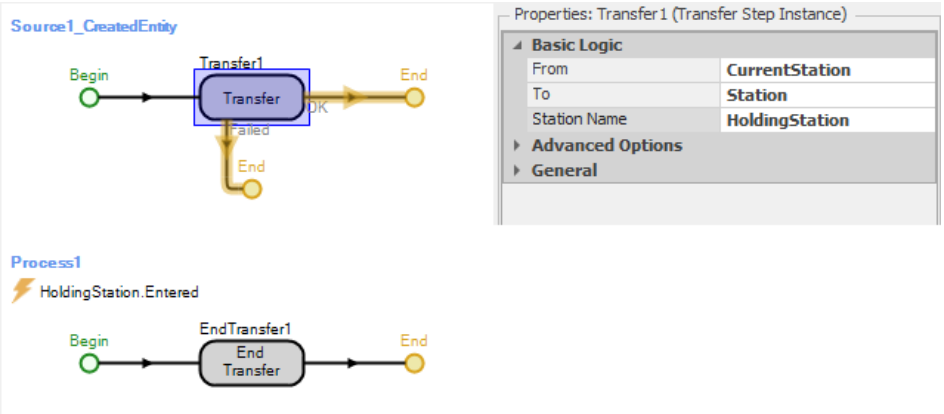


図 10.3　Source1 から HoldingStation までの 2 つの搬送プロセス

これまでに説明したロジックにより、生成されたエンティティは待機ステーションに直接搬送され、シミュレーション終了まで無期限に留まることになる。エンティティを定期的に待機ステーションから取り出して処理するため、タイマーで実行されるプロセスを使用する。このプロセスは待機ステーションを検索し、ProcessA の入力ノードにエンティティを搬送する。

図 10.4 に示すプロセス（PullEntities）では、Search ステップのプロパティとして、Collection Type に QueueState、Queue State Name に Holding Station.Contents、Limit に Infinity を設定する。これにより、Search ステップは待ち行列 Holding Station の全エンティティを検索し、見つかったエンティティを ProcessA の入力ノードへ搬送する。なお、このプロセスはタイマーイベント（Timer1.Event）によって実行される点に注意が必要だ（タイマーの定義は図 10.2 参照）。この例ではタイマーを使用してステーションからエンティティを定期的に取り出しているが、同じ Search ロジックは他の多くのプル・プッシュメカニズムにも応用できる。

このモデルは、到着プロセス（Source1）やタイマー間隔（Time1）を変更して実験することを推奨する。これでモデルへの影響を観察できる。また、Trace をオンにして実行すれば、モデル実行時の正確なイベントの順序を確認できる。

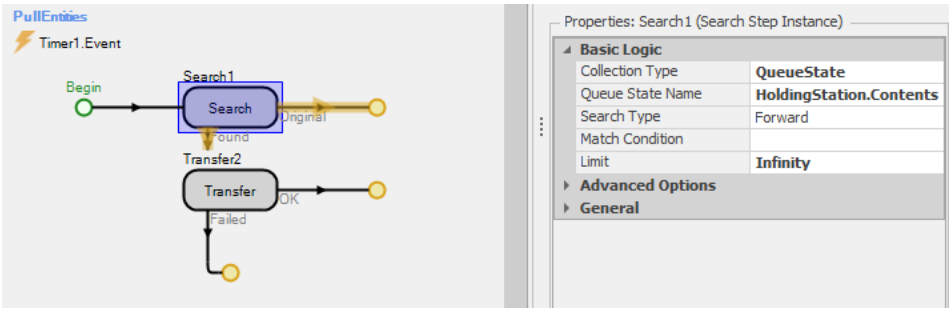


図 10.4　待機行列から ProcessA へエンティティを搬送する検索プロセス

10.1.2　Model 10-2：バッチ内の総処理時間の蓄積

この 2 つ目の例では、Search ステップを使用してエンティティをバッチにまとめ、バッチを構成する各エンティティの処理時間を蓄積することで、バッチ全体の処理時間を算出する。図 10.5 に完成したモデルを示す。Source1 はバッチの親エンティティとなるタイプ A のエンティティを生成し、Source2 はバッチのメンバーとなるタイプ B のエンティティを生成する。タイプ B エンティティの個別処理時間とバッチ全体の総処理時間（タイプ A エンティティ）を記録するため、ユーザ定義エンティティ状態 ProcessTime を使用する。図 10.5 に示すように、タイプ B エンティティの処理時間は三角分布（パラメータ 1, 2, 3）を用いてランダムに設定される。Source1 の到着間隔は平均 1 分の指数分布に従い、バッチの形成過程を視覚的に確認できるよう、エンティティインスタンス A の Batch Members 待ち行列にアニメーションを設定した。

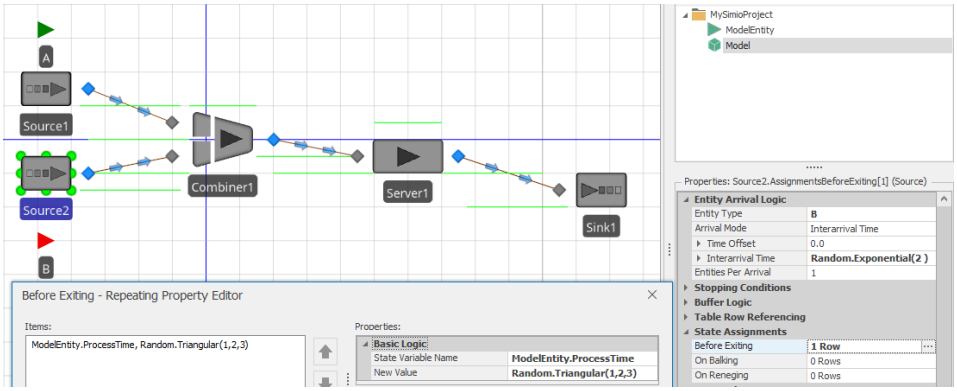


図 10.5　Model 10-2

オブジェクト Combiner1 は、タイプ A エンティティ 1 つと、タイプ B エンティティ 5 つをバッチにまとめる。Server1 でバッチの処理時間として利用するため、バッチ形成前にタイプ B エンティティに割り当てられている 5 つの値の合計を求める必要がある。このために、Combiner1 でバッチが形成される前に、Processing アドオンプロセストリガーを使用する。図 10.6 にこのプロセスを示す。このプロセスでは、Search ステップを使用してバッチ待ち行列（ModelEntity.BatchMembers）内のすべてのエンティティを検索し、Search Expression プロパティで指定した式（Candidate.ModelEntity.ProcessTime）の値を合計する。

合計値はトークンの戻り値として格納され、続く Assign ステップでバッチの状態変数 ProcessTime にこの合計値（Token.ReturnValue）を代入する。この状態変数は Server1 のプロパティ Processing Time で使用される。結果として、タイプ B の各バッチ構成要素は個別の処理時間を保持し、バッチ全体ではそれらの合計値を処理時間として持つことになる。

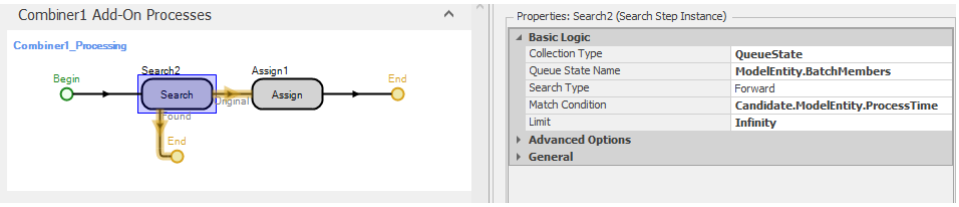


図 10.6　Model 10-2 における処理時間を蓄積するアドオンプロセス

Search ステップについて、最後に重要な注意点がある。Search ステップは非常に強力で柔軟な機能だが、過度な使用には潜在的な問題がある。たとえば、10,000 行のテーブルや 10,000 エンティティのコレクションから次のアクションを決定するために全件検索を行う場合、Simio では容易に実装できるものの、実行速度が著しく低下する。このような検索の過剰な使用は、パフォーマンス低下の主要な原因となりうる。

10.2　Model 10-3：退去と放棄

退去と放棄は、モデルを扱いにくくする一般的な待ち行列のふるまいである。**退去**（Balking）は、到着エンティティが待ち行列に入らないことを選択する際に発生する（たとえば、待ち行列が長すぎるため、システムに入らない潜在顧客）。**放棄**（Reneging）は、待ち行列内のエンティティがサービスを受ける前に離れることを選択する際に発生する（たとえば、列の進みが遅すぎるため、待っていた客が離れてしまう場合）。**ジョッキーイング**（Jockeying）、つまり待ち行列乗換は、顧客がある待ち行列から別の待ち行列に移動する際に発生する待ち行列のふるまいである。Simio では Standard Library オブジェクトのプロパティ Buffer Logic を使用して、退去と放棄の両方を効果的に実装できる。Model 10-3 は、4.4 節の ATM モデルを修正し、ATM 顧客の基本的な退去と放棄のふるまいを組み込んだものである。このモデルでは、待ち行列に 4 人の客がいる場合に新規客が退去し、また待ち行列での滞留時間が 3 分を超えると確率的に放棄する（これらの値は例示的なものであり、実際の適用時には状況に応じて調整が必要である）。

図 10.7 は、オブジェクト ATM1 のプロパティを表示した Model 10-3 を示している。このモデルには、2 つの Sink オブジェクトを追加した：退去するエンティティ用の Sink モジュール Balks と、放棄するエンティティ用の Sink モジュール Reneges である（各 Sink に入るエンティティ数を表示するステータスラベルも追加し、視認性を高めてもよい）。プロパティウィンドウでは、ATM1 オブジェクトの Input Buffer Capacity が 4 に設定され、Balk Decision Type が Blocked に設定されている点が重要である。これにより、エンティティが満杯のバッファ

に到着した場合、ネットワークでブロックされる（既定の動作）のではなく、Input@Balks に転送される。

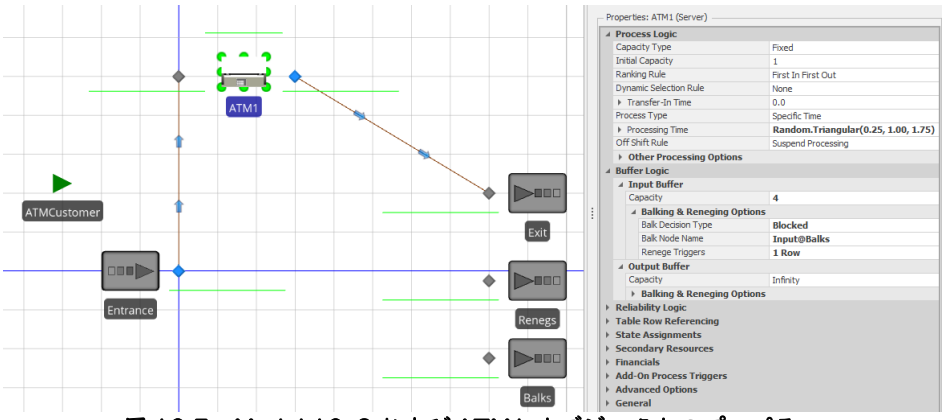


図 10.7　Model 10-3 および ATM1 オブジェクトのプロパティ

放棄のふるまいは、プロパティグループ Renege Triggers で設定する（図 10.8）。このモデルでは、エンティティが待ち行列に 3 分間滞留した後、0.5 の確率で放棄するように設定している。放棄を選択したエンティティは Input@Reneges（Sink の入力ノード）へ移動し、放棄しないと判定されたエンティティは待ち行列に残る。

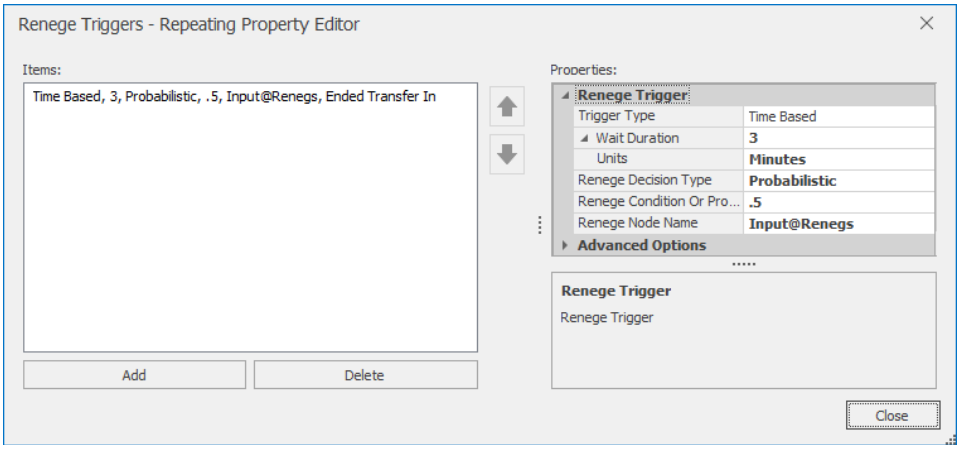


図 10.8　Model 10-3 の Renege Triggers プロパティ

Model 10-3 の退去および放棄のふるまいは基本的なものだが、同様の方法を応用することで、より複雑で包括的な退去、放棄、待ち行列乗換のふるまいを構築できる。他の Simio トピックスと同様に、退去や放棄の実装例として、以下のような SimBit Solutions がある：

- BalkingOnSourceBlockingOnServer：バッファ容量のない窓口での基本的な退去のシミュレーション。
- ChangingQueuesWhenServerFails：サービス窓口が故障した際に、顧客（エンティティ）が別の窓口の待ち行列に移動する。

・ServerQueueWithBalkingAndReneging：各エンティティが独自の最大待ち人数の閾値（退去用）と最大待ち時間の閾値（放棄用）を持つ。
・MultiServerSystemWithJockeying：エンティティが、自身が待機している待ち行列よりも短い待ち行列を見つけた場合、その短い方の待ち行列に移動（ジョッキーイング）する。

10.3　マテリアル要素とインベントリ要素

本節ではマテリアル（Material）要素とインベントリ（Inventory）要素を説明する。これらの要素は、材料の消費と生産が異なる場所で行われる場合や、部品表（Bill of Materials；BOM）構造でアイテムを表現するサプライチェーン型の事例で特に有用である。これらの要素には多くの機能とオプションがあるが、本節では基本的な紹介と、使用方法を示す SimBit リファレンスの提供にとどめる。

マテリアル要素は、物の数量（原材料、完成品、サブアセンブリなど）の追跡が必要であるものの、エンティティ関連の機能は不要な場合に使用する。マテリアル要素では、モデル開始時に**在庫数量**の初期値を設定し、実行中に材料を**生産**（在庫数量増加）または**消費**（在庫数量減少）できる。これらの操作は、Simio プロセスの Produce ステップと Consume ステップ、またはタスクシーケンスの一部として実行できる（10.4 節参照）。マテリアルの重要な機能は以下のとおりである：

・無限在庫量：マテリアル要素の Assume Infinite プロパティが False の場合、在庫数量が消費量を下回ると、プロセス実行（Consume ステップ）またはタスク実行（タスクシーケンス）がブロックされ、Produce ステップやタスクで必要数量が確保されるまでブロック状態が続く。一方、このプロパティが True の場合、マテリアル要素の在庫数量はマイナスになることが許可され、ブロックは発生しない。
・部品表：マテリアル要素は単一の材料、またはユーザ指定の数量で構成される他の材料からなる**部品表**（BOM）を表現できる。この機能は製造プロセスで特に有用である。部品表マテリアルが消費されると、その構成要素である個々のマテリアル要素が自動的に消費される。たとえば、製品組立では、単一の Consume ステップ/タスクで組立部品を消費し、Produce ステップ/タスクで完成品を生産できる。
・補充方針：マテリアル要素は、複数の標準的な**補充方針**をサポートしており、その方針に従って材料を自動的に補充できる（最小/最大、発注点/発注量、補充目標水準など）。これにより、モデル開発者は柔軟に補充方針を実装できる。
・ロケーションベース：マテリアル要素は、モデル全体に対して「グローバル」に設定することも、インベントリ要素（以下参照）を用いてモデル内の特定のオブジェクト（Server や Node など）に関連づけることもできる。このロケーションベースの機能により、サプライヤから倉庫、製造施設、小売店といった各拠点間での材料の移動を容易にモデル化できる。

インベントリ要素は、特定のマテリアル要素を格納する「バケット」として定義され、モデルオブジェクトに関連づけられる。マテリアルがロケーションベース

でない場合、「グローバルバケット」に対して生産・消費が行われる。一方、ロケーションベースの場合は、特定のインベントリに対して生産・消費が行われる。たとえば、サプライチェーンモデルでは、サプライヤ、倉庫、製造施設それぞれのオブジェクトに個別のインベントリ要素が関連づけられる。サプライチェーンでのマテリアルの「移動」は、あるインベントリ要素から消費し、次のインベントリ要素で生産することで実現される。サプライヤから倉庫へ、さらに倉庫から製造施設へと、この消費と生産の連鎖で表現される。

マテリアル要素とインベントリ要素の詳細な情報と実践例は、以下の SimBit モデルで確認できる：

- SimBit：Server With Material Consumption はマテリアルの基本的な使用方法を説明する。
- SimBit：Inventory and Materials はマテリアルとそれに対応するサイトベースのインベントリ要素の使用方法を説明する。
- SimBit：Inventory Replenish は在庫補充の使用方法を説明する。
- SimBit：Constraint Logic はマテリアル要件の制約が満たされていない場合にエンティティを保持するための Constraint Logic 要素の使用方法を説明する。

10.4　タスクシーケンス

4 章で Server オブジェクトを最初に紹介して以来、プロパティ Process Type を `Specific Time` に設定したままで説明してきた。これは単一の処理時間を持つ、単一工程の処理を表している。では、複数の工程が連続的に実行されたり、並列で行われたり、あるいは条件によって処理が分岐するような、より複雑なケースはどのように扱えばよいのか。このような場合に対して、Simio はタスクシーケンスという解決策を用意している。

タスクシーケンス（Task Sequences）は、タスクを処理する順序を構造的に定義し実行するものである。もっとも簡単な利用方法は、Server の Process Type を `Task Sequence` に設定することで、これにより処理する Tasks を指定する Processing Tasks リピートグループが表示される。この機能は Combiner や Separator オブジェクトでも使用でき、Task Sequences 要素として直接利用することも可能である。Task には、単純な処理時間の設定から複雑なオプションまで含めることができる。また、単一のタスクだけでなく、複数のタスクを組み合わせることもできる。複数のタスクを使用する場合は、直列実行、並列実行、条件分岐、確率的実行、およびこれらの組み合わせによって、さまざまな処理を表現できる。

それでは、簡単なシナリオで説明する。2 工程からなる塗装作業を考える。第 1 工程の Prepare は 0.5 分から 1.5 分の一様分布に従う時間がかかる。第 2 工程の Paint は三角分布に従い、最小 0.5 分、最頻 1.0 分、最大 2.0 分の時間を要する。部品は平均 2.5 分の指数分布に従う間隔でシステムに到着する。このシステムは 2 つのサーバで簡単にモデル化できるが、すべての作業を 1 か所で行うものとして、1 つのサーバで 2 工程を表現することにする。

・Source、Server、Sink を配置し、Source のプロパティを設定する。
・Server の Process Type を Task Sequence に設定する。
・Processing Tasks をクリックして Tasks リピートグループを開く。
・最初のタスクを追加し、Name を Prepare、Processing Time を Random.Uniform(0.5, 1.0)に設定する。他のフィールドはデフォルト値のままにする。
・次のタスクを追加し、Sequence Number を 20 に設定する（これにより、最初のタスク「10」の後に実行される）。Name を Paint、Processing Time を Random.Triangular(0.5, 1.0, 2.0)に設定する。他のフィールドはデフォルト値のままにする。
・図 10.9 と一致することを確認する。

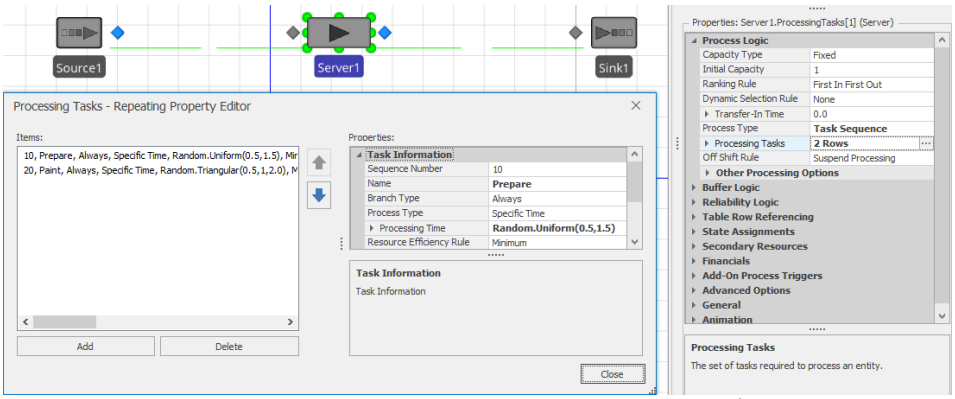


図 10.9　Model 10-4 の Task Sequences リピートグループ（初期設定）

現時点でモデルを実行して動作を確認できるが、正確に動作しているかの判断は容易ではない。Trace ウィンドウの Step 機能を使用して、システム内を移動する特定のエンティティの挙動を詳しく観察することを勧める。

モデルをさらに拡張して、より多くの機能を学び、モデルの挙動をより明確に理解しよう。このモデルでは、Source の近くにあるホームノード BasicNode1 に Prepper という作業者を配置する。また、Sink 近くのホームノード BasicNode2 には Painter という 2 番目の作業者がいる。両作業者は、作業していないときは各自のホームノードに待機する。さらに、エンティティが Server で処理される際の待機場所として、その真上に 3 つ目の BasicNode（WorkPlace）を設置する。

Server の Task Sequences リピートグループを開き、先ほど説明したリソース定義に類似した Resource Requirements セクションを確認する。2 つのタスクを修正して、適切な作業者の割当と WorkPlace ノードでの作業実行を設定する（図 10.10 に 1 つ目のタスクを示す）。モデルを実行すると、Prepper が Prepare タスクのために左側から、続いて Painter が Paint タスクのために右側から作業場所に向かう様子を確認できる。

Resource Requirements を指定したのと同様に、同じタスクリピートグループで Material Requirements（10.3 節参照）も設定できる。ここで、Materials の生産もしくは消費、あるいは Bill of Materials（たとえば、他の原材料を構成するために必要な原材料のリスト）の生産もしくは消費を選択できる。

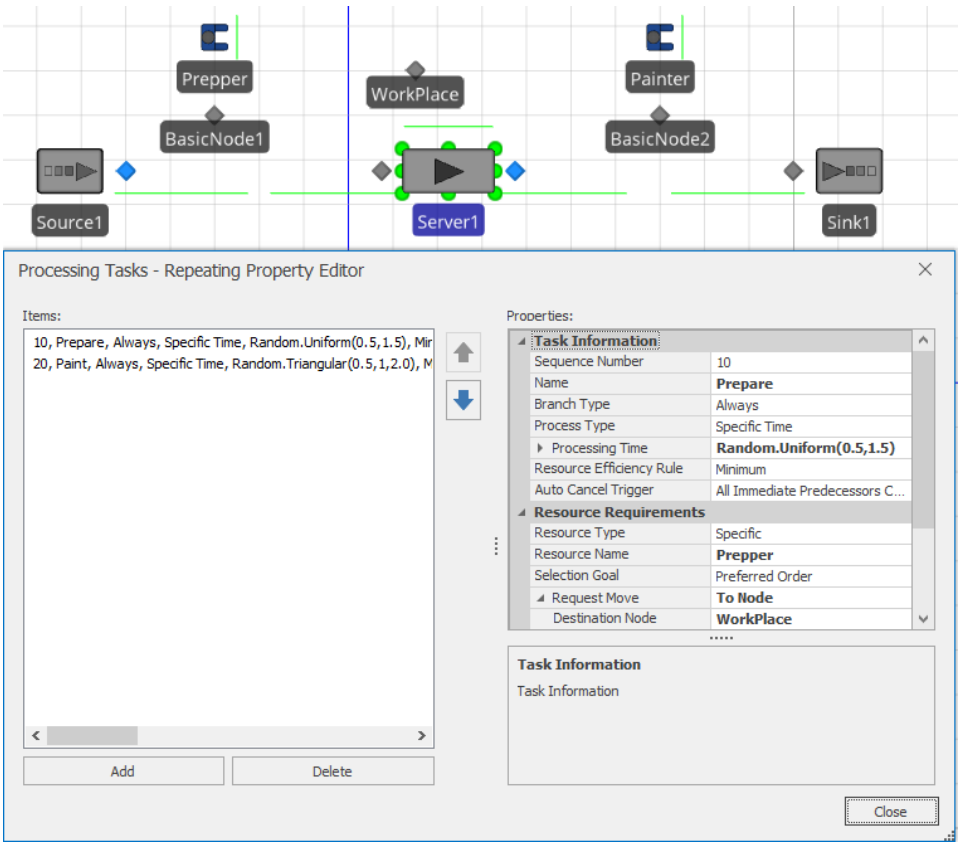


図 10.10　Model 10-4 の Task Sequences リピートグループ（完成）

すべてのタスクには、実行条件を決定するための Branch Type を設定できる。上記の例では、既定値の Always を使用した。これは文字通り、常に両方のタスクを実行することを意味する。その他の選択肢として、Conditional（条件）、Probabilistic（確率）、Independent Probabilistic（独立確率）がある。たとえば、Prepare タスクに分岐タイプ Probabilistic を設定すれば、45%の確率でのみ実行されるように指定できる。ここで注意したいのは、タスクの実行判断には他の要素も関係する点である。特に重要なのが、先行タスクの取り消し状態である（例：Prepare が不要な場合、Paint は実行すべきか）。プロパティ Auto Cancel Trigger は、既定では直前のすべての先行タスクが取り消された（All Immediate Predecessors Cancelled が選択された）場合にタスクを取り消すが、アクションを None に設定することで、タスクを継続することも可能である。

複数のタスクを使用する場合、その処理順序を指定する必要がある。上記のモデルでは、連続するタスク番号（10 および 20）を使用した。比較的単純な順序づけではこの方法がもっとも扱いやすい。ただし、Server の Other Task Sequence Options には、Task Precedence Method という別の選択肢もある。すでに使用した Sequence Number Method の他に、Immediate Successor と Immediate Predecessor が選択可能である。これら 2 つの方法を使えば、図 10.11 に示すような複雑なシーケンス図もより簡単に作成できる。

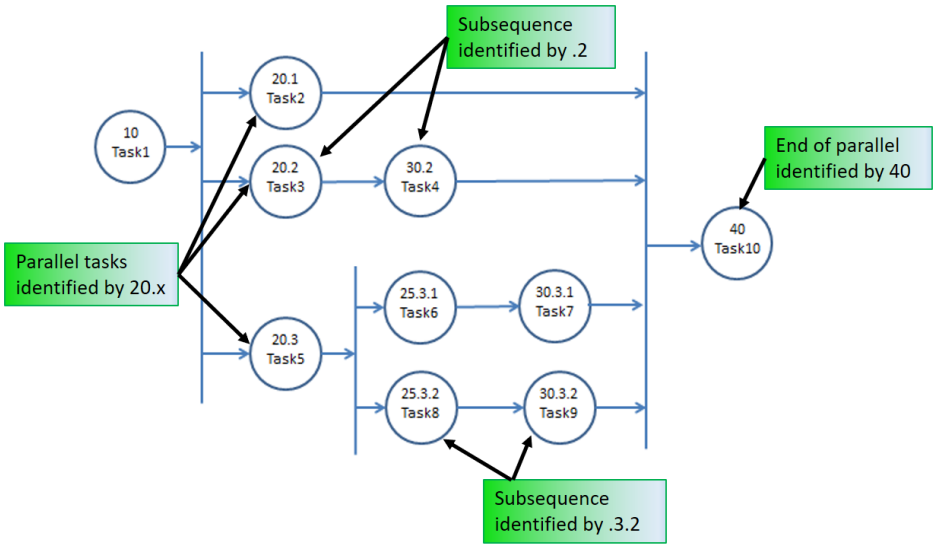


図 10.11　順序番号による複雑なタスクシーケンスの例

以前 Server を解説した際、プロパティ Process Type の説明を省略し、Specific Time のみを扱った。これは、Task Sequences の理解が前提となるためだ。ここまで学習してきたので、そのプロパティについて考えてみる。順序内の各タスクには、これまで説明を省いてきた Process Type が存在する。詳細な説明は割愛するが、タスクは単純な処理時間だけでなく、より複雑な機能を持つことができる。たとえば、Process Name を選択すると任意の長さのプロセスを実行でき、Submodel ではエンティティの複製をサブモデル（オブジェクトのセット）に送って完了を待つことができる。また、Sequence Dependent Setup では、Changeover Logic 要素で指定されたデータに基づき処理時間を自動計算できる。

タスクシーケンスに関する重要な注意点として、1 つのステーションで数百に及ぶような大量のタスクが存在する場合がある。たとえば、航空機の組立では、ある地点で数日間にわたり、数十人の作業員が数百のタスクを実行し、その後別の地点に移動して同様の作業が続く。このような場合、上記のようなタスクシーケンスの手入力は非常に非効率的だ。そこで、7.7 節で説明した**データ駆動型**モデルが有効である。Server プロパティの Other Task Sequence Options には、データ駆動型のタスクシーケンスをサポートする機能が用意されている。

さらに詳しい情報は、次のモデルを検討してほしい：

- SimBit：ServerUsingTaskSequenceWithWorkers は、複数の作業者による直列 4 タスクの実行方法を説明する。
- SimBit：TaskSequenceAndWorker は、移動する作業者とタスクの同期方法を説明する。
- SimBit ： ServerUsingTaskSequenceAlternativeMethodsForDefiningTaskPrecedence は、タスク間の先行関係を定義する代替手法を説明する。
- SimBit：ServersUsingTaskSequenceWithDataTablesJobShop は、データ

テーブルの活用方法を説明する。
・例題：HealthCareClinic は、関係データテーブルを用いてすべてのタスクのデータを指定する方法を説明する。

10.5　イベントに基づく決定ロジック

5.1.1.1 で学んだとおり、イベントはオブジェクト間の通信のために発生するメッセージである。イベントは、重要な事象の発生を他のオブジェクトに通知したり、カスタムロジックを組み込んだりするために活用される。このトピックについて、さらに詳しく見ていこう。

Simio でイベントを生成する方法は複数あるが、多くのイベントは自動的に定義・実行される。たとえば、エンティティがノードに出入りするたびにイベントが生成される。ほとんどのモデルでは、大部分のオブジェクトが入口・出口のノードを持つため、重要な状態変化を容易に検知し、その発生を伝えることができる。また、Standard Library オブジェクトの多くは、選択したイベントに応答できる。たとえば、ジャストインタイム方式でシステム内にパレットを到着させたい場合、Source の Arrival Mode を使用して、パレットが指定ノードを離れるときのみオブジェクトを生成できる。詳細は SimBit の RegeneratingCombiner を参照してほしい。

Standard Library オブジェクトには数多くの自動イベントが実装されている：

・エンティティが移動を開始または終了したとき、および破棄されたとき。
・リソースが割当または解放されたとき、容量が変化したとき、および故障または修理完了したとき。
・サーバの容量が変化したとき、および故障または修理完了したとき。
・作業者または車両がノードに進入または離れるとき、搬送を開始または終了するとき、割当または解放されたとき、容量が変化したとき、および故障または修理完了したとき。

同様に、Standard Library オブジェクトの多くのアクションはイベントによって実行される：

・ソースがエンティティを生成する（前述のとおり）。
・ソースがエンティティの生成を停止する。
・サーバ、リソース、コンベヤ、車両が故障状態になる。
・車両または作業者が休止状態を終了する。

Standard Library オブジェクトのプロパティと自動イベントを組み合わせることで、多くの動的な意思決定が可能となる。しかし、その詳細に入る前に、もう一歩踏み込んで学習する。これらの自動イベントに加えて、独自のイベントを定義することができる。要素やステップを使用して、柔軟な方法でこれらのイベントを実行できる。要素とステップは、Standard Library と同じようにイベントを実行・応答するが、より高度な柔軟性と制御性を備えている。また、独自の**ユーザ定義**イベントは Definitions タブの Events ウィンドウで定義できる。では、独自

のイベントを実行（発火）するための方法をいくつか見ていく：

・Station 要素：容量制限のある場所を定義し、離散的な品目を表すエンティティを保持するために使用される。Entered、Exited、CapacityChanged の各イベントを生成する。
・Timer 要素：指定された IntervalType に基づいてイベント系列を発火する。タイマー満了時に Event という名称のイベントを生成する。また、外部イベントに応答して次回のタイマーイベントを発火したり、タイマーをリセットしたりすることもできる。
・Monitor 要素：離散値の変化や、指定した状態変数（または状態変数群）の閾値超過を検知するために使用される。監視対象の状態変数が変化するたびに Event という名称のイベントを生成する。
・Fire ステップ：ユーザ定義イベントを発火するために使用される。

上述のとおり、Standard Library オブジェクトの多くは、イベントを受信したときに動作するプロパティを持つ。ただし、これらの定義済みの反応だけでなく、独自の反応も定義できる。この独自の反応を実装するには、プロセス内で Wait ステップを使用するか、プロセスそのものを発火させるかの 2 つの方法がある。

これまで使用してきたプロセスは、Simio によって自動的に開始（発火）されるか（例：OnRunInitialized）、プロセス内で定義されたオブジェクト（例：アドオンプロセス）によって開始されていた。しかし、プロセスを開始する別の有用な方法として、Triggering Event Name プロパティで特定のイベントを予約する方法がある。このプロパティに指定したイベントを受信すると、プロセスが開始され、トークンが生成されてプロセスの実行が始まる（Triggering Event Condition を設定した場合、プロセス開始前に条件が True（真）と評価される必要がある）。また、Subscribe ステップを使用すると、同様の関係を動的に確立でき（主に特定のエンティティとの間で）、Unsubscribe ステップでその関係を動的に解除できる。

イベントへの反応をより細かく制御するには、Wait ステップを使用する。トークンが Wait ステップに到達するまで、そのステップは非アクティブ状態（すべてのイベントを無視）となる。トークンは、指定されたイベントが発生するまで Wait ステップで「待機」する（通常は単一のイベントを指定するが、複数のイベントも指定可能で、トークン解放前に 1 つまたはすべてのイベントの発生を要求するよう選択できる）。この機能は、特定の条件が満たされるまで、モデル内の特定の場所でトークン（および関連オブジェクト）を待機させるために使用される。たとえば、あるノード（コンベア）上で、別のコンベアで処理中の優先順位 1 の部品がすべて完成するまで、優先順位 2 の部品群を待機させる場合がある。最後の優先順位 1 の部品が完成したときに AllDone イベントを発火し、待機中の優先順位 2 の部品を解放する。

Wait と Fire には多くの簡単な用途があるが、特に興味深い一般的な用例として、2 つのエンティティが連携地点で同期を取る場合がある。たとえば、患者が

MRI などの画像診断を受けようと待機しているが、医療保険の事務処理が承認されるまで診断を受けられない。一方で、患者が MRI の準備を整えるまで保険の事務処理も進められない。この状況で、各エンティティが相手を待つために PatientReady と PaperworkReady のイベントを設定すれば解決すると考えがちだが、どちらのエンティティが先に到着するか予測できないため、相手が待機状態でないときにイベントが誤って発火してしまう問題が生じる。よりよい解決策は、両オブジェクトの準備状態を表す状態変数を用意し、Decide ステップでその状態をチェックしてから、相手のエンティティを発火または待機させることだ。このソリューションの詳細な実装については、読者の検討に委ねる。

本節の最後に、イベントベースのロジックの代替案について考えてみる。イベントベースのアプローチは一般的に非常に効率的だが、ときにはより柔軟な対応が必要となる：

- Scan ステップは、指定した条件が真になるまで処理トークンを保持するために使用される。トークンが Scan ステップに到着すると、スキャン条件が評価される。条件が真の場合、トークンは即座にステップを離脱できる。そうでない場合、トークンは条件が真になるまで Scan ステップで保持される。任意の式の値を監視できる柔軟性は大きな利点だが、2 つの重要な注意点がある。1 つ目は、条件判定が時間進行時にのみ行われるため、瞬時（ゼロ時間）の条件変化を見逃す可能性があり、条件自体も時間進行まで認識できないことである。2 つ目は、式の監視によりイベントベースのアプローチと比べて処理速度が遅くなることである。この速度低下は多くのモデルでは問題にならないものの、一部のモデルでは顕著な実行速度の低下を引き起こす可能性がある。
- Search ステップは、本章の冒頭で扱ったが、イベントベースのロジックの一部として活用できる。たとえば、注目すべきイベントが発火したときに、最適なアクションを選択するため、候補群の中から検索を行うことができる。

10.6　その他のライブラリとリソース

これまで Simio Standard Library に注目し、15 のオブジェクトについて学んできた。ここからは、さらなる可能性を探るため、Simio がサポートする Flow ライブラリと Extras ライブラリについて説明する。また、ユーザフォーラムの SI Shared Items には他のユーザによる興味深いアイテムが多数投稿されており、Simio の GitHub リポジトリでは Simio チームが提供するさまざまなオープンソースプロジェクトのコードを確認できる。

10.6.1　Flow ライブラリ

Standard Library は主に離散的なエンティティ移動のモデル化に焦点を当てているが、液体や液状の動作は、連続的な**フロー**（**Flow**）概念を用いることでより正確にモデル化できる。代表的な応用例として、医薬品、食品、飲料品、鉱業、石油化学製品、化学製品などがある。特に、パイプラインのネットワーク（あるいはコンベヤ）、ミキサーとタンク（あるいは貯蔵パイル）間の移動、コンテナの充填など、定常的または断続的な原材料の流れを扱う場面で有用である。

他のシミュレーション製品ではフロー機能を持つものは少ないが、Simio は包括的な機能を提供している。さらに、多くの製品と異なり、Simio は単一のモデル内でフローと離散をシームレスに統合できる。このコンセプトは、エンティティの物理的な特性に表れている。離散的なエンティティは 3 次元の物理的な大きさ（**体積**）に加えて**密度**も持つ。これにより、樽や山などの離散的な表現と、100 リットルのオイルや 100 トンの石炭などの液体量や質量の表現の両方が可能となる。離散的なエンティティは、フロー要素として扱うことも、体積を基準に液体との相互変換も可能である。また、リンクやタンク内の原材料の流れについて、製品種類、目的地、構成データなどの属性を追跡できる。

Flow ライブラリには、次のオブジェクトが含まれている：

・FlowSource：液体または指定したエンティティタイプの集団の無限の流れを生成するソース。
・FlowSink：処理を完了した流れを消滅させるオブジェクト。
・Tank：流れを保持し、体積または重量で制限された容量を持つ保管場所。
・ContainerEntity：液体または集団の量を表すフローエンティティを運ぶ、移動可能な容器（例：樽）。
・Filler：コンテナエンティティに内容物を充填する装置。
・Emptier：コンテナエンティティから内容物を排出する装置。
・ItemToFlowConverter：離散的なアイテムをフローエンティティに変換する装置。
・FlowToItemConverter：フローエンティティを離散的なアイテムに変換する装置。
・FlowNode：他のオブジェクト（例：Tank）やリンクネットワークの制御地点で、フローの流入・流出を調整するオブジェクト。
・FlowConnector：フローノード間の即時接続（移動時間ゼロ）を提供するオブジェクト。
・Pipe：フローノード間を接続し、移動時間とパイプライン容量を考慮した搬送を行うオブジェクト。

Flow ライブラリの詳細は Simio ヘルプを参照してほしい。特に、SimBit の FlowConcepts に含まれる 8 つのモデルを確認するとよい。これらのモデルはこのライブラリの多様な機能を分かりやすく解説している。

10.6.2　Extras ライブラリ

Extras ライブラリは、一般的な構造物を追加するための補助ライブラリとして設計されている。このライブラリの多くのオブジェクトは、複数の基本オブジェクトを組み合わせた複合オブジェクトである。これらの複合オブジェクトは、グラフィックス、動作、プロパティを必要に応じて編集でき、さまざまな用途に対応できる。

Extras ライブラリには、以下のオブジェクトが含まれている：

・Bay：橋形クレーン（天井クレーン）が移動できる四角形の領域を示す。フロア

の空間などがその例である。クレーン同士の衝突を防ぐため、各ベイは複数のゾーンに分割できる。

・Crane：橋形クレーンまたは天井クレーンを表す複合オブジェクト。ブリッジ本体、ブリッジ上を移動する運転室、垂直移動用のリフト、およびエンドエフェクタ（フックや掴み装置）で構成される。車両と同様の特性（積載・積降、待機、初期ノード、遊休状態）を持ち、全方向への加速と速度を制御できる。
・Robot：エンティティを一つのノードから別のノードへ移動させる関節ロボット。回転可能な固定土台、連動する下腕と上腕、先端の手またはエンドエフェクタで構成される複合オブジェクト。車両と同様の特性を持ち、全軸方向のピッチと変化率を制御できる。
・Rack：メインラック（固定フレーム）と複数の棚で構成される複合オブジェクト。棚の数と各棚の容量（保持可能な最大エンティティ数）が主要なプロパティとなる。
・LiftTruck：車両と昇降装置からなる複合オブジェクト。棚へのエンティティの出し入れのため、昇降装置を適切な高さまで上下させる。標準的な車両のプロパティに加え、垂直方向の速度と昇降距離を設定できる。
・Elevator：垂直方向に移動し、Elevator Node でエンティティの乗降を行うトランスポータ。
・ElevatorNode：エレベータの各停止位置を表すノード。対応するエレベータと連携する。

Support リボンから Sample SimBitSolutions を開き、検索ボックスに「Extras Library」と入力すると、使用方法を説明する SimBits を確認できる。また、サンプルモデル（EngineRepairUsing ExtrasLibrary）も参照可能である。

10.6.3　Shared Items フォーラム

SI Shared Items ユーザフォーラムは、ユーザがオブジェクト、モデル、ライブラリ、パス装飾、テクスチャ、シンボル、カスタム API コード、文書、その他の便利なツールを投稿・共有できる場所である。このフォーラムは Support リボンからアクセスでき、豊富な便利ツールが常時更新されている。以下では、その概要を説明する。

まず、いくつかのライブラリがある：

・Cranes ライブラリには、一般的な橋形クレーン（Extras ライブラリの Cranes オブジェクトと同様）に加えて、クレーン運転室がブリッジ間を移動し天井に移動ネットワークを形成する天井クレーンも含まれている。
・Transportation ライブラリには、複数車両による鉄道システムや液体貨物を運搬する Tanker オブジェクトなどが含まれている。

このフォーラムには、以下のような便利なオブジェクトも用意されている：

・Vehicle with Driver：移動可能なオブジェクトが別の移動可能なオブジェクトを運転してエンティティを輸送する。

・Conveyor Transfer Device：3 つのコンベヤを接続してエンティティの搬送方向を変更する。
・Combiner without Parent：親子関係を使用せずにエンティティを結合する。
・その他、多数。

さらに、このフォーラムには、便利なアドインやマクロなどの機能も含まれている。主な例を以下に示す：

・Import Objects from a Spreadsheet：スプレッドシートからオブジェクトやリンクなどのモデル要素をインポートするアドイン。
・Import Experiment from CSV File：CSV ファイルから実験に係わる要素をインポートするアドイン。
・Save Step：モデル実行前に自動保存を行う機能。
・Simio Quick Reference Card：モデル構築に必要な重要事項をまとめたクイックリファレンス。

これらのコンテンツを積極的に活用し、さらに独自のコンテンツを投稿することで、ユーザコミュニティの発展に貢献できる。

10.6.4　Simio GitHub

Simio LLC GitHub サイトは、Simio チームメンバーがユーザ向けにオープンソースプロジェクトのソースコードと情報を公開する場所である。サイトのアドレスは `https://github.com/SimioLLC` である。各プロジェクトは独自のライセンスの下で公開されているが、サポートは保証されない。

このリポジトリは、次の 6 つのトピックで構成される：自動化、データ接続と変換、設計アドイン、ユーザ定義ステップと要素、サードパーティ連携、その他。

図 10.12 に Simio GitHub サイトの人気リポジトリを示す。

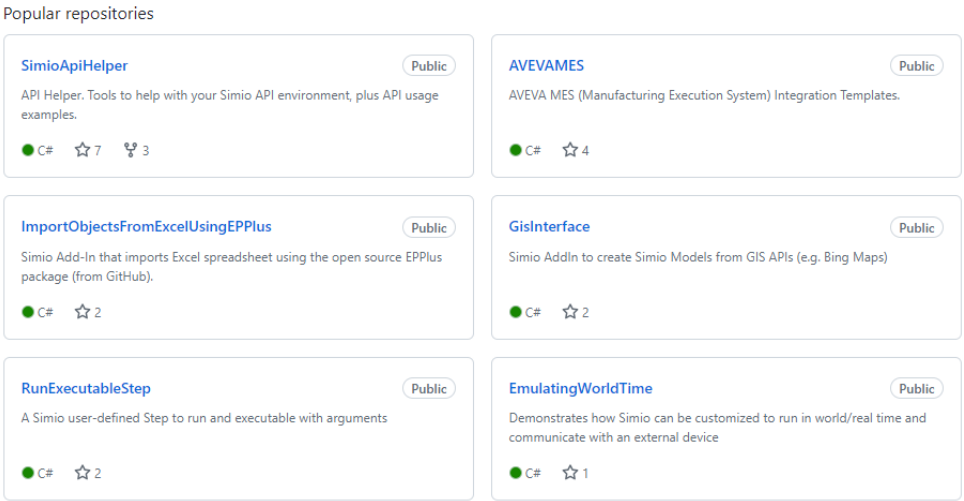


図 10.12：Simio GitHub サイトの人気リポジトリ

10.7　実験

4.2.3 項で説明した実験のトピックに戻ろう。これまでに、統計的に正しい結果を得るための複数回の反復実行と、最適解を導くための複数シナリオ実行の重要性について理解を深めてきた。9 章では、多くのシナリオから最適な代替案を見つけるための OptQuest の活用方法を解説した。

しかし、これらに共通する課題として、大量の反復実行が必要となる点がある。1 回の実行が 1 秒で 100 回の反復が必要な場合は 2 分未満で済み、それほど問題にはならない。ところが、500 シナリオそれぞれに 9 つの実験があり、最低 5 回ずつの反復実行が必要な場合はどうだろうか。1 回の実行に 2.5 分かかるとすれば、総所要時間は 900 時間を超える。順次処理では 1 か月以上かかる実験計画は、現実的ではないだろう。この例は極端ではあるが、大規模な実プロジェクトではよく直面する状況である。本節では、このような大量の処理需要への対処法を解説する。

10.7.1　並列処理

多くのコンピュータは並列処理の能力を持っており、Simio の全バージョンがこの機能を活用できる。たとえば、デュアルスレッドのクアッドプロセッサを搭載したコンピュータでは、標準構成で 8 つの並列「スレッド」を処理できる。つまり Simio では、8 つの反復実行を同時に実行できる。この機能により、先の例の 900 時間を 113 時間まで短縮できるが、それでもかなりの時間を要する。

16 個以上のプロセッサを搭載している場合（本書執筆時点ではまだ一般的ではない）、Simio は最大 16 個までの反復実行を並列処理できる。Simio の上級エディション（Professional と RPS）では、ローカルネットワーク上の他のコンピュータと連携した分散実行（Distribute Runs）が可能だ（設定は Experiment Properties→Advanced Options→Distribute Runs で行える）。これらの機能を活用できる環境であれば、16 個の並列実行により、先の例の処理時間を約 60 時間まで短縮できる。まだかなりの時間を要するが、より現実的な範囲となる。

サーバファーム（ローカルネットワーク上の多数のコンピュータ群）へのアクセス権がある場合、Simio は大規模な並列処理のためのライセンスを提供している。たとえば、追加で 100 個のプロセッサを実行できるライセンスがあれば（計 116 プロセッサとなり）、900 時間かかっていた処理を 8 時間未満に短縮できる。

10.7.2　クラウド処理

前項を読んで「クアッドプロセッサのコンピュータを持っていても、900÷4＝225 時間も待てない」と思うかもしれない。「クラウド」という用語を耳にしたことがあるだろう。私たちの多くは、意識するしないに関わらず、クラウド上のアプリケーションを日常的に利用している。Simio には、Simio Portal Edition という製品がある。これは商用製品であり、残念ながら本書執筆時点ではアカデミックバージョンは提供されていない。Simio Portal を使えば、モデルをクラウドにアップロードし、クラウドのリソースを活用して実験の設定・実行が可能だ。実用上の制限はあるが、理論的には前述の問題で必要な反復実行を、1 回の実行時間（2.5

分）よりも短時間で完了できる。ただし、現実的な制約により、すべての結果を得るまでには約 1 時間程度かかる可能性がある。それでも、900 時間という処理時間と比べれば、じゅうぶんに高速な実行といえる。

10.8　まとめ

本章では、まず Search ステップを紹介し、オブジェクトの集合やテーブル行から選択する強力な仕組みについて解説した。次に、退去と放棄について扱い、処理中のステーションからエンティティを簡単にバイパスまたは取り出す方法を説明した。続いて、マテリアル要素とインベントリ要素を紹介し、これらの強力な要素の活用法を解説した。その後、タスクシーケンスの概念を導入し、Server タイプのオブジェクトで並列・直列のサブタスクを柔軟に実行できることを示した。最後に、イベントによる決定ロジックを通じて、処理ロジックを制御する新しいアプローチを提示した。

また、Flow および Extras ライブラリの概要を説明し、Standard Library では対応が難しい領域をカバーする Simio 提供の機能を紹介した。さらに、Shared Items フォーラムでユーザが提供している便利なオブジェクトやツールについても触れた。11 章では、独自のカスタムルール、オブジェクト、ライブラリの作成方法を解説する。

10.9　問題

1．各窓口の容量が 2 である直列 3 工程の CONWIP（CONstant Work In Process）システムを作成する。窓口の処理時間とバッファ容量は既定値のままとする。システム内の WIP を常に 8 に維持したい。この 8 つのエンティティの位置は問わない。システム内のエンティティ数の平均値と最大値を示し、予測される動作を説明せよ。

2．図 10.13 のシステムを考える。エンティティは、平均 2.5 分の指数分布に従う到着間隔で各ソースから到着する。BasicNode1 への移動時間は 1 分から 3 分の一様分布に従い、BasicNode2 への移動時間は平均 1 分の指数分布に従う。各 BasicNode から Combiner までの移動時間は正確に 1 分である。エンティティは Combiner に完全に同期して到着し、待ち時間は発生しない（Combiner では、親エンティティと 1 つのメンバーが組み合わされる）。BasicNode にイベントを用いたプロセスロジックを追加し、先行エンティティが後続エンティティを待つように設定せよ。

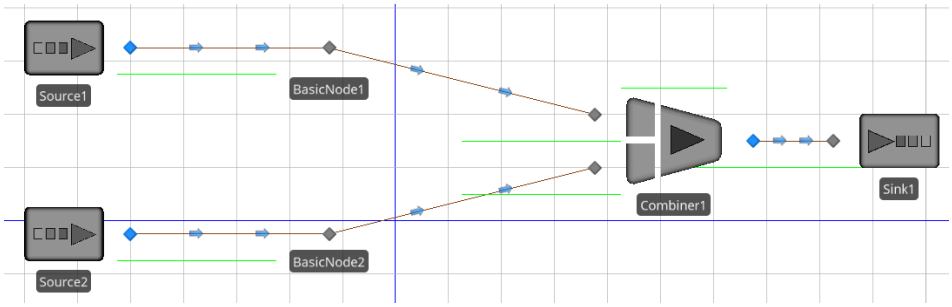


図 10.13　問題 2：同期システム

3．問題 2 を拡張し、すべてのエンティティにオーダ番号を持たせ、各 Source は 1 から順

にオーダ番号を割り当ててエンティティを生成するものとする。このとき、オーダ番号に基づいて組み合わせを行うため、BasicNode で同一オーダ番号のエンティティを同期させ、ペアとして一緒に解放する必要がある。

4. 図 10.14 の生産システムについて検討する。部品は Receiving に約 8 分間隔で到着し、天井クレーンによって Grinding まで搬送される。研磨工程には 3～7 分を要する。研磨後、80%の部品はクレーンで Shipping へ、20%は Rework へ搬送される。Rework 完了後の部品は、必ずクレーンで Grinding に戻される。通常、1 台の天井クレーンは待機時に Receiving 付近で待機している。クレーンの積載・荷降ろし時間は各 1 分で、横方向の移動速度は 1m/秒である。1 日 24 時間のモデルを 5 日間実行し、クレーンと機械の期待利用率を求めなさい。

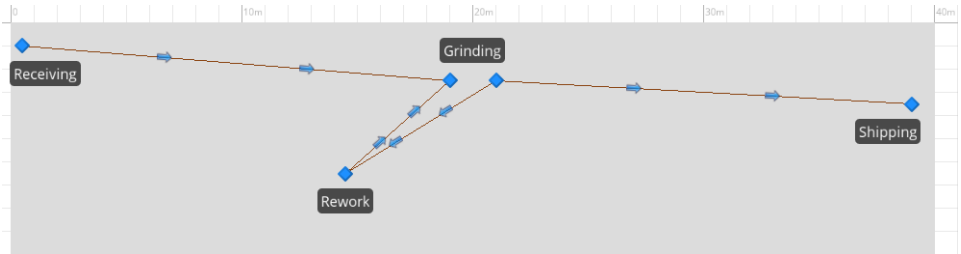


図 10.14　問題 4：クレーンシステム

5. 問題 4 のシステムにおいて、生産量を増加させたいと考えている。現在とは反対側の端に駐機する 2 番目の同一性能のクレーンを追加し、Grinding に 2 台目のユニットを加えた場合、合理的な注文の受入率はどの程度になるか。また、そのスループットの向上についても考察しなさい。

6. 入口と出口が 1 階にあり、2 階に 3 名の医師が待機する小規模な診療所をモデル化する。患者は 1 日 8 時間の診療時間中、1 時間当たり約 30 名の割合で到着する。2 基のエレベータ（各定員 4 名）により、患者はフロア間を移動する。患者は医師 1 名から 3～7 分の診察を受けた後、1 階の出口に戻る。Extras ライブラリの Elevator オブジェクトを使用して、このモデルを作成しなさい。

第 11 章　カスタマイズと Simio の拡張

多くの場合、Simio Standard ライブラリのオブジェクトやアドオンプロセスを使用して、シミュレーションの問題を解決できる。ただし、これらの標準的な方法ではじゅうぶんな機能が得られない場合もある。以下のような例が考えられる：

・アドオンプロセスでは対応できず、カスタマイズされたオブジェクトが必要な場合。
・同じ拡張機能の複数回使用や、独自の再利用可能なオブジェクトが必要な場合。
・経験豊富なモデル作成者と初心者が同じチームで働き、状況に応じたシミュレーション活用の支援や独自のカスタムツールが必要な場合。
・商用ライブラリの開発を目指す場合。

これらの要件は、Simio のカスタムオブジェクトとライブラリを作成することで実現できる。これまでの章と例題を理解していれば、カスタムオブジェクトの定義に必要な知識の大部分をすでに習得していることになる。本章の前半では、既習の概念と新しい概念を組み合わせて解説する。

カスタムオブジェクトは非常に柔軟で強力だが、意欲的なモデル作成者はさらなる可能性を追求するだろう。Simio は、経験豊富なユーザ向けに追加機能も提供している。プログラミングの知識（Visual Basic、C++、C#、または.NET 言語）があれば、以下のことが可能である：

・設計の際に使用する独自アドオンを作成する（例：外部データからのモデル自動生成）。
・新しい動的選択ルールを Simio に追加する（例：Vehicle、Worker、Server に対する包括的な作業選択ルールの作成）。
・テーブルのインポート・エクスポート用カスタムルーチンを追加する。
・実験の設計（例：シナリオの設定・分析）や実行（例：最適化）のためのカスタムアルゴリズムを追加する。
・Simio エンジンを拡張するための独自の Step と Element を追加する。
・特定の用途に合わせて Simio のリボンをカスタマイズする。

本章では、これらの機能の概要を説明する。なお、本章の内容の多くは、Simio ソフトに付属の電子書籍「Introduction to Simio」（Pegden and Sturrock 2013）から許諾を得て引用している。

11.1　オブジェクト定義に関する基本的なコンセプト

これまで使用してきた Standard ライブラリは、モデル構築のための選択肢の一つに過ぎない。目的に合わせて独自のオブジェクトライブラリを構築したり、企業内で共有するためのライブラリを作成したりすることができる。Simio の基本概念の一つは、すべてのモデルをオブジェクト定義として扱えることだ。モデルはオブジェクトの基本的なふるまいを定義するものであり、作成した Simio モ

デルは容易にパッケージ化して他のモデルの一部として利用できる。1 つのプロジェクト内で複数のサブモデルを構築し、それらを自身のモデルの構成要素として活用することも可能だ。

本節での議論は、4 章、5 章、9 章で取り上げた Simio プロセスを発展させた内容である。これまでの章では、オブジェクト定義の核心部分を変更せずに、プロセスを活用して特定のオブジェクトインスタンスの基本機能を拡張する方法を学んだ。ここでは、オブジェクト定義の核心部分の設定変更や拡張、さらには新規オブジェクトの構築に焦点を当てる。これにより、オブジェクトのすべてのインスタンスのふるまいを変更することが可能になる。

オブジェクト定義には、5 つの主要な**構成要素**がある。プロパティ、状態変数、イベント、外部ビュー、ロジックである（図 11.1）。プロパティ、状態変数、イベント、外部ビューはすべてモデルの Definitions ウィンドウで定義される。一方、ロジックはモデルの Facility ウィンドウと Processes ウィンドウを組み合わせて定義する。

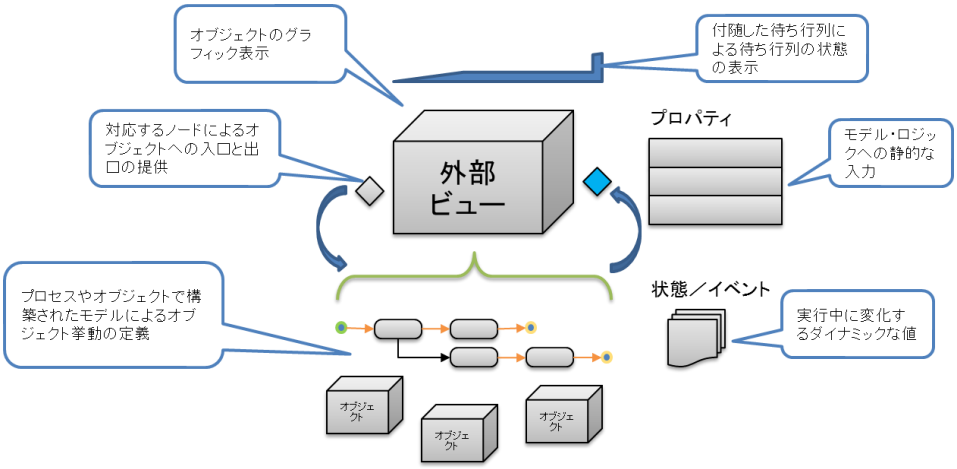


図 11.1　オブジェクトの構成要素

まずモデルプロパティを利用して、モデルで参照されたり実験で使用されるプロパティを作成する。これらのプロパティは、オブジェクト定義用のモデルへの入力を定義する際にも活用できる。**モデルプロパティ**は、モデルロジックへの入力を提供し、実行中は固定値となる。**状態変数**は、オブジェクトの現在の状態を定義する。シミュレーション実行中に変更可能な項目はすべて、状態変数としてオブジェクト内に表現される。**イベント**は、エンティティのステーション進入やノードからの離脱など、瞬間的に発生する論理事象である。オブジェクトは独自のイベントを定義でき、それを始動させると同時に他のオブジェクトへイベント発生の合図を送ることができる。**外部ビュー**は、オブジェクトのインスタンスをグラフィカルに表現したもので、モデルにオブジェクトを配置する際に表示される。オブジェクトの**ロジック**は、各イベントに対するオブジェクトの反応を定義するモデルであり、オブジェクトにふるまいを与える。このロジックは、既存のオブジェクトを用いて階層的に構築することも、グラフィカルプロセスのフローとして

定義することもできる。

11.1.1　モデルのロジック

固定オブジェクトのロジックは、Facility モデルと Processes モデルによって定義され、ノードシンボルの Input Logic Type で決定される。エンティティ、トランスポータ、ノード、リンクなどはノードシンボルを持たないため、主に Processes ロジックを用いて構築される。

新しいモデルを構築することは、本質的に新しいオブジェクト定義のロジック部分を作成することと同じである。モデルの入力用プロパティ、グラフィカルな外部ビュー、そしてエンティティの出入り用ノードを追加することで、どのようなモデルでもオブジェクト定義として容易に活用できる。

オブジェクトのモデルロジックを定義するには、3 つのアプローチがある：

- 1 つ目のアプローチは、Facility モデルを利用して階層的なモデルを構築する。Facility オブジェクトのカスタムなふるまいを定義するために、アドオンプロセスと組み合わせることも可能である。このアプローチは、たとえば機械 2 台、作業員 1 人、治具からなるワークセンターといった高レベルの施設要素を構築する際に活用される。
- 2 つ目のアプローチは、もっとも柔軟性の高いもので、Processes モデルを利用してゼロからオブジェクト定義のふるまいを構築する。このアプローチは、Standard ライブラリのオブジェクトの構築に用いられている。
- 3 つ目のアプローチは、既存のオブジェクト定義を**サブクラス**化し、プロセスを用いて新しいオブジェクトのふるまいを変更・拡張することである。このアプローチは、目的とするオブジェクトに類似したふるまいを持つ既存オブジェクトがあり、プロパティ名や概要、ふるまいの微調整で要件を満たせる場合によく用いられる。

11.1.2　外部ビュー

外部ビューは、固定または動的なオブジェクトのインスタンスのグラフィカル表現である。ノードとリンクには外部ビューは使用しない。図 11.2 に示すように、外部ビューはモデルの Definitions ウィンドウの External パネルをクリックして定義する。

外部ビューは、モデルにオブジェクトのインスタンスを追加する際に表示されるものである。外部ビューのグラフィックスは、シンボルライブラリや Trimble 3D Warehouse からダウンロードしたシンボルで構成できる。また、静的なグラフィックスは、Drawing リボンの描画ツールを使って作成することができる。さらに、外部ビューには、Animation ウィンドウから各種アニメーション要素を追加できる。これには待ち行列、状態ラベル、プロット、円グラフ、四角・円形のゲージ、およびボタンが含まれる。動的オブジェクト（エンティティなど）の場合、アニメーション要素はシステム内でそのオブジェクトとともに移動する。たとえば、エンティティは状態変数の値を表示する状態ラベルを持つことができ、またクリックでアクションを実行するボタンを持つこともできる（Model 9-2 参照）。

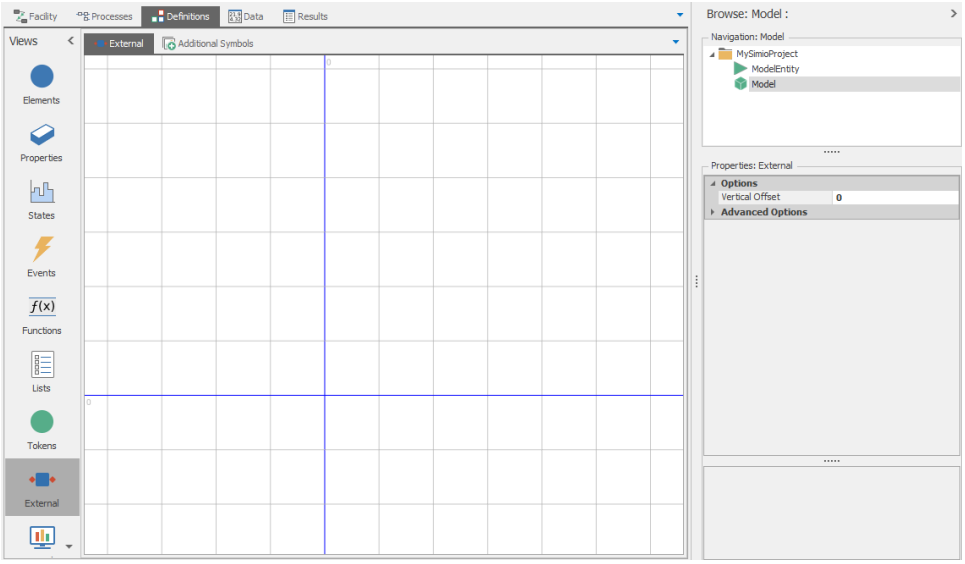


図 11.2　外部ビュー

新しい固定オブジェクトを構築する場合、一般にそのオブジェクトに入力および出力ノードを関連づけて、動的エンティティが固定オブジェクトに出入りできるようにする。たとえば、Server オブジェクトには必ず入力および出力用のノードオブジェクトが関連づけられている。このような関連オブジェクトの特性は、Drawing リボンを使用して外部ビューにノードシンボルを配置することで定義する。ここで重要な点は、これらのノードシンボルはノードオブジェクトそのものではなく、元のオブジェクトが配置される際に作成される関連ノードオブジェクトの位置を示すシンボルだということである。

外部ビューでノードシンボルを配置する際は、そのプロパティを定義する必要がある。まず、**ノードクラス**からノードのタイプを指定する。ドロップリストには、次の選択肢が表示される：Node（空のモデル）、BasicNode（基本的な接点）、TransferNode（目的地設定とトランスポータ選択を支援する接点）、その他のライブラリやプロジェクトで定義されたノード。Standard ライブラリでは、通常 BasicNode クラスを入力ノードとして、TransferNode を出力ノードとして使用する。

Input Logic Type は、到着エンティティの処理方法を指定する：None（エンティティが許可なしで進入）、ProcessStation（ステーション進入イベントによりプロセスロジックを開始。Processes モデル使用時に選択）、FacilityNode（エンティティを Facility モデル内部のノードへ送信。Facility モデル使用時に選択）。

また、General セクションでノードシンボルの Name を設定できる。ノードシンボル名は関連ノードオブジェクトの名称生成に使用され、NodeSymbolName@ObjectName の形式で生成される。たとえば、シンボル名が Input、オブジェクト名が Lathe の場合、関連オブジェクトは Input@Lathe と命名される。Standard ライブラリでは、入力ノードシンボルは Input、出力ノードシンボルは Output と命名される。

11.1.3　オブジェクト定義のサブクラス化

オブジェクト定義の**サブクラス化**の基本的な考え方は、新しく構築するオブジェクトが既存のオブジェクト定義からプロパティ、状態変数、イベント、およびロジックをそのまま継承することである。サブクラス化されたオブジェクトは、最初は元のオブジェクトと同じプロパティとふるまいを持つ。元のオブジェクト定義を更新すると、サブクラス化されたオブジェクトも新しいふるまいを継承する。

サブクラス化を行った後は、継承されたプロパティの再命名や非表示が可能である。また、継承されたプロセスロジックの一部を上書きしたり、プロセスを追加してふるまいを拡張したりすることもできる。たとえば、Server からサブクラス化した新しいオブジェクト定義を MRI と命名し、医療システムをモデリングするためのライブラリに組み込むことができる。この場合、通常の Server プロパティ（例：Capacity Type）を非表示にしたり、プロパティを再命名（例：Process Time を Treatment Time に変更）したりできる。さらに、他のプロセスを Server から継承しながら、通常の継承プロセスを MRI の故障パターンとしてモデル化する新しいプロセスに置き換えることも可能である。

オブジェクト定義をサブクラス化するには、Navigation ウィンドウの Project をクリックし、Models パネルを選択する。Edit リボン（図 11.3）で、新しい空のモデルの追加、選択したモデルのサブクラスの作成、ライブラリのオブジェクト定義からのサブクラスの作成が可能である。また、Facility ウィンドウ内でライブラリのオブジェクト定義を右クリックして表示されるメニューからも、ライブラリオブジェクトをサブクラス化できる。

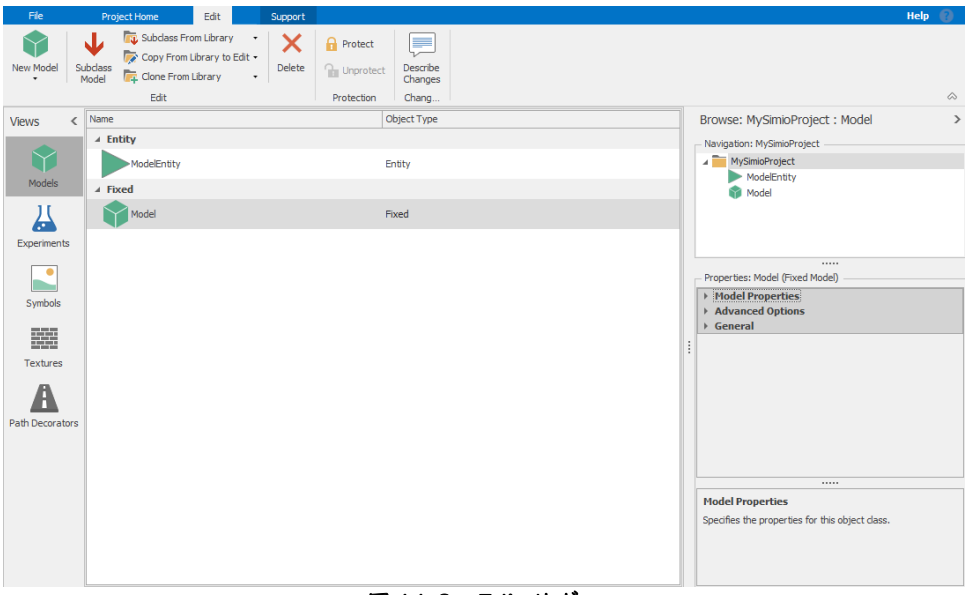


図 11.3　Edit リボン

サブクラス化でよく見られるケースは、複数のデータテーブルを使用して複数のプロパティに値を設定し、それを類似したオブジェクトに適用する場合である。たとえば、複数の Server オブジェクトの容量をあるデータテーブルから、処理時

間を別のデータテーブルから取得したい場合がある。これは各サーバに個別に設定することもできるが、単一のオブジェクトインスタンスに対してまとめて変更を行う方がはるかに効率的である。1 つのオブジェクトインスタンス(例:Server1)の変更が完了したら、それを右クリックして Create Object From This を選択できる。すると、Server の親オブジェクト定義から自動的にサブクラスが作成され、名前（例：MyServer）を設定すれば、編集中のオブジェクトのすべてのプロパティが新しいオブジェクト定義のデフォルトとして適用される。

モデルの構築中、インスタンスにさらなる変更を加える必要が生じることがある。これらの変更をオブジェクト定義の一部として、関連するすべてのインスタンスで共有したい場合は、変更したインスタンスを右クリックして Update 'MyServer' Property Defaults from This オプションを選択する。このテクニックは 12.13 節で Model 12-03 を構築する際に使用する。

ライブラリオブジェクトを基にして新しいオブジェクトを定義する方法には、以下の 3 つがある：

- SubClass From Library：ライブラリオブジェクトからサブクラス化（派生）を行う。この方法では、既存のオブジェクト定義からプロセスロジックを継承するため、元のオブジェクトを変更すると、サブクラス化されたオブジェクトのふるまいも自動的に更新される。
- Copy From Library for Edit：オブジェクトを単純にコピーする。新規作成されたオブジェクト定義はライブラリオブジェクトの独立したコピーとなり、元のオブジェクトとの継承関係は持たない。そのため、元のオブジェクト定義が変更されても、新規作成されたオブジェクトには影響しない。
- Clone From Library：元のオブジェクトと完全に同一のオブジェクトを作成する（Simio では新規オブジェクトと元のオブジェクトを区別できない）。この機能は、異なるライブラリから構成されているモデルを整理する際によく使用する。

オブジェクト定義はパスワードで保護することができる。保護された場合、オブジェクトの内部モデルの表示や変更時にパスワードの入力が必要となる。

モデルプロパティには、Model Name、Object Type（固定、リンク、ノード、エンティティ、トランスポータ）、Parent Class（親クラス：このオブジェクトのサブクラス化元）、Icon（モデル表示用）、Author、Version Number、Description が含まれる。また、オブジェクトがリソースとして使用される場合は Resource Object プロパティが、モデルとして実行されるかサブモデルとしてのみ使用される場合は Runnable プロパティが含まれる。

11.1.4 プロパティ、状態変数、イベント

オブジェクト定義を作成すると、元のオブジェクトクラスからプロパティ、状態変数、およびイベントを継承し、新しいメンバーを追加することができる。これらの継承されたプロパティ、状態変数、およびイベントは、モデルの Definitions ウィンドウで確認できる。継承されたメンバーと新規作成されたメンバーは、そ

れぞれのカテゴリに分類され、個別に展開・折りたたみが可能である。図 11.4 の Properties パネルには、ReworkTime という名称の新しいプロパティのみが Fixed（固定）として表示されている。このプロパティは、モデル内のオブジェクトとプロセスの両方から参照可能である点に注意が必要である。

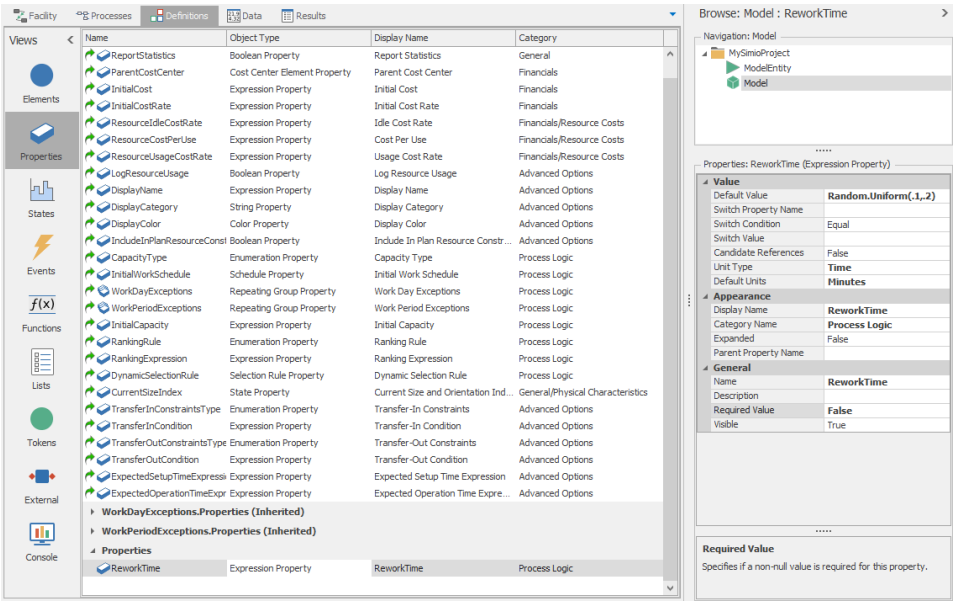


図 11.4　Definitions ウィンドウの Properties パネル

ReworkTime の特性は Properties ウィンドウに表示され、Default Value、Unit Type/Default Units、Display Name、Description、Category Name などを含む。これらの項目に対して、必須入力項目とするか（Required Value）、ユーザに表示するか（Visible）を設定できる。また、Switch Property Name、Switch Condition、Switch Value の設定も可能だ。これらの設定により、他のプロパティの値に応じて特定のプロパティの表示・非表示を制御できる。たとえば、プロパティ X の値が 0 より大きい場合にのみプロパティ Y を表示するといった制御が可能で、ユーザの入力に応じて Property ウィンドウを動的に変更できる。

継承されたプロパティについては、Default Value、Display Name、Description、Category に Visible フラグがあり、表示・非表示の切り替えや概観の変更が可能だ。ただし、状態変数やイベントについては、新規追加は可能だが、継承されたものの名称変更や非表示化はできない。

11.2　Model 11-1：階層的オブジェクトの構築

この例では、Standard ライブラリの 2 つの Server と 1 つの Connector を組み合わせた、タンデムサーバという新しいオブジェクト定義を構築する。1 つ目の Server は、容量が 1 で**アウトプットバッファを持たない**。2 つ目の Server も同様に容量が 1 だが**インプットバッファを持たない**。そのため、2 つ目の Server が稼働中の場合、1 つ目の Server は**ブロック**される。このオブジェクトには、2

つの Server それぞれの処理時間を指定するプロパティがある。さらに、各 Server のプロセス内のエンティティと、リソース状態を示す円グラフの両方をアニメーション表示する。

11.2.1　モデルロジック

階層的なオブジェクトを構築するため（つまり、Facility ウィンドウ内の他のオブジェクトを組み合わせて）、まず Facility ウィンドウでオブジェクトを作成し、ロジックを定義する。

- 新しいプロジェクト（New Project）を開始する。Project Home リボンから New Model→Fixed Class を選択し、Nagivation ペインでモデル名を TandemServer に設定する。
- TandemServer を選択してアクティブモデルにする。次に、2 つの Server と 1 つの Connector を Facility ウィンドウに配置する。Server1 の Output Buffer と Server2 の Input Buffer をそれぞれ 0 に設定する。

次に、TandemServer のプロパティを定義し、Properties ウィンドウの表示をカスタマイズする。

- Definitions ウィンドウの Properties パネルで、Expression タイプの Standard Properties を 2 つ追加する。プロパティ名をそれぞれ ProcessTimeOne および ProcessTimeTwo に設定する。Default Value、UnitType/Default Units、Display Name、Category Name の設定については、図 11.5 を参照。

Properties: ProcessTimeOne (Expression Property)

Value	
Default Value	**Random.Triangular(.1,.2,.3)**
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	**Time**
Default Units	**Minutes**
Appearance	
Display Name	**ProcessTimeOne**
Category Name	**Process Logic**
Expanded	False
Parent Property Name	
General	
Name	**ProcessTimeOne**
Description	**Processing time on first server**
Required Value	True
Visible	True

図 11.5　TandemServer の ProcessTimeOne プロパティの設定

・Properties(Inherited)（継承されたプロパティ）を展開する。CapacityType、InitialWorkSchedule、InitialCapacity の 3 つの継承プロパティは、TandemServer オブジェクトとの関連性が低いため、図 11.6 に示すように、それぞれの General カテゴリの Visible 特性を False に設定し非表示にする。

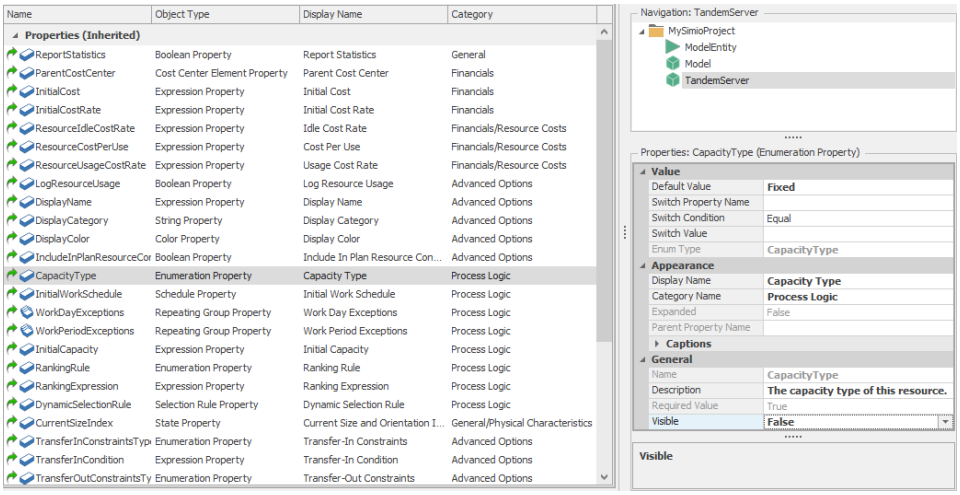


図 11.6　継承プロパティの非表示

11.2.2　外部ビュー

次に、外部ビュー（External View）を利用して、オブジェクト配置時にユーザが目にする項目を定義する。Simio のバージョン 4 以降では、Facility ウィンドウで定義されたオブジェクト、ノード、リンク、および状態変数のアニメーションが自動的に外部ビューに追加される。これは、オブジェクト内のリンク上のエンティティ移動を確認する際に特に便利である。ただし、この例では、より細やかな制御を行うため（そして操作経験を積むため）、手動でユーザビューを作成する。そのため、まずデフォルトのユーザビューを非表示にする。

・Facility ウィンドウを選択する。外部ビューの自動表示を非表示にするには、Facility ビューで各アイテムを右クリックし、Externally Visible オプションの選択を解除する。複数のアイテムがある場合は、Ctrl キーを押しながらドラッグしてすべてのアイテムを選択し、Externally Visible オプションを一括で解除できる。

・Definitions ウィンドウの External パネルを選択し、リボンの Place Symbol ボタンを使用して Server のシンボルを 2 つ連続して配置する。

・External Node をインプット側に配置する。Node Class を BasicNode に設定し、Input Location Type を Node に設定する。次に、Node を Input@Server1 として、Name を Input と定義する。

・Animation リボンの Queue ボタンを使用して、インプットバッファ、2 つの加工ステーション、およびアウトプットバッファの待ち行列を描画する。各待ち行列を配置後、Appearance リボンの Alignment で None を選択する。Queue States には、Server1.InputBuffer.Contents、Server1.Processing.

Contents 、 Server2.Processing.Contents 、 Server2.OutputBuffer.Contents を指定する。

・Animation リボンの Status Pie ボタンを使用して、縦 5×横 6 目盛の円グラフを 2 つ追加する。Data Type を ListState に設定し、それぞれの ListState に Server1.ResourceState と Server2.ResourceState を指定する。この時点で、画面は図 11.7 のような状態になる。

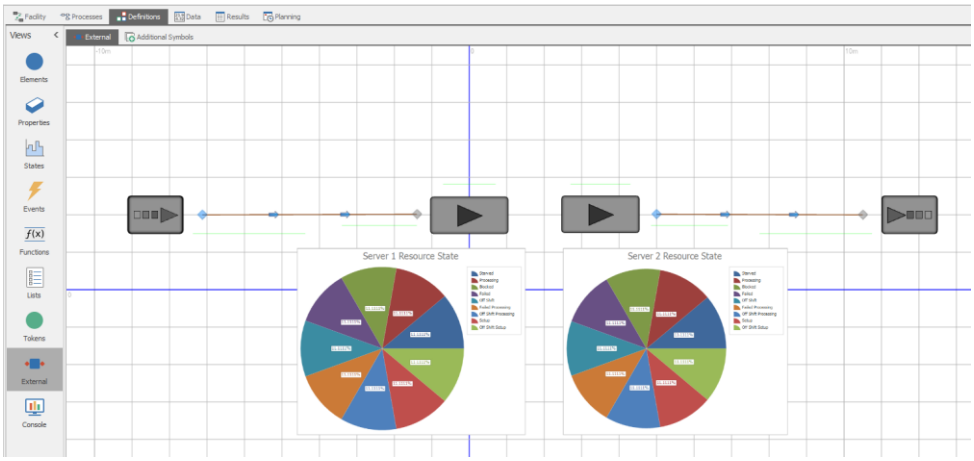


図 11.7　TandemServer の外部ビューの定義

外部ビューにおいて、Input ノードシンボルで作成された関連ノードに到着するエンティティは、TandemServer の Facility モデル内の Input@Server1 というノードへ送られる。エンティティは 2 つの Server 内で処理され、Server2 の出力ノードに達した時点で、Output という名前の外部ノードシンボルで定義された関連ノードへ送られる。

外部出力ノードを設定するには、TandemServer の Facility ウィンドウで Server2 の出力ノードを右クリックし、プルダウンメニューから Bind to New External Output Node を選択する。External Node Name には Output を指定する。これにより、TandemServer を離れるエンティティは、Output として作成された関連出力ノードを通過するようになる。External View では、この新しく作成したノードを 2 番目の Server シンボルの右側に配置する。

次に、Processing Time を設定する。Server1 には ProcessTimeOne、Server2 には ProcessTimeTwo を割り当てる。各サーバの Processing Time プロパティを右クリックし、Set Referenced Property を選択して、先ほど定義した対応するプロパティ値を選択する。これにより、これらの処理時間が TandemServer の新しいプロパティとして設定される。

11.2.2.1　本項のまとめ

本項では、階層的なアプローチを用いてオブジェクトを構築した。具体的には、以下の手順で進めた。まず Facility ウィンドウでオブジェクトのロジックを定義し、次にプロパティの表示・非表示を設定した。さらに、外部ビューを利用して、Facility ウィンドウ上でのオブジェクトの外観を定義した。

これで新規オブジェクト定義を利用する準備が整った。メインモデル（Model）をクリックしてアクティブモデルにし、Source、TandemServer（左下の Project Library から選択）、Sink を 1 つずつ設置して Path で接続する。TandemServer をクリックすると、Property ウィンドウに設定したカスタムプロパティが表示される。図 11.8 に TandemServer オブジェクトを使用した簡単なモデルを示す。

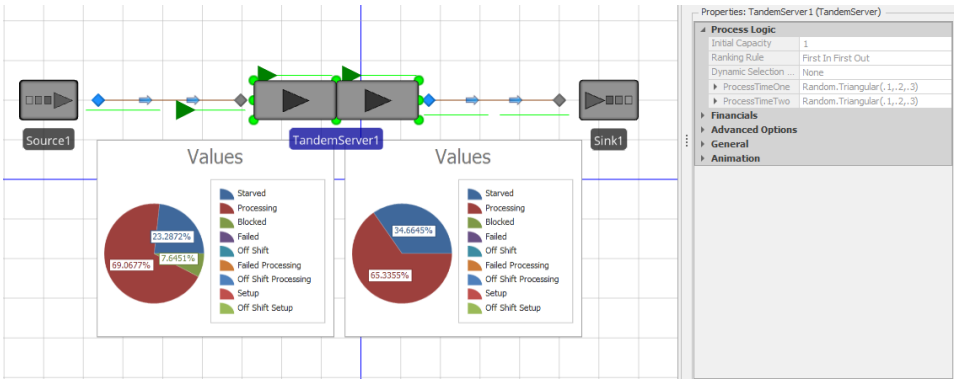


図 11.8　実行中の TandemServer

11.3　Model 11-2：ベースオブジェクトの作成

Model 11-1 では、Facility モデルを利用してオブジェクト定義を構築した。Model 11-2 では、プロセスモデルを使用して新しいオブジェクト定義を作成する。このオブジェクトは旋盤（工作機械の一種）で、一度に 1 つの部品しか加工できない。旋盤には 2 つのバッファがある：加工待ちの部品を保持するインプットバッファと、次工程への移動を待つ部品を保持するアウトプットバッファだ。

11.3.1　モデルロジック

新しいプロジェクト（ModelEntity および Model）を開始し、Project Home リボンから Fixed Class モデルを追加して Lathe と名づける。なお、Fixed Class の下に Processor オプションがあるが、今回はこれを使用しない。Processor オプションを使用すると、ベースオブジェクト構築時に便利なフレームワークが自動生成されるが、この例題ではフレームワークの作成過程をより深く理解することを目的としている。Lathe をアクティブモデルにした後、Definitions ウィンドウの Properties パネルを選択する。新しいオブジェクトでは、まず 4 つの新規プロパティを定義し、その後、不要な継承プロパティを非表示に設定する。

- Expression タイプの Standard Property を新規追加し、名前を TransferInTime に設定する。Default Value は 0.0、UnitType は Time、Default Units は Minutes、Display Name は Transfer In Time、Category Name は Process Logic とする。
- 同じく Expression タイプの Standard Property を追加し、名前を ProcessingTime にする。Default Value は Random.Triangular(0.1, 0.2, 0.3)、Unit Type は Time、Default Units は Minutes、Display Name は Processing Time、Category Name は Process Logic とする。

・バッファサイズを定義するため、Integer タイプ（整数型）の Standard Property を 2 つ追加する。名前は InputBufferCapacity（表示名：Input Buffer）と OutputBufferCapacity（表示名：Output Buffer）とし、Default Value は Infinity、Category Name は Buffer Capacity にする。
・継承プロパティを展開し、CapacityType、WorkSchedule、InitialCapacity については、Lathe オブジェクトと関連が低いため、General カテゴリでそれぞれの Visible 特性を False に設定して非表示にする（図 11.6 参照）。

Process ウィンドウのロジック構築のため、まず必要なステーションを追加する。Definitions ウィンドウの Elements パネルから、3 つのステーションエレメント（InputBuffer、Processing、OutputBuffer）を追加する。InputBuffer ステーションの Initial Capacity には、InputBufferCapacity プロパティを設定し（右クリックから Set Referenced Property を選択）、OutputBuffer ステーションの Initial Capacity には OutputBufferCapacity プロパティを設定する。Processing ステーションの Initial Capacity は 1 に設定する。

次に、Lathe オブジェクトのまま Process ウィンドウへ移動し、これら 3 つのステーション間の移動を行うプロセスフローを構築する。エンティティが Lathe オブジェクトと InputBuffer に入る際には、InputBufferTransferIn という名前のプロセスが実行される。

・Create Process ボタンをクリックして最初のプロセスを作成し、InputBufferTransferIn と名づける。Description には「Transfer in from outside the object. Hold the entity until the processing capacity is available.」という説明を加える。
・このプロセスを実行するトークンは、エンティティがステーションに進入するたびに自動生成される InputBuffer.Entered イベントによって開始されるため、プロセスの Triggering Event に InputBuffer.Entered を設定する。
・Delay、EndTransfer、Transfer の各ステップをこの順序で設置する。
・DelayTime に TransferInTime を設定する。この時間は、エンティティがステーションに進入するまでに要する時間を表す。F2 キーを押して Delay ステップの名前を TransferringIn に変更する。EndTransfer ステップは、搬送完了を外部世界（コンベヤ、ロボット、作業者など）に通知する。搬送完了時、InputBuffer ステーションに空きがあれば新しい搬送が開始される。注：ステップの「名前」は、各プロセスステップにつけられる固有のラベルである。意味のある名称を使用することで、プロセスの理解が容易になり、モデルの追跡も効率的になる。この命名規則は常に心がけるべきである。
・搬送完了後、トークンは、InputBuffer ステーションから Processing ステーションへのエンティティ搬送を開始する。Transfer ステップでは、From プロパティを CurrentStation、To プロパティを Station、Station Name プロパティを Processing に設定する。

この InputBufferTransferIn プロセスの完成図を、図 11.9 の最上段に示す。

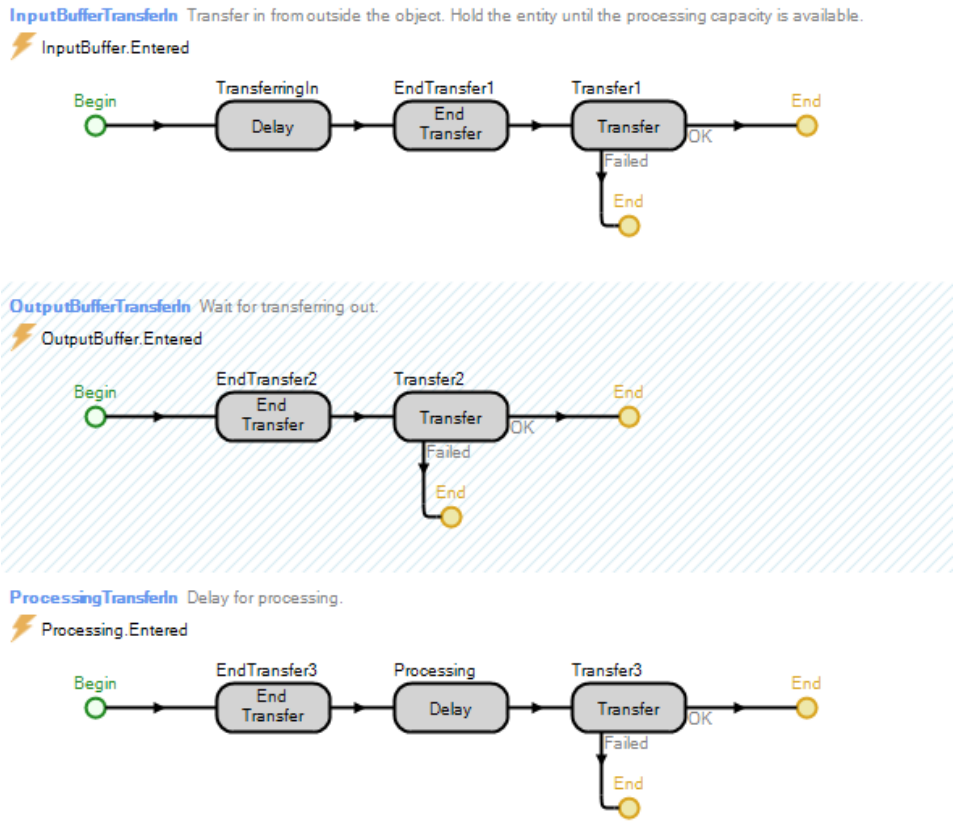


図 11.9　Lathe オブジェクトの動作を定義する 3 つのプロセス

Processing ステーションでは、搬入時間はゼロだが処理時間が必要となるため、以下のように異なるロジックを使用する：

- Create Process ボタンをクリックして新しいプロセスを作成し、ProcessingTransferIn と名づける。Description に「Delay for processing.」と入力する。
- このプロセスは、エンティティがステーションに進入するたびに自動生成される Processing.Entered イベントによって開始される。そのため、プロセスの Triggering Event に Processing.Entered を設定する。
- EndTransfer、Delay、Transfer の各ステップをこの順序で設置する。
- 搬入時間がゼロのため、EndTransfer ステップは搬入を即座に完了させる。搬送完了時、Processing ステーションに空きがあれば新しい搬送が開始される可能性がある。ただし、このオブジェクトの処理能力は 1 に固定されているため、現在のエンティティが搬出されるまで次の搬入は発生しない。
- DelayTime に ProcessingTime を設定する。これはユーザが指定する処理時間となる。F2 キーを押して Delay ステップの名前を Processing に変更する。
- 処理完了後、トークンは Processing ステーションから OutputBuffer ステーションへエンティティを搬送する。Transfer ステップでは、From プロパティ

を CurrentStation、To プロパティを Station、Station Name プロパティを OutputBuffer に設定する。

ProcessingTransferIn プロセスの完成図は、図 11.9 最下段に示されている。

OutputBuffer ステーションのプロセスはもっとも単純で、エンティティをオブジェクトから退出させる機能のみを持つ：

- Create Process ボタンをクリックし、最後のプロセスを作成する。OutputBufferTransferIn と名づけ、Description に「Wait for transferring out.」という説明を追加する。
- このプロセスを実行するトークンは、エンティティがステーションに進入するたびに自動生成される OutputBuffer.Entered イベントにより開始される。そのため、プロセスの Triggering Event を OutputBuffer.Entered に設定する。
- EndTransfer と Transfer ステップをこの順序で設置する。
- EndTransfer ステップは、搬入時間がゼロであるため、搬入を即座に完了させる。
- トークンは、OutputBuffer から、Output という名称の ParentExternalNode で定義された関連ノードオブジェクトへエンティティを搬送する。Transfer ステップでは、From プロパティを CurrentStation、To プロパティを ParentExternalNode に設定する（注：External Node Name プロパティは、外部ビューの定義後に指定可能）。

OutputBufferTransferIn プロセスの完成図は、図 11.9 の中段に示している。

11.3.2 外部ビュー

この項では、外部ビューを定義する。Definitions ウィンドウの External パネルをクリックし、以下の設定を行う。

- シンボルライブラリから、Lathe（旋盤）のシンボルを外部ビューの中央に配置する。
- Draw リボンから External Node（外部ノード）を旋盤の左側に 1 つ追加する。このノードは入力専用であるため、Node Class を BasicNode に設定し、Input と命名する。
- External Node をもう 1 つ旋盤の右側に追加する。出力側に能力を付与するため、Node Class を TransferNode に設定し、Output と命名する。
- Input 外部ノードの Input Location Type を Station とし、ドロップリストから InputBuffer を選択する。到着したエンティティは、この外部ノードシンボルの関連ノードオブジェクトに到着次第、InputBuffer ステーションへ搬送される。
- InputBuffer ステーションのアニメート表示のため、Animated Queue を旋盤シンボルの左側に追加し、Queue State を InputBuffer.Contents にする。
- OutputBuffer ステーションのアニメート表示のため、Animated Queue を旋盤シンボルの右側に追加し、Queue State を OutputBuffer.Contents にする。

・Processing ステーションのアニメート表示のため、旋盤シンボルの隣に Animated Queue を追加し、Queue State を Processing.Contents にする。そして、8 章で説明した Shift キーを押しながらドラッグする手法を使用して、この待ち行列を上方へ移動し、旋盤の上部に配置する。

Lathe の外部ビューを、図 11.10 に示す。

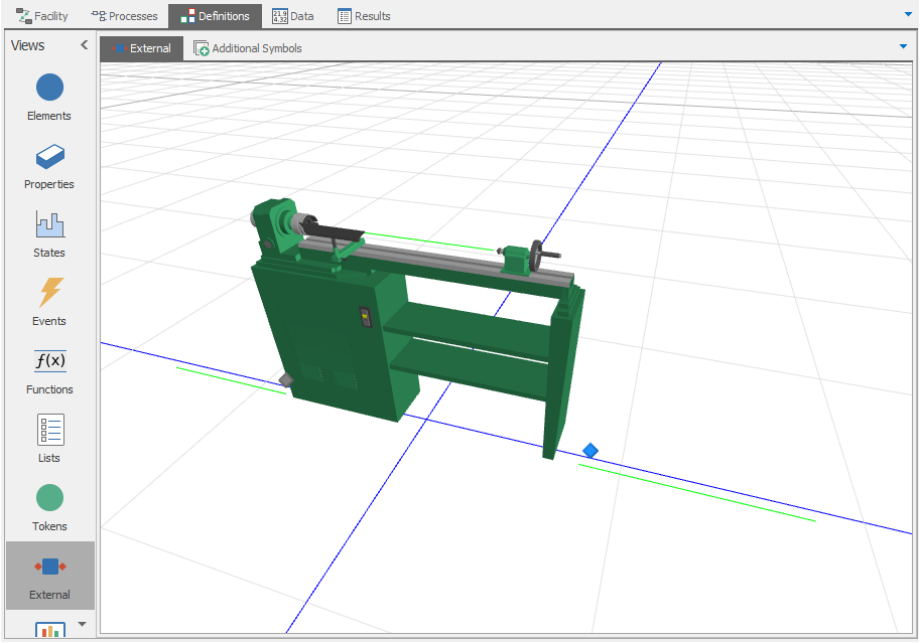


図 11.10　Lathe オブジェクトの外部ビュー

外部ノードの定義が完了したので、Process ウィンドウに戻り、OutputBuffer Transfer ステップで、Output という名称の ParentExternalNode を指定する。

11.3.2.1　本項のまとめ

これで新しいオブジェクト定義を利用する準備が整った。Navigation ウィンドウで Model をクリックしてモデルをアクティブにし、Source、Lathe（Project Library から選択）、Sink を順にドラッグして配置し、これらを Path で接続する。Lathe のプロパティを設定してモデルを実行すると、図 11.11 のような結果が表示される。

11.4　Model 11-3：オブジェクトのサブクラス化

本節では、MRI（磁気共鳴撮像装置）という医療機器の新しいオブジェクト定義を例として構築する。この装置の動作は Standard ライブラリの Server と類似しているため、Server をサブクラス化して MRI を構築する。具体的には、一部のプロパティの非表示化や名称変更を行い、MRI の修理に必要な修理担当者を選択するための新しいプロパティを定義する。さらに、修理担当者の占有・解放を制御する修理ロジックを実装する必要がある。

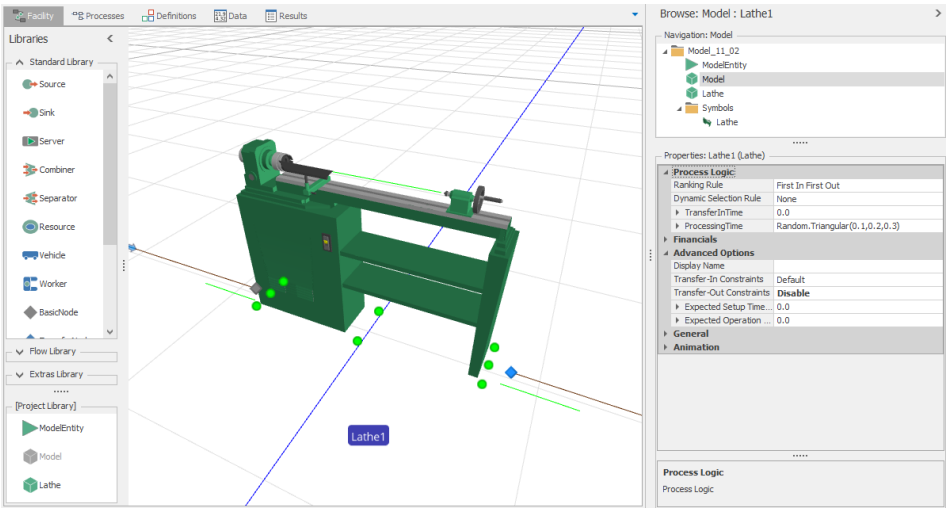


図 11.11　Lathe オブジェクトのモデルでの活用

11.4.1　モデルロジック

新しいプロジェクトを開始し、Facility ウィンドウで Server を右クリックして、Subclass を選択する（Project ウィンドウでも同様の操作が可能）。これにより、MyServer という名前の新しいモデルがプロジェクトに追加されるため、その名称を MRI に変更する。

次に、ユーザが利用する修理担当者を指定できるように、新しいプロパティを追加する：

- MRI をクリックしてアクティブモデルし、Definitions ウィンドウの Properties パネルをクリックする。
- 新しい Object Reference（オブジェクト参照）プロパティを追加し、RepairPerson と命名して、カテゴリを Reliability Logic にする。
- Switch Property Name を FailureType、Switch Condition を NotEqual、Switch Value を NoFailures に設定する。ユーザが NoFailures を選択した場合、このプロパティは非表示になる。
- Filter to Resources プロパティを True に設定する
- Required Value フラグを False に設定する。

このプロパティの定義を図 11.12 に示す。

次に、このオブジェクトで利用される用語を、汎用的な表現から医療専門用語に変更する。ProcessingTime プロパティの Display Name を Treatment Time に変更し、Description を「The time required to process each patient.」に変更する。変更後の ProcessingTime プロパティを図 11.13 に示す。また、Capacity Type、WorkSchedule、InitialCapacity という 3 つの継承プロパティを非表示にする。

Properties: RepairPerson (Object Property)

Value	
Default Value	null
Switch Property Name	**FailureType**
Switch Condition	**NotEqual**
Switch Value	**NoFailures**
Filter to Resources	**True**
Object Type	
Default Value Instantiati...	None
Appearance	
Display Name	**RepairPerson**
Category Name	**ReliabilityLogic**
Expanded	False
Parent Property Name	
General	
Name	**RepairPerson**
Description	
Required Value	**False**
Visible	True

図 11.12　RepairPerson のプロパティの定義

Properties: ProcessingTime (Expression Property)

Value	
Default Value	**Random.Triangular(.1,.2,.3)**
Switch Property Name	**ProcessType**
Switch Condition	Equal
Switch Value	**SpecificTime**
Candidate References	False
Unit Type	Time
Default Units	**Minutes**
Appearance	
Display Name	**Treatment Time**
Category Name	**Process Logic**
Expanded	False
Parent Property Name	
General	
Name	ProcessingTime
Description	**The time required to process each patient.**
Required Value	True
Visible	True

.....

Description

The description for this property.

図 11.13　ProcessingTime を Treatment Time に変更するプロパティの定義

そして、メンテナンス中に修理担当者を活用するため、継承されたプロセスロジックを修正する。まず、Processes ウィンドウをクリックして、ベースとなっ

た Server の定義から継承したプロセスを表示する。この時点で表示されているプロセスは、Server が所有し MRI が継承して使用するため、直接変更することはできない。この継承関係は、プロセス名の左側にある緑色の矢印で示されている。

最初の継承プロセスである FailureOccurrenceLogic を選択し、Process リボンの Override をクリックする。これにより、Server から継承したプロセスのコピーが作成され、MRI で使用できるようになる。この上書きされたプロセスは自由に編集が可能で、上書きの状態は左側の下向き矢印アイコンで確認できる。なお、Processes リボンの Restore をクリックすることで、いつでも元のプロセスに戻すことができる。

それでは、このプロセスのロジックを変更する。TimeToRepair という名称の Delay の前後に、Seize ステップと Release ステップをそれぞれ追加する。これらのステップで、RepairPerson プロパティをオブジェクトとして占有・解放できるように設定する（右クリックして Set Reference Property を使用）。図 11.14 では、修正したプロセスの先頭部分を示している。

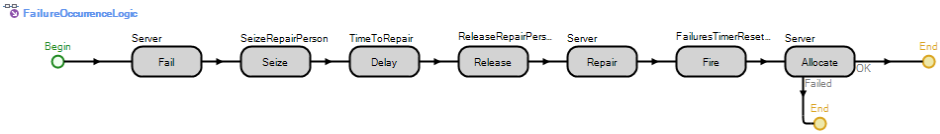


図 11.14　修理担当者の Seize/Release ステップを追加した FailureOccurrenceLogic

11.4.2　外部ビュー

次に、MRI の外部ビューをカスタマイズする。これまでのオブジェクトと同様に、ユーザの視点からアニメーションをカスタマイズする必要がある。Server オブジェクトから継承されているため、ノードとフレームワークについては考慮する必要はない。Definitions ウィンドウの External パネルをクリックし、Place Symbol ドロップリストから Download Symbol を選択する。次に、Trimble 3D Warehouse から MRI の画像をダウンロードする。InputBuffer.Contents、Processing.Contents、OutputBuffer.Contents のアニメーション表示のために待ち行列を追加する。なお、2 つの外部ノードシンボルは Server から継承されているため、位置の調整以外の変更はできない。

11.4.2.1　本項のまとめ

これで新しい MRI オブジェクト定義を利用する準備が整った。Model をクリックしてメインモデルに戻り、Source、MRI（Project Library から選択）、Sink を順にドラッグして配置し、それらを Path で接続する。MRI のプロパティを設定してモデルを実行する。実行中のサンプルモデルを図 11.15 に示す。

11.5　ユーザ拡張機能の利用

本書のこれまでの内容で、Standard ライブラリを使用したモデルの構築、アドオンプロセスによる Standard ライブラリの拡張、そして（本章で説明した）カスタムオブジェクトの構築について学んだ。さらに、中級程度のプログラミング

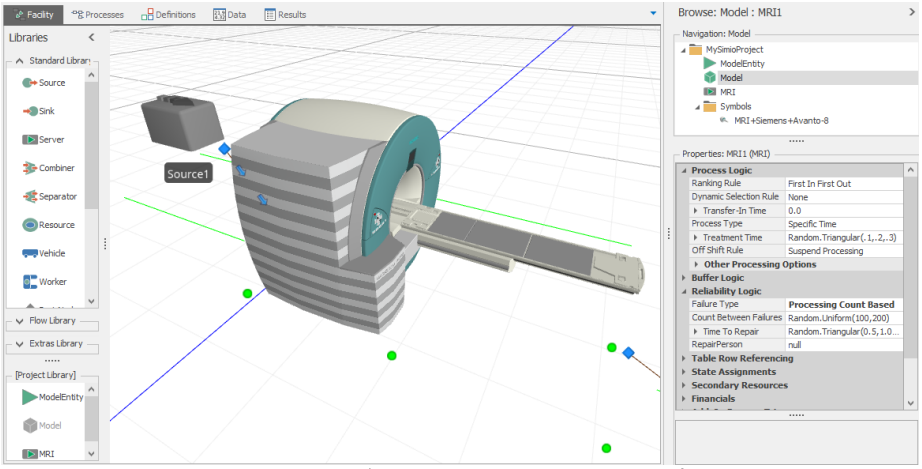


図 11.15　新しい MRI オブジェクトの実装とカスタムプロパティの設定

能力があれば、Simio の機能をより発展させ、内蔵シミュレーションエンジン自体を拡張できる。Simio のアーキテクチャは、Visual C#や Visual Basic.Net などの.Net 言語で作成したカスタム機能を、システムのさまざまな部分に統合できる。

サポートされているユーザ拡張機能には、以下の種類がある：

・ユーザ定義ステップ
・ユーザ定義エレメント
・ユーザ定義選択ルール
・走行・操縦挙動
・設計時アドイン
・スケジューリングアドイン
・テーブルのインポートとバインディング
・実験の設計・実行

Simio 拡張機能は、一連のインタフェースとして公開している。これらのインタフェースには、ユーザが作成した拡張コンポーネントが実装すべきメソッドと呼び出し規則を定義している。このインタフェース群は **Simio API**（Application Programming Interface）と呼ばれる。Simio API の詳細については、Simio のインストールディレクトリにあるファイルを参照してほしい(通常は `C:¥Program Files(x86)¥Simio` にある)。

.Net プログラミング言語を使用して拡張機能を開発する場合、Simio は C#ユーザに特に充実したサポートを提供している。開発をスムーズに始められるよう、Microsoft Visual Studio で使用できる多数の定義済み Simio プロジェクトとプロジェクト項目テンプレートを用意している。これらのテンプレートは再利用可能で、カスタマイズも容易だ。テンプレートを活用することで、新規プロジェクトやアイテムをゼロから作成する必要がなく、開発時間を大幅に短縮できる。

また、Simio の標準インストールには、さまざまなユーザ拡張機能のサンプルが含まれている。このサンプルは参考例として利用できるほか、独自の要件に合わせてカスタマイズすることも可能だ。サンプルには以下のものが含まれている：

・Binary Gate：ゲートを通過する流れを制御する 1 つエレメントと 3 つのステップ。
・TextFileReadWrite：テキストファイルの読み書きを行う 1 つのエレメントと 2 つのステップ。
・DbReadWrite：データベースファイルの読み書きに使用する 1 つのエレメントと 4 つのステップ。
・ExcelGridDataProvider：テーブルと Excel ファイル間のデータ転送をサポート。
・CSVGridDataProvider：テーブルと CSV ファイル間のデータ転送をサポート。
・SelectBestScenario：実験データ分析アドインの機能を提供。
・SimioSelectionRules：Simio の動的選択ルールの実装一式。
・SourceServerSink：コードを使用して基本的な Facility モデルを作成する設計時アドイン。
・SimioTravelSteeringBehaviors：フリースペース内でのエンティティの移動制御をサポート。
・SimioScheduling：スケジューリングアプリケーション用のリソース、リスト、テーブルを割り当て。

これらのサンプルは、Simio 例題モデルの UserExtensions サブフォルダにある。例題モデルのフォルダは、Public もしくは All Users フォルダ内にある。

11.5.1　ユーザ拡張機能の作成・展開の手順

ユーザ拡張機能を作成・展開する際の推奨手順は以下のとおりである：

・Visual Studio で新規.Net プロジェクトを作成するか、Simio Visual Studio テンプレートを使用して既存のプロジェクトにアイテムを追加する。Microsoft は、Visual Studio の商用バージョンに加え、無料の Express 版と Community 版を提供している（www.microsoft.com/express/Windows から入手可能）
・ユーザ拡張機能を実装し、.Net アセンブリファイル（.dll）をビルドする。
・拡張機能を展開するには、.dll ファイルを [MyDocuments]¥SimioUserExtensions フォルダにコピーする。このフォルダが存在しない場合は作成する必要がある。また、[Simio のインストールディレクトリ]¥UserExtensions にコピーすることも可能だが、適切な権限があることを確認する。

正しく展開された拡張機能は、Simio の適切な場所に自動的に表示される。拡張機能は、状況に応じてユーザアドインとして認識される。以下に例を示す：

・モデルの Processes ウィンドウでは、すべてのユーザ定義ステップが左側の User Defined パネルに表示される。
・モデルの Definitions ウィンドウで Elements を定義する際は、ユーザ定義エレメントがリボンの Elements タブの User Defined ボタンから利用できる。

また、他の場面でも、Simio に新しい機能が追加されたように表示される：

・ユーザ定義選択ルールはモデル内で動的選択ルールとして利用できる。
・アプリケーションアドインは Project Home タブの Select Add-In ボタンからプロジェクトに追加する。
・テーブルを作成すると Table リボンの Bind To ボタン配下に Table Imports and Binding アドインが表示される。

Simio のヘルプで「extensions」（拡張機能）または「API」を検索すると、拡張機能に関する基本的な情報を確認できる。詳細な技術情報は Program Files の Simio フォルダにある SimioAPI Reference Guide.chm（500 ページ以上）を参照してほしい。現時点では、ユーザ拡張機能の作成に関する専用のトレーニングコースはないが、Learning Simio のスライド付録にステップ・バイ・ステップの解説がある。また、Simio Insider（https://forum.simio.com/）に登録すると、追加のサンプルや Shared Items、API に関する議論を閲覧できる。

11.6 まとめ

本章では、Simio でのオブジェクト定義の構築に関する基本的な概念を復習した。Simio の重要な特徴の一つは、プログラマでなくても特定のモデルや応用領域向けのカスタムオブジェクトを構築できることである。

本章の例題では、Project Library からオブジェクトを配置する方法を示した。もう一つの方法として、独自のプロジェクトでオブジェクト定義を構築し、Project Home リボンからそのプロジェクトをライブラリとして読み込むこともできる。

Simio を使用する際、アドオンプロセスを追加した Standard ライブラリを使用するか、カスタムロジックを持つ新しいライブラリを作成するかの選択に直面することがある。同様の応用事例に頻繁に遭遇する場合は、カスタムオブジェクトを構築する方が効率的で、アドオンプロセスを繰り返し読み込む手間が省ける。なお、独自のオブジェクトでもアドオンプロセスを活用できる。これは、プロセス名をプロパティとして定義し、プロセス内の Execute ステップにそのプロパティを渡すだけで実現できる。

オブジェクトの定義は、経験を重ねるにつれて容易になり、モデリング作業を効果的に支援する強力なツールとなることを実感できるだろう。

11.7 問題

1. モデルロジックを作成する 3 つの方法について比較し、特定のオブジェクトを作成する際にどの選択肢を選ぶべきか、その根拠とともに説明しなさい。
2. External View と Console について、それぞれの特徴と制約を比較しなさい。
3. 塗装ラインのモデルを構築している場合を考える。このモデルを、より大規模なモデル（例：複数の異なる塗装ラインを持つ自動車工場）のオブジェクトとして再利用可能にするために必要な一般的な変更点を説明しなさい。
4. Model 11-2 を **Fixed Class** の **Processor** を使用して再作成しなさい。このアプローチの長所と短所について考察しなさい。

5. Model 11-2 に時間基準の故障を追加しなさい。Time Between Failures、Time To Repair、Repair Resource の選択肢を含め、修理担当者が作業のために訪れるよう外部ノードを設定しなさい。作成したオブジェクトの有効性を検証するため、これを使用して新しいモデルを構築しなさい。
6. 入力ノード 1 つ（箱の搬入用）と出力ノード 1 つ（パレットに積まれた箱の搬出用）を持つ Palletizer オブジェクトを作成する。パレットは Palletizer 内で必要に応じて生成され、箱はユーザが指定する数量でパレットに積み込まれる。適切なアニメーションを設定する。
作成した Palletizer オブジェクトを使用して、2 種類の箱を扱うモデルを構築する。ただし、異なる種類の箱は同じパレットに積み込まない。満載になったパレットは出荷エリアへ搬送され、そこで箱がパレットから降ろされ、すべての箱とパレットの数がカウントされる。
7. 実験ウィンドウで使用するアドインを独自に作成する。このアドインは、外部データファイルからシナリオのデータを読み込み、Experiment Design ビューに適切なシナリオを作成する機能を持つ。
8. 醸造所の抽出プロセスをシミュレートするため、離散バッチ処理のモデル化が必要である。このシステムには多数の発酵タンクがあり、長時間の加工・処理を行う。全体のモデルを効率的に構築するため、発酵タンク（FermentationTank）オブジェクトを作成する。
実験を容易にするため、システムの小規模なサブセットを設計する。このサブセットは、図 11.16 に示すスケジュールに従ってビールのバッチを生成する。検証システムでは各バッチの発酵時間を約 2 時間とする（実際の発酵は数日かかる）。発酵タンクは、同じ種類のビール（Beer Type）で、かつ 2 番目のバッチが最初のバッチ到着から 60 分以内に到着する場合、最大 2 つのバッチを受け入れることができる。2 つのバッチをまとめて処理する場合、最初のバッチが単独で完了する時間で終了する。ただし、異なる種類のビールは決してまとめて発酵できない。
これらの特徴を備えた発酵タンクオブジェクトを作成し、図 11.16 の到着スケジュールを用いて単一の発酵タンクによるシンプルなモデル実験を行う。テーブルコメントで指定された結果が得られることを確認する。
注：テーブルコメントは次のとおり：1）即座に処理を開始、2）前のバッチとともに処理、3）次のバッチを待機、4）前のバッチとともに開始、5）バッチが満載のため、次のバッチを待機、6）異なる種類のため、次のバッチを待機、7）到着が遅いため、次のバッチを待機、8）前のバッチとともに開始。

Batch Production

	Job Number	Beer Type	ArrivalTime (Minutes)	Comment
1	1	Light	0	Start processing immediately
2	2	Light	30	Process with previous
3	3	Dark	60	Wait for next batch
4	4	Dark	120	Start with previous
5	5	Dark	120	Batch full, wait for next batch
6	6	Light	130	Wrong type, wait for next batch
7	7	Light	425	Too late, wait for next batch
8	8	Light	445	Start with previous

図 11.16　醸造所のバッチ到着スケジュール（テストデータ）

第 12 章　インダストリー4.0 時代の シミュレーションスケジューリング

1 章で述べ、本書全体で説明したように、シミュレーションは主にシステムの設計改善に活用されてきた（例：代替案の比較、パラメータの最適化、パフォーマンスの予測）。システム設計の改善はたしかに重要だが、効率的なシステムの**運営**においても多くの課題が存在する。

スマートファクトリーの構想は、近年、必要な技術が飛躍的に進歩し、世界的な注目を集めている。この新しいアプローチによる工場運営において、残された課題の多くは、シミュレーション技術の発展と新領域への応用によって解決できる。

本章では、まずインダストリー4.0 の発展と現状、そしてその課題と可能性について説明する。次に、スマートファクトリーにおけるシミュレーションの従来の役割に加えて、デジタルツイン領域での応用を検討する。さらに、計画とスケジューリングに用いられる主要技術を概説し、リスクに基づく計画とスケジューリング（RPS：Risk-based Planning and Scheduling）について解説する。なお、本章の一部の資料は Simio の Web サイトでも公開されており、許諾を得て本書に再録している（Simio LLC 2018）。

12.1　産業革命の進化と変遷

インダストリー4.0 は、「完全に接続された自動化製造システム」への現在のトレンドを指す一般的な呼称である。世界経済フォーラムの創設者兼会長であるクラウス・シュワブ氏は、著書『The Fourth Industrial Revolution（第四次産業革命）』の中で、『私たちは、生活や仕事の仕方、さらには他者との関わり方を根本から変える大変革の入口にいる』と述べている（Schwab 2017）。この声明は 2017 年初頭になされたが、興味深いことに、第一次産業革命が文明を再定義する 200 年前にも同様の見解が示されていた。

では、インダストリー4.0 の詳細に入る前に、私たちが現在いる時代について考察してみよう（図 12.1）。

12.1.1　第一次産業革命：機械生産

1760 年代以前、世界中の人々は完全に閉鎖的な環境で暮らしており、限られた技術と食料しか持たず、家族の伝統に従って変化のない生活を送っていた。しかし、英国を発祥として、蒸気機関と蒸気動力の発明・改良により、農業が急速に発展し、食料が豊富になったことで人口が増加し、新たな市場の創出につながった。

工場は水車のある場所に限らず、どこにでも設立できるようになり、工業化が急速に広がっていた。これに伴い、消費者があふれる都市が形成され、鉄道網の発展により、さらなる拡大と都市化が進んだ。従来型の農業は次第に衰退し、繊維、鉄鋼、食品などの製品生産が主流となった。新たな雇用が生まれ、メディア、交通システム、医療サービスといった分野が革新され、人々の日常生活は大きく変容した。

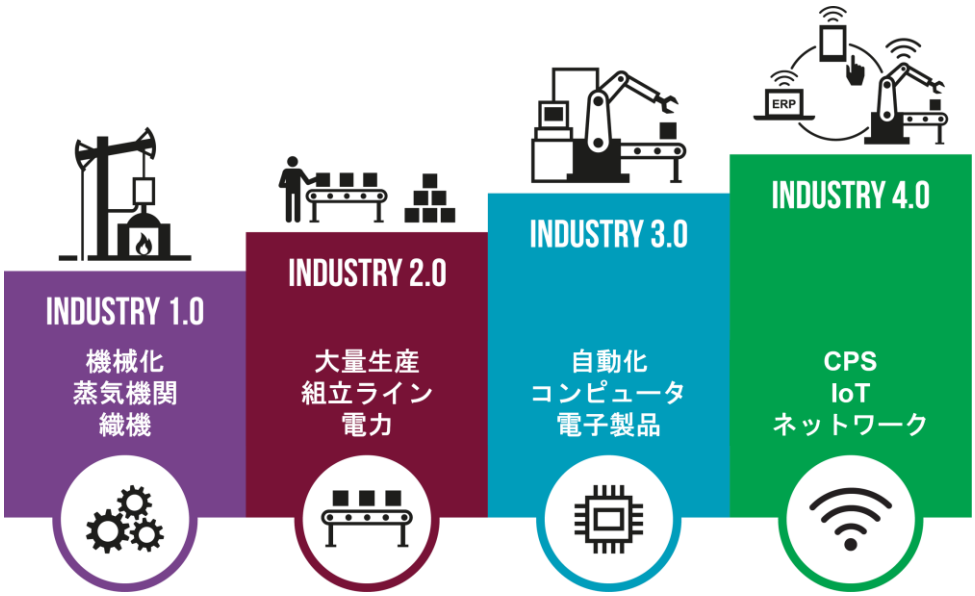


図 12.1　産業革命の歴史

こうした変革は約 80 年間にわたり進行し、1800 年代半ばまでにヨーロッパやアメリカ大陸にその影響が広がった。

12.1.2　第二次産業革命：大量生産

20 世紀の幕開けと共に、最新の発明と電力の普及を追い風に、人口増加と都市化が加速した。第二次産業革命では、組立ラインによる食品、衣料、物資の大量生産が発展の契機となり、輸送、医療、軍需産業も大きく発展した。この時代はヨーロッパが主導的な立場を維持していたものの、他の国々も独自の工業力と技術を築き上げていった。

12.1.3　第三次産業革命：デジタル時代

1960 年代には、二度の世界大戦を経て、文明は平穏を取り戻した。戦時中に開発された技術の民生利用が新たなイノベーションを生み、平和な時代における建設的な展望をもたらした。シリコンウエハーの実用的価値が見出され、これが半導体とメインフレームコンピュータの発展の基礎となった。その後、1970 年代から 80 年代にかけて、コンピュータ技術はパーソナルコンピュータという形で一般に普及していった。1990 年代にはインターネットが世界中に広がり、デジタル革命はグローバル規模で着実な進展を遂げた。

12.2　第四次産業革命：スマートファクトリー

第四次産業革命は人類史上最大の飛躍的な成長とされ、今後 20 年間で世界の GDP に 10～15 兆ドルもの増加をもたらすと予測されている（Visual Components 2015）。ドイツで考案された**インダストリー4.0（Industry 4.0）**は、**Industrie 4.0**、**スマートファクトリー**（Smart Factory）、または**スマートマニュファクチャリング**（Smart Manufacturing）とも呼ばれている。この呼称は、

企業の製品設計、製造、サポート方法に革新的な変革をもたらすことに由来する。仮想世界と現実世界の融合は、製造業に新たな成長の原動力をもたらしている。人件費削減と生産性向上を実現した次のステップとして、製造業のデジタルトランスフォーメーションが進められている。この新しい製造システムでは、統合的にリンクされた機械と人員から得られるリアルタイムの情報に基づき、生産に関する意思決定を最適化することが可能となる。

インダストリー4.0 を構成する主要なデジタル技術は以下のとおりである：

・モノのインターネット（Internet of Things：IoT）および産業用モノのインターネット（Industrial IoT：IIoT）
・ロボティクス
・クラウドコンピューティングと SaaS（Software as a Service；サービスとしてのソフトウェア）
・ビッグデータと高度なデータ分析
・付加製造（3D プリンティング）
・システム統合
・AR/VR（拡張現実とバーチャルリアリティ）
・シミュレーション
・IT セキュリティとサイバーセキュリティ

インダストリー4.0 は、これらの技術を組み合わせることで、プロセスの大幅な改善と生産性の最適化を実現する。こうしたイノベーションは、情報とサービスを基盤とした革新的なビジネスモデルの開発へとつながっていく。

現在、多くの製造業者にとってインダストリー4.0 は目指すべきビジョンの段階に留まっているが、実際には先進的な製造ロボットや 3D プリンティングの活用を通じて、業界のデジタルトランスフォーメーションはすでに進行している。特にプラスチックと金属の分野では、生産の柔軟性、品質、安全性、スピードにおいて、具体的な改善が実現されている。一方で、情報管理、生産スケジューリング、サイバーセキュリティの分野では新たな課題も浮上している。

インダストリー4.0 は当初の構想を超えて発展し、単品から数千個規模のバッチまで、同じリードタイムとコストで生産できる新時代を創出している。さらに、消費者により近い場所に小規模で機動力のある工場を設立する動きが加速しており、これは地域貿易の促進につながる大きな経済的転換となっている。

インダストリー4.0 は製品のライフサイクルとバリューチェーン全体に適用可能であり、すべての利害関係者に影響を及ぼす可能性を持つ。ただし、その成功はシステム全体における効果的な情報流通にかかっている。そのため、データの収集、計画、スケジューリングには、最先端のシミュレーションとスケジューリングソフトウェアが重要な役割を果たすことになる。

12.2.1　次は？

コラボレーションと知識共有により新たな発見が続々と生まれ、現代社会にはかつてない速度と推進力が集約されている。デジタル革命を土台として、今日で

は、広範なモバイルインターネット、高性能な記憶装置、そして強力なコンピューティング資源が整備されている。これに加えて、小型で高性能なデバイスやセンサーの活用により、ロボティクス、ビッグデータ、シミュレーション、機械学習、人工知能といった技術を駆使し、より高度な知見を得ることが可能である。

スマートファクトリーは、物理システムと仮想システムの連携を実現し、予兆診断と生産性向上に向けた新しい運用モデルを構築でき、さらにそれをデジタルツインとして表現できる。これらの可能性を現実のものとし、潜在的な解決策を実践的な成果へと転換するには、さらなる取り組みが必要である。

さらに、ナノテクノロジー、再生可能エネルギー、遺伝学、量子コンピューティングなどの分野でも目覚ましい進展を遂げており、この最新の革命は物理・デジタル領域を生物学的側面から融合させようとしている。第一次産業革命のゆるやかな進展と比較すると、現代の変革速度は驚くべきものである。テクノロジーはグローバルに接続されたデバイスとともに急速に進化し、世界を一つにつなぎ、技術やアイディアの交換を通じて、異分野間での革新を促進している。

しかし、第一次産業革命という小さな一歩がなければ、私たちは今なお、移動や探検の機会に乏しく、限られた食料供給に依存し、健康リスクに脆弱で、革新の見込みのない孤立した生活を送っていたことだろう。確かに今日、重大な変革と国際的な発展は避けられない現実となっているが、私たちはこの発展を受け入れ、歴史の歩みの中で自らの視野を広げていく必要がある。

クラウス・シュワブ氏が指摘するように、『これは人類がこれまで経験したことのない、規模、範囲、複雑さの大変革』となるだろう（Schwab 2017）。

12.3　デジタルツインの必要性

消費者環境では、**モノのインターネット**（IoT）が接続されたデバイスのネットワークを提供し、デバイス間の安全なデータ通信による協調動作を実現する。製造現場では、産業用モノのインターネット（IIoT）により、製造プロセス内の機械と製品が相互に通信を行い、効率的な生産を実現する。インダストリー4.0 への移行が進むにつれ、デジタル化の進展は製造業に固有の課題や懸念をもたらしているが、その解決策の一つとしてデジタルツインの活用が挙げられる。

デジタルツインとは、製品、部品、システム、またはプロセスの仮想的な表現である。デジタルツインにより、実物が存在する前からその動作や性能を確認できる。この概念はデバイスレベルにも適用され、デバイスのデジタルツインは実機とほぼ同様に仮想空間で動作する。さらに重要な応用例として、工場全体のデジタルツインがあり、これは現実の工場の動作を仮想空間で再現する。この工場全体のデジタルツインは、従来の定義を超えた概念だが、決して実現不可能ではない。デザイン指向モデルで述べたように、私たちの目標は「完璧なモデル」の作成ではなく、目的達成に必要な「じゅうぶんに近い」モデルの作成である。では、この目標をどのように達成できるか探ってみよう。

一部の実務家によると、デジタルツインと呼べるのは、必要なデータを含むすべてのシステムと完全に接続されている場合のみである。スタンドアロンのシミ

ュレーションモデルは「バーチャルファクトリーモデル」と呼ばれ、ERP、MES などのシステムからのデータによってリアルタイム（もしくはほぼリアルタイム）で駆動され、完全に接続されるまでは、デジタルツインとは見なされない。そのため、デジタルツインとして効果的に機能するには、7.8 節で説明したように、データからモデルを生成できることが重要となる。これにより、最新データのインポートだけで、リソースやマシンの追加、モデルとスケジュールの自動作成など、データの変更に応じてモデルが自動的に対応できるようになる。

インダストリー4.0 時代では、技術革新の指数関数的な成長により、かつてない規模でデータの収集・保存・操作が可能になっている。小型センサー、低価格なストレージ、高速プロセッサがワイヤレスでネットワークに接続されることで、物体をデジタル世界に投影する動的シミュレーションとモデリングが容易になった。この仮想モデルは、運用データ、履歴データ、環境データを受け取ることができる。

技術の進歩により、大量のデータ収集と共有が飛躍的に簡単になり、モデルへの適用も容易になった。これにより、さまざまなシナリオの予測と評価が可能になっている。ただし、重要なリソースのデジタルモデリングと同様、デジタルツインにおいてもデータセキュリティは常に重要な考慮事項である。

デジタルツインは実物資産のデジタル版として、多様な目的に活用できる。たとえば、残存耐用年数の決定、故障やプロジェクトパフォーマンスの予測、財務収益の見積もりなどが可能である。このデータを活用することで、設計、製造、運用を最適化し、潜在的な機会から価値を創出できる。

有用なデジタルツインを実装するには、以下の 3 段階のプロセスが必要となる：

- **モデルを確立する**：3D ソフトウェアを使用して対象物をシミュレートし、デジタルデータを実物に統合する。実物との間で相互作用を確立し、すべての関連パラメータがモデルと通信を行う。入力されたデータを基に、モデルは類似性を通じて動作方法を「学習」する。
- **モデルをアクティブにする**：シミュレーション実行により、モデルは既存データと新規データに基づいて継続的に更新される。履歴、他のモデル、接続デバイス、予測、コストなどの情報源からデータを取得し、リスクと信頼性に関する知見を得るためにシナリオを順次実行する。
- **モデルから学習する**：モデルから得られた解決策に基づき、稼働率、コスト、パフォーマンスを最適化する計画を実施する。これにより、潜在的な問題領域や収益性の低い状況を回避できる。

デジタルツインは、単なるデバイスやシステムの設計図や概略図ではない。それは、これらの要素が他の要素や環境とどのように動的に相互作用するかというふるまいを含む、実践的な仮想表現である。この要素を監視し、診断・予測を改善し、問題の根本原因を調査することで、性能と生産性を全体的に向上させ、大きな利益をもたらすことができる。

適切に構築されたデジタルツインを活用することで、製品ライフサイクルの各

段階（設計・構築・運用）において、運用環境を動的に調整し、好ましい影響を与えることができる。このような応用では、デジタルツインモデルの作成前に、目的と期待を明確に理解しておく必要がある。そうすることで初めて、目的に適い、意図した便益を反映する忠実なモデルを作成することが可能となる。便益の例は次のとおりである：

・設備は自身の状態を自動監視し、メンテナンスのスケジューリングや、交換部品の自動発注が可能となる。
・混流生産方式により、納期を守りながら設備稼働率を最大限に高めるスケジューリングが可能となる。
・設備構成が変更された場合でも、納期を優先しながら負荷を最適化する迅速な再スケジューリングにより、損失を最小限に抑えることができる。

12.4　インダストリー4.0 におけるデザインシミュレーションの役割

数十年にわたり、シミュレーションは主に施設設計の改善に活用されてきた。1.2 節では、シミュレーションが日常的に活用されている主要な応用分野を以下のように示した：

・**サプライチェーンロジスティクス**：ジャストインタイム生産、リスク低減、発注点管理、生産配分、在庫水準の最適化、緊急対応計画、経路評価、情報フロー、データモデリング
・**輸送**：原材料・労働力の輸送、配車計画、交通管理（列車、船舶、トラック、クレーン、リフトトラック）
・**人員配置**：スキルレベルの評価、人員配置の最適化と割当、トレーニング計画、スケジューリングアルゴリズム
・**設備投資**：成長戦略への投資、投資判断の最適化、投資収益率の客観的評価
・**生産性**：生産ラインの最適化、製品ミックスの調整、設備の効率的割当、人件費の適正化、キャパシティプランニング、予知保全、変動分析、分散型意思決定

シミュレーションのインダストリー4.0 における有用性を一言で表すと、「前述のすべてが当てはまる」ということだ。つまり、スマートファクトリーも基本的には工場であり、従来の工場と同様の課題を抱えている。そのため、シミュレーションは従来と同じ分野で同様の価値を提供できる。シミュレーションを活用することで、システムを客観的に評価し、最適な構成と運用に関する知見を得ることが可能だ。

ただし、スマートファクトリーの実装は「従来の工場」の枠を超え、多くの重要な点で異なる。特徴的なのは、スマートファクトリーが一般的に大規模で、より**多くの**構成要素と、より**洗練された**構成要素を備えていることだ。「洗練された」という表現について、楽観主義者は「問題が少ない」と解釈するかもしれないが、悲観主義者は「故障のリスクが高い」と捉えるかもしれない。いずれにしても、相互

作用の多い大規模システムは分析が複雑になるため、従来型のシミュレーションの重要性がさらに増す。**特定の高度な機能の影響を評価する**のは容易ではない。そのため、シミュレーションは各構成要素の相互作用と貢献度を客観的に評価し、統合システムを設計・最適化できる唯一の手段となる可能性がある。

スマートファクトリーでは、ビッグデータやクラウド運用などの IT イノベーションにより、リアルタイムデータの活用が格段に進んでいる。大量のデータを効果的に処理することは、すべてのシミュレーション製品が得意とする分野ではないものの、最新の製品ではそうしたデータをモデルに統合できる。この拡張されたデータアクセスは、システムパフォーマンスの向上をもたらすだけでなく、より多くの故障ポイントを発見し、**実装前にリスク領域を特定する**ための詳細なモデルを構築する機会も提供する。

スマートファクトリーのもう一つの特徴的な違いは、高度な自動化と自律性にある。スマートファクトリーの動的プロセスは、システム故障に対してインテリジェントに反応し、自動的な修正措置を講じる運用の柔軟性を実現する。これにより、故障の修復や、適切な経路変更による回避が可能となる。シミュレーションは、**代替案のパフォーマンスを評価する**ことで、これらの対応策の有効性を検証できる。

通常の工場と同様に、スマートファクトリーも異なる構成や設定で実際に何度も試行することはできない。一方、シミュレーションは繰り返し実験ができるように設計されており、将来の運用を効果的に予測し、**数日分の検証をわずか数秒で実現する**。また、システムの規模拡大や縮小の影響を検討する際も、シミュレーションモデルを容易に調整できる。これにより、プロセスの所要時間、機器の使用頻度、不良品の発生頻度など、システム全体のプロセスや挙動に関する基本的な問題への回答が得られる。さらに、遅延、設備稼働率、ボトルネックといった、システム改善に向けた**パフォーマンス指標を予測できる**。

この仮想工場モデルは、大規模組織にとって**データ、システム、およびプロセスの標準化**という大きな利点をもたらす。一般的に、大企業の各工場ではシステムの実装方法が異なっているため、それらを単一の ERP インスタンスに統合すると重大な問題が生じる。すべてのユーザが同一のプロセスとワークフローを使用する必要があるものの、最適なプロセスや正確なデータの判断基準が明確でないためである。この課題に対し、仮想工場モデルを用いてさまざまな運用方針とデータを検証することで、最適なグローバルプロセスを特定し、全工場の調整を効率的に進めることができる。このようなデータ生成型のシミュレーションアプローチは、グローバルに複数の工場を展開する大企業にとって、極めて有益な手法となっている。

シミュレーションの他の 2 つの利点である**ナレッジベースの確立**と関係者間の**コミュニケーションの支援**は、スマートファクトリーにおいて特に重要である。複雑なシステムの機能を説明し理解することは極めて困難だが、シミュレーションモデルの作成過程で、各サブシステムの機能を深く理解し、それをモデルに正確に表現することができる。シミュレーションモデルは、構成要素に組み込まれ

た直接的な知識と、実行結果から得られる間接的な知見の両方を含む、重要な知識リポジトリとなる。さらに、2D や 3D モデルのアニメーションは、システムの理解を促進する効果的なツールとなり、利害関係者がシステムの仕組みを把握し、問題解決に積極的に参加することで、より確実な合意形成が可能となる。

モデリング段階では、時間を要するものの、プロセスに精通したオペレータやスタッフの参加が不可欠である。この参加は当事者意識を醸成し、後の調査結果の実装段階でも大きな価値を持つ。実践的なシミュレーションは、システム全体のパフォーマンス向上を検証し理解する上で、スプレッドシートなどの従来型ツールと比べ、はるかに迅速かつ有効な手法となっている。特に、ユーザや意思決定者へのデモンストレーションにおいて、その価値が顕著に表れる。

このように、シミュレーションは以下の機能を提供する：

・システムの運用結果とパフォーマンスの予測。
・システム構成要素間の相互作用の解明。
・パフォーマンス指標の測定と比較分析。
・システム構成と全体パフォーマンスに関する知識基盤の構築。
・効果的なコミュニケーションツールとしての活用。

結論として、従来型の設計プロセスにシミュレーションを組み込むことで、スマートファクトリーの構築・導入・運用全体において競争優位性が得られる。これにより、システムの展開時間が短縮され、問題が最小化され、最適な収益性をより迅速に達成することが可能となる。

12.5　シミュレーションによるスケジューリングの役割

インダストリー4.0 の台頭により、高価で競合するリソースを含む複雑なシステムの日次スケジューリングにおいて、シミュレーションの需要が高まっている。この運用上のニーズにより、シミュレーションの価値は、従来のシステム設計改善という役割を超え、より迅速で効率的なプロセス管理と、性能・生産性の向上を実現する領域へと拡大した。特に、リスクに基づく計画とスケジューリング（RPS）などの最新技術により、システムの設計・評価のために構築されたモデルが、インダストリー4.0 環境での日次オペレーションスケジューリングにおいても、重要なビジネスツールとなっている。

デジタル化された製造環境では、コネクテッド・テクノロジーがスマートファクトリーを形成し、生産プロセスの分析と制御に有用なデータをリアルタイムで送信する。センサーとマイクロチップは機械、工具、さらには製品自体にも組み込まれ、インダストリー4.0 の工場で製造される「スマート製品」は、原材料から最終製品に至るまでの全工程においてステータスレポートを送信することが可能となっている。

製造プロセス全体にわたるデータ利用の拡大により、生産の柔軟性と即応性が向上し、小規模バッチ生産や受注生産への移行が実現可能となる。

このような適応性を最大限活用するため、インダストリー4.0 のスケジューリ

ングシステムには以下の要件が求められる：

・システムのすべての要素を正確にモデル化すること。
・スケジュールを迅速に算出すること。
・結果を容易に可視化できること。

フィジカルとデジタルの世界のギャップを埋めるため、IoT デバイス、ビッグデータ、クラウドコンピューティングといったインダストリー4.0 の特徴的な技術を活用したスケジューリングシステムの必要性が増している。

スケジューリングの手法は従来から、手動、制約ベース、シミュレーションの 3 つに分類される。

小規模または単純なシステムにおいては、ホワイトボードやスプレッドシートを用いた**手動スケジューリング**が効果的な場合がある。ただし、このアプローチは担当者の経験と判断に大きく依存する。そのため、大規模で動的な生産環境では、データ量と複雑さから現実的な選択肢とはならない。

制約ベースのスケジューリングは、システム全体の制約条件を数式化して解を求める手法である。スマートファクトリーの全要素を数学モデルとして構築できるものの、データ入力と計算に多大な時間を要する。さらに、得られた解の解釈や可視化、実装が困難な場合、実現可能にするために重要な制約条件を省略または簡略化せざるを得ない。

手動および制約ベースのスケジューリングでは、複雑さを軽減するため、多くの場合、施設の単一の部門やセクションのみを対象としてスケジューリングが行われる。このようなローカライズされたスケジュールは、セクション間にプロセスバッファを必要とし、結果として時間、在庫、そしてキャパシティの無駄を生じさせる。

シミュレーションによるスケジューリングは、インダストリー4.0 アプリケーションにおける最適なソリューションとなっている。このアプローチでは、システムの各要素をモデル化し、データを割り当てることが可能だ。設備、工具、作業者などのリソースに加え、プロセスで消費・生産される材料も表現できる。これにより、システム全体でのジョブの流れをシミュレートし、各段階におけるリソースと材料の使用状況を正確に把握し、リアルタイムでステータスを更新できる。

意思決定ロジックでは、段取り替え時間の最小化や作業者の経験に基づくカスタムルールなどをモデルに組み込むことができる。これらのアルゴリズムを組み合わせることで、システム内の材料と部品の実際の流れを正確にモデル化するルールセットが構築される。

この技術により、シミュレーションベースのスケジューリングソフトウェアは、生産プロセスのあらゆる側面における計算と最適化を実現する。デジタルワークステーションから提供される大量のリアルタイムデータと組み合わせることで、迅速かつ詳細で正確なスケジューリングが可能となる。

スマートファクトリーにおけるスケジューリングの 3 つの主要要件は、シミュレーションによるスケジューリングソフトウェアで満たされている：

・すべての要素の正確なモデリング：運用上の制約やカスタムルールを含め、システム全体を正確に再現する柔軟なモデルを構築できる。
・スケジュールの高速計算：スケジュールの作成、代替案の比較、配布までを迅速かつ正確に実行できる。
・容易な可視化：コンピュータシミュレーションにより、組織全体でスケジュールを明確かつ効果的に共有できる。

シミュレーションによるスケジューリングは、労働効率の向上という重要な利点ももたらす。生成される詳細な情報により、「スマートグラス」などの先進的な技術の活用が可能となる。スマートグラスは作業者に対してリアルタイムで詳細な指示を提供する。シミュレーションモデルは、実データと最新情報に基づいてスケジュールを継続的に評価し、各作業者に対して次に行うべきタスクの最適な手順を指示する。

このようなスケジューリングはスマートファクトリーにとって不可欠な要素だが、次節で説明するように、モデルはスケジューリング以上の重要な役割を果たすことができる。

12.5.1　デジタルツインとしてのシミュレーション

インダストリー4.0 の IT イノベーションにより、スマートファクトリーのデジタル化されたシステムから収集したデータを活用し、離散事象型シミュレーションソフトウェアで生産ライン全体をシミュレートできるようになった。在庫レベル、コンポーネント履歴、有効期限、輸送、ロジスティクスなどのリアルタイム情報をモデルに取り込み、複数の計画やスケジュール案を作成できる。これにより、潜在的な損失や中断を最小限に抑えつつ、代替的な供給元や生産計画を比較評価することが可能となる。

品切れ、機器の故障、大規模な自然災害といった変化が生じた際、シミュレーションモデルは生産工程と下流のサービスへの影響を正確に把握できる。これに基づき、対応策を手動または自動で評価し、最適な解決策を実行に移すことができる。

インダストリー4.0 環境におけるシミュレーションを用いたスケジューリングは、リスクを軽減しながら一貫した生産を実現する。これにより、コストの効果的な管理と、あらゆる状況下での品質維持が可能となる。

このスケジューリング機能により、高度なデータ駆動型シミュレーションモデルはデジタルツインとして機能する。図 12.2 が示すように、シミュレーションモデルはスマートファクトリーの中核として、すべての重要なサブシステムと連携する。これにより、計画とリアルタイムの実行情報を収集し、短期スケジュールを自動生成して、その構成要素と結果を各サブシステムへ配信することが可能となる。

高度なシミュレーションによるスケジューリングソフトウェアは、次の理由から、このようなアプリケーションに最適である：

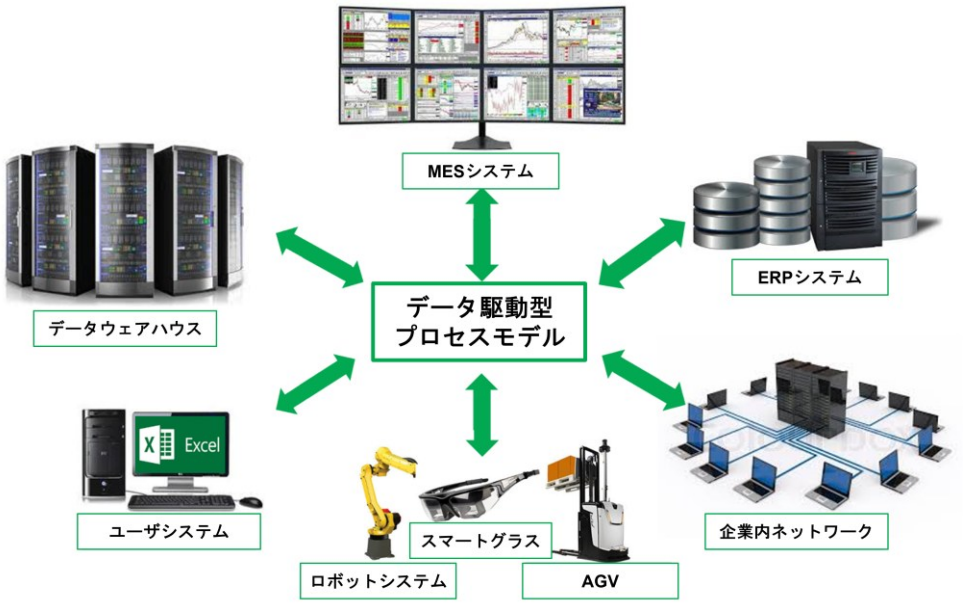


図 12.2　デジタルツインによるスマートファクトリーの実現

・サブシステムとのバッチ処理やリアルタイム通信が可能。
・工場の複雑な動作を正確にモデル化できる機能。
・高度な技術を活用した最適なスケジュールの生成。
・関係者への効果的なスケジュール共有と展開。
・計画からの逸脱をリアルタイムで監視する機能。

これにより、スマートファクトリー計画における重要な課題を解決できる。

12.6　計画とスケジューリングの課題

計画とスケジューリングは密接に関連した応用分野であり、通常は一体として議論される。**計画**は「全体像」の分析である。つまり、生産量、生産方法、そして必要な原材料とリソースの調達時期・場所・方法を検討することである。計画は一般的に、無限の原材料を想定した能力の集計ビューに基づいて行われる。**スケジューリング**は運用上の具体的な詳細を扱う。現在の生産状況、実際の生産能力、リソースの可用性、および仕掛品（WIP）を考慮し、重要な目標をもっとも効果的に達成するための優先順位・順序・戦略的決定を行う。計画が実施の数日から数ヶ月前に立てられるのに対し、スケジューリングは数分から数日前に行われることが多い。多くの現場では、これらは別々に実施される。そのため、一方のみが完了し、もう一方が無視されることで、潜在的な性能がじゅうぶんに発揮されていないケースも少なくない。

もっとも基本的な計画手法の一つは、**リードタイム**に基づくものである。たとえば、ある種の部品が過去の平均から「標準的に」発注後 3 週間で出荷されることがわかっている場合、他の要因を考慮しなければ、生産計画には 3 週間を見込めばよい。ただし、この方法ではリソースの稼働率がじゅうぶんに考慮されてい

ないことが多い。「標準」以上の仕掛品が存在する場合、実際の所要時間が予測を上回り、リードタイムが楽観的な見積もりとなる可能性がある。

もう一つの基本的な計画手法として、**マグネットボード、ホワイトボード、スプレッドシート**を使用し、部品の移動経路やリソースの利用状況を手作業でガントチャートに表現する方法がある。しかし、これは非常に労働集約的な作業となる。また、システムの複雑さと計画担当者の経験レベルによって、作成される計画の品質にばらつきが生じやすい。

三番目の計画方法は、プログラミング言語で実装されたカスタムアルゴリズムを用いた**特注のシステム**である。これは特定の領域やシステムに合わせて高度にカスタマイズされている。効果的に機能する可能性はあるものの、開発費用と実装時間が膨大になりがちで、カスタマイズの程度が高いほど再利用が難しくなる。

もっとも広く採用されている方法の一つが、**先進的計画スケジューリング**（Advanced Planning and Scheduling：APS）である。**APS** は生産需要に応じて、生産能力、リソース、原材料を最適に配分する。統合基幹業務システム（ERP）と詳細な生産スケジューリングを統合するように設計された APS 製品は数多く市販されているが、これらのソリューションには広く認識された課題がある。特に、複雑性と変動性という二つの制約要因により、ERP システムと日々の生産活動の連携がじゅうぶんに図れていないケースが多い：

- **複雑性**：一つ目の限界は、あいまいで複雑なシステムを効果的に扱えないことである。特注のシステムはあらゆるシステムを表現できる可能性があるが、その開発に必要な費用と時間が、実用的なソリューションとしての導入を妨げることが多い。システムが標準的なベンチマークに近い場合は既存の技術で対応できるが、ベンチマークと異なる場合、適切なソリューションを提供するための詳細な機能が不足する。重要だが対応できない状況として、複雑なマテリアルハンドリング（クレーン、ロボット装置、トランスポータ、作業者など）、特殊な作業とリソース割当（段取り替え、順序依存の設定、作業者など）、経験に基づく意思決定ロジックと運用規則（注文の優先順位、作業選択の規則、バッファリング、注文の順序など）がある。
- **変動性**：二つ目の限界は、システム内の変動に効果的に対応できないことである。すべての処理時間を事前に把握する必要があり、他の変動要素は通常無視される。予期せぬ中断や機械の故障は考慮されず、作業者や原材料の問題、その他の予期せぬ事態も想定されない。その結果、作成される計画は過度に楽観的なものとなる。図 12.3 は、典型的なスケジューリング出力をガントチャートで示している。緑色の破線は、完成予定日（黒）と納期（グレー）の間の余裕時間を表すが、この余裕時間がじゅうぶんかどうかの判断は困難である。当初は実行可能なスケジュールでも、変動要因や想定外の出来事によってパフォーマンスが低下し、時間とともに実行不可能となることが多い。予測スケジュールと実績との間に大きな差異が生じるのは一般的である。遅延を防ぐため、スケジューラは余分な時間、在庫、能力をバッファとして確保する必要があるが、これらはすべてシステムのコストを増加させる要因となる。

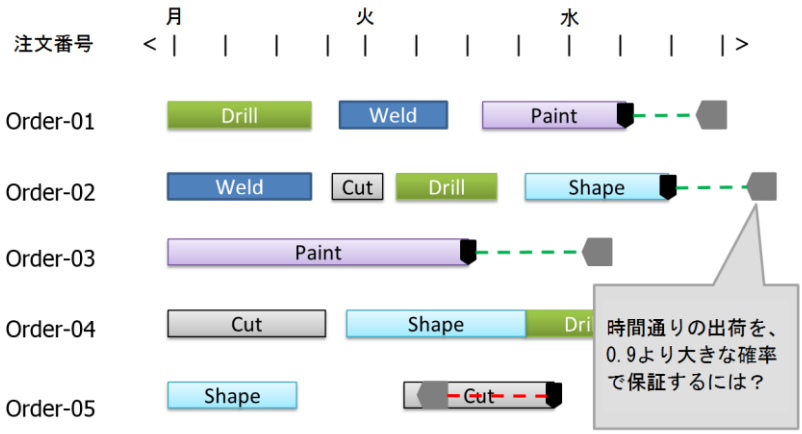


図 12.3　典型的な計画用ガントチャート

所与の能力制限のあるリソース（作業者、機械、移動手段など）を考慮して実行可能なスケジュールを生成する問題は、一般に**有限能力スケジューリング**（Finite Capacity Scheduling）と呼ばれる。有限能力スケジューリングには二つの基本的アプローチがあり、この節と次の節で説明する。

一つ目のアプローチは、制約条件を数学的な関係式で表現してシステムを定義する**数学的最適化法**である。このアプローチでは、遅延ジョブ数の最小化などの目標を達成しながら制約条件を満たす解を、アルゴリズムベースのソルバーを用いて探索する。しかし、これらの数学モデルは **NP 困難**として知られており、最適解を見つけるための効率的な**アルゴリズム**が存在しない問題に分類される。そのため実務では、最適解の発見よりも「良好な」解を見つけることを目的として、ヒューリスティックなソルバーが用いられる。この手法を採用した代表的な製品には、IBM の ILOG 製品ファミリー（CPLEX）と SAP の APO-PP/DS がある。

スケジューリングの数学的最適化手法には、広く認識された欠点がいくつかある。システムを数学的な制約条件で表現するのは、非常に複雑でコストのかかるプロセスだ。また、システムは常に変化するため、時間の経過とともに数学モデルの維持が困難になる。さらに、実システムには数学的制約条件で正確にモデル化できない重要な制約条件が存在し、それらを無視せざるを得ない場合がある。そのため、作成されたスケジュールは数学モデルの要件は満たすものの、実システムでは実行不可能なことが多い。加えて、数学モデルのソルバーが良好な候補スケジュールを生成するには、通常数時間を要する。その結果、これらのスケジューリングは夜間や週末に実行されることが一般的だ。機械の故障、原材料の到着遅延、作業者の病欠といった予期せぬ事態が発生すると、スケジュールはすぐに陳腐化してしまうため、その有効期限は短くなる。

本節では、すべての項目を網羅的に解説するのではなく、主要な概念と一般的な課題の概要を示すにとどめた。より詳細な内容については、優れた参考文献である『Factory Physics』（Hopp and Spreaman 2008）を参照されたい。

12.7　シミュレーションを用いたスケジューリング

有限能力スケジューリングの二番目のアプローチは、システム内の有限リソースを表現するためにシミュレーションモデルを活用する方法である。シミュレーションツールを計画とスケジューリングの支援に利用するという考え方は、数十年の歴史がある。著者のひとりは、1980 年代前半に製鋼スケジューリングシステムの開発にシミュレーションを適用した経験を持つ。スケジューリングへの応用では、現状のシステムでシミュレーションモデルを初期化し、計画された実際の仕事の流れをモデル内でシミュレートする。ただし、スケジュールを生成する際は、すべての変動要素と計画外のイベントを除外する必要がある。

シミュレーションを用いたスケジューリングは、**ヒューリスティック**な解を生成する。この解は、最適化手法と比べてより短時間で得られる。生成されるスケジュールの質は、モデル内で有限リソースを活動に割り当てる意思決定ロジックに依存する。たとえば、機械などのリソースが遊休状態になったとき、次に処理するエンティティの選択にはモデル内のルールが使用される。このルールにはさまざまな種類がある。もっとも優先度の高い仕事を選ぶという単純な静的ルール、段取り時間を最小化する順序を決定する複雑な動的選択ルール、あるいは納期までの残り時間を残り作業時間で割った値（クリティカルレシオ）が最小の仕事を選ぶことで緊急度に基づいて判断するルールなどである。

データ駆動型の伝統的な（あるいは「懐古的な」）ジョブショップシステムのモデルを対象として、多くのシミュレーションによるスケジューリングシステムが開発されてきた。これらのシステムでは、製造現場はワークステーションの集合として扱われ、各ワークステーションは前処理、加工、分解の各フェーズで構成される。仕事は決められたルートに従って、ワークステーション間を移動する。このソフトウェアは、ワークステーション、原材料、仕事の記述にデータを使用するように設計されている。適用事例が懐古的なモデルと適合する場合は良好な解が得られる可能性があるが、そうでない場合はモデルのカスタマイズ機会が限られ、実システムの重要な制約条件を無視せざるを得なくなる。

また、有限能力スケジューリングには汎用の離散イベントシミュレーション（DES）製品も利用可能だ。図 12.4 は、計画とスケジューリングシステムのコアとして DES エンジンを使用する際の典型的な構造を示している。このアプローチには以下の利点がある：

- 柔軟性に富む：汎用ツールは、システムのあらゆる重要な側面をモデル化できる。これはシステム設計用に作成されたモデルの特徴である。
- 拡張性が高い：設計用シミュレーションと同様に反復が可能（かつ推奨される）。問題の一部から解決を始め、徐々に改善できる。反復を重ねることで、望ましいスケジュール精度が得られるまでモデルの範囲と詳細度を適切に拡張できる。
- 先行事例を活用可能：スケジューリングに必要なシステムモデルは、設計の最適化に使用したモデル（理想的にはすでに活用済みのもの）と非常に類似している。そのため、既存の設計モデルを計画やスケジューリング用に拡張できる。

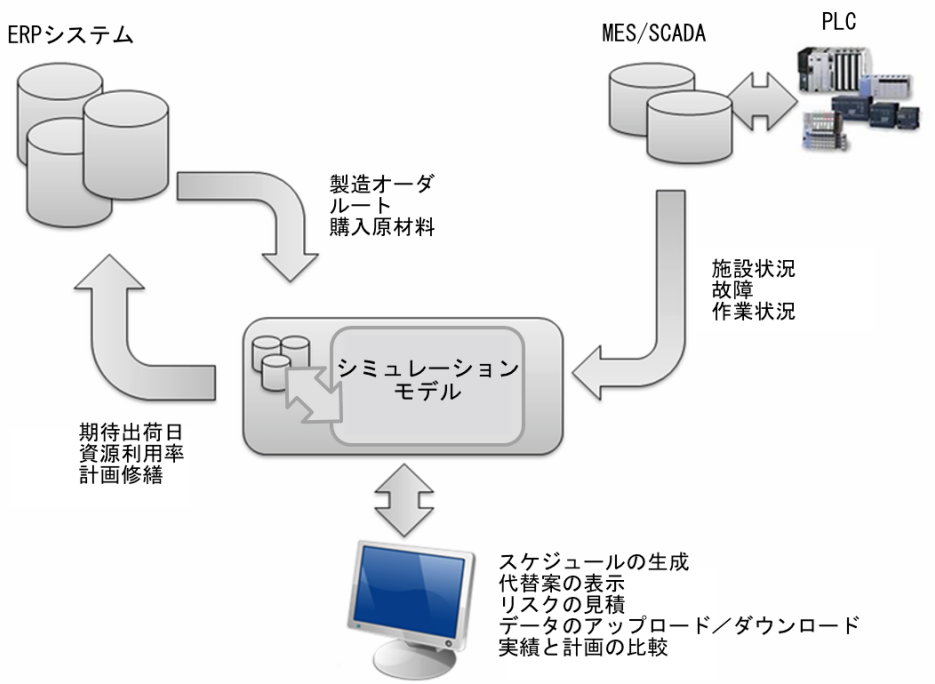


図 12.4　シミュレーションを用いたスケジューリングシステムの基本構造

・確率的な運用が可能：設計モデルがシステム構成を確率的に分析するように、計画モデルも就業規則や他の運用特性を確率的に評価できる。これにより、より適切な意思決定が可能な「スマートな」スケジューリングシステムを最初から構築できる。
・確定的な運用が可能：確率的な機能を無効にすることで、確定的な計画を生成できる。これは楽観的なスケジュールとなるが、詳細さの水準が比較的高いため、他のツールによるスケジュールより正確な傾向にある。また、スケジュールの楽観度も評価可能だ（次の項目参照）。
・リスクの評価可能：確定的な計画を生成した後で、確率的な機能を実行に用いることができる。変動性（想定されるすべての不具合）を再び有効にし、その計画に対して複数回の反復実行を行うことで、重要なパフォーマンス目標の達成度を評価できる。この情報を基に、もっとも費用対効果の高い方法でリスクを管理するようスケジュールを客観的に調整できる。
・必要なパフォーマンス指標をすべて利用可能：モデル実行中のどの時点でも、パフォーマンス目標に関する主要な情報を収集できるため、スケジュールの実行可能性とリスクを実用的に測定できる。

ただし、汎用の DES 製品をスケジューリングに使用する場合、これらのツールは本来その目的のために設計されていないため、独特の課題がある。以下に、発生する可能性のある問題をいくつか挙げる：

・スケジューリング結果：汎用の DES は通常、スループットや稼働率などの主要

なシステムパラメータのサマリ統計量を出力する。これは関連性があるものの、スケジューリングでは個々の仕事（エンティティ）とリソースが主な焦点となり、ガントチャートや詳細な追跡ログの形で提供される必要がある。しかし、このレベルの詳細さは汎用の DES 製品では自動的に記録されない。

・モデル初期設定：システム設計のシミュレーションでは、空かつ遊休状態のモデルから開始し、バイアスを排除するため初期部分を切り捨てる。一方、スケジューリングでは、処理中の仕事やシステム内の各所に存在する仕事を含めた現状をモデルの初期状態として設定できる必要がある。ほとんどの DES 製品ではこれを実現するのが困難だ。

・ランダム性の制御：DES モデルは通常、処理時間や機械故障などのランダムな時間とイベントを含む。計画生成時には時間の期待値を使用し確率的イベントを無効にする一方で、計画策定後はリスク評価のために変動性を加える必要がある。しかし、典型的な DES 製品はこの両方の運用モードに対応していない。

・企業データとの連結：計画・スケジューリングモデルに必要な情報は、通常、企業の ERP システムやデータベースに存在する。これらの情報は複数のデータテーブルにまたがる複雑なリレーションを含むが、ほとんどの DES 製品はリレーショナルデータソースとの連携に対応していない。

・状況の更新：計画・スケジューリングモデルは、機械故障などの実システムの変化に継続的に対応する必要がある。そのためには状況変化を入力できる対話型インタフェースが不可欠となる。

・スケジューリングのユーザインタフェース：典型的な DES 製品は設計モデルの構築と実行のためのインタフェースを備えている。しかし、スケジューリングでは、既存モデルを使用して計画を生成し、残業追加や原材料発送催促といった業務的意思決定のリスクを評価するための専用インタフェースが必要となる。

新しいアプローチであるリスクに基づく計画とスケジューリング（Risk-based Planning and Scheduling：RPS）は、シミュレーションの主要な利点を最大限に活用しつつ、上記の課題を解決するように設計されている。

12.8　リスクに基づく計画とスケジューリング

リスクに基づく計画とスケジューリング（RPS）は、業務計画やスケジューリングに確定的・確率的シミュレーションを組み合わせ、従来の DES の能力を最大限に活用するツールだ（Sturrock 2012a）。RPS の技術的詳細は、『Deliver On Your Promise – How Simulation-Based Scheduling Will Change Your Business』（Pegden 2017）に詳しく記載されている。RPS は、生産システムに内在する変動性を完全に考慮することで従来の APS を拡張し、スケジューリング担当者がリスクと不確実性を事前に低減するための情報を提供する。RPS は基盤となるシミュレーションモデルを二つの方法で活用し、このモデルはさまざまな詳細レベルで構築でき、実システムのあらゆるランダム変動を組み込むことができる。

RPS はまず、ランダム性を無効にした（確定的な）シミュレーションモデルを実行して、確定的なスケジュールを生成することから始まる。これは従来の APS

ソリューションが生成するスケジュールと同等だが、より詳細な表現が可能だ。次に、RPS は同じモデルでランダム性を有効にし（確率的モード）、複数のプロセッサを使用して複数回のスケジュール生成を繰り返し、各実行結果の統計データを収集する。測定される指標には、目標達成（納期など）の確率、予想される完了時期（システムの変動により計画日程より遅延する傾向がある）、完了時期の楽観的・悲観的予測（変動性に基づくパーセンタイル）などがある。図 12.3 の RPS 分析と図 12.5 を比較すると、リスク分析では、Order-02 は余裕があるように見えても、注文自体やそれに必要なリソース・原材料のリスクを考慮すると、納期遵守の確率は 47%と低くなっている。このようにリスクを客観的に把握できることで、もっとも効果的なリスク低減策を講じることができる。

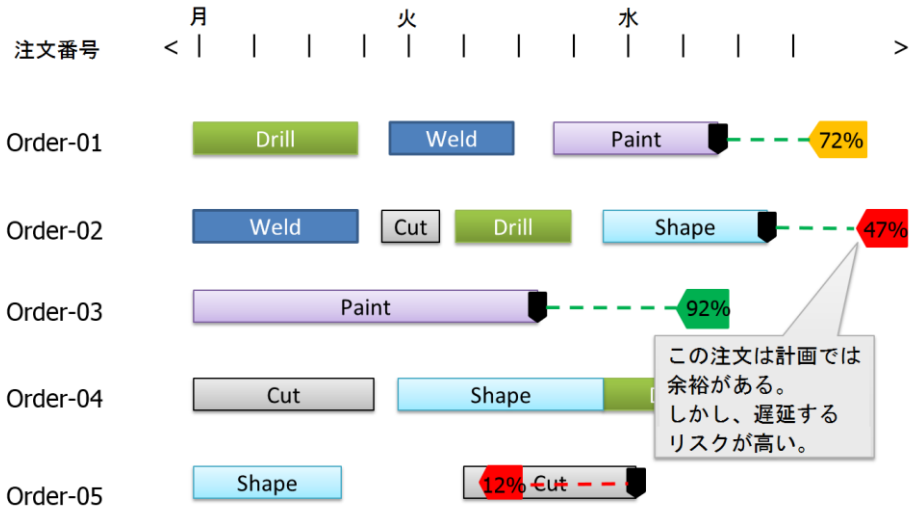


図 12.5　高リスクの案件を示すガントチャート

RPS は、対象システムの専用シミュレーションモデルを構築するシミュレーションベースのアプローチを採用している。このアプローチの最大の利点は、シミュレーションソフトウェアのモデリング能力を活用して、システムの制約条件を完全に表現できることだ。これにより、あらゆるシミュレーションツールでシステムのモデル化が可能となる。また、シミュレーションソフトウェアが対応している場合、カスタムオブジェクトを使用して複雑なシステムをモデル化できる。クレーンやコンベア、フォークリフト、AGV などの搬送装置の動き（経路上の混雑を含む）も表現可能だ。さらに、オーブンや治具を切り替えられるマシニングセンターなどの複雑なワークステーションもモデル化できる。

RPS は制約条件の種類や数に制限を設けない。そのため、生産システムの重要な制約を無視する必要がなくなった。複雑な生産やサプライチェーンの実態を正確に反映したモデルを用いることで、確定的な計画とそれに対応するリスク分析の両方を作成できる。また、施設設計の評価用に構築したモデルを RPS にも活用できる。つまり、1 つのモデルで施設設計と日程計画の両方を改善できる。

デジタルツインとして実装された RPS は、日々のスケジュール作成と同時に、

運用戦略の継続的な見直しや仮説分析を行う改善プラットフォームとして機能する。新部品の導入や新規機械・ラインの設置などをオフラインでテストできる。システムの更新後は、評価モデルを実運用モデルに切り替えるだけで本稼働が可能だ。設計検証に使用したモデルは、ソフトウェアの再実装や高コストの更新なしで、システムの変更をスケジュールに即座に反映できる。

また、サプライチェーンの性能向上のため、計画期間を拡張してマスタープランと詳細な工場スケジュールの整合性を高められる。同一モデルを計画用に 3～6 週間、スケジューリング用に 1～3 週間、詳細な生産スケジュール実行用に 1～2 日運用できる。これにより、適切な所要日数に基づく調達が可能となり、原材料の確保が保証される。この正確な情報を用いて、ERP システムの更新（SAP へのフィードバックなど）も可能だ。

RPS は OptQuest などの最適化プログラムとも連携できる。企業の KPI を設定し、自動実験を実行することで、バッファサイズ、リソーススケジュール、ディスパッチルールなどの最適な設定を見出し、効果的な工場運営とスケジューリングが可能になる。シミュレーションモデルの設計モードと実行モードを組み合わせることで、需要主導型 MRP（Demand Driven MRP：DDMRP）システムを構築・実装する理想的なツールとなる。シミュレーションベースの DDMRP システムは、ほぼ瞬時の再スケジューリングと動的な設定更新という利点を備えている。

12.9　Simio RPS を用いた計画とスケジューリング

Simio RPS Edition は、リスクに基づく計画とスケジューリングに特化した機能を備えた Simio の拡張版だ。Simio バージョン 16.256 以降では、この拡張機能がすべてのアカデミックパッケージに標準搭載されている。未導入の場合、教職員は academic@simio.com へリクエストを送信してアップグレードを申請できる。RPS 機能の有無は、Facility ビュー上部の Planning タブ（図 12.6）の存在で確認できる。主要な RPS 機能はこの Planning タブから利用可能だ。なお、Simio RPS のアカデミック版は商用版とほぼ同等の機能を持つが、ライセンス規約により商用利用は認められていない。

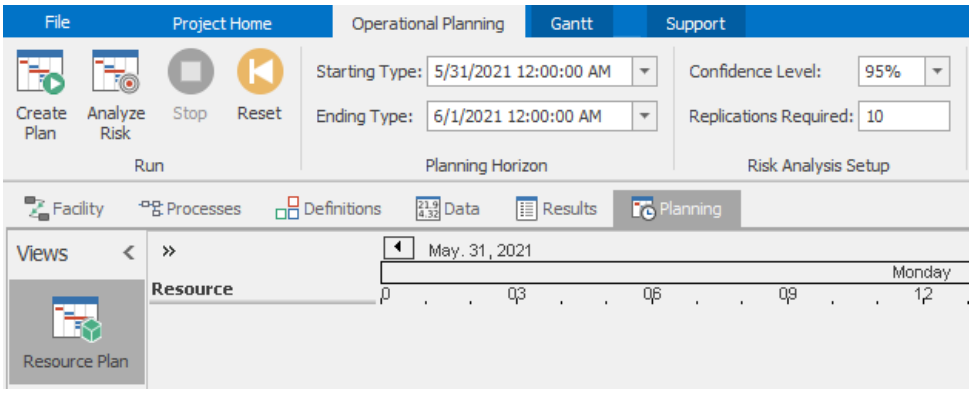


図 12.6　Planning タブから RPS スケジューリング機能へのアクセス

Simio RPS ソリューションの基本モデルはどの Simio 製品でも構築できるが、RPS での使用に向けた準備には Simio RPS 機能が必要となる。この準備には、アクティビティログの有効化、出力状態変数とスケジューリング**目標**のテーブルへの追加、およびスケジューリング担当者向けのユーザインタフェースのカスタマイズが含まれる。

従来の Simio では、データテーブルはモデルへの入力提供にのみ使用されていた。一方、RPS ではデータテーブルをモデル実行中の値の記録にも活用する。これは標準プロパティの列に加えて、テーブル定義に状態変数列を追加することで実現される。たとえば、テーブル状態変数 Date-Time は仕事の出荷日を、Real は仕事の累積費用を記録する。状態変数の記録は、Standard Library のオブジェクトでの状態変数の割当、もしくはプロセスロジックでの Assign ステップを使用してより柔軟に設定できる。出力の状態変数列は、Simio RPS Edition の Data タブにある States リボンまたは State ボタンでテーブルに追加できる。

スケジューリングの**目標**とは、スケジュールで達成したい結果の値を指す。典型的な例は仕事の出荷日（納期前の完了）だが、目標は完成期日だけに限らない。シミュレーションで測定可能なすべての指標（材料到着、熟練度、費用、生産量、品質、設備総合効率など）や、あらゆる水準（全社的パフォーマンス、部門別、サブアセンブリ別、その他の主要管理点）で目標を設定できる。

目標は値を指定する式で定義され、その上限下限も設定される。これらの境界値に基づいて結果が分類される。たとえば、納期目標では上限を超えた範囲を「Late（遅延）」、下回る範囲を「On Time（定刻）」と分類する。Simio はこれらの用語を使って統計を報告する。図 12.7 は、テーブル状態変数 ManufacturingOrders.ShipDate をテーブルプロパティ ManufacturingOrders.DueDate と比較する目標（TargetShipDate）の定義を示している。ここでは、OnTime、Late、Incomplete（未完成）の 3 つの結果が想定される。同様に、費用目標では「Cost Overrun（予算超過）」と「On Budget（予算通り）」に分類される。目標には日付・時刻や総生産費用などの一般的な値も使用可能だ。納期や費用などの目標は上限を下回ることが望ましく、完了仕事数などの目標は下限を上回ることが望ましい。Target 列は、Simio RPS の Data タブにある Targets リボンまたは Target ボタンで追加できる。

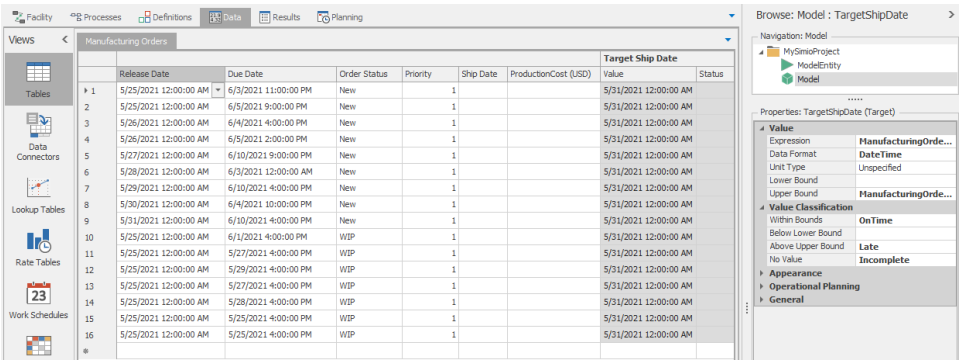


図 12.7　テーブル状態変数とプロパティを用いた目標の定義

Simio は、定義された境界と各目標値を比べたパフォーマンスを自動的に記録する。確定的な計画の実行では、定義された限界（On Time、Cost Overrun など）と終了時の目標値を比べたデータを記録する。一方、変動や予期せぬイベントを含むリスク分析を実行する場合、Simio はモデルの全反復実行における各目標値のパフォーマンスを記録する。この情報を基に、定刻出荷の確率や、期待・楽観的・悲観的な出荷日などのリスク指標を算出する。

標準の Simio ユーザインタフェースは、モデルの構築と実験に重点を置いている。しかし、RPS の実務では、計画およびスケジューリング担当者向けの専用インタフェースが必要となる。これらのユーザは、モデル構築は行わず、むしろ計画とスケジュール作成のために、設定可能なモデルを利用する。Simio RPS では、担当者のニーズに合わせた専用インタフェースを容易に作成できる。さらに、スケジューリング担当者向けの表示データや編集可能なデータ型をすべて構築でき、特定の用途に応じて完全なカスタマイズも可能だ。

12.10　スケジューリングのインタフェース

これまで Simio RPS をデザインモードで使用してきたが、RPS にはスケジューラモードも搭載されている。このモードでは、既存のモデルを基にスケジュールの生成・表示が可能だ。スケジューラモードには、計画とスケジューリングに特化したカスタマイズ可能なユーザインタフェースが用意されている。このインタフェース（図 12.8）は新規モデルの構築機能を省いているため、標準の RPS インタフェースより小型でシンプルだ。スケジューラモードでは、標準の Simio 製品で作成したモデルを実行し、その後 RPS のデザインモードで展開するための準備を行う。このモードは File→Settings から切り替えることができる。

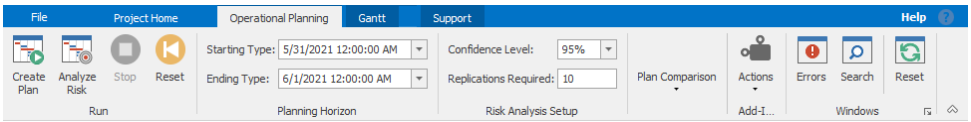


図 12.8　スケジューラモードの簡素化されたリボン

スケジューラモードの主な目的は、確定的モードでモデル内の計画された仕事を実行し、計画やスケジュールを生成することだ。この際、すべてのランダムな時間は期待値に置き換えられ、ランダムイベントは抑制される。この確定的スケジュールは他の確定的スケジュールと同様に楽観的だが、後にランダムイベントを分析する機能も備えている。

スケジューラモードは、専用レポートとともに、生成された計画やスケジュールの静的・動的なグラフィカル表示を提供する。グラフィカル表示には、スケジュール生成の 3D アニメーションやエンティティとリソースのガントチャートがある。エンティティのガントチャートでは、各エンティティ（仕事やオーダーなど）がチャートの行に表示され、水平方向の時間軸上に各リソースの使用期間が長方形で示される。リソースのガントチャートでは、各リソース（機械や作業者など）がチャートの行に表示され、水平方向の時間軸上にそのリソースを使用するエン

ティティが長方形で示される。図 12.9 は、リソース制約を部分的に満たしていないリソースのガントチャートを示している。この図からは、Order-04 の遅延原因として、Weld（溶接）装置の長時間待ちと作業者の短時間待ちが確認できる。

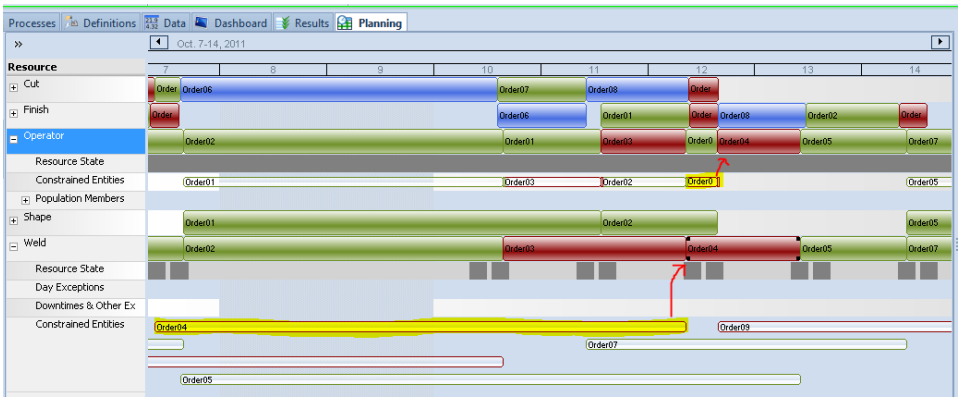


図 12.9　Order-04 における遅延要因の制約条件分析

専用レポートには、各リソースの作業リストが含まれ、計画期間中の作業開始・終了時間が示される。作業リストは各ワークステーションの作業者に提供され、予想作業量の確認に使用される。さらに、計画期間中のリソース状態を示すリソース利用率レポートや、非付加価値をもたらす制約条件（原材料不足や作業者不在など）をまとめた制約条件レポートも用意されている。図 12.10 は、作業者向けのリソース作業手配リストの例を示している。これらのレポートの詳細は次項で説明する。

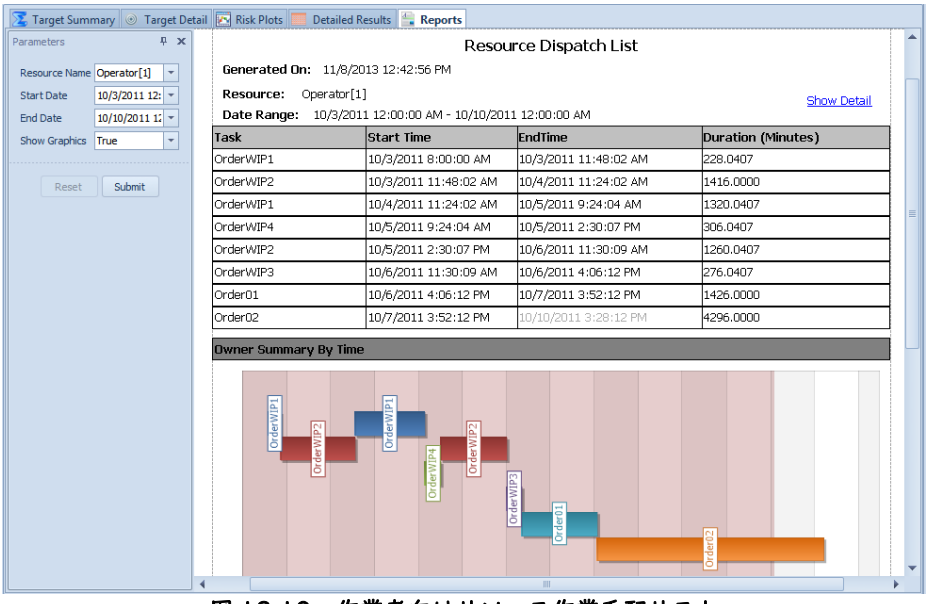


図 12.10　作業者向けリソース作業手配リスト

スケジューラモードでは、デザインモードで設定された独自のダッシュボードを通じて、特定の仕事、機械、作業者、原材料の情報を一覧で確認できる。ダッシ

ュボードは、任意の仕事や機械に関する要約画面に、複数のグラフやテーブル形式のスケジュール情報を統合し、計画やスケジュールの詳細な分析を可能にする。

スケジューラモードを活用することで、計画・スケジューリング担当者は、リスクの高いスケジュールを改善するための代替案を検討できる。残業の効果、原材料の配送タイミング、目標達成のための作業優先順位の変更など、さまざまな観点から各計画案を評価することができる。

12.11　Model 12-1：モデルファーストアプローチによるスケジューリング

本節では、スケジューリングモデルを構築するためのモデルファーストアプローチについて説明する。このアプローチは、新規施設や、MES などのサブシステムが開発中で、モデル構成データがすぐに利用できない施設に最適である。時間はかかるものの、外部データの要件が少ないため、シミュレーションに基づく設計改善を早期に、大きなリスクなく実行できるという利点がある（12.4 節参照）。

最初に、基本的なモデルを構築し、計画機能を有効にして生成された計画を確認する。次に、未決注文ファイルをインポートし、そのデータを使用できるようモデルを調整する。続いて、スケジュールリスクを評価できるよう、出荷予定日と目標日の追跡機能を追加してモデルをカスタマイズする。最後に、スケジュール評価・運用に役立つ組み込み分析ツールの概要を説明する。

12.11.1　基本的なスケジューリングモデルの作成

New Project を起動し、1 つの Source、3 つの Server、1 つの Sink を配置する。図 12.11 のように、各 Server に名前をつけ、Path で接続する。この時点では、すべてのオブジェクトプロパティをデフォルト値のままにしておく。次に、3 つの Server を同時に選択し、Advanced Options プロパティグループ内の Log Resource Usage を True に設定する。最後に、Run リボンの Ending Type を Run Length : 1 hour に変更する。

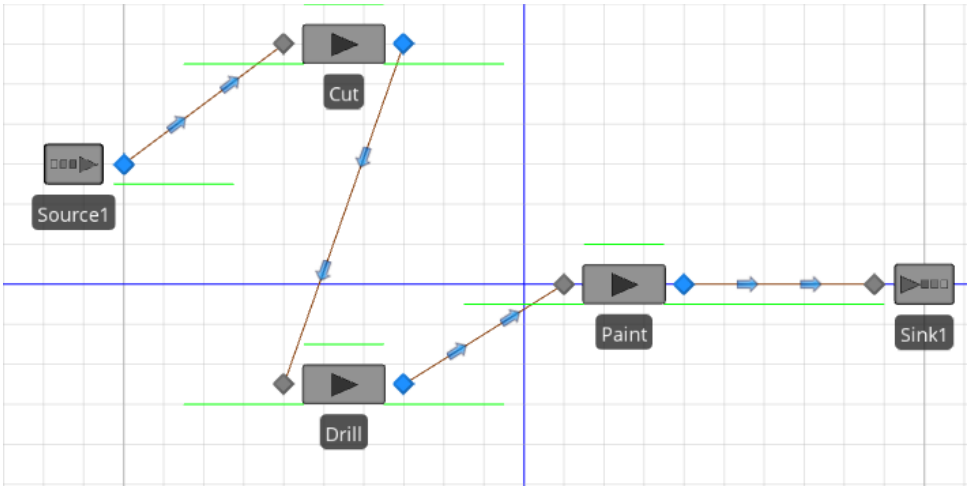


図 12.11　Model 12-1 の Facility ウィンドウ

Planning タブの Operational Planning リボンを選択し、Create Plan ボタンをクリックする。Resource Plan（左上のボタン）を選択すると、各リソースのリストとそのアクティビティが表示される。各リソースについて、エンティティの処理開始時間と終了時間を確認できる。Gantt リボンの Zoom In や Zoom Range 機能を使用し、時間軸上でマウスをスクロールすると、図 12.12 のように、アクティビティの詳細を表示できる。

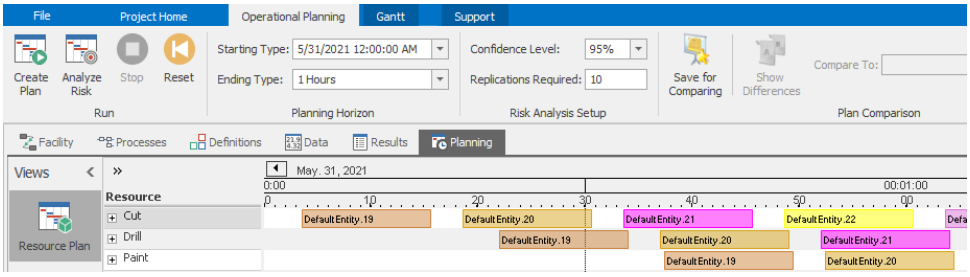


図 12.12　Model 12-1 の Resource Plan の拡大図：エンティティの詳細表示

左側の Entity Workflow ボタンをクリックすると、各エンティティの行に、使用リソースと処理時間が Gantt ビューで表示される。Zoom 機能を使用すると、最初のいくつかのエンティティの詳細なアクティビティを確認できる（図 12.13）。なお、エンティティ ID は文字列としてソートされるため、最初に作成されたエンティティ（DefaultEntity.19）は DefaultEntity.189 と DefaultEntity.190 の間に配置される（非直観的だが、この解決方法は後述する）。

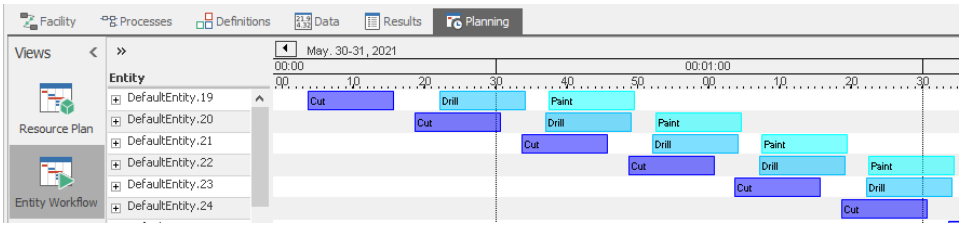
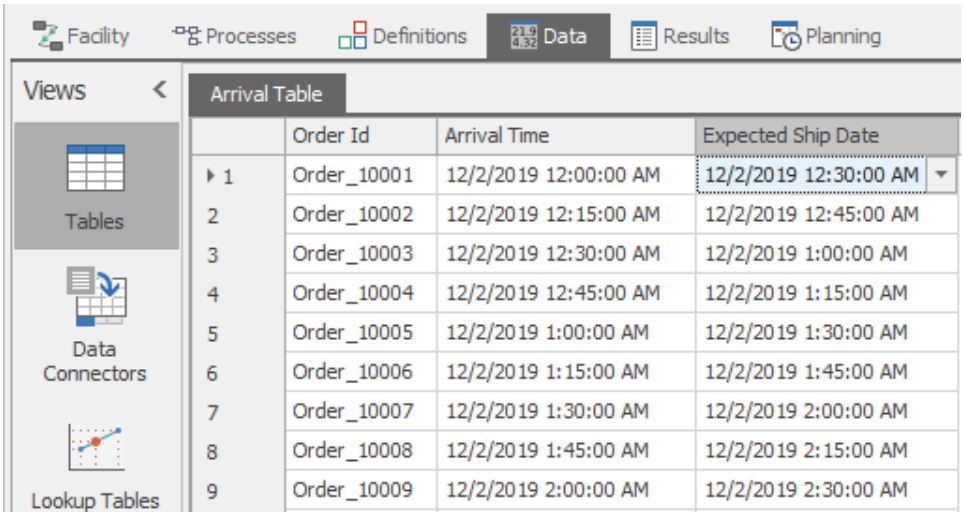


図 12.13　Model 12-1 の Entity Workflow の拡大図：リソースの使用状況の詳細

12.11.2　より実践的なモデルの作成

注文データファイルと実際の処理時間を使用して、モデルをより現実的なものに改良しよう。まず、Data タブの Tables ビューを開き、Tables リボンから Add Data Table ボタンでテーブルを追加する。このテーブルに ArrivalTable という名前をつける。students ダウンロードファイルの Model_12_01_DataFiles フォルダには ArrivalTableData.csv という CSV ファイルがある。Tables リボンの Create Binding ボタンを使って、この CSV ファイルをテーブルに関連づける。これでテーブルとファイル間の関係が確立され、データ取り込み用のスキーマ（列定義）が作成される。この手順で、ファイルをテーブルにインポートできるようになる（図 12.14）。このテーブルには実際の日付データが含まれているため、モデルの実行期間を実データに合わせる必要がある。最初の注文到着時刻に合わせて、Run リボンの Starting Type を Specific Starting Time（12/2/2019 12:00:00 AM）に、Ending Type を Run Length : 1 Days に設定する。



	Order Id	Arrival Time	Expected Ship Date
1	Order_10001	12/2/2019 12:00:00 AM	12/2/2019 12:30:00 AM
2	Order_10002	12/2/2019 12:15:00 AM	12/2/2019 12:45:00 AM
3	Order_10003	12/2/2019 12:30:00 AM	12/2/2019 1:00:00 AM
4	Order_10004	12/2/2019 12:45:00 AM	12/2/2019 1:15:00 AM
5	Order_10005	12/2/2019 1:00:00 AM	12/2/2019 1:30:00 AM
6	Order_10006	12/2/2019 1:15:00 AM	12/2/2019 1:45:00 AM
7	Order_10007	12/2/2019 1:30:00 AM	12/2/2019 2:00:00 AM
8	Order_10008	12/2/2019 1:45:00 AM	12/2/2019 2:15:00 AM
9	Order_10009	12/2/2019 2:00:00 AM	12/2/2019 2:30:00 AM

図 12.14　CSV ファイルをインポートした Model 12-1 の ArrivalTable

Facility ビューで Source1 オブジェクトを選択し、新しいテーブルを使用してエンティティを生成するように設定する。Arrival Mode を `ArrivalTable` に、Arrival Time プロパティを `ArrivalTable.ArrivalTime` に設定する。次に、サーバの処理時間をより現実的な値に変更しよう。3 つのサーバを同時に選択し、Processing Time の Units を Minutes から Hours に変更する。続いて、エンティティをテーブルの OrderID で識別できるようにするため、`ModelEntity` を Facility ビューにドラッグし、プロパティを編集する。Advanced Options カテゴリの Display Name と Animation カテゴリの Dynamic Label Text をそれぞれ `ArrivalTable.OrderId` に設定する。

これらの設定が完了したら、結果を確認してみよう。Planning タブの Entity Workflow Gantt を開くと、前回の実行から変更された部分が赤いバーで表示される。Create Plan ボタンをクリックしてガントチャートを更新し、新しい時間軸に合わせてズームすると、ArrivalTable で定義された意味のある名前でエンティティが表示される（図 12.15）。Resource Plan でも、各リソースの表示は以前と同じままだが、エンティティにより分かりやすい名前がつけられている（図 12.16）。

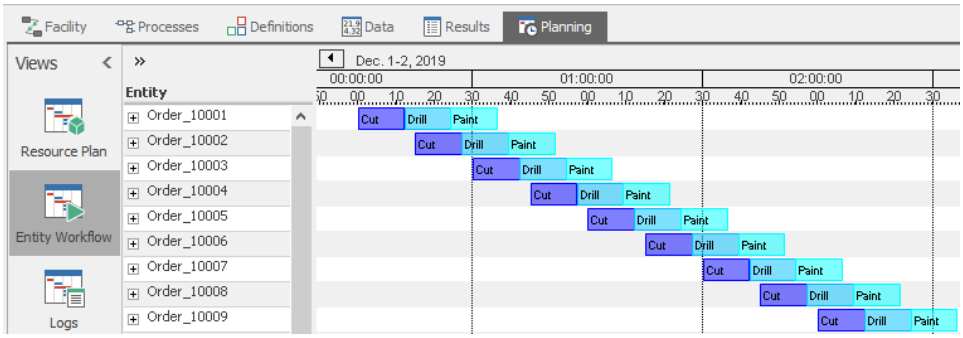


図 12.15　時刻とラベルを更新した Model 12-1 の Entity Workflow Gantt

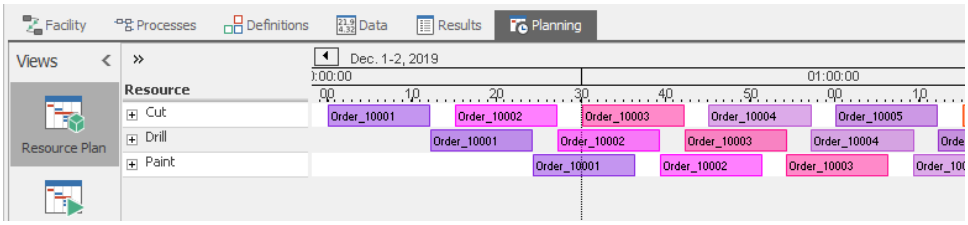


図 12.16　時刻とラベルを更新した Model 12-1 の Resource Plan Gantt

12.11.3　パフォーマンストラッキングとターゲットの追加

モデルが動作するようになったので、**スケジュール**の運用状況を評価するための機能を追加しよう。まず重要なのは、各注文の出荷予定日を記録することだ。状態変数をテーブルに追加するため、テーブルに戻って States を選択する。状態変数 DateTime をテーブルに追加し、この列に ScheduledShipDate という名前をつける。この列は実行中に値が代入される出力列なので、最初は値がないことを示すエラーが表示される。注文が完了して Sink1 に到達したときに値を代入するため、Sink1 の State Assignments セクションで ArrivalTable.ScheduledShipDate に TimeNow を代入する。

次に重要なのは、注文が納期通りに出荷できるかを評価することだ。これには、テーブルにターゲット（Target）を追加する。テーブルに戻り、Targets リボンを開く。Add Targets ボタンをクリックすると、「Value（値）」と「Status（状態）」の 2 列がテーブルに追加される。このターゲットに TargetShipDate という名前をつける。評価対象の式は、Data Format が DateTime の ArrivalTable.ScheduledShipDate で、比較対象は ArrivalTable.ExpectedShipDate だ。この値は超過させたくないため、上限（Upper Bound）として設定する。わかりやすくするため、デフォルトの用語を変更する。Value Classification カテゴリで、Within Bounds を On Time、Above Upper Bound を Late、No Value を Incomplete に設定する。モデルを実行すると、すべての注文の状態が Late となっている。

システムのバランスを改善するため、すべてのサーバの処理時間を Random.Triangular(0.05, 0.1, 0.2) Hours に変更する。モデルを再実行してテーブルを確認すると、今回はほぼすべての注文が On Time となっている。Planning タブの Entity Workflow に移動して Create Plan を実行すると、目標出荷日（Target Ship Date）を示す灰色のフラグが各エンティティに表示される。このフラグが最終オペレーションの右側にあれば、余裕時間があることを示している（つまり、注文が目標より早く完了できる）。ただし、この結果の確実性はまだ不明だ。Analyze Risk ボタンをクリックすると、モデルは複数回の変動を含む反復実行を行い、フラグの色が変化して注文が On Time となる確率が表示される。

このモデルにさらなる変動性を加えるため、3 つの要素を追加しよう。まず、注文到着時刻の変動性として、注文が最大 15 分早く、または最大 30 分遅れて到着するよう、Source の Other Arrival Stream Options の Arrival Time Deviation を Random.Triangular(-0.25, 0.0, 0.5) Hours に設定する。次に、Drill は他

の機械より処理時間が長く、予測も困難という想定で、Processing Time を Random.Exponential(0.2)に設定する。最後に、3 台のサーバすべてに信頼性の問題を導入する。Reliability に Calendar Time Based の故障を追加し、Uptime Between Failure と Time To Repair はデフォルト値のままとする。Entity Workflow Gantt で Analyze Risk を実行すると、確定的な計画では大半の注文が納期どおりに完了するものの、これらの変動要素を考慮すると、実際には納期どおりの完了率が大幅に低下することが判明する（図 12.17）。

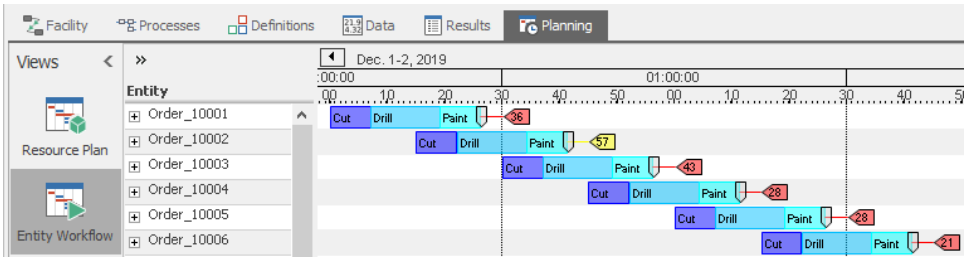


図 12.17　Model 12-1 における変動性を含むリスク分析

12.11.4　その他の評価機能

シミュレーションによるスケジューリングを実行して得られた結果と出力データは、すべて Planning タブに表示される。本書では分析ツールの詳細な説明は割愛するが、特に重要な機能をいくつか紹介する。

左側のパネルには、スケジューラモードで利用可能なツールが表示される。これまでに 2 つのガントチャートについて簡単に説明したが、スケジュールの評価と改善に役立つガント関連機能が他にも多数ある。たとえば、いずれかのガントチャート上のアイテムをクリックすると Properties ウィンドウに注文の関連情報が表示され、マウスをその上に置くと特定のタスクの詳細情報がポップアップで表示される（図 12.18）。

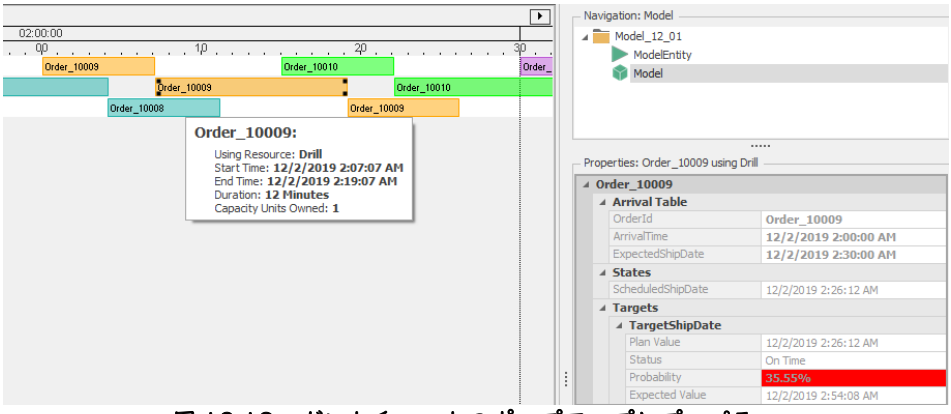


図 12.18　ガントチャートのポップアップとプロパティ

これまでの例では使用していないが、行の左側にある「+」ボタンをクリックすると、その行の項目に関する詳細（Constraints など）が表示される。また、Gantt リボンのボタンから表示するオプションデータを選択できる。これらの 2 つの機能は図 12.19 に示されている。

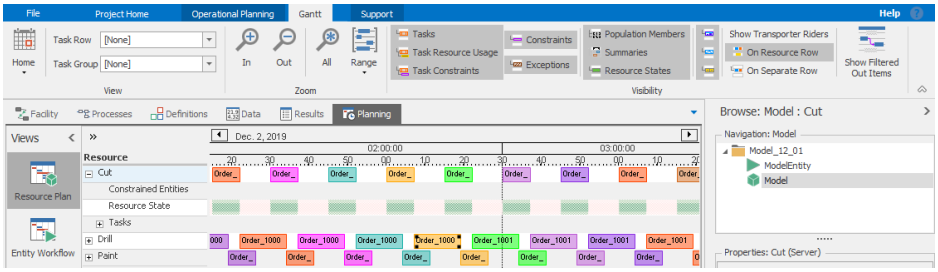


図 12.19　ガントチャートの行データ詳細表示

左側の 3 番目のボタンは Logs で、リソース関連、制約、トランスポータ、原材料、統計など 10 種類のカテゴリのイベントを記録・表示する。ガントチャートや後述する分析機能の多くは、これらのログデータに基づいている。ログはフィルタリングで読みやすく整理でき、列を追加してデータの識別も容易にできる。

左側の Tables ボタンをクリックすると、モデルの Data タブで使用したテーブルと同様の、タブつき表示が生成される。主な違いは、モデル作成者が各テーブルの列の表示・非表示や、計画立案者による編集可否を設定できる点である。これによりテーブルを簡素化し、計画立案者による不適切な変更（出荷予定日の変更など）を防ぐことができる。リスク分析を実行すると、Planning ビューのテーブルに分析結果の列が追加される。

左側の一番下の Results ボタンをクリックすると、3 行目のタブが有効になる。図 12.20 の Target Summary をはじめ、Target Detail と Risk Plots にも詳細なリスク分析データが表示される。Detailed Results タブには、通常の実行時と同様のピボットテーブルが表示される。Reports、Dashboard Reports、Table Reports では、計画立案者や関係者向けのカスタムレポートや対話型ダッシュボードを提供する。

このように、これらのツールを使用することで、計画立案者は現在の計画の実行状況を理解し、改善点を分析し、結果を関係者と共有できる。

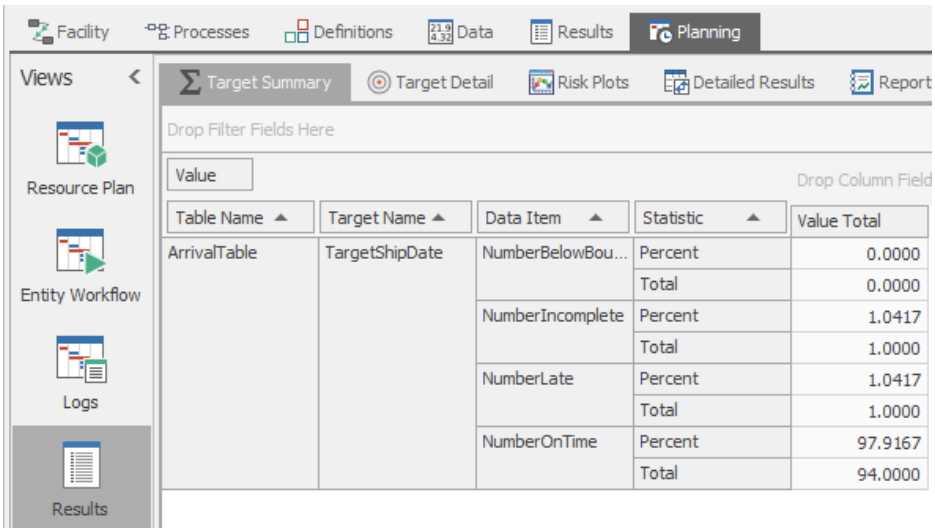


Table Name	Target Name	Data Item	Statistic	Value Total
ArrivalTable	TargetShipDate	NumberBelowBou...	Percent	0.0000
			Total	0.0000
		NumberIncomplete	Percent	1.0417
			Total	1.0000
		NumberLate	Percent	1.0417
			Total	1.0000
		NumberOnTime	Percent	97.9167
			Total	94.0000

図 12.20　Planning タブの Target Summary 表示

12.12　Model 12-2：データファーストアプローチによるスケジューリング

12.11 節では、モデルファーストアプローチによる部分的なデータ駆動型モデルの構築を練習した。7.8 節では、データ生成モデルの背後にある理論と実践について説明した。データファーストアプローチは、既存のシステムがあり、モデル構成データがすでに ERP（SAP など）、MES（Wonderware など）、スプレッドシートなどに存在する場合に最適である。このアプローチの最大の利点は、ベースラインモデルを短時間で作成できることだ。すでにある程度のモデリングとスケジューリングの知識を持っているので、B2MML データファイル（7.8 節参照）からモデルを構築し、そのモデルの強化方法を探ってみよう。

図 12.21 に示すように、対象システムには 4 つの工程があり、各工程に 2 台ずつの機械が配置され、製品ごとに固有の工程順序が設定されている。はじめに組み込みツールでデータテーブルを設定し、インポートしたデータで使用する定義済みオブジェクトでモデルを構成する。次に、B2MML データファイルのセットをインポートしてテーブルを生成する。さらに、データ分析用のダッシュボードレポートとテーブルレポートもインポートする。

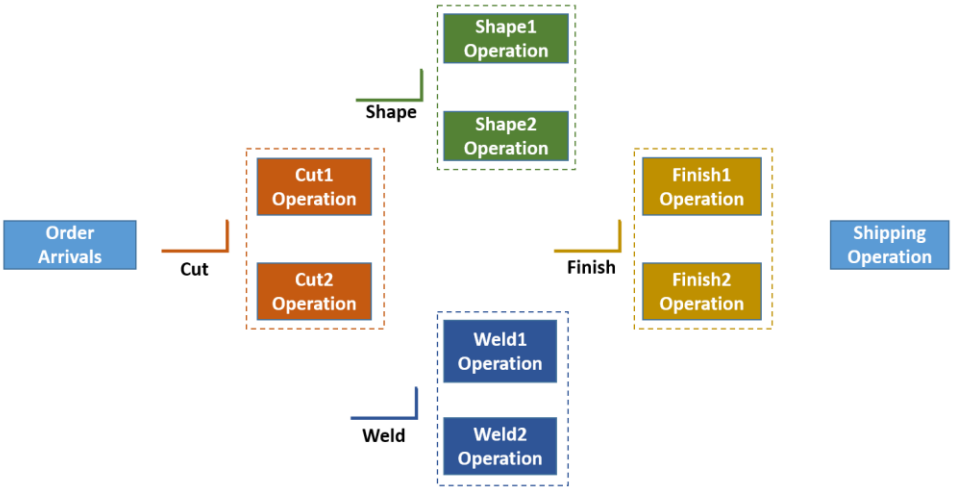


図 12.21　Model 12-2 のシステム概要

12.12.1　データインポートのためのモデル設定

Simio B2MML 準拠テーブルには次のものがある：Resources、Routing Destinations、Materials、Material Lots、Manufacturing Orders、Routings、Bill Of Materials、Work In Process、Manufacturing Orders Output。これらのテーブルをすべて作成するが、インポートは最後のテーブルを除いて行う。まず、モデルの構成を行うため、File ボタンをクリックし、New From Template を選択する。これにより、図 12.22 のテンプレート一覧が表示される。

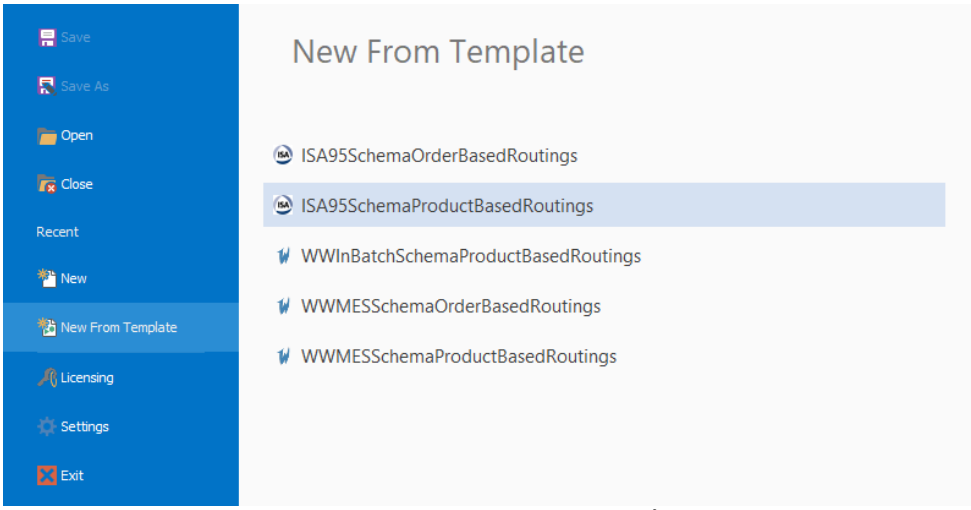


図 12.22　New From Template のテンプレート一覧

このモデルには、B2MML に準拠したテーブルを生成する ISA95SchemaProductBasedRoutings テンプレートを選択する。このテンプレートを選択すると、Data タブに Resources、Routing Destinations、Materials、Material Lots、Manufacturing Orders、Routings、Bill Of Materials、Work In Process、Manufacturing Orders Output の各テーブルが表示される。

12.12.2　データインポート

これでデータをインポートする準備が整った。Resources テーブルを選択し、Create Binding オプションで CSV を選択後、student ダウンロードの Model_12_02_DataFiles フォルダから Resources.csv を指定する。Table リボンの Import ボタンをクリックし、Facility ビューに移動すると、リソースがモデルに追加されていることが確認できる。

続いて Data タブに戻り、残りの 7 つのテーブルに対して同様の手順を実行する。各テーブルを対応する CSV ファイルにバインドし、インポートを行う。Facility ビューで完成したモデルを確認できる。図 12.23 の Navigation ビューには、インポートによって追加された全オブジェクトが表示されている。選択されている Shape1 オブジェクトは、SchedServer オブジェクトとして設定されており、Work Schedule、Processing Tasks、Assignments などの多くのプロパティが、テーブルから自動的にデータを取得するよう事前設定されていることが Properties ウィンドウに示されている。

12.12.3　モデルの実行と評価分析

この節のモデルはデータファイルのみで構築されており、インタラクティブに実行してアニメーションを確認できる。実行前に注意すべき重要な点がある。インポートしたサンプルデータには 2017 年 12 月の注文の Release Dates と Due Dates が含まれているため、現在の日付でモデルを実行すると、すべての注文に大幅な遅延が発生してしまう。この問題に対するもっとも簡単な解決策は、モデルの開始時間を 12/2/2017 8:00 に設定することだ。

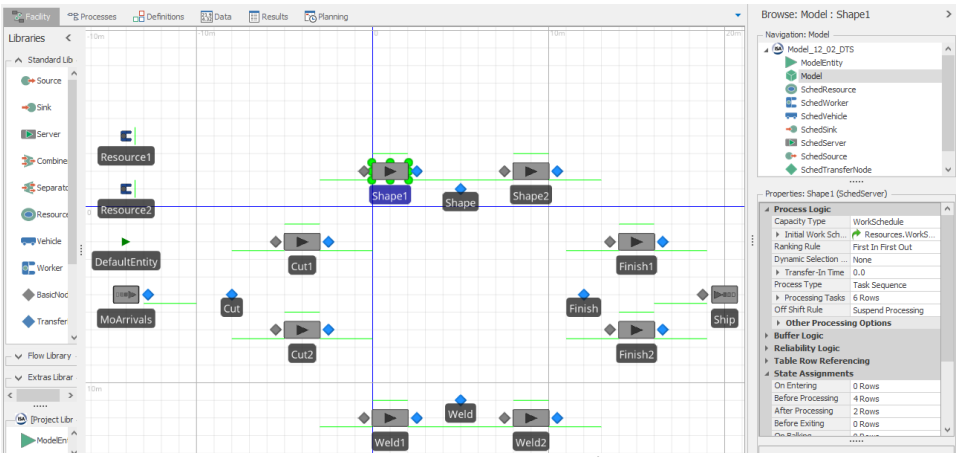


図 12.23　Model 12-2 のデータインポート完了後の状態

このモデルでスケジューリングを行う前に、Run リボンの Advanced Options から Enable Interactive Logging を選択する必要がある。各カスタムオブジェクトには、独自のリソース利用履歴を記録する機能が設定されている。その後、Planning タブに移動して Create Plan ボタンをクリックすると、ガントチャートなどの分析結果が生成される。

次に、このデータスキーマ用に設計された定義済みダッシュボードをインポートする。これらのダッシュボードは XML ファイル形式で、CSV ファイルと同じフォルダに保存されている。用意されている 3 つのダッシュボード（原材料の詳細、オーダの詳細、オペレータ用の差立リスト）をインポートするには、以下の手順で行う。まず、Results タブの Dashboard Reports ウィンドウ（Planning の Results ウィンドウ**ではない**）に移動し、Dashboards リボンを選択する。次に、Import ボタンをクリックし、同じフォルダから Dispatch List.xml を選択する。同様の手順で Materials.xml と Order Details.xml もインポートする。最後に、Planning タブ→Results ウィンドウ→Dashboard Report サブタブで、3 つのレポートから表示したいものを選択できる。図 12.24 は、Order Details ダッシュボードレポートの表示例である。

最後に、いくつかの従来型レポートを追加しよう。これらのレポートをインポートするには、Results タブの Table Reports ウィンドウ（Planning の Results ウィンドウ**ではない**）に移動し、Table Reports リボンを選択する。ManufacturingOrdersOutput の Import ボタンをクリックし、以前と同じフォルダから DispatchListReport.repx ファイルを選択する。同様の手順で ManufacturingOrders に OrderDetails.repx ファイルをインポートする。これら 2 つのファイルをインポートすると、Planning タブで使用するレポートが定義される。Planning タブ→Results ウィンドウ→Table Reports サブタブで、新しく追加された 2 つのカスタムレポートから表示したいものを選択できる。

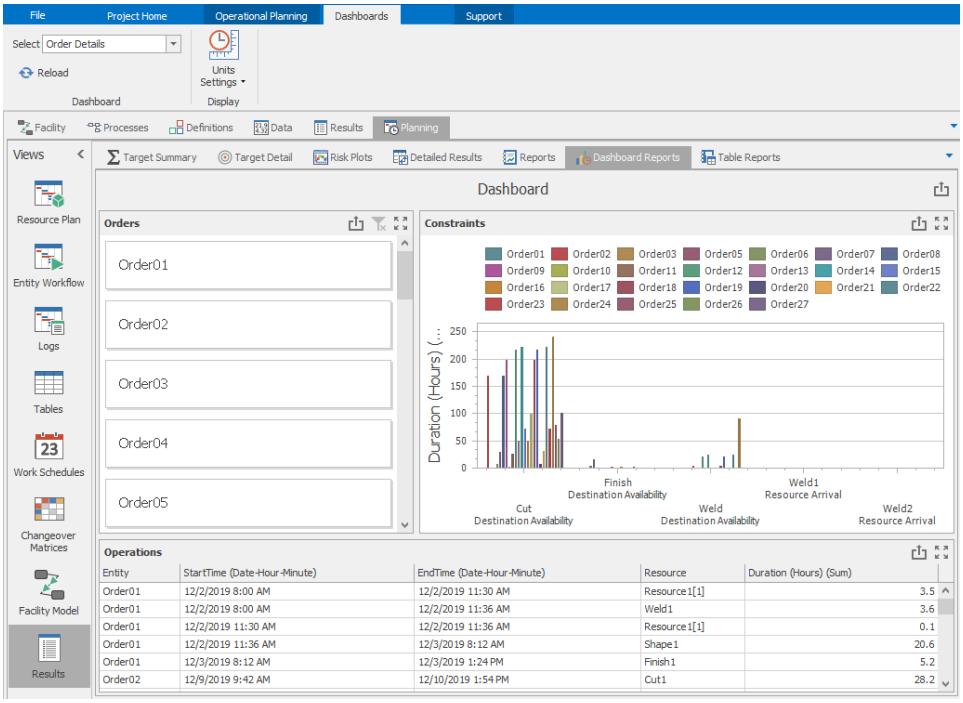


図 12.24　Model 12-2 の Order Details ダッシュボードレポート

この例は単純なものだが、B2MML、Wonderware MES、SAP ERP システムなどの既存のデータソースからモデル全体を構築できる可能性を示している。このアプローチでは、少ない労力で実用的な初期モデルを作成できる。さらに、詳細な設定とロジックを追加することで、より優れたソリューションへと発展させることも可能だ。これは非常に強力なアプローチといえる。

12.13　Uniko スケジューリング事例

Uniko Designs Grupo（Uniko）は 1999 年設立のグアテマラの小規模企業で、家族とともに過ごすユニークな空間の創造に特化している。建築用仕上げ材、木製家具（キッチンなど）、メラミンや MDF ボードを使用した室内外用家具を製造している。提案開発、製造、設置を行うプロフェッショナルチームを擁し、キッチン、クローゼット、パーゴラ（日陰棚）の需要が拡大するローカル市場で事業を展開している。会社は現在、成長と近代化を進めている。

事業拡大に伴う生産量の増加、新規従業員の採用、新しい製造技術の導入により、既存のインフラへの負荷が増している。小規模企業として、多くの従業員が複数のスキルを持っているため、最適な人員配置の決定が困難である。これまでは納期見積もり（納期回答）や、人員配置・優先順位づけの日々の管理判断に、シンプルなスプレッドシートベースのシステムを使用してきた。新しい設備とソフトウェアの導入で生産性向上の機会が生まれたものの、多くの変更が同時に発生したため、リソースの最適な割り当てによる生産最大化の方法を把握することが難しくなっている。そこで彼らは、既存のスプレッドシートに代わる新しいツール

を求めた。このツールには、機械、作業員、工程、代替処理オプションを含め、以下の機能が必要とされる：

・工程の詳細と相互の関連性を容易に理解できる。
・さまざまな人員配置と生産順序を試行し、生産需要への最適な対応を見出せる。
・日次作業スケジュールを生成できる。顧客注文の優先順位づけとリソースの再配分を自動化し、全体の最適化と重要注文の納期遵守を実現する。
・納期回答（CTP：Capable to Promise）を設定し、全注文の納品までを正確なスケジュールで管理できる。新規注文については、確実な完了時期の見積もりを実現する。
・上記の機能に加え、従業員が普段から使用している Excel をデータ入力に活用できる。

これらから、データ駆動型のシンプルな Simio モデルで Uniko のニーズを満たせることがわかった。シミュレーション技術に不慣れだった彼らのために、モデルを小さなフェーズに分けて構築し、各段階での試用期間を設けた。この段階的なアプローチにより、技術の習得、重要ニーズの特定と優先順位づけ、そしてモデルとデータインタフェースの簡素化が実現できた。各フェーズのテストを経て、ツールの性能と実用性を高めるための改良要望が出された。

12.13.1　Model 12-3：Uniko の基本モデル

本項では、Uniko の基本モデルを一緒に構築する。その後、課題として要求される機能拡張について検討する。モデル構築にあたり、学習と使用を容易にするため、モデルとそのデータファイルをできるだけシンプルに保つことを目標の一つとする。そのため、アジャイルモデリングの前提に基づき、当面の目的を満たすもっともシンプルな実装から始め、その後パフォーマンスを評価してモデルを必要に応じて拡張する。

プロジェクト開始時点で把握している内容の概要は以下のとおりである：

・Uniko の主力製品は 4 つある：小型キッチン（Small Kitchen）、中型キッチン（Medium Kitchen）、大型キッチン（Large Kitchen）、パーゴラ（Pergola）だ（図 12.25）。各製品には多様なバリエーションがあり、他の小規模製品も製造しているが、この 4 製品が事業の中心を占めている。モデルではこれら 4 製品をエンティティタイプとして追加する。
・作業は 16 か所の作業場（Server）、入口（Source）、出口（Sink）で実施される。作業場には多くの共通点があるが、将来的なカスタマイズに備えて、Standard ライブラリの Server からサブクラス化したカスタムサーバオブジェクトを作成することにした。
・各作業場では 20 種類の職種（Resource）が作業を担当し、職種ごとに特定の処理能力（人数）が設定されている。労働力は変動する可能性が高いため、これらの情報はスプレッドシートから読み込める Resources データテーブルで管理する。職種も将来的なカスタマイズに備え、Standard ライブラリの Resource からサブクラス化したカスタムリソースオブジェクトを用意する。

図 12.25　Unikoの製品例（各種キッチンとパーゴラ）

・4 製品は同じ作業場とリソースを共有するが、それぞれ異なる処理時間とリソース要件を持ち、製品によって省略される工程もある。工程は改善によって変更される可能性があるため、これらの情報はスプレッドシートからインポート可能な Routings シーケンステーブルで管理する。
・最後に、Orders テーブルで顧客注文の情報（納期などの重要事項を含む）を管理する。このテーブルは更新が頻繁なため、他のテーブルと同様にスプレッドシートからインポートできるようにする。

多くのスケジューリングモデルはテンプレートを使用して作成するが、今回のデータ要件は比較的シンプルなため、そのアプローチは取らない。このシンプルなモデルでは、追加のデータテーブルやカスタムオブジェクトによるオーバーヘッドは必要ない。それでは、基本モデルの構築を始めよう：

・UnikoBaseModel.spfx という名前の新しいモデルを作成する。ユーザが主にスプレッドシートで操作できるデータ生成型モデルとする。作業を始める前に 7.7 節と 7.8 節を確認しておくとよい。
・Tables ウィンドウで、JobTypes というデータテーブルを追加する。
・XLoc と ZLoc という 2 つの Real プロパティを追加する。
・JobType という名前の Object Reference 列を追加する。
・外部ファイルからエンティティタイプを自動生成できるよう、Default Value Instantiation プロパティを AutoCreateInstance に設定する。
・Object Type プロパティを ModelEntity に設定する。
・Initial Physical Characteristics を開き、X プロパティを XLoc、Z プロパティを ZLoc に定義する。これにより生成されるエンティティの X/Z 座標が設定される。また、モデル内での視認性を高めるため Length と Width を 2 とする。
・このテーブルは小規模なので、図 12.26 に示すように 4 つのエンティティタイプを直接入力する。Facility ウィンドウで(0, 0)中心点の左上に 4 つのエンテ

ィティタイプが表示されることを確認する。

・すべてが正常に動作することを確認したら、Data ウィンドウに戻り、Content タブで Save to CSV を選択して生成ファイルを確認する（CSV ファイルは通常 Excel などに自動的にロードされる）。Simio テーブルと同様の内容が表示されるはずだ。確認後、CSV ファイルを閉じる。

・次に、データのインポートのためのコネクタを設定する。Content タブで Create Binding を選択し、プルダウンから CSV Data Importer を選ぶ。これを CSV Data Importer JobType と名づける。左側の Data Connectors ボタンで、ファイル名 JobTypes.CSV を追加する。「+」キーを押すと、図 12.27 のようなファイルの内容が表示される。

・Content タブに戻り、Import Table ボタンでファイルをインポートする。手動での変更がなければ内容は同じままだが、インポートの完了を示すようにテーブルのヘッダーが更新される。

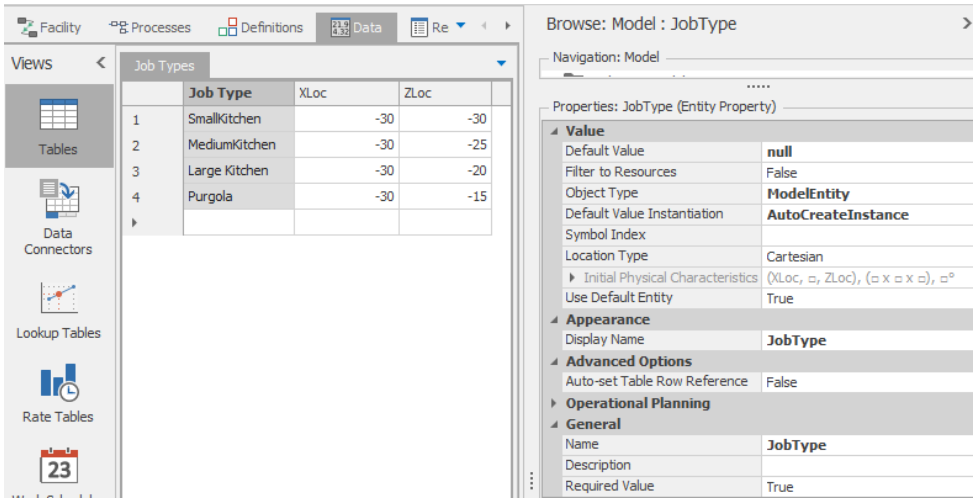


図 12.26　Model 12-3 の初期 JobTypes テーブル

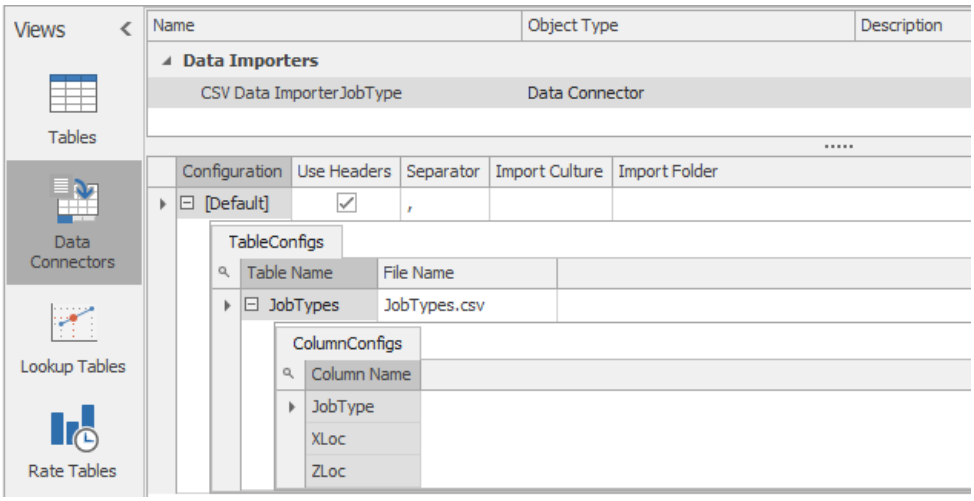


図 12.27　Uniko の JobTypes テーブルデータコネクタ

次はリソース（作業者）の設定である。ここでは既存のデータファイルからテーブルを作成する別のアプローチを採用する。

- リソースのカスタマイズが必要になる可能性が高いため、まずカスタムリソースを作成する。Facility ウィンドウで、Standard Library の Resource を右クリックして Subclass を選択する。すると右上の Browse ウィンドウに新しいオブジェクト MyResource が表示される。これは現時点では Resource の完全なコピーだが、必要に応じて編集可能だ。MyResource を右クリックして UnikoWorker に改名する。
- UnikoWorker の編集はここまでとし、（メイン）Model に戻る。その後、メインモデルの Data と Tables ウィンドウを開く。
- Resources という名前のテーブルを追加する。
- Content タブで CSV Data Importer タイプのバインディングを作成（Create Binding）し、CSV Data Importer Resources と名づける。
- Data Connectors ページで、提供された Resources.CSV ファイルを追加する。
- Table ウィンドウの Content タブに戻り、Import Table を実行する。データは取り込まれるが、デフォルトではすべてが文字列と数値として認識されるため、追加の設定が必要となる。
- Schema タブで、Capacity 列をクリックして Change Type を選択し、Standard Property の Integer を設定する。
- Resource Name 列をクリックして Change Type を選択し、Object Reference の Object を設定する。
- Object Type に UnikoWorker を選択する。
- Default Value Instantiation プロパティを AutoCreateInstance に設定する。
- Initial Physical Characteristics プロパティを開き、X プロパティを XLoc、Z プロパティを ZLoc として定義する。
- Auto-set Table Row Reference を True に設定する。

Facility ウィンドウで確認すると、すべての作業者（リソース）が配置されている。次のセクションでは、作業場所（主に機械）を追加する。これらの場所の動作を柔軟に調整できるよう、カスタマイズしたサーバオブジェクトを作成する。

- Facility ウィンドウで、Standard Library の Server を右クリックして Subclass を選択し、MyServer を右クリックして UnikoLocation に改名する。
- メインモデルに戻り、Locations という名前のテーブルを作成し、前述の手順で Locations.CSV ファイルに接続して、データをインポートして列タイプを設定する。
- 設定が成功すれば、Facility ウィンドウに 16 の作業場所が表示される。
- プロセスフローを完成させるには、Source と Sink の 2 つのオブジェクトを追加する必要がある。OrderEntry という名前の Source を座標(-10, -30)（最初の場所の左側）に配置する。プロパティは現時点ではデフォルトのままとする。17OrderComplete という名前の Sink を座標(50, 40)（最後の場所の右側）

に配置する。

これでモデルはほぼ完成だ。Facility ウィンドウは図 12.28 のような状態になる。定義したオブジェクトはすべて配置されているが、コネクタは使用していないことに注意してほしい。システム内のエンティティの移動は、シーケンステーブルで定義してインポートする Rootings によって制御される。

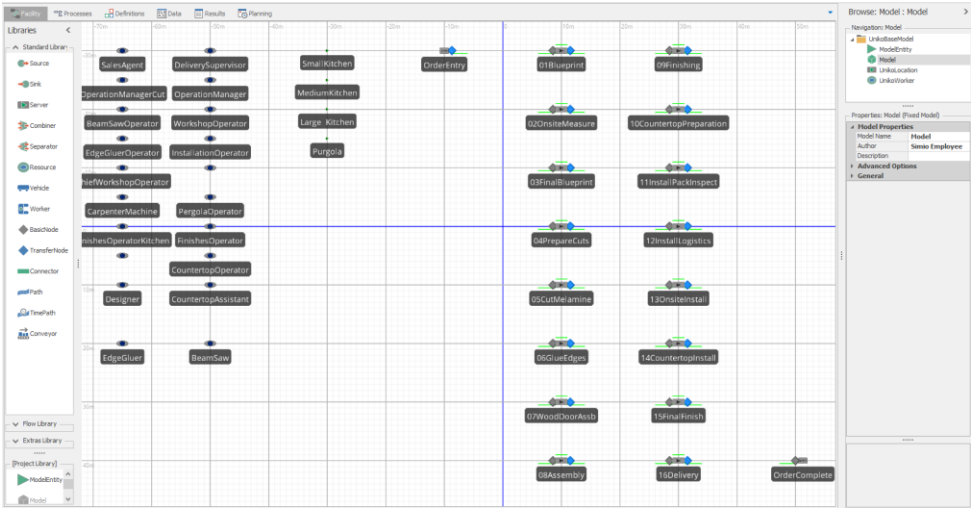


図 12.28　Model 12-3 の Facility ウィンドウ

次にインポートするのは Rootings（シーケンス）テーブルで、各製品がどのように生産されるかを記述する。各ステップで製品には特定のリソースチームが割り当てられる。このチームは最大で 3 種類の異なるリソースタイプを含み、さまざまな数量で構成される。また、チームには製品固有の処理時間が必要となる。

- Routings という名前のテーブルを作成し、先述の手順と同様に Routings.csv という提供ファイルに接続してデータをインポートする。
- 以下の指定に従って列タイプを変更する：
  - ProductType は外部キーとなり、JobTypes テーブルと関連づける。まず、JobTypes テーブルに移動して JobType 列を選び、Set Column as Key ボタンをクリックする。成功すると、JobType 列ヘッダーに鍵のアイコンが表示される。このテーブルが製品を定義する唯一の場所となり、他のすべての場所はこのテーブルを参照することになる。その後、Routings テーブルの ProductType 列に戻り、Change Type で Foreign Key に設定する。成功すると、その列の各フィールドに鍵のアイコンが表示される。また、Table Key を JobTypes.JobType に設定して、使用するキーを指定する。
  - JobStep は Integer として定義する。これは、シーケンス内での移動を容易にするためである。
  - Location は Standard Property の Sequence Destination として定義する。これはエンティティがシステム内を移動する主な方法になる。操作を簡単にするために、入力ノードの代わりにサーバ名を使用し、Accepts Any

Node プロパティを False に設定する。
- ProcessTime は Real プロパティだが、Unit Type を Time に、Default Units を Hours に指定し、Required Value を False に設定する。
- 最後の 6 列は、実際には 3 つの列のペアで構成されている。ほとんどの場所では 1 つのリソースクラスが必要だが、複数のリソースが必要な場所、あるいはリソースがまったく不要な場所もある。
  - Resource1 は、その場所で必要な主要なリソースだ。Object Reference の Object プロパティとして指定する。Filter to Resources プロパティを True に設定する。一部の場所ではリソースが不要な場合があるため、Required Value は False に設定する。
  - Quantity1 は、必要な Resource1 の数量である（必要な場合）。デフォルトを 0 にし、Required Value は False に設定する。
  - Resource2、Quantity2、Resource3、Quantity3 についても同じように変更する。

最後のテーブルは、何を生産するかを指定する Orders テーブルである。最終的には、顧客はこのファイルを外部ファイルとして利用することを想定しているが、必要に応じて拡張できるよう、まずはデータテーブルとして作成する。

・Orders という名前の新しいデータテーブルを作成する。
・以下の列を追加する：
  - OrderID（Integer）
  - JobType（Foreign Key。Table Key として JobTypes.JobType を使用）
  - Priority（Integer）
  - ReleaseDate（現時点では Real。UnitType は Time、Units は Hours）
・テスト用に次の 2 つの注文を追加する。#1 は SmallKitchen、#2 は LargeKitchen で、他のプロパティはデフォルトの 0 のままにする。

Schema タブの Show Table Schema Viewer を使って、現在のテーブルスキーマの概要を確認する（図 12.29）。Routings テーブルと Orders テーブルが JobTypes テーブルの JobType キーを参照している様子がグラフィカルに表示される。

現時点ですべてのデータが完成しているが、そのデータをモデルで使用できるように修正する必要がある。まずは注文をモデルに取り込むことから始めよう。

・Source オブジェクト OrderEntry を編集し、以下の設定を行う：
  - Entity Type は JobTypes.JobType。
  - Arrival Mode は Arrival Table。
  - Arrival Time Property は Orders.ReleaseDate。
  - 出力ノードの Entity Destination Type は By Sequence

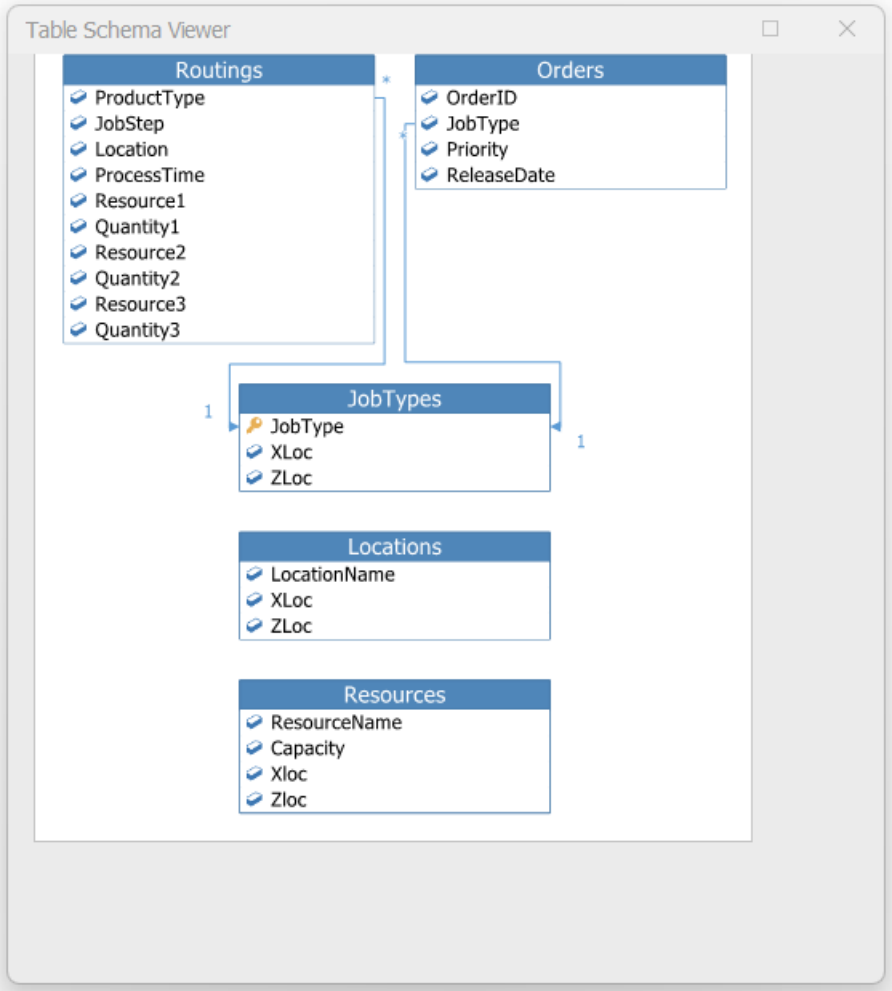


図 12.29　Model 12-3 の初期テーブルスキーマ

最初の作業場（01Blueprint）を、テーブルデータを使用するように更新する：

- Ranking Rule を Smallest Value First に設定し、Ranking Expression に Orders.Priority を指定する。
- Processing Time を Routings.ProcessTime に設定する。
- Secondary Resources カテゴリで、Resources For Processing リピートグループを開いて、各処理タプルに次を追加する：
  - Resource Name に Routings.Resource1、Units Per Resource に Routings.Quantity1 を指定する。
  - Resource Name に Routings.Resource2、Units Per Resource に Routings.Quantity2 を指定する。
  - Resource Name に Routings.Resource3、Units Per Resource に Routings.Quantity3 を指定する。
  - 一部のフィールドが空白でも問題ない。Seize では、リソース名が空白、または数量が 0 の場合、その行は無視される。完了後は図 12.30 のようになる。

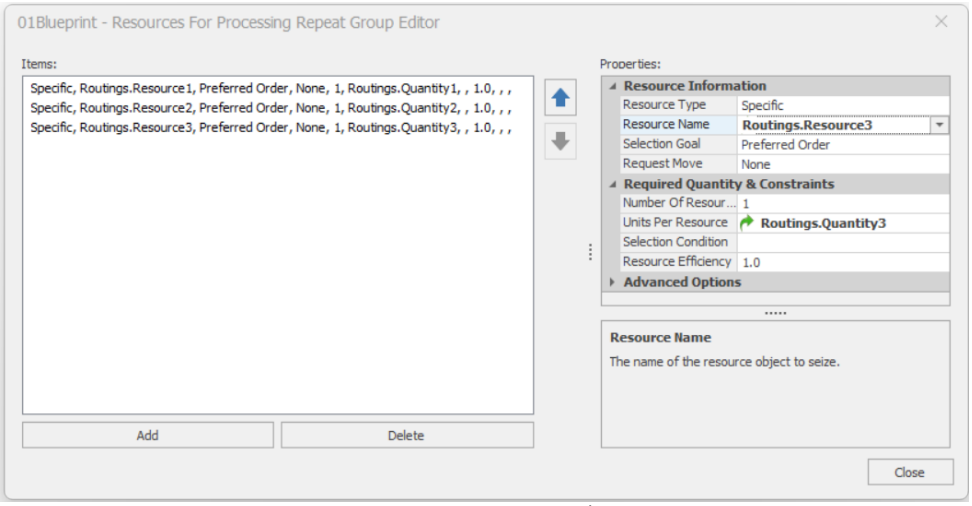


図 12.30　Model 12-3 の Routings テーブルで指定されたリソース

・Advanced Options カテゴリで、Log Resource Usage プロパティを True に設定し、ガントチャートでこの項目を確認できるようにする。
・出力ノードの Entity Destination Type を By Sequence に設定する。

この時点で、他の 15 か所の作業場にも同様の変更を加える必要があると考えるかもしれない。これらの作業場はすべて同じカスタムオブジェクト（UnikoLocation）に基づいているため、より簡単な方法がある。変更を加えた 01BluePrint オブジェクトを右クリックして、Update 'UnikoLocation' Property Defaults from This オプションを選択する。名前が示すとおり、01Blueprint オブジェクトインスタンスに加えたすべての変更が UnikoLocation オブジェクト定義にコピーされる。これで、16 か所すべてのロケーションオブジェクトインスタンスが同じ方法でデータテーブルにリンクされる。

UnikoWorker についても同様の更新が必要だ。SalesAgent インスタンスを編集し、Initial Capacity を Resources.Capacity に設定する。その後、変更した SalesAgent オブジェクトを右クリックして、Update 'UnikoWorker' Property Defaults from This オプションを選択する。これで、20 個すべてのワーカーオブジェクトインスタンスがデータテーブルから能力値を取得するようになる。

これで、モデルを実行する準備が整ったので、Facility ウィンドウでモデルの様子を確認しよう。Routings では各作業に 4 時間から 40 時間かかるため、実行が非常に遅く感じるかもしれない。Run リボンの Speed Factor を 1000 などに設定することを推奨する。実行すると、2 つの注文がデフォルトで指定された 24 時間の実行時間内に、最初の列のロケーションを順番に移動していく様子が確認できる。

Run リボンで Ending Type を Unspecified（Infinite）に設定する。これにより、イベントがなくなるまでモデルが実行され続ける。この場合、すべての注文が完了してから停止することを意味する。Planning ウィンドウを開き、Operational Planning タブの Create Plan ボタンを押す。その後、Gantt タブ

の All ボタンを押すと、すべての注文が完了するまで（またはモデル終了まで）の Gantt ウィンドウが自動的にスケール調整される。Resource Plan ウィンドウには、各ロケーションの行と、時間経過とともにどのエンティティが使用していたかが表示される（図 12.31）。ウィンドウの上半分では、最初の注文がスムーズに進み、すべてのロケーションを妨げられることなく通過する様子が確認できる。一方で、2 番目の注文は最初の注文を待つ必要があるため、当初は制約を受けている。同じウィンドウの下半分では、各作業者が時間とともにどの作業を行っているのかが確認できる。

Entity Workflow ウィンドウに切り替えると、各エンティティ（注文）の行と、そのエンティティが時間経過とともにどのリソースを使用していたのかが表示される（図 12.32）。「+」を選択して制約を展開すると、2 番目の注文の開始が 01Blueprint の空き待ちで遅延し、その後 03FinalBluprint でも若干の制約を受けていることが明確にわかる。この場合、2 番目の注文（LargeKitchen）は各ステップの処理時間が一般的に長いため、追加の制約は見られない。しかし、注文が逆の順序でリリースされていた場合、より重大な競合が発生していただろう。

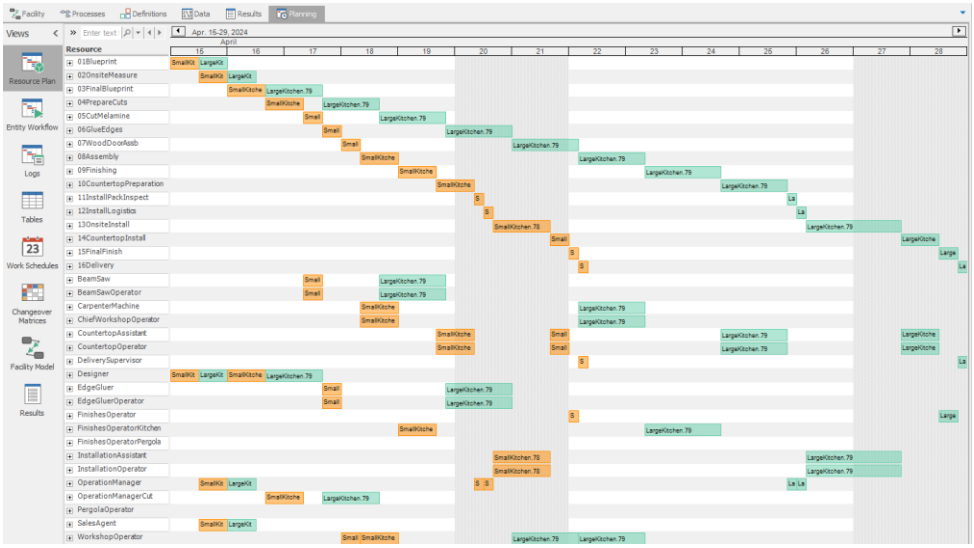


図 12.31　2 件の注文を含む Model 12-3 の Resource ガントチャート

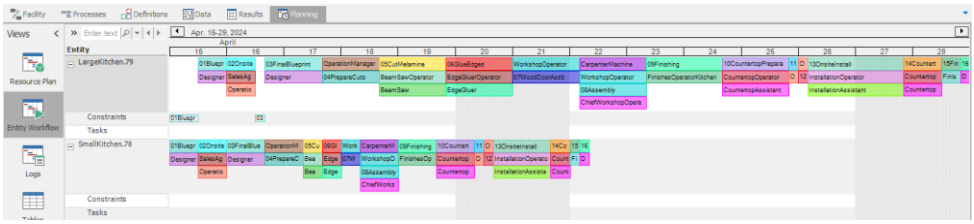


図 12.32　2 件の注文を含む Model 12-3 の Entity Workflow ガントチャート

12.13.2　Model 12-4：Uniko の拡張モデル

以前、顧客は使いやすいモデルを求めていた。つまり、**ニーズを満たす可能性のあるもっともシンプルなモデル**を持つことを意味していた。現在、完成した非常

にシンプルなモデルを用いて少し試行錯誤ができる状態なので、追加のニーズが特定され次第、段階的に機能を追加できる。本項では、それらの追加要件のいくつかに対応する。

機能を追加する前に、モデルの検証を助けるためにさらなるテストを行う必要がある。この検証は、より複雑な機能を追加する前に行うほうが簡単だ。最低限のテストとして、4 つの注文（各ジョブタイプ 1 つずつ）をシステムで実行することが考えられる。そして、アニメーション、トレース、ガントチャートの表示を確認し、モデルが設計通りに動作していることを確認する。各タイプの注文を 1 つずつ、定義された順序で追加した場合、Resource Plan の上半分は図 12.33 のように、Entity Workflow は図 12.34 のように表示される。

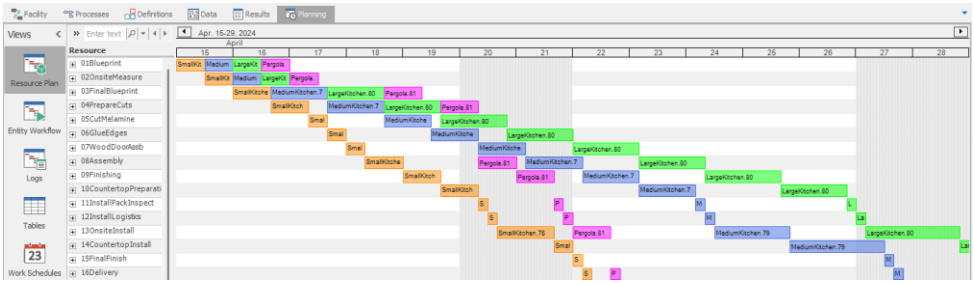


図 12.33　4 件の注文を含む Model 12-4 の Resource ガントチャート

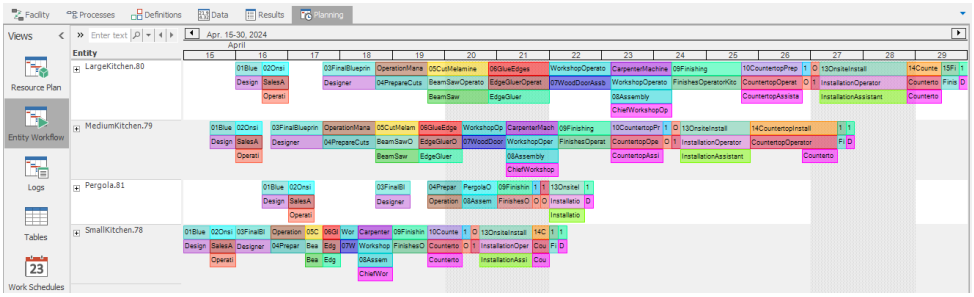


図 12.34　4 件の注文を含む Model 12-4 の Entity Workflow ガントチャート

システムにより多くの作業が入ったことで、より現実的に見えるようになった。ただし、**あまりにも**忙しそうに見える。顧客が特定した最初の追加要件の 1 つは、全スタッフが週 5 日のフルタイム勤務と土曜日の半日勤務というスケジュールを守ることだった。作業場は常に利用可能だが、作業者全員が 1 日 1 シフトの共通スケジュールに従う必要がある。そこで、以下のようにスケジュールを追加する：

- Data の Work Schedules ウィンドウに移動すると、Simio がすでに 8 時間×4 の 7 日間で構成される StandardWeek という作業スケジュールを定義しているのがわかる。これは必要な条件にかなり近いため、これを基に構築していく。
- StandardDay は、昼休みを挟んで 2 つの 4 時間の作業期間として定義されている。これは適しているが、ここで能力（Value プロパティ）は Resources テーブルからデータを取得する必要がある点に注意が必要だ。そのため、両方の Value を `Resources.Capacity` に変更する。

・さらに、土曜日の半日勤務用に新しいデイパターンを作成する必要がある。StandardDay の午前中と同じ形式の HalfDay という新しい Day Pattern を作成する。
・次に、StandardWeek の定義に戻り、Saturday に HalfDay を追加する。これらの手順を完了すると、図 12.35 のように表示される。

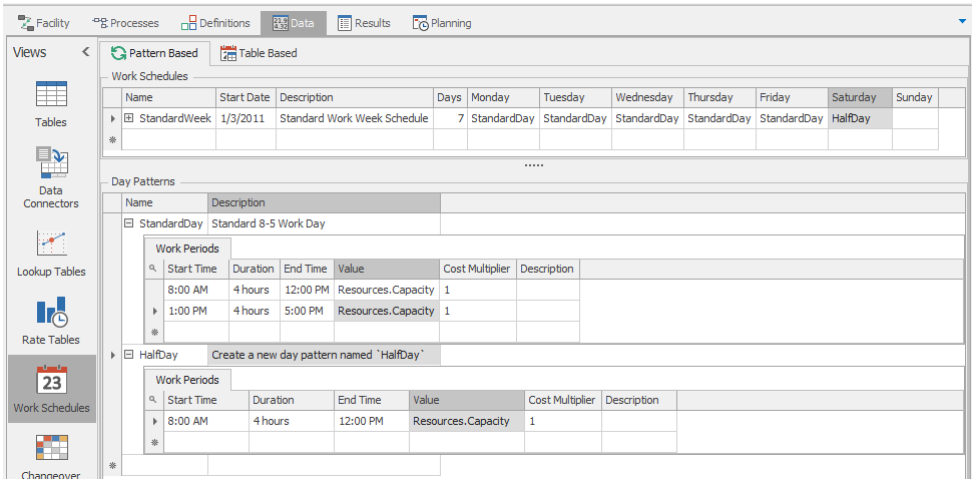


図 12.35　週 5.5 日勤務を定義した Model 12-4 のスケジュール

・スケジュールを作成したので、UnikoWorkers にそれを適用するよう指示する必要がある。
　- SalesAgent（または任意の UnikoWorker）を選択し、Properties ウィンドウで Capacity Type を WorkSchedule に変更し、Initial Work Schedule が StandardWeek であることを確認する。
　- このオブジェクトを右クリックし、Update 'UnikoWorker' Property Defaults from This オプションを選択する。これにより、20 個すべてのワーカーオブジェクトインスタンスが、テーブルで指定された各自の能力に基づいたスケジュールに従うようになる。

モデルを実行して Resource Plan を再確認すると（図 12.36）、一見変わらないように見える。しかし、4 つの注文全体のスケジュールが以前は約 14 日で完了していたが、現在は約 55 日かかるようになった。これは、リソースの利用可能時間が週 168 時間から週 44 時間に減少したためである。

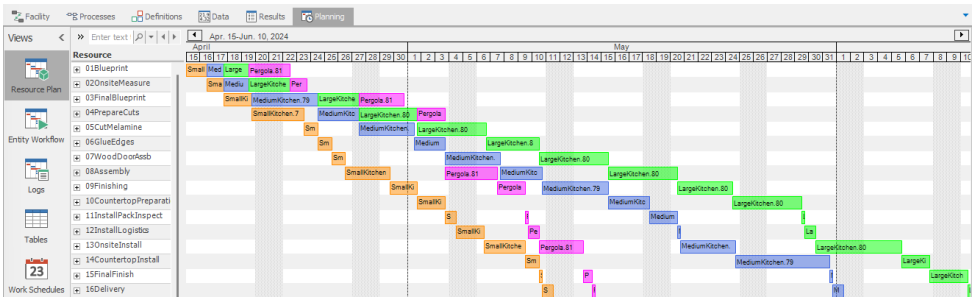


図 12.36　スケジュールに従う Model 12-4 の Resource ガントチャート

次の顧客要望による機能拡張は、注文の定義をより明確にすることだ。具体的には、各注文に顧客名（文字列）を関連づけ、その文字列とOrderIDを組み合わせて、モデル内で注文を識別できるようにしたい。

・Ordersテーブルの`OrderID`の型を`String`に変更する。
・`Customer`という名前の`String`型の列を追加し、2番目の列にする。

また、整数の代わりに実際の日付を使用し、着手日だけでなく納期も設定したい。Ordersテーブルで以下の変更を行う：

・Release Dateの型を`Standard Property`の`DateTime`に変更し、デフォルトを`1/1/2025`にする。
・すべてのリリース日を`01/01/2025`に変更する。
・`DateTime`型の新しいプロパティ`PromisedDate`（納期）を追加し、デフォルトを`1/1/2025`にする。
・標準的な処理リードタイムを6週間と想定し、すべての納期を`2/15/2025`に変更する。
・Runリボンで`Starting Type`を`1/1/2025 12:00:00 AM`に変更する。

上記の両方の変更をモデルに実装するため、SmallKitchenオブジェクトインスタンスを編集する：

・Advanced Optionsカテゴリで、Due Date Expressionを`Orders.PromisedDate`に変更する。
・Advanced Optionsカテゴリで、Display Nameを`Orders.OrderID + ":" + Orders.Customer`に変更する。
・Animationカテゴリで、Dynamic Label Textを`Orders.OrderID + ":" + Orders.Customer`に変更する。
・このオブジェクトを右クリックし、`Update 'ModelEntity' Property Defaults from This`を選択する。この手法ではすべてのプロパティが常に更新されるわけではない。たとえば、各エンティティオブジェクトインスタンスのDynamic Labelテキストとして新しいデフォルト値を使用するには、右クリックして`Reset`する必要がある。

ここで対話的にモデルを実行し、プランを生成して、今回加えた変更が正しく機能しているか確認する。たとえば、追加したサンプルデータを含むOrdersテーブルは図12.37のように表示される。

図12.38では、新しいラベルづけと1月15日にリリースされる予定の2つの新しい注文を確認できる。

ここで取り上げる最後の顧客要望による機能拡張は、基本的なWIP（仕掛品）の実装だ。たとえば、2月1日に1月など以前に開始された注文を含む計画を生成できるようにすることを目指す。各注文の詳細な進捗状況は把握していないが、どの作業ステップが**完了した**かはわかっている。これには、Ordersテーブルへのいくつかの変更とモデルへの軽微な変更が必要になる。

Job Types　Locations　Resources　Routings　Orders

	Order ID	Customer	Job Type	Priority	Release Date	Promised Date
1	1	Ajax	SmallKitchen	0	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM
2	2	Thompson	MediumKitchen	0	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM
3	3	XYZ 27	LargeKitchen	0	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM
4	4	Centurian	Pergola	0	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM
5	22	Smith	MediumKitchen	0	1/15/2025 12:00:00 AM	3/1/2025 12:00:00 AM
6	23	Smith	MediumKitchen	0	1/15/2025 12:00:00 AM	3/1/2025 12:00:00 AM

図 12.37　テスト用の Model 12-4 のサンプルオーダ

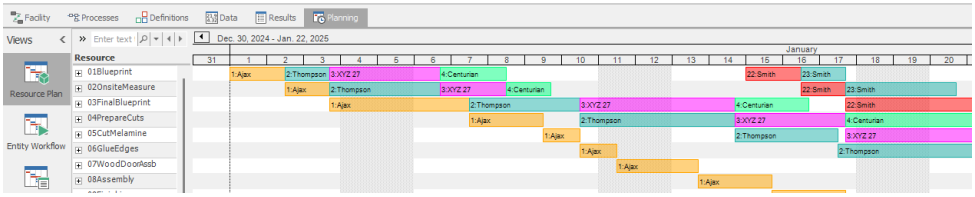


図 12.38　新しい OrderID と遅らせた着手日を含む Model 12-4 の Resource Plan

- JobType の右側に Integer 型の新しい列 LastCompletedStep を追加する。この列は、顧客が完了した最後のステップ番号を記録するためのものだ。たとえば、ステップ 7 が完了している注文の場合、このフィールドに 7 を設定する。
- Expression 型の新しい列 CalcReleaseDate を追加する。Unit Type は Time、Default Units は Hours とする。デフォルト値には、式 Math.Max(TimeNow, Orders.ReleaseDate)を設定する。この列の値は常に現在時刻に基づいて計算され、「過去」（モデル実行開始前など）にリリースされた注文は、モデル開始時にリリースされるようになる。これは LastCompletedStep と組み合わせて WIP を表現する。

Facility ビューの OrderEntry（Create）オブジェクトにいくつかの変更を加える必要がある：

- Entity Arrival Logic で Arrival Time Property を Orders.CalcReleaseDate に設定する。
- Add-On Process Triggers の Created Entity で SetRow ステップを含むプロセスを作成する。Table Name を Routings、Row Number を Orders.LastCompletedStep と指定する。これにより、デフォルトの関連づけが上書きされ、各注文は指定された作業ステップの直後から開始される。値が 0 の場合は通常通りステップ 1 から開始し、それ以外の値の場合は経路内の次の指定されたステップから開始する。
- 最後に State Assignments の Before Exiting で、State Variable Name の ModelEntity.Priority に New Value として Orders.Priority を割り当てる。これは前述の変更とは無関係だが、エンティティの優先度などの基本情報を既存のエンティティ状態で簡単に利用できるようにすると便利だ。顧客は優先度の値が低いほど重要度が高いとみなすため、「標準」優先度を 10 以上に設定し、より低い数値を優先度の高い注文に残しておくことを推奨する。

拡張された Orders テーブルは図 12.39 のようになる。

Job Types　Locations　Resources　Routings　Orders

	Order ID	Customer	Job Type	Last Completed Step	Priority	Release Date	Promised Date	CalcReleaseDate (Hours)
1	1	Ajax	SmallKitchen	0	10	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)
2	2	Thompson	MediumKitchen	0	10	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)
3	3	XYZ 27	LargeKitchen	0	10	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)
4	4	Centurian	Pergola	0	10	1/1/2025 12:00:00 AM	2/15/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)
5	22	Smith	MediumKitchen	0	10	1/15/2025 12:00:00 AM	3/1/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)
6	23	Smith	MediumKitchen	0	10	1/15/2025 12:00:00 AM	3/1/2025 12:00:00 AM	Math.Max(TimeNow,Orders.ReleaseDate)

図 12.39　テスト用の Model 12-4 の拡張オーダ

ここでも、注文が指定されたステップの直後から処理を開始することを確認するため、Orders テーブルでさまざまなシナリオを試してみるとよい。また、一部の注文がすでに開始された後の時間に Starting Type（日付）を変更して、WIP が正しく表現されているか確認してみよう。

12.13.3　Uniko モデルの結果

基本モデルとその拡張（ここで説明していないものも含む）を活用することで、Uniko は新しい設備の導入に伴い、生産を維持しつつ向上させることができた。新しいプロセスの流れを単なる推測に頼るのではなく、このモデルは客観的な見通しを提供し、潜在的な問題を事前に予測することを可能にした。また、「what if（もし～なら）」シナリオを試すことで、これまで想定していなかった新たな方法を発見することもできる。

彼らはこのモデルを定期的に使用して、作業の割り当てや注文の優先順位の管理を実施している。注文が計画通りに進まない場合（材料の到着が遅れたり、重要な人員が休暇を取ったりした場合）、モデルは代替案の評価を支援する。高い信頼性で納期を見積もることができ、発生しつつある問題を早期に把握することで、納期遅延や顧客の不満を招く前に解決することが可能だ。

このツールは新規案件の受注や価格設定の支援も行う。潜在顧客が短い納期を要求した場合、Uniko はこのモデルを「納期回答」ツールとして活用し、注文日が実現可能か、必要な追加リソース（例：残業）、および発生する可能性のあるトレードオフ（例：他の顧客注文への影響）を評価できる。

12.14　追加情報と例

12.14.1　Simio の計画・スケジューリングに関する電子書籍

Simio の Web サイトには、電子書籍『Manufacturing Digital Twin, Solution Description』がある。この電子書籍は、Textbook Publication ページ（https://textbook.simio.com/）または Support リボンの Books ボタンから入手できる。シミュレーションベースのスケジューリングについてさらに学びたい人に最適な資料だ。この本では、標準的なデータスキーマと一般的なスケジューリングの概念、そしてそれらが Simio でどのように実装されているかを詳しく解説している。

さらに、同じウェブサイトで『Deliver On Your Promise: How Simulation-Based Scheduling will Change Your Business』も入手可能だ。この電子書籍は、スケジューリングの複雑なプロセスについて深く理解したい管理者向けに書

かれている。本章で触れたトピックの詳細な解説と、実践的なケーススタディを提供している。スケジューリングの課題に取り組むマネージャには、この PDF（またはオンラインで購入可能な印刷版）をぜひ参照することを勧める。

12.14.2　スケジューリング例

Simio ソフトウェアには 6 つのスケジューリングのサンプルファイルが同梱されており、各ファイルの詳細は付属の PDF ファイルで確認できる。これらのサンプルファイルは Support リボンの Examples ボタンからアクセス可能だ。

12.14.2.1　個別部品の生産計画

この例のシステムは個別部品製造工場のジョブショップで、完成品を生産している。3 つの完成品はそれぞれ、独自のルーティング、セットアップ時間、処理時間、および各機械での資材所要量を持つ。限られたシステムリソースを考慮しながら、30 日間の生産スケジュールの作成を目指している。この工場では、部品がスケジューリングルールに従って動的に移動し、機械は機能別にグループ化されている。一部の機械では、セカンダリリソースが必要となるか、あるいは工程順序によってセットアップ時間が変わる。生産データの管理には、先に説明した B2MML ベースのデータスキーマを使用している。

12.14.2.2　自転車組立のスケジューリング

この例のシステムは自転車組立工場のジョブショップで、最終組立ラインで完成品を製造する。顧客注文に基づいて最終組立用の部品も製造し、多数の購入部品も組立工程に含まれる。限られた資産と部品を考慮しながら、7 日間の生産計画を作成することが目標である。

12.14.2.3　自転車組立のスケジューリング：MO と PO の更新

これは上記の自転車組立のスケジューリング例を拡張したものである。システム内の初期在庫に基づいて、販売オーダから**製造オーダ**（Manufacturing Orders：MO）と**購買オーダ**（Purchase Orders：PO）を生成する。この工場では限られた資源と部品を考慮しながら、14 日間の生産スケジュールの作成を目指している。

12.14.2.4　バッチ方式の飲料生産スケジューリング

この例では、限られたリソースを考慮しながら、工場の 30 日間の生産スケジュールを生成する。材料と完成品の両方のオーダを含み、原材料も制約としてモデル化している。製造工程では、3 つの材料をミキサーで混合した後、タンク（Tank）に移す。完成品は充填機（Filler）での処理後、包装機（Packing）で梱包される。各工程で作業者が必要で、完成品の処理には適切な材料がタンク内になければならない。また、ボトルやラベルなどの原材料も充填プロセスに必要な要素としてモデル化されている。

12.14.2.5　労働力の効率的なスケジューリング

この例では、作業効率の異なる労働者が働く工場のオーダをスケジューリングするため、データを基にモデルを構築する。このジョブショップでは 4 種類の完成品を生産している。モデルは、システム内の限られたリソースをじゅうぶんに

考慮しながら、6ヶ月の生産スケジュールを作成する。

工場はPress、Anodize、Mill、Paintという4つのワークステーションで構成され、機能別にグループ化されている。各ワークステーションには待機場所があり、そこから複数の機械へオーダが振り分けられる。

このモデルのテンプレートは、Standard Libraryオブジェクトの Subclass機能を使用して作成された独自のオブジェクト定義を提供する。SchedSource やSchedServerなどと呼ばれるこれらのオブジェクトは、スケジューリングモデル専用に設計されている。この例は、データテーブルに格納されたオブジェクトデータから自動的にモデルが生成される**データ駆動型**モデリングの実例となる。

12.14.2.6　並列ルート制御のスケジューリング

この例は「個別部品の生産計画」に類似しているが、1つの部品ルーティング内で発生する並列アクティビティを管理するための**並列ルート制御**の活用を示している。このシステムは Server としてモデル化されている。モデルの初期化時にPopulateImmediateSuccessorSequences プロセスが実行され、各部品種類のルーティングステップにおける後続順序番号を Immediate Successor Sequences 出力テーブルに生成する。ParallelRouteController サーバでは、Parallel RouteController_TaskProcessNameプロセスを使用して、ルーティング内の並列アクティビティの完了を待つ各ステップのエンティティを生成する。オーダごとの代表エンティティは、すべての処理タスクが完了するまでParallelRoute Controllerサーバ内に留まる。

12.15　まとめ

本章では、インダストリー4.0 とその起源、およびデジタルツインの必要性について説明した。インダストリー4.0 プロジェクトにおけるシミュレーションの役割を、従来の分析手法から、計画とスケジューリングによるデジタルツインの実現を支援する新たな手法まで解説した。また、計画とスケジューリングの一般的なアプローチとその長所・短所を検討した上で、リスクに基づく計画とスケジューリング（RPS）へのシミュレーションの応用、計画・スケジュールの課題、そしてスケジューラの新機能について紹介した。さらに、RPS を活用するためのSimioの機能についても概説した。

最初に、従来の部分的なデータ駆動方式による例を示し、次にB2MML準拠のデータファイルから完全に自動生成されたデータ駆動型モデルの例を通じて、自動モデル構築の可能性を実証した。続いて、実際のケーススタディで使用した比較的シンプルなモデルの有用性を検証し、最後に Simio ソフトウェアに搭載された6つの包括的な例題を通じて、さらなる概念の理解を深めた。

本章ではRPSについて概要のみの説明となったが、この技術には多くの可能性が秘められている。特にデジタルツインの設計、評価、実装に関する新しい応用分野が広がりつつある。RPSは商用アプリケーションはもちろん、学生プロジェクトや研究調査においても非常に有望な領域となるだろう。

12.16　問題

1．インダストリー3.0 とインダストリー4.0 の違いを比較して説明しなさい。
2．デジタルツインの定義、その構成要素、および利点について説明しなさい。
3．一般的なスケジューリング手法が直面する共通の課題と、シミュレーションおよび RPS によるその解決方法について説明しなさい。
4．Simio ダッシュボードと Simio テーブルレポートの違いを説明しなさい。
5．Model 12-1 を開き（または再構築し）、Cut と Drill 工程間の部品搬送用に分速 4 フィートの AGV（搬送車両）2 台を追加しなさい。リソースとエンティティのガントチャートを用いて、AGV によって移動が制約された最初のエンティティを特定し、そのスクリーンショットを作成しなさい。
6．Model 12-2 を考えよう。データファイルを変更して再インポートすることで、Cut 工程に Cut 3 という機械を追加しなさい。スケジュールパフォーマンスの変化を評価するには、Planning タブの Operational Planning リボンにある Save for Compare と Show Differences 機能を使用する。また、この変更による影響を示す Target Detail レポートのスクリーンショットを作成しなさい。
7．問題 6 において、さらに 1 台の機械を追加できるとすれば、どの機械を追加すべきか。その理由も説明しなさい。
8．Model 12-4 から始める。Uniko には Closet（クローゼット）という 5 番目の製品がある。クローゼットの製造工程はパーゴラと同一で、同じ設備を使用するが、処理時間が大幅に短い。必要な時間は次のとおりである：ステップ 1～4 は 4、4、8、8、ステップ 8～9 は 8、8、ステップ 11～13 は 4、4、8、ステップ 15～16 は 0、4 である（すべての時間単位は時間（hours））。この新しい製品タイプとその関連工程を追加しなさい。
a）クローゼット 1 台の製造に必要な最小の標準的な日数を算出しなさい。
b）システムが空の状態から開始した場合、5 台のクローゼットを同時に製造するのに要する時間を求めなさい。
c）クローゼットが「通常」製品と統合された低優先度の製品として、利用可能なリソースでのみ製造される場合、妥当な納期はどの程度か。理由とともに説明しなさい。
9．問題 8 の解答（グローゼットの注文が含まれていない状態）から始める。Uniko は注文管理を外部スプレッドシート（CSV 以外）で行いたい。注文データテーブルを外部スプレッドシート（例：Excel）に変換し、重要な情報（時刻ではなく日付のみなど）だけを入力できるよう簡素化する。UnikoLocation オブジェクト（全インスタンス）を更新し、Standard Dispatching Rule に Dynamic Selection Rule を追加する。Dispatching Rule を LeastSlackTime に設定し、Tie Breaker Rule を SmallestPriorityValue に設定する。この設定の意味と、予想されるモデルパフォーマンスの変化を説明しなさい。
10．問題 9 の解答から始めて、Uniko は納期回答の遵守状況を Simio で報告したい。
a）注文に目標を設定する。約束した納期より 5 日以上前の納品を「Early（早期）」、2 日以上遅れの納品「Late（遅延）」とし、その間を「On Time（納期通り）」とする。
b）6 件の注文の納期を調整し、すべてを「納期通り」または「早期」にしなさい。
c）既存の 6 件の注文に、低優先度のクローゼット注文を 4 件追加する。この処理方法と他の予定された注文への影響を評価し、プランナーが取るべき対応を提案しなさい。
11．問題 10b の解答（クローゼットなしの状態）から始める。Uniko は、キッチンを標準化している会社から中型キッチン 5 台の製造依頼を受けた。最初の 4 工程をスキップできるこの案件は、既存の仕事より重要度は低いが、クローゼットより重要度が高い。5 台のキッチンについて最早の完成日を求め、既存のコミットメントへの影響を分析しなさい。

第 13 章　AI を活用したシミュレーションモデリング

注：本章は Mohammad Dehghani と Rylan Carney の執筆・寄稿による。

デジタルツイン（DT）は、インダストリー4.0 イニシアチブを実現する重要な要素として注目されている。この変革期において、組織はデジタルツインが業務の最適化とデータ駆動型の意思決定に不可欠な役割を果たすことを認識している。物理的な資産やシステムを仮想空間に再現することで、企業は効率性と柔軟性を大幅に向上させることができる。

デジタルツインの技術が進歩する中、人工知能（AI）の統合は、この急速に発展する分野に革新をもたらしている。AI はデジタルツインの知能性と適応性を高め、インダストリー4.0 においてさらに重要な存在となっている。機械学習と予測分析を基盤とする AI は、デジタルツインに状況適応力を与え、パターンの発見、問題の予測、最適解の提案を可能にする。AI の活用により、組織はデジタルツインの潜在能力を最大限に引き出し、業務効率を向上させ、ダウンタイムを削減し、全体的な生産性を高めることができる。

離散事象シミュレーションソフトウェアのパイオニアである Simio は、ニューラルネットワークを組み込み、デジタルツインモデルに AI の機能を実装している。これにより、ユーザはモデルロジック内でニューラルネットワークを活用でき、トレーニングデータの収集が容易になり、直感的なインタフェースでニューラルネットワークモデルを学習させることができる。具体的には、以下が可能となる：

- コーディング不要でフィードフォワード型ニューラルネットワークを構築し、モデル開発時間を大幅に短縮できる。
- 意思決定の精度を向上させ、生産サイクル、リソース管理、生産能力計画の最適化が可能になる。
- 予測分析により、ダウンタイムとその影響を最小化できる。
- 学習済みのニューラルネットワーク（ONNX ファイル）を Simio に直接インポートして、モデルの知能性を向上させることができる。

Simio は 2021 年 10 月リリースのバージョン 14.230 で初めてニューラルネットワークと関連ツールを実装し、以降も継続的に機能を強化している。本章では、このツールの使用方法と、学習済みモデルをシミュレーションの意思決定モジュールとして活用する方法を解説する。

13.1　デジタルツインのレベル

ニューラルネットワークなどの AI 対応技術のシミュレーション環境への適用は、比較的新しい概念である。そのため、デジタルツインの各レベルでの適用方法には慎重な検討が必要となる。この理解は、デジタルツインの発展過程を把握し、ニューラルネットワークツールを効果的に導入する上で重要な役割を果たす。

デジタルツインは 5 つの進化レベルで構成される：1）監視、2）実験、3）最適化、4）予測、5）自動化（図 13.1）。組織は、それぞれの業務要件と意思決定プロセスに応じて、適切なデジタルツインレベルを選択する必要がある。上位レベルへの移行には、より高度なツールが必要だ。この傾向は、特に高度なデジタルツインモデルにおいて、AI 対応ツールが不可欠であることを明確に示している。

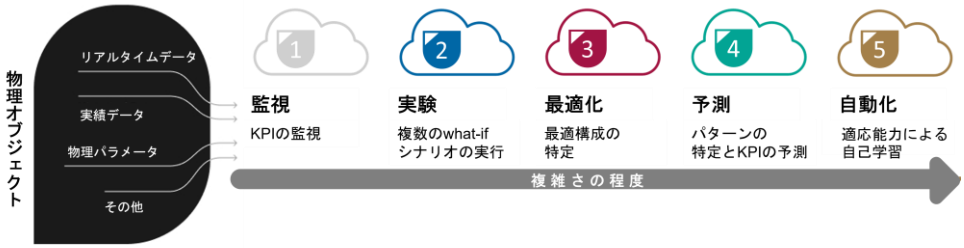


図 13.1　デジタルツインの進化レベル：インダストリー4.0 における展開

- **レベル 1　監視**：このレベルでは、システムの問題点、ボトルネック、期待値からの逸脱を特定できる。
- **レベル 2　実験**：組織はシミュレーションモデルを用いて複数の what-if シナリオを実行し、各種変数がシステムのパフォーマンスに与える影響を分析する。この実験を通じて、意思決定者は最適な結果と KPI（Key Performance Indicator：重要業績評価指標）を達成するようモデルを改良できる。
- **レベル 3　最適化**：デジタルツインの発展とともに、最適化が重要な焦点となる。多数のシナリオや意思決定変数が存在する場合、最適化モデルが不可欠となる。これらのモデルはさまざまなシナリオの作成とテストを可能にし、システムに最適なパラメータの組み合わせを特定できる。Simio は 3 種類のシナリオ選択・最適化ツールを提供している（図 13.2 参照）：

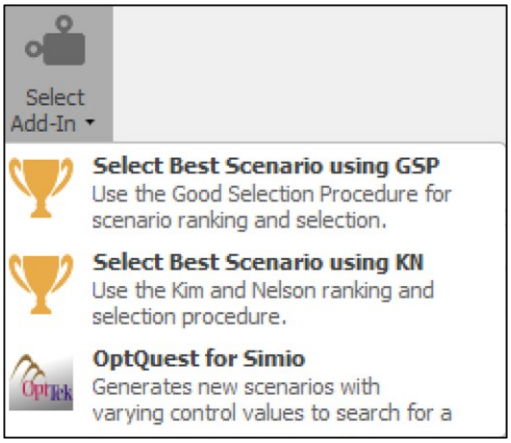


図 13.2　Simio の実験機能における最適化ツール

GPS（Good Selection Procedure）と **KN**（Kim and Nelson）ランクづけ・選択手順は、事前に定義された選択肢から最適なシナリオを特定する。加えて、実

験用最適化ツール **OptQuest** も提供している。OptQuest は制御パラメータを最適化するブラックボックス最適化ツールで、目的関数の最大化または最小化を目指し、制御設定を段階的に改良しながら最適解を探索する。

- **レベル 4　予測：**高度なシミュレーションモデルは、質の高い合成データを生成し、システムの挙動とパターンを正確に予測できる。ニューラルネットワークなどの予測モデルの組み込みにより、デジタルツインの適応性が向上し、複雑なシナリオでも的確な意思決定が可能になる。Simio の Neural Network ツールはこうした予測モデルの実装と調整に特化しており、さまざまな産業環境で高精度な予測とスマートな意思決定を実現する。
- **レベル 5　自動化：**このレベルでシステムは、自己適応と自律的な問題解決が可能な成熟段階に達する。シミュレーションモデルに組み込まれた強化学習（Reinforcement Learning：RL）モジュールが自動化を促進し、システムの中断を最小限に抑えながら、予測可能および予測不可能な事象に対応した意思決定を行う。このレベルには入念な訓練と大量のデータが必要だが、豊富なデータを扱うデジタルツインシミュレーション環境と相性がよい。

13.2　ディープラーニングの基礎

ディープラーニングは、AI 分野における機械学習（Machine Learning：ML）の一分野で、人間の脳の仕組みを参考に設計されている。ディープラーニングのニューラルネットワークは、生物学的なニューロンの信号伝達の仕組みを模倣し、脳の情報処理方式を取り入れている。図 13.3 に示すように、ニューラルネットワークは 3 つの主要な層から構成され、それぞれの層が学習と予測の過程で特有の役割を担っている：

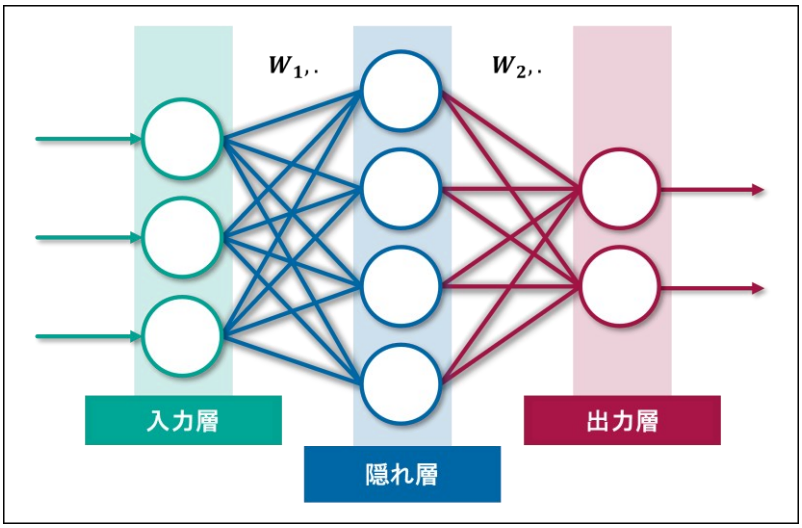


図 13.3　フィードフォワード型ニューラルネットワークの基本構造

1. **入力層：**ニューラルネットワークへの入口として、特定の入力特徴を持つユニットを通じて生データを受け取る。受け取ったデータは整理され、後続の層へと送られ、学習の基盤となる。

2. **隠れ層**：入力層と出力層の間に位置する、ニューラルネットワークの中核的要素である。複数の層とノードから構成され、この複数の隠れ層の存在が「ディープ」という名称の由来となっている。
3. **出力層**：ニューラルネットワークの最終層であり、出力を生成する役割を担う。隠れ層で処理された情報を、画像のオブジェクト分類や数値予測といった、目的のタスクに応じた形式へと変換する。

これらの層は**フィードフォワード方式**で接続され、入力層から隠れ層を経て出力層へと一方向の情報の流れを形成する。この順次的な流れは、ニューロン間の接続によって制御され、各接続は重みという値で強さが特徴づけられる。重みの符号（−または＋）は後続層の出力に影響を与え、重み間の相互作用により層を通じて信号が変換・伝搬される。重みの値が小さいほど影響は穏やかになり、大きいほどニューラルネットワークの出力への影響が顕著になる。このように、重みはニューロン間の接続強度を制御し、入力の変化が出力に与える影響を決定する重要な役割を担う。

データが隠れ層を通過する過程で、重みは特徴の階層的な変換を行う。各層において特定の特徴が抽出・表現され、重み（W）が各要素の重要度を決定する。この階層的な特徴抽出はディープニューラルネットワークの特徴であり、画像認識や自然言語処理など、幅広い分野で優れた性能を示している。ディープラーニングについて詳細を知りたい読者は、Goodfellow et al.（2016）を参照されたい。

自動車価格を予測するニューラルネットワークのトレーニング例を見てみよう。このトレーニングには、多数の自動車の属性値と実際の価格データが必要となる。これらのデータを用いて、ニューラルネットワークの最適な重みを決定する。入力特徴として、走行距離、気筒数、燃費、その他の関連属性を使用し、最終的な目標はこれらの入力特徴から自動車価格を正確に予測できるモデルを構築することである（図 13.4）。

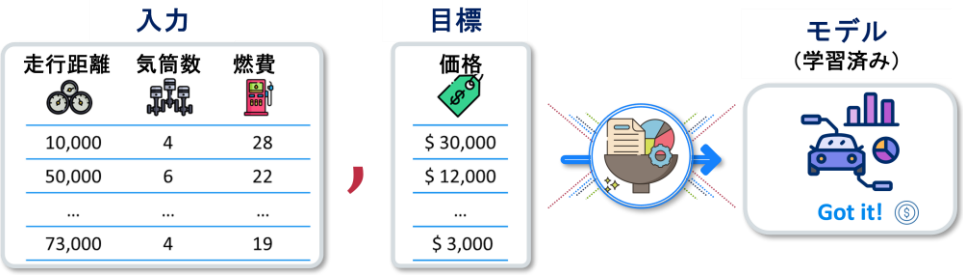


図 13.4　自動車価格予測用ニューラルネットワークの入力特徴

ニューラルネットワークは、隠れ層と接続の重みによってデータのパターンと関係性を学習していく。モデルは学習を繰り返すたびに重みを自動調整し、予測誤差を最小化して精度を高めていく。トレーニングが完了後は、新しい自動車の特性データを入力するだけで、学習済みモデルが価格を予測できるようになる。

13.2.1　Simio のニューラルネットワークツール

離散事象シミュレーションとニューラルネットワークの統合により、予測能力

を最大限に活用でき、組織はデジタルツインをレベル 4 まで高度化できる。両者は相互に補完し合う特性を持つ。

シミュレーションモデルは複雑なシステムのモデリングに強固なフレームワークを提供し、プロセスとイベントを詳細に分析できる。シミュレーションモデルの検証と妥当性確認により、研究対象システムから質の高い構造化データを生成でき、ニューラルネットワークの大規模データ要件に効果的に対応できる。さらに、ニューラルネットワークはこのシミュレーション生成データを活用して学習モデルを開発し、システム内の潜在的なパターンを特定できる。

この相乗効果により、意思決定プロセスが強化され、大きな利点がもたらされる。トレーニング完了後は、学習済みモデルをシミュレーションと連携した意思決定モジュールとして活用でき、プロセスの最適化とシステム全体のパフォーマンス向上が実現される。

Simio は、シミュレーション環境で生成された合成データを活用できるニューラルネットワークの組み込みサポートを提供している。この機能により、実際の工場現場からのデータ収集にかかる時間とリソースを大幅に削減しながら、効率的にトレーニングデータを作成できる。また、Simio はニューラルネットワークモデルのトレーニング用インタフェースを提供し、シミュレーションデータをトレーニングプロセスにシームレスに統合する。これにより、ユーザは以下の利点を得られる：

- **統合プラットフォーム**：Simio はニューラルネットワークモデリングとシミュレーション機能を単一のプラットフォームで提供するため、外部パッケージは不要である。ユーザは追加のソフトウェアなしで、ニューラルネットワーク機能をシミュレーション作業にシームレスに統合できる。
- **ノーコード**：Simio はニューラルネットワークツールの使用を簡素化し、プログラミングの知識がなくても利用できる。このコードレスアプローチにより、より多くのユーザがニューラルネットワークのモデリングに取り組める。また、パラメータや重みの微調整のための最適化機能に加え、トレーニングの進捗やモデルの精度を視覚的に確認できる機能も搭載している。
- **データの可用性**：Simio のような離散事象シミュレーションパッケージは、イベントベースのデータログ収集に優れており、ニューラルネットワークの学習に必要な堅牢なデータソースを提供する。ニューラルネットワークモデルの性能はトレーニングデータの品質に依存するため、正確な入力特徴の収集が不可欠である。Simio はシステムのあらゆる要素（エンティティ、サーバ、待ち行列など）からデータを収集でき、高品質なトレーニングデータを豊富に生成できる。これにより、ユーザはシステム全体から必要なデータを効率的に収集し、モデリングのニーズに対応できる。また、開発者はモデルの性能向上のために、必要に応じて入力特徴を追加することも可能である。
- **効率的なデータ収集**：シミュレーションモデリングにおけるデータ収集は、現実世界での収集と比較して、迅速、安全、かつ効率的である。この効率性により、ニューラルネットワークのトレーニングプロセスが大幅に向上する。

・**ONNX との互換性**：Simio は ONNX（Open Neural Network Exchange）に対応しており、学習済みニューラルネットワークや他の機械学習モデルのインポート・エクスポートが可能だ。ONNX により、高度なトレーニングが容易になり、異なるディープラーニングフレームワーク間での相互運用性が向上する。
・**性能評価**：Simio 内での訓練済みモデルの性能評価は、分析における重要なステップである。仮想環境でのシミュレーション実験により、実世界に近いシナリオでのモデル性能について貴重な知見が得られる。このアプローチにより、ユーザは実運用前にモデルを検証し、実用的な適用での有効性を確認できる。

Simio のニューラルネットワークの実装は、フィードフォワード型アーキテクチャに従っており、データがネットワークの入力ニューロンから出力ニューロンへと一方向に流れ、入力ニューロンへのループバックは発生しない。現在の実装では単一出力のみをサポートしているため、複数出力の予測には複数のネットワークを訓練して適用する必要がある。

13.2.2　ニューラルネットワークツールの実装手順

Simio のニューラルネットワークツールは、シンプルなステップで利用できる体系的なアプローチを提供し、正確な予測と洞察を得ることができる。本章では、製造環境での効果的な活用方法を、具体的な実装ステップとケーススタディを通じて解説する。図 13.5 は、ニューラルネットワークツールを使用する際の主要なステップを示している。これには、ベースラインモデルの作成、データ収集、トレーニングモジュールの適用、テストの実施が含まれる。これらの各ステップは、ニューラルネットワークツールを成功裏に活用するために不可欠な要素となる。

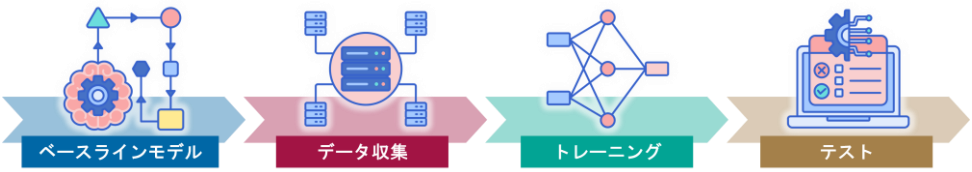


図 13.5　ニューラルネットワークツールの実装手順

・**ステップ 1　ベースラインモデル**：ニューラルネットワークツールを効果的に活用するための基盤となるシミュレーションモデルを開発する。このモデルは、後続のデータ収集ステップで必要なデータを適切に収集できるよう設計し、必要に応じて調整を加える。
・**ステップ 2　データ収集**：ベースラインモデルの完成後、Simio のニューラルネットワークのデータ収集モジュールを使用する。この段階で、モデルは設定された仕様に基づいて必要なデータポイントを収集し、分析用に保存する。
・**ステップ 3　トレーニング**：収集したデータを用いて、ニューラルネットワークのトレーニングを実施する。データをニューラルネットワークの構造に統合することで、システム内の潜在的なパターンを特定する。この過程で、フィードフォワード型ニューラルネットワークを用いた予測モデルが構築され、対象システムの正確な予測が可能となる。

・**ステップ 4 テスト**：トレーニング完了後、モデルの性能評価を行う。優れたモデルの開発には、データ収集、トレーニング、テストの反復が不可欠である。初期段階では、データポイントとトレーニングの関連性について理解が限られる場合があるが、このプロセスを繰り返すことで、モデルの性能を目標レベルまで向上させることができる。

13.3　例題 13-1：ニューラルネットワークによるインテリジェント作業手配

本節では、ニューラルネットワークツールの効果的な実装と展開についてケーススタディを用いて説明する。単純な製造環境での実例を通じて、正確な予測と洞察を得るための段階的なプロセスを解説する。このモデルでは、訓練されたニューラルネットワークが作業手配の意思決定を行う「AI 駆動型アプローチ」を採用している。このアプローチの核心は、各ワークステーションにおける注文ごとの**予測**完了時間にある。各オーダは、その特性、ワークステーションの状態、システム全体の状況を考慮した上で、予測完了時間が最短となるワークステーションに割り当てられる。このデータ駆動型の手法により、ランダムな作業手配や周期的な作業手配、さらには従来のヒューリスティックベースのアプローチ（最短処理時間や最短待ち行列長など）を超える、より高度な意思決定を実現できる。

13.3.1　ベースラインモデル：製造環境

本章で紹介するインテリジェントアプローチは、ニューラルネットワークによる予測を基盤とする。このニューラルネットワークは、オーダがワークステーションに到着した時点で、その予測完了時間を算出する。そして、最短の予測完了時間を示すワークステーションにオーダを振り分ける。この予測では、処理時間、オーダ種類、ワークステーションの状態、システム全体の状況など、多様なオーダの特性を考慮する。ベースラインモデルを図 13.6 に示す。

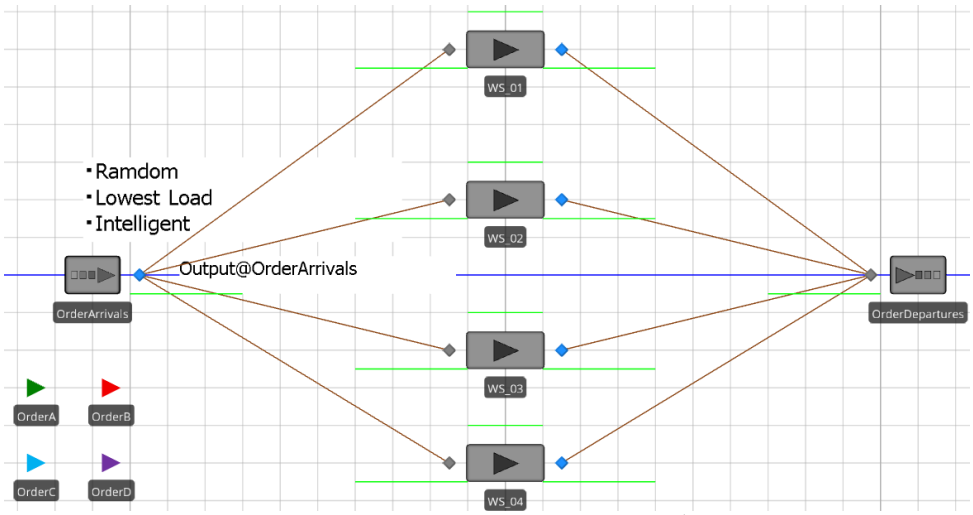


図 13.6　例題の製造環境の Simio モデル

本ケースでは、AI 活用アプローチの有効性を評価するため、以下の 3 つのワークステーションへのオーダ作業手配シナリオを分析する：

- **ランダム作業手配**：オーダを無作為にワークステーションに割り当てる。
- **ヒューリスティック作業手配**：オーダをもっとも負荷の低いワークステーションに割り当てる（5.4 節参照）。
- **インテリジェント作業手配**：ニューラルネットワークによる予測完了時間に基づいてオーダを割り当てる。

ニューラルネットワークモデルの詳細に入る前に、まずベースラインモデルの設定と構成について概説する。

13.3.1.1　Source：OrderArrivals

オーダ作成は、オーダミックスを定義する tblOrderTypes テーブルのデータに基づく。オーダの内訳は、OrderA（40%）、OrderB（15%）、OrderC（25%）、OrderD（20%）となる（図 13.7）。

tbl Order Types

	Order Type	Mix
1	OrderA	0.4
2	OrderB	0.15
3	OrderC	0.25
4	OrderD	0.2

図 13.7　tblOrderTypes テーブルで定義したオーダミックスの構成

Source は、Table Row Referencing の Row Number プロパティを tblOrderTypes.Mix.RandomRow に設定し、これらのオーダミックスの確率に従って注文を生成する。システムは Mix 値に基づいてテーブルから行を確率的に選択し、対応する注文を生成・送信する（図 13.8）。

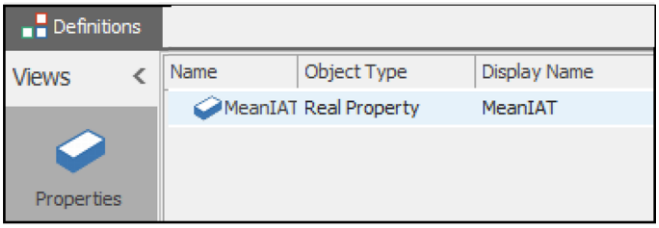
Properties: OrderArrivals (Source)

Entity Arrival Logic	
Entity Type	**tblOrderTypes.OrderType**
Arrival Mode	Interarrival Time
Time Offset	0.0
Interarrival Time	**Random.Exponential(MeanIAT)**
Entities Per Arrival	1
Stopping Conditions	
Buffer Logic	
Table Row Referencing	
Before Creating Entities	
Action Type	Reference Existing Row
Table Name	**tblOrderTypes**
Row Number	**tblOrderTypes.Mix.RandomRow**
On Created Entity	

図 13.8　Source オブジェクト OrderArrivals のプロパティ

到着率を実験目的で調整するため、指数分布 Random.Exponential(MeanIAT)

を使用し、平均到着時間間隔（MeanIAT）を変更可能にしている。初期設定では平均 IAT は 4 分に設定しており、これは 1 時間当たり平均 15 オーダの到着率に相当する。平均 IAT を高く設定すればオーダの到着頻度が下がり混雑が緩和され、低く設定すれば混雑が発生しやすくなる。この実験により、さまざまな混雑レベルにおける各作業手配ルールの性能を評価できる。プロパティの設定については図 13.9 を参照してほしい。



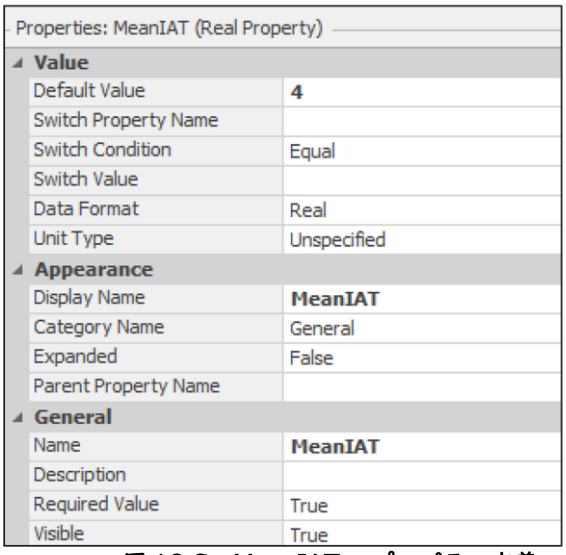


図 13.9　MeanIAT のプロパティ定義

13.3.1.2　Transfer ノード：Output@OrderArrivals

オーダが作成されると、OrderArrivals の出力ノード（Output@OrderArrivals）で設定された作業手配ロジックに基づいて、いずれかのワークステーションに割り当てられる。作業手配を動的に決定するため、Selection Goal プロパティは WSSelctionRule という新しい参照プロパティに設定される（図 13.10）。このプロパティにより、**Smallest Distance**、**Largest Distance**、**Preferred Order**、**Cyclic**、**Random**、**Smallest Value**、**Largest Value** などの動的な目標選択が可能になる。ユーザは実験の一環として、これらの作業手配アプローチから 1 つを選択してパフォーマンスを比較できる。デフォルトの選択規則は「Random」に設定されている。

このプロパティは、Output@OrderArrivals の作業手配方式として機能する。Entity Destination Type を Select From List に設定すると、オーダは指定された作業手配方式に従って処理される（図 13.11）。すべてのワークステーション

入力ノードを含む tblWorkstations テーブル（図 13.12）を通じて、ランダムまたはインテリジェントな作業手配が実行される。Transfer ノードは Selection Goal に基づいてワークステーションを選択し、プロパティ WSSelctionRule を選択基準として参照する。これにより、ユーザが選択した方式に応じてオーダが適切なサーバに振り分けられる。具体的なシナリオは以下のとおりである：

Properties: WSSelectionRule (Enumeration Property)		
Value		
Default Value	Random	
Switch Property Name		
Switch Condition	Equal	
Switch Value		
Enum Type	**NodeSelectionGoal**	
Appearance		
Display Name	**WSSelectionRule**	
Category Name	General	
Expanded	False	
Parent Property Name		
Captions		
SmallestDistance	☑	SmallestDistance
LargestDistance	☑	LargestDistance
PreferredOrder	☑	PreferredOrder
Cyclic	☑	Cyclic
Random	☑	Random
SmallestValue	☑	SmallestValue
LargestValue	☑	LargestValue
General		
Name	**WSSelectionRule**	
Description		
Required Value	True	
Visible	True	

図 13.10　WSSelectionRule プロパティ参照

Properties: Output@OrderArrivals (TransferNode)	
Crossing Logic	
Initial Traveler Capacity	Infinity
Entry Ranking Rule	First In First Out
Entry Exemption Condition	TransferNode.RideOnTransporter && !Entity.Is.Transporter
Routing Logic	
Destination	
Entity Destination Type	**Select From List**
Node List Name	**tblWorkstations.Node**
Selection Goal	**WSSelectionRule**
Value Expression	Candidate.Node.AssociatedStationLoad
Selection Condition	
Blocked Destination Rule	Select Available Only
Route Constraint Logic	0 Rows
Route Request Ranking Rule	First In First Out
Route Request Dynamic Selection Rule	None
Network Path	
Outbound Travel Mode	Continue
Outbound Link Preference	Any
Outbound Link Rule	Shortest Path
Path Planner Name	

図 13.11　Output@OrderArrivals のプロパティ設定

tbl Order Types | tbl Workstations | tbl Order Destinations

	Workstation	OType	Node	x Loc	z Loc
1	WS_01	Server	Input@WS_01	20	-6
2	WS_02	Server	Input@WS_02	20	-2
3	WS_03	Server	Input@WS_03	20	2
4	WS_04	Server	Input@WS_04	20	5

図 13.12　tblWorkstations テーブル

1. **Random**（デフォルト）：このオプションは**ランダム作業手配**を実装し、各ワークステーションに均等な確率（25%）で割り当てる。選択に対して式は使用されず、オーダは予測なしで配分される。
2. **Lowest Load**：オーダは、**最小**負荷（負荷は処理中、移動中、または待機中のオーダ総数）のワークステーションへ割り当てられる。選択式は Candidate.Node.AssociatedStationLoad に設定される。
3. **Intelligent Dispatching**：このオプションはインテリジェント作業手配を実装する。Smallest Value が選択された場合、選択式が評価の参照として使用される。式は CompletionTimePredictor.PredictedValue(Candidate.Node.AssociatedObject)に設定される。
   これにより、Transfer ノードは訓練済みのニューラルネットを使用して各ワークステーションでの予想完了時間を予測し、最短完了時間が見込まれるワークステーションを選択する。このロジックは、ニューラルネットワークのトレーニングが完了し、じゅうぶんな精度が確認された後に適用される。Candidate.Node.AssociatedObject は、tblWorkstaions に登録された全ノードを表し、追加プロセスなしで動的な評価を可能にする。オーダ（エンティティ）が出力ノードに到達すると、トークンがワークステーションの入力ノードを巡回し、各ワークステーションの特性に基づいて完了時間を算出する。

この設定により、ベースラインモデル（Random）と学習済みモデル（Intelligent Dispatching）の性能を比較評価する実験が可能になる。

この Transfer ノードには、状態割当と、ニューラルネットワークの入力を保存するためのアドオンプロセス Output_OrderArrivals_Exited もある。このプロセスの詳細は 13.3.2 項で説明する。

13.3.1.3　Server：WS_01、WS_02、WS_03、WS_04

オーダがリリースされると、モデル内のいずれかのワークステーション（Server）に手配される。これらのワークステーションは並列に稼働し、それぞれ容量 1 を持ち、すべての種類のオーダを処理できる。ただし、オーダの処理時間は各ワークステーションで異なる。この実装を効率化するため、新しいテーブル tblOrderDestinations を設計した。

図 13.13 に示すように、このテーブルは 3 つの列で構成される。OrderType は tblOrderTypes テーブルの OrderType を参照する外部キープロパティ、

Workstation は tblWorkstations テーブルの Workstation を参照する外部キープロパティである。Expression プロパティの ProcessingTime はオーダの処理時間を指定する。このテーブルには 16 行あり、各行は 4 つのオーダと 4 つのワークステーションの組み合わせに対する固有の分布を表している。処理時間は、**最小値**、**最頻値**、**最大値**を持つ三角分布と、**最小値**、**最大値**を持つ一様分布を組み合わせて定義される。これらの値は任意に選択される。そのため、効果的な作業手配ルールは、処理時間の長いワークステーションへのオーダ割当を避け、適切な処理時間を持つワークステーションを優先する必要がある。

tbl Order Types | tbl Workstations | tbl Order Destinations

	Order Type	Workstation	Processing Time
1	OrderA	WS_01	Random.Uniform(.5, 1.5)
2	OrderB	WS_01	Random.Triangular(10, 15, 20)
3	OrderC	WS_01	Random.Uniform(5, 13)
4	OrderD	WS_01	Random.Triangular(8, 16, 24)
5	OrderA	WS_02	Random.Exponential(5)
6	OrderB	WS_02	Random.Triangular(7, 9, 11)
7	OrderC	WS_02	Random.Uniform(.25, 1.75)
8	OrderD	WS_02	Random.Triangular(18, 22, 26)
9	OrderA	WS_03	Random.uniform(5, 9)
10	OrderB	WS_03	Random.Triangular(1, 2, 3)
11	OrderC	WS_03	Random.Uniform(10, 26)
12	OrderD	WS_03	Random.Uniform(1, 5)
13	OrderA	WS_04	Random.Triangular(10, 15, 20)
14	OrderB	WS_04	Random.Uniform(6, 12)
15	OrderC	WS_04	Random.Triangular(2, 4, 6)
16	OrderD	WS_04	Random.Exponential(2)

図 13.13　tblOrderDestinations テーブル

ワークステーションの設定には、処理時間と信頼性ロジックの定義のみが必要である。図 13.14 にワークステーションのプロパティ設定例を示す。処理時間の式（分）は以下のように設定する：

Properties: WS_1 (Server)

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	**tblOrderDestinations[1].ProcessingTime**
Off Shift Rule	Suspend Processing

図 13.14　WS_01 のプロパティ設定（他のワークステーションも共通）

```
tbl_OrderDestinations[1].ProcessingTime
```

この設定は tblOrderDestinations テーブルを参照し、ProcessingTime で定義された分布を使用して処理時間を生成する。このテーブルは、オーダタイプとワークステーションを照合し、適切な処理時間分布を選択する。たとえば、OrderB が WS_01 に割り当てられた場合、Random.Triangular(10,15,20)が処理時間の生成に使用される。

13.3.1.4　Sink：OrderDepartures

処理終了後、オーダは OrderDepartures という Sink ノードからシステムを離脱する。この部分には追加の設定は必要ない。モデル内のすべてのオブジェクトは、シンプルな設計方針に基づきコネクタで接続されている。

13.3.1.5　初期結果

ベースラインモデルをテストするため、2 つの作業手配方法を比較した。1 つ目は Random 方式、2 つ目は関連負荷（Candidate.Node.AssociatedStationLoad）に基づく Lowest Load 方式である。公平な比較を行うため、両シナリオの平均到着間隔率（MeanIAT）を 4 に設定した。

分析の結果、関連ステーション負荷を最小化する 2 番目のシナリオが、Random 方式を大きく上回る性能を示した。具体的には、平均システム内時間（TIS）とシステム内オーダ数（NIS）が大幅に削減された。さらに、各ワークステーション間でより効率的な作業負荷の分散が実現され、利用率の違いも明確に確認された（図 13.15）。

Scenario			Replications		Controls		Responses					
☑	Name	Status	Required	Completed	WSSelectionRule	MeanIAT	TIS (Minutes)	NIS	WS_1_Util	WS_2_Util	WS_3_Util	WS_4_Util
☑	Random	Completed	30	30 of 30	Random	4	15.6431	3.9111	50.6743	50.188	51.0841	54.4439
☑	Lowest Load	Completed	30	30 of 30	SmallestValue	4	9.44041	2.3593	72.4565	59.0013	43.4008	29.9159

図 13.15　Random と Lowest Load による作業手配の比較

13.3.2　データ収集：トレーニングデータの生成

ニューラルネットワークのトレーニングにおいて、データの質はもっとも重要な要素である。モデルの精度はトレーニングデータの質に直接比例するため、じゅうぶんな量の高品質なデータを確保することが不可欠である。

第 2 ステップでは、シミュレーションモデルから、システムの動態を正確に反映し、予測精度の向上に寄与するデータポイントを特定する必要がある。トレーニングの目的に応じて、エンティティ、サーバ、システム全体の測定値など、さまざまなデータを収集する。

本モデルでは、各ワークステーションにおけるオーダの完了時間を予測することを目的としている。そのため、以下のデータポイントが予測に重要となる：

1. **ワークステーションの負荷**：各ワークステーションの処理待ちまたは待機中のオーダ数を表す。
2. **期待処理時間**：完了時間を決定する重要な要素である期待処理時間を計算する。定義された分布から、ワークステーションでのオーダ処理に要する時間を算出する。

3. **ワークステーションのステータス**：信頼性ロジックに基づき、starved、processing、failed などのワークステーションの稼働状態を示す。これらは状態把握に不可欠なデータである。
4. **目的ワークステーション**：オーダが割り当てられる対象のワークステーションを示す。

この変更を実装するには、ベースラインモデルを複製して NN_model という新しいモデルを作成することを推奨する（図 13.16）。

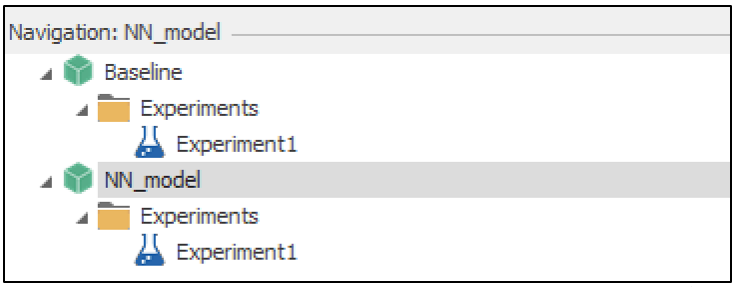


図 13.16　ニューラルネットワークツールを実装するための NN モデルの作成

モデルの変化に伴い、入力データの調整が必要になる。ニューラルネットワークへの入力データを選択する際は、予測値の計算に必要な情報を体系的に特定し、それらを入力として活用することが標準的なアプローチである。このプロセスを通じて、完了時間の予測精度を向上させる新たな入力変数を発見できる可能性がある。これらの原則に基づき、以下にデータ収集の設定手順を示す。

13.3.2.1　Transfer ノードの更新：Output@OrderArrivals

Output@OrderArrivals の初期設定については 13.3.1 項で解説した。データ収集機能を拡張するため、以下の変更を実施する必要がある：

1. **イベントの定義**：ニューラルネットワークのデータ収集用の入力データを保存するアドオンプロセスとして、SaveInputs イベントを作成する（図 13.17）。
2. **状態変数の定義**：Definitions タブで、CurrentOrder という新しいエンティティ参照状態変数を設定する（図 13.18）。
3. **状態割当の設定**：学習済みのニューラルネットワークシナリオでは、Transfer ノードがエンティティの目的地を選択する前に予測を問い合わせる。その後、単一プロセスが tblWorkstations のエンティティ行を設定し、現在のエンティティを CurrentOrder エンティティ参照状態変数に割り当て、**Save Inputs** イベントを発生させる。このイベントにより Neural Network 要素が入力データを記録する。この変更により、冗長なノード、コネクタ、状態割当の削除が可能となる（図 13.19）。
4. **アドオンプロセスの設定**：このプロセスは学習済み・未学習の両方のニューラルネットワークで使用する。学習済みの場合、Transfer ノードがエンティティをノードに割り当てた後、適切なテーブルでエンティティの行参照を確

立する。これにより、Server 上での Processing Time 式の評価時に正しい行が参照される。未学習の場合は、Assign ステップでニューラルネットワークの入力式がエンティティ参照プロパティにアクセスできることを確認し、Fire ステップで入力データの記録を指示する（図 13.20）。

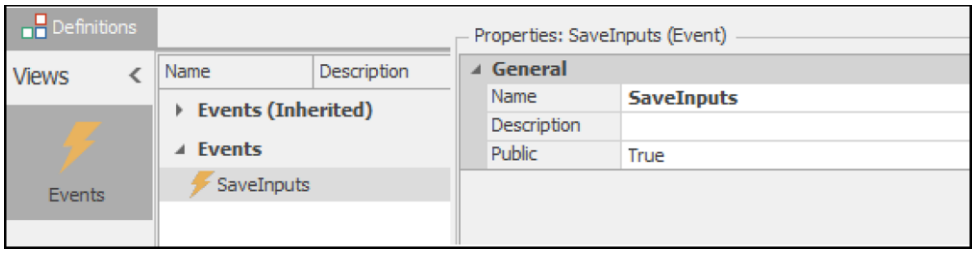


図 13.17　SaveInputs イベントの設定

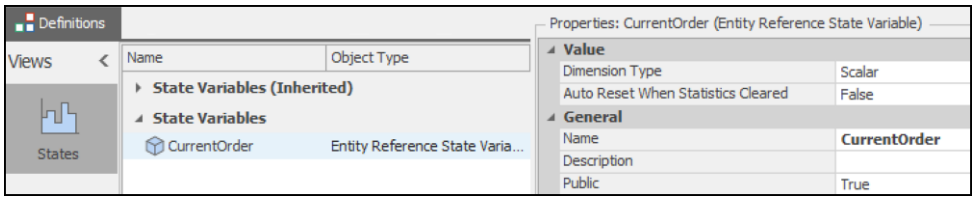


図 13.18　CurrentOrder エンティティ参照状態変数の定義

	状態	整数	One-Hot
▶	Starved (Idle)	0	[1, 0, 0]
▶	Processing (Busy)	1	[0, 1, 0]
▶	Failed (Broken)	3	[0, 0, 1]

注：状態の整数2はBlockedで予約済みであり、本モデルでは適用しない。

図 13.19　ワークステーション状態の One-Hot エンコーディング

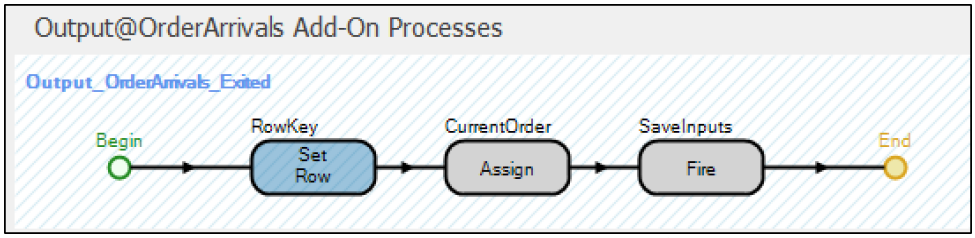


図 13.20　データ処理を効率化するアドオンプロセスの設定

13.3.2.2　ニューラルネットワークモデル：CompletionTimePredictorModel

Simio 内でニューラルネットワークを統合する場合、最初に**ニューラルネットワークモデル**を作成する。Simio は機能を継続的に強化しているため、ソフトウェアを最新版に保つことが大切だ（更新手順は付録 C を参照）。ユーザは各自の要件に合わせてさまざまなニューラルネットワークモデルを作成できる。具体例として、CompletionTimePredictorModel は注文の完了時間を予測するためのモデルである。ニューラルネットワークモデルを設定する際は、以下のプロパティ設定

が必要となる：

1. **ニューラルネットワークモデルの作成**：Simio の Data ビューの Neural Netowork Model セクションから、Add Feedforward Neural Network Model を選択する（図 13.21）。これにより、指定した数の数値入力と単一の数値出力を持つ完全連結フィードフォワード型ニューラルネットワーク回帰モデルが作成される。このモデルは複数の隠れ層を持つことができ、学習フェーズの前に層数とノード数をハイパーパラメータとして調整できる。一般的に隠れ層には tanh 活性化関数を、出力層には identity 関数を使用する。
2. **モデルプロパティの設定**：この段階で、モデル名、入力ノード数、データリポジトリの最大サイズなどの基本プロパティを設定する（図 13.22 参照）。隠れ層の構成やその他のハイパーパラメータについては、13.3.2.4 のトレーニングフェーズで説明する。

- Number Input Nodes：9（設定の重要性はニューラルネットワーク要素を説明する次項で扱う）
- Maximum Records Limit：5000
- Name：CompletionTimePredictorModel

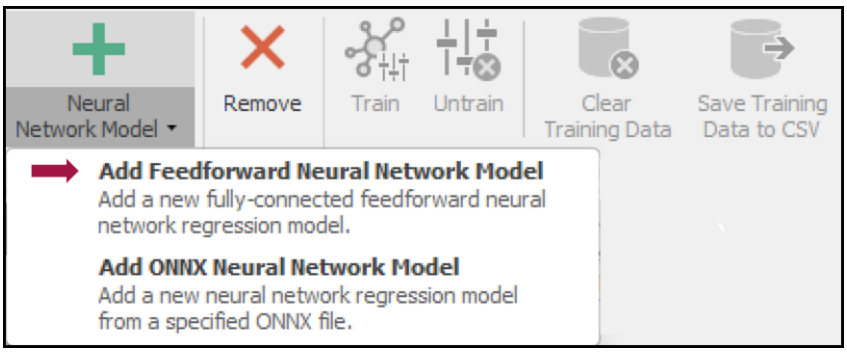


図 13.21　フィードフォワード型ニューラルネットワークモデルの設定

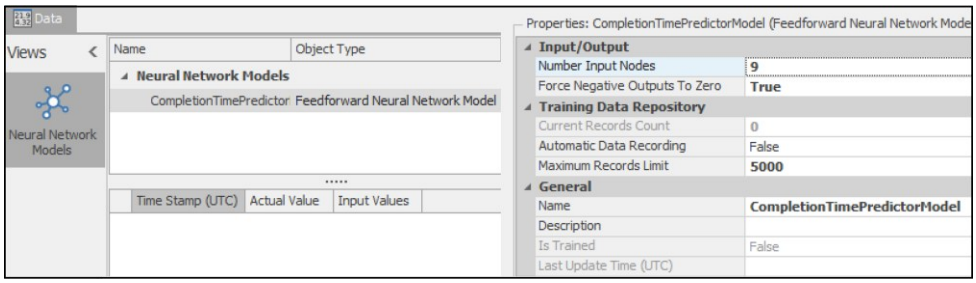


図 13.22　ニューラルネットワークモデルのプロパティ設定

13.3.2.3　要素：CompletionTimePredictor

Simio の Neural Network 要素は、ニューラルネットワークベースのモデルをシミュレーションの意思決定フレームワークに統合する。Input Value Expressions プロパティを使用して、モデルの入力層に式を動的に関連づけることができる。シミュレーション実行中に予測が必要な場合は、PredictedValue 関

数を使用して特定のオブジェクトの入力式を評価する。これにより、ニューラルネットワークモデルが入力を処理し、予測値を生成する。

1. **ニューラルネットワーク要素の作成**：Simio の Definitions タブで Elements ビューを開き、Data Science カテゴリの Newral Network 要素を追加する。
2. **要素のプロパティを設定**：ニューラルネットワークモデルの適切な構成は、モデルの有効性を確保する基本となる。入力データ（独立変数）と出力データ（従属変数）には、精密な式の定義が不可欠である。また、モデルの実行全体を通じて一貫したデータを記録するため、正確なトリガー設定も必要である。ニューラルネットワークの性能はこれらの初期設定の質に直接影響されるため、この段階は特に重要である。

- Basic Logic
  - **Neural Network Model Name**：定義済みのニューラルネットワークモデルリストから CompletionTimePredictorModel を選択する。
  - **Inpuut Value Expressions**：ニューラルネットワークの入力値を生成する式を定義する。これらの式は、13.3.2 項で解説した入力を表すものである。図 13.23 にモデル入力を記録するために使用する式を示す：最初の式（a）は、関連するワークステーションの負荷を捕捉する。2 番目の 3 つの式（b）は、ワンホットエンコーディングでワークステーションの状態を記録する（starved または idle：0、processing または busy：1、failed または broken：3）。続く 4 つの式（c）は、ワークステーションの行き先を記録する。最後の式（d）は、ワークステーションでのオーダの予想処理時間を記録する。
  - **Untrained Predicted Value Expression**：Random.Uniform(0, 1)。ニューラルネットワークモデルが未訓練の場合、この式で一時的な予測出力値を提供する。学習済みモデルのデバッグにも使用できる。

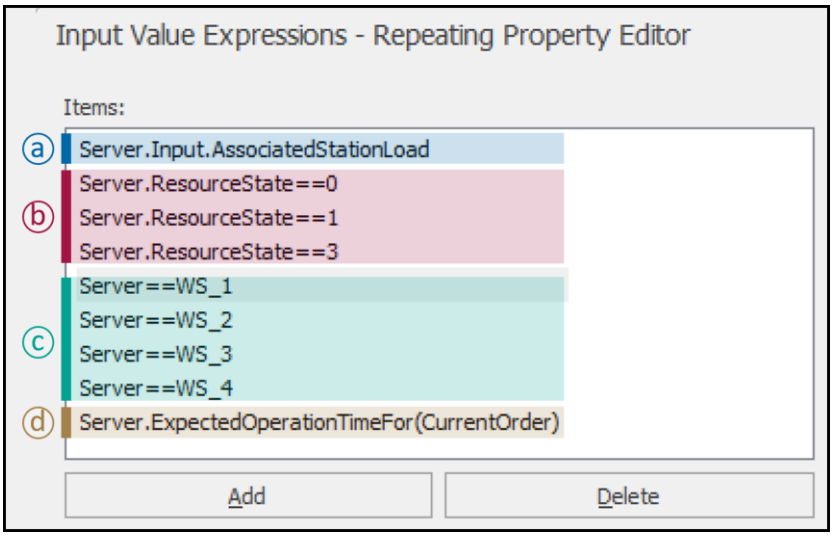


図 13.23　Neural Network 要素の入力式

- Training Data Recording
  - **Save Inputs Triggers**：入力値を 1 行ずつ記録するように設定する。SaveInputs イベントトリガーにより、特定のイベント発生時に入力データの保存を開始する（図 13.24）。
  - **Save Actual Triggers**：Save Actual トリガーが発生した際に、実測値を記録する。Input@OrderDepartures.Entered イベントの発生時に、観測された値を保存する（図 13.25）。
  - **Actual Value Expression**：Save Actual トリガーが発生した際に、実測値を記録する。出力変数（実測値）は、TimeNow-ModelEntity.TimeCreated で計算されるオーダの完了時間（システム内時間）として定義される。

図 13.24　Save Inputs Triggers の設定（SaveInputs イベントの使用）

図 13.25　Save Actual Triggers の設定（完了時間の出力データ保存）

設定完了後、シミュレーションモデルを実行してデータポイントを生成し、リポジトリに保存する。データ収集を開始するには、Data タブの Neural Network Models ビューから Generate Training Data ツールを選択し、Run ボタンをクリックする。Simio は、各ニューラルネットワークのトレーニングデータが Maximum Record Limit（この場合 5,000）に達するまで、モデルを自動的に実行する。任意のニューラルネットワークモデルを選択すると、そのトレーニングレコードを確認できる。図 13.26 は、CompletionTimePredictorModel のトレーニングデータ記録を示しており、データの記録時刻（Time Stamp）、実際の値（Actual Value）、および Neural Network 要素で定義された 9 つの入力値が表示されている。

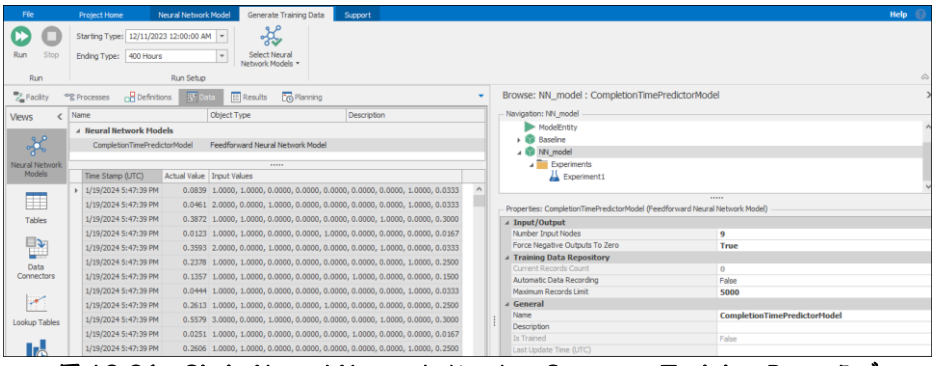


図 13.26　Simio Neural Network ツールの Generate Training Data タブ

図 13.27 の保存データの最初のレコードを解釈する。数値の順序は、要素の Inpuut Value Expressions で定義された入力式の順序と一致する。Actual Value の 0.0839 は、モデルが予測対象とする実際の完了時間（時間単位）を示す。Input Values には、ワークステーションの負荷（青）、状態（赤）、特定のワークステーション ID（図では WS_4）を示す項目（緑）が含まれる。最後の入力値 0.0333 は、予想処理時間を表す。この構造化されたデータ形式により、ニューラルネットワークはさまざまな運用要因と実際の完了時間との相関を学習し、信頼性の高い予測モデルを構築できる。必要なデータの収集が完了したら、モデルのトレーニングを開始する。

Actual Value	Input Values								
0.0839	1	1	0	0	0	0	0	1	0.0333
Completion Time (hr)	WS_Load = 1 order	State =1 (Idle or Starved)			Workstation = WS_4				Processing Time (hr)

図 13.27　トレーニングデータの構造と解釈

13.3.2.4　トレーニングプロセス：モデルの準備

高品質なデータを収集した後、次のステップはニューラルネットワークモデルを機械学習技術で訓練する。Simio は、ニューラルネットワークモデルの設定と微調整のための直感的なインタフェースを提供している。このインタフェースでは、ネットワークのアーキテクチャ、学習率、オプティマイザの種類を設定できる。さらに、トレーニングの進捗状況をモニタリングし、モデルが必要な精度と信頼性の基準を満たしているか評価することができる。

トレーニングを開始するには、Data タブの Neural Network Models リボンで CompletionTimePredictorModel を選択し、Train をクリックしてトレーニングダイアログを開く（図 13.28）。Train ツールには以下の設定オプションがある：

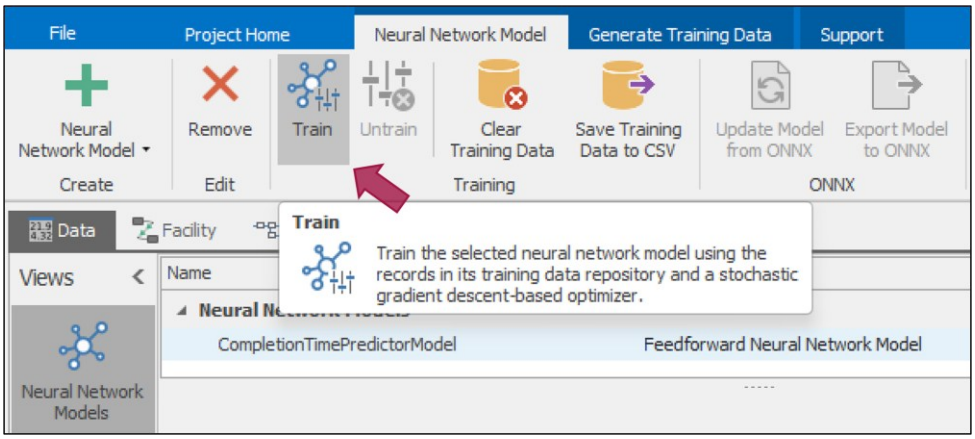


図 13.28　ニューラルネットワークのトレーニング

- **Data**
  - **Total Records Available**：ニューラルネットワークのトレーニング、検証、テストに使用可能な全データレコード数。
  - **Validation Holdout Percentage**：モデルの性能評価用に確保するデータの

割合。
- **Test Holdout Percentage**：学習後の性能評価と汎化性能確認用に確保するデータの割合。

・**Network Architecture**
- **Number Hidden Nodes**：隠れ層の各層におけるノード（ニューロン）数。

・**Training Algorithm**
- **Max Epochs**：学習データ全体の最大処理回数。
- **Optimizer Type**：学習に使用する最適化アルゴリズム。
- **Learning Rate**：パラメータ更新時のステップサイズを制御し、学習の収束速度と安定性を決定する値。
- **Beta1 Parameter**：Adam などの最適化アルゴリズムにおける第一モーメント推定値の減衰率を制御するハイパーパラメータ。
- **Beta2 Parameter**：Adam などの最適化アルゴリズムにおける第二モーメント推定値の減衰率を制御するハイパーパラメータ。
- **Early Stopping Min Delta**：学習早期停止を判断するための検証損失改善の最小閾値。
- **Early Stopping Patience**：検証損失が改善しない場合に許容される最大エポック数。

・**Progress Plot**
- **Skip First Epochs**：初期の学習変動を除外し、詳細な学習過程の可視化に焦点を当てるオプション。

Neural Network Training ツールを効果的に使用するには、まず適切な隠れノード数とレイヤ数を選択する。単一の隠れ層を使用する場合は、ノード数を直接指定する（例：9 ノードの場合は「9」）。複数の隠れ層を使用する場合は、各層のノード数をカンマ区切りで定義する（例：9 ノードと 4 ノードの 2 層構成の場合は「9, 4」）。続いて、トレーニングエポック数を設定する。これは、ニューラルネットワークがトレーニングデータセット全体を処理する回数を指す。最適なエポック数は状況によって異なるため、実験的に決定していく必要がある。

設定が完了し、図 13.29 の段階に到達したら、トレーニングプロセスを開始する。トレーニング中は、エポックの完了状況と報告されるエラーを注意深く監視する。これにより、ネットワークの進捗状況を正確に把握し、必要な調整を適切に行うことができる。ツールは、トレーニング、検証、テストの各データセットの平均二乗誤差（Mean Squared Error：MSE）を計算・表示し、モデルの性能に関する詳細な情報を提供する。

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

ここで、nは観測数、Y_iは実測値、$\hat{Y}_i$は予測値を表す。MSE は 0 に近いほど望ましく、これはモデルの予測値が実測値とより正確に一致することを意味する。

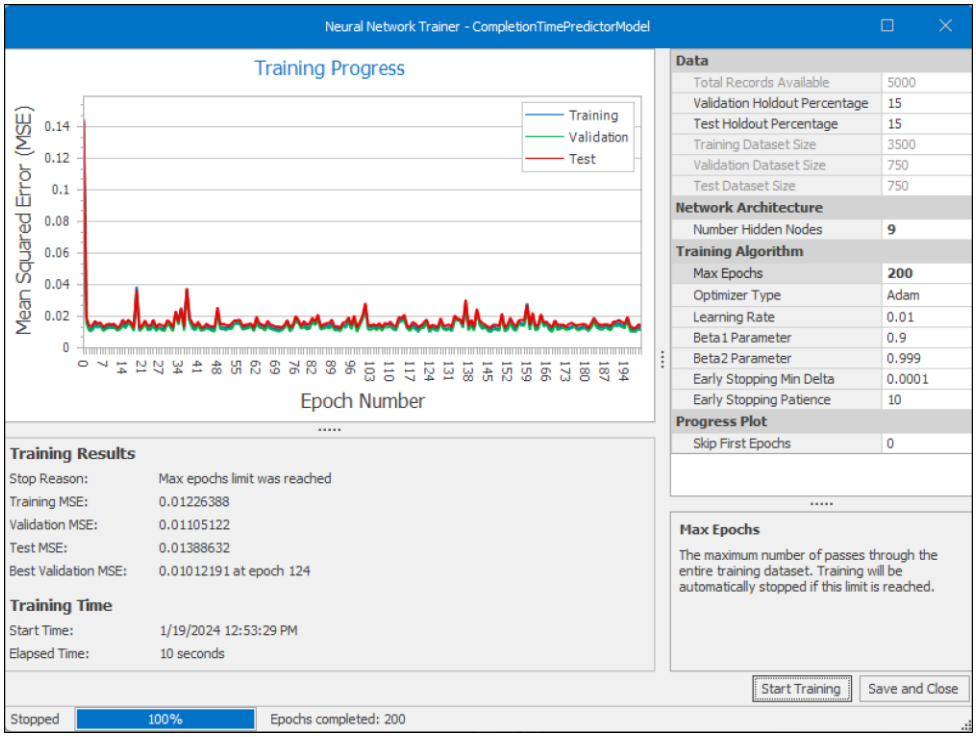


図 13.29　Neural Network Trainer による平均二乗誤差（MSE）の収束過程の可視化

トレーニング、検証、テストの MSE 値が互いに近い値を示す場合、そのモデルは既知データと未知データの両方で安定した予測性能を持ち、過学習を起こすことなく優れた汎化性能を達成していると評価できる。

トレーニング結果がじゅうぶんであれば、ニューラルネットワークモデルを保存し、モデルが完全にトレーニングされ更新されたことを確認する。その後、新しいデータセットを用いてモデルの性能を評価することができる。

13.4　テストと性能評価

トレーニング完了後、Property ウィンドウのトレーニングステータスが更新され、IsTrained が True となり、Last Update Time が最新の時刻に更新される（図 13.30）。

システム性能を評価するために、ベースラインモデルと訓練済みニューラルネットワークを組み込んだモデルの比較実験を行う。ただし、ニューラルネットワークのトレーニングごとに実験結果が変動する可能性がある。図 13.31 では、次の 2 つのモデルを比較する：

- **Lowest Load**：ワークステーションの Associated Load に基づいて WSSelectionRule を `Smallest Value` に設定した単純なベースラインシナリオ。特定のロジックや基準を用いずにオーダを配置し、比較のためのコントロールグループとして機能する。
- **Intelligent Dispatching**：WSSelectionRule を `Smallest Value` に設定し、

ニューラルネットワークの予測を活用してオーダ処理の効率を最適化する。

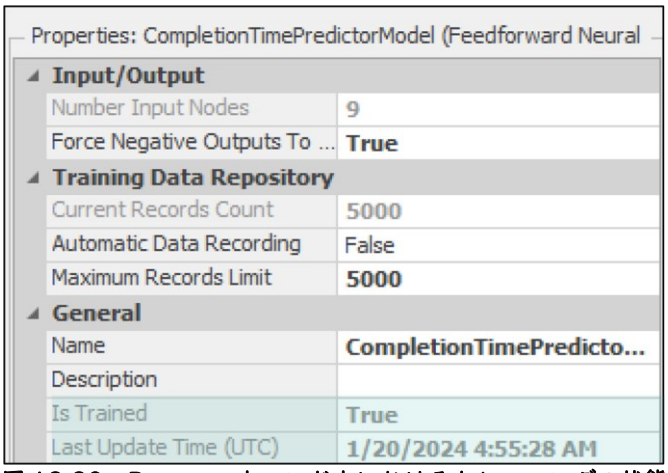
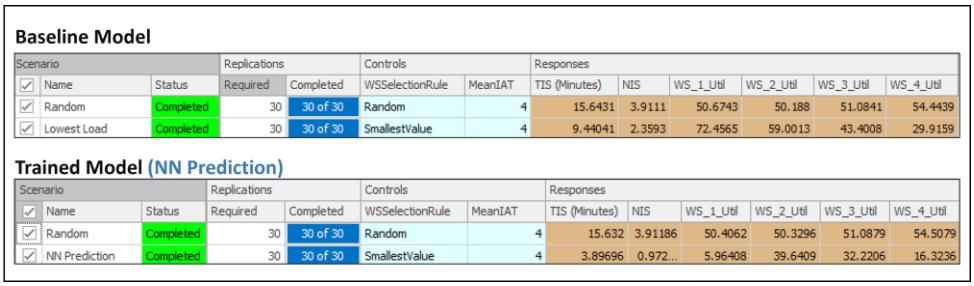


図 13.30　Property ウィンドウにおけるトレーニングの状態



Baseline Model

Scenario			Replications		Controls		Responses					
Name	Status	Required	Completed	WSSelectionRule	MeanIAT	TIS (Minutes)	NIS	WS_1_Util	WS_2_Util	WS_3_Util	WS_4_Util
Random	Completed	30	30 of 30	Random	4	15.6431	3.9111	50.6743	50.188	51.0841	54.4439
Lowest Load	Completed	30	30 of 30	SmallestValue	4	9.44041	2.3593	72.4565	59.0013	43.4008	29.9159

Trained Model (NN Prediction)

Scenario			Replications		Controls		Responses					
Name	Status	Required	Completed	WSSelectionRule	MeanIAT	TIS (Minutes)	NIS	WS_1_Util	WS_2_Util	WS_3_Util	WS_4_Util
Random	Completed	30	30 of 30	Random	4	15.632	3.91186	50.4062	50.3296	51.0879	54.5079
NN Prediction	Completed	30	30 of 30	SmallestValue	4	3.89696	0.972...	5.96408	39.6409	32.2206	16.3236

図 13.31　学習済みニューラルネットワークの予測モデルによる性能比較

結果は、ニューラルネットワーク予測因子を活用した Intelligent Dispatching（NN Prediction）が、Random および Lowest Load の両手法を上回る性能を示した。図 13.31 が示すように、学習済みのニューラルネットワーク予測モデルは、Lowest Load アプローチと比較して業務効率を大幅に向上させた。具体的には、システム内の処理時間とオーダ数を効果的に削減している。また、Random アプローチでは見られたワークステーション間の均等なタスク配分とは対照的に、ニューラルネットワークモデルはより戦略的なリソース活用を実現し、その予測能力によってワークフローを効率的に最適化できることが実証された。

13.5　まとめ

AI は多くの分野で目覚ましい成果を上げているにもかかわらず、デジタルツイン技術ではその潜在能力がじゅうぶんに活用されていない。そのような中、Simio の Neural Network は、デジタルツインプラットフォームに AI を導入する革新的なツールとして注目されている。本章では、このツールの概要、設定方法、そして実践的な応用例について解説した。

このツールの有用性は、製造業のケーススタディで実証された。具体的には、4 つの並列ワークステーション間で 4 種類のオーダを配置するためのニューラルネ

ットワークの学習例（インテリジェントな作業配置）を示した。各オーダはワークステーションごとに異なる処理時間を要するため、効率的な作業配置には、ワークステーションの負荷状況や処理時間などの複数要因を考慮する必要がある。Simio の Neural Network ツールは、データ収集とトレーニングの 2 段階で適用される。まず、ニューラルネットワークの学習に必要なデータを収集し、適切なデータセットを得た後にモデルのトレーニングを行う。この予測モデルは、各ワークステーションでのオーダ完了時間を予測し、最短完了時間のワークステーションへの作業配置を実現する。予測には、オーダの種類に関する情報と、ワークステーションの状態や現在の負荷といった運用データを活用する。

このインテリジェントな作業手配の性能を、ランダム作業手配と最低負荷作業手配という 2 つのベンチマークと比較した。結果として、適切に学習されたインテリジェントアプローチは、業務目標を大きく最適化できることが実証された。

この実証例は、デジタルツイン技術による運用最適化、予測分析、意思決定支援における AI の可能性を明確に示している。ただし、この分野で AI の能力を最大限に活用するには、詳細な理解と慎重な実装プロセスが不可欠である。

13.5.1 ニューラルネットワークモデルの継続的な改善

ニューラルネットワークツールをデジタルツイン技術に統合することは、効果的な実装に不可欠なプロセスである。この統合を成功させるには、包括的なデータ収集を確保するため、定期的なモデル修正が必要となる。具体的には、ロジック、プロセス、変数、属性の調整が含まれる。たとえば、新規オーダをモデルに組み込む際には、システムの動的な変化を正確に反映するための修正が求められる。これらの調整によって、データの関連性と精度が維持され、以降のニューラルネットワーク再学習の基盤が形成される。

モデル修正後は、包括的なデータ収集フェーズに移行する。このフェーズは必要に応じて更新され、モデルパラメータと同期を取りながら繰り返し実施される。これにより、ニューラルネットワークが最新の関連データを確実に学習できる環境が整う。

学習フェーズでは、ネットワークアーキテクチャ、学習率、オプティマイザの種類などのハイパーパラメータを段階的に最適化していく。この反復的な学習プロセスによってモデルの性能が向上し、デジタルツインはより正確な予測と洞察に基づいた意思決定支援が可能となる。

このモデル修正、データ収集、学習という継続的なサイクルを通じて、ニューラルネットワークは着実に性能を向上させていく。その結果、ユーザはより確実な情報に基づく意思決定が可能となり、デジタルツインにおける AI の潜在能力を最大限に引き出すことができる。

13.6 問題

1. Simio のようなシミュレーション環境において、OptQuest などの最適化ツールとニューラルネットモデリングの主な違いを説明しなさい。それぞれのアプローチについて、長所と限界、適切な使用事例を述べなさい。

2. デジタルツインの定義、構成要素、および利点について説明しなさい。デジタルツインモデルに AI、特にニューラルネットワークを統合する利点を説明しなさい。AI を活用したデジタルツインは、従来のデジタルツインの機能をどのように向上させるか。
3. シミュレーション環境にニューラルネットワークを統合する具体的なメリットを説明しなさい。
4. 下図は Simio の Neural Network ツールを使用したデータ収集段階で生成されたデータのスナップショットを示している。図 13.27 を参照し、入力値（サーバの使用状況や状態など）を解読しなさい。

Collected Data

Actual Value	Input Values
0.7258	6.0000, 0.0000, 1.0000, 0.0000, 0.0000, 1.0000, 0.0000, 0.0000, 0.1500

NN Element Inputs

Input Value Expressions - Repeating Property Editor

Items:

```
Server.Input.AssociatedStationLoad
Server.ResourceState == 0
Server.ResourceState == 1
Server.ResourceState == 3
Server == WS_01
Server == WS_02
Server == WS_03
Server == WS_04
Server.ExpectedOperationTimeFor(CurrentOrder)
```

5. 例題 13.1 の 4 台のワークステーション（WS_01、WS_02、WS_03、WS_04）に、バックアップとして 5 台目のワークステーション WS_05 を追加するケースを検討する。WS_05 は、主ワークステーションの予測完了時間が、CompletionThreshold という一定の閾値を超えた場合にのみ使用される。WS_01 から WS_04 の予測完了時間が Completion Threshold を超える場合、Neural Netowork ツールを使用した意思決定プロセスにより、オーダを WS_05 に振り分ける。このバックアップワークステーションを追加したシステムのパフォーマンスを分析・視覚化するため、以下のステップで実装する：

a) WS_05 をモデルに追加する。
b) tblOrderDestinations テーブルを更新し、WS_05 のすべてのオーダの処理時間を含める。

Order Type	Workstation	ProcessingTime (Minutes)
OrderA	WS_05	Random.Triangular(10, 12, 20)
OrderB	WS_05	Random.Triangular(10, 12, 20)
OrderC	WS_05	Random.Triangular(10, 12, 20)
OrderD	WS_05	Random.Triangular(10, 12, 20)

c) WS_05 の処理時間を tblOrderDestinations[1].ProcessingTime に設定する。
d) CompletionThreshold プロパティの初期値を 20 分に設定する。

e) システムの混雑度を上げるため、MeanIAT を 2 に設定する。
f) OrderArrivals の Transfer ノード（Output@OrderArrivals）にリルートロジックを実装し、Value Expression と Selection Condition を調整してオーダのルーティングを設定する。
g) モデルの動作をテストし、要件通りに機能することを確認する。
h) `CompletionThreshold` を 20 分、50 分、100 分と変更して実験を行い、5 つのワークステーションの使用率、平均システム内時間（TIS）、システム内オーダ数（NIS）を比較・分析する。

6. 機械学習では、予測値を実際の値に対してプロットすることが重要である。この可視化により、モデルのパフォーマンスを視覚的に確認し、パターン、傾向、異常値を容易に識別できる。さらに、モデルの精度を迅速に評価することが可能となる。プロットした点が線分 $y = x$ に沿って密接に分布している場合は高い予測精度を示し、大きく逸脱している場合はモデルの改善が必要な領域を特定できる。
Simio の Neural Network ツールでは、トレーニングデータを CSV ファイルとして保存できる。このデータをダウンロードし Excel 形式に変換することで、実績値に対する予測値の散布図を作成できる。この可視化から、どのような洞察と結論が得られるか。

a) 例題 13-1（NN モデル）において、`Save Training Data to CSV` オプションを使用し、Simio から学習済みデータを CSV ファイルにエクスポートする。
b) エクスポートした CSV ファイルを Excel で開く。
c) ファイルを Excel 形式（.xlsx）で保存する。
d) Actual Value（実測値）を X 軸、Predicted Value（予測値）を Y 軸とする散布図を作成する。
e) プロットを分析し、データのパターン、傾向、および偏差を特定する。
f) 作成した可視化に基づき、モデルの精度と性能について結論を導き出す。

付録 A　Simio を活用したケーススタディ

本章には、Simio モデルを用いたシステム分析に関する 6 つのケーススタディが含まれており、4 つは「導入」編、2 つは「発展」編である。これらのケースは、これまでの問題よりも広範な内容を扱っており、詳細な記述は省略されている。最初の 2 つのケースではサンプルモデルに基づく分析結果を示すが、これまでの章とは異なり、モデルそのものの詳しい**説明はしない**。

第 1 のケース（A.1 節）では、機械加工、検査、清浄作業からなる生産システムについて、2 つの改良案に基づく現行システムの分析を行う。第 2 のケース（A.2 節）では、遊園地における「ファストパス」チケットオプションが待ち時間に与える影響を分析する。第 3 のケース（A.3 節）では、レストランのスタッフ配置と収容能力に関する重要課題についてモデル化を行う。第 4 のケース（A.4 節）では、銀行支店におけるスケジューリングと設備選択が運営費用に与える影響を考察する。

発展的な 2 つのケーススタディでは、実務における大規模な問題を扱う。現実世界の問題は、教科書的な課題とは異なり、明確な形や完璧な条件を持っていない。多くの場合、情報の欠落やあいまいさ、複雑な状況、そして解釈によって異なる複数の解決策に直面する。実務では、このようなケースと同様の状況に遭遇する可能性が高い。そのような状況では、適切な仮定を立て、それらを必ず記録に残すべきである。

授業向けケーススタディの動画はこの URL にある：
https://www.simio.com/solution-series/

A.1　機械加工・検査工程

A.1.1　問題記述

あなたは、現在の機械加工と検査工程（図 A.1）の評価と、2 つの改善案の調査を依頼されている。両案とも、検査ステーションと検査員 1 名を自動化された検査工程に置き換えることを提案している。これらの変更は、システムパフォーマンスの向上を目的としている。

現在のシステムでは、4 種類の部品が自動化されたマシニングセンターに到着し、1 個ずつ処理される。機械加工完了後、各部品は人間の検査員が個別に検査を行う。検査工程では 2 種類の誤りが発生する可能性がある。「第 1 種の誤り」は「合格（good）」の部品を「不合格（bad）」と分類してしまう場合、「第 2 種の誤り」は「不合格（bad）」の部品を「合格（good）」と分類してしまう場合である。検査の結果、「合格」と分類された部品は工場の別のゾーンへ送られて加工される。一方、「不合格」と分類された部品は、1 個用の自動洗浄機で洗浄された後、新しい部品と同じ処理パラメータを用いてマシニングセンターで再加工される。

その他のシステムの詳細は以下のとおりである：

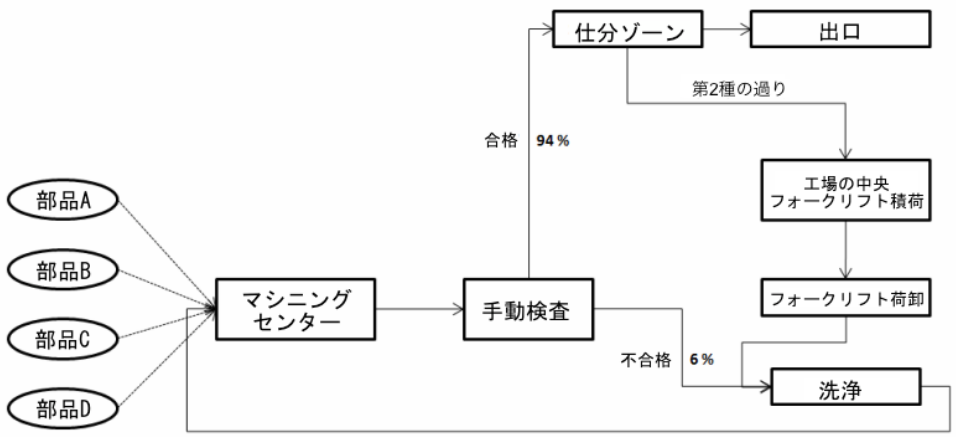


図 A.1　機械加工・検査工程のレイアウト

・表 A.1 に、部品到着および機械加工時間の詳細を示す。4 つの到着過程は、定常ポアソン過程であり、互いに独立している。

表 A.1　部品の到着間隔と加工時間データ

種類	到着率（個／時間）	機械加工時間分布
A	5.0	tria(2.0, 3.0, 4.0)分
B	6.0	tria(1.5, 2.0, 2.5)分
C	7.5	expo(1.5)分
D	5.0	tria(1.0, 2.0, 3.0)分

・すべての部品は、機械加工後に検査工程へ進む。検査時間は部品タイプに関係なく、平均 2 分の指数分布に従う。検査された部品の 6%が「不合格」となり、それらは洗浄・再加工される（この確率は洗浄・再加工の回数に依存しない）。洗浄後に検査に戻る部品は、機械加工待ち行列において「新規」部品より優先される。洗浄時間は部品タイプに関係なく 12 分である。
・「合格」と判定され別のゾーンへ送られた部品のうち、3%が後に「不合格」と判明する（第 2 種の誤り）。これらの部品は工場中央のパレットに置かれ、フォークリフトが定期的に回収して洗浄機械へ運ぶ。これらの部品は自動洗浄機の待ち行列で優先的に処理される。フォークリフトの回収間隔は平均 3 時間の指数分布に従う。
・「不合格」と判定され自動洗浄機へ送られた部品のうち、7%は実際には合格品であり、洗浄・再加工は不要である（第 1 種の誤り）。表 A.2 に、不要な洗浄と処理にかかる 1 部品当たりの費用を示す。
・マシニングセンターと洗浄機械は、ともに時間依存のランダムな故障が発生する。両機械とも、稼働時間（故障間隔）は平均 180 分の指数分布、修理時間は平均 15 分の指数分布に従う。機械が停止したとき部品を処理中の場合、その処理は中断されるが部品は破棄されない。
・システムは 1 日 2 シフト、週 6 日操業する。各シフトは 8 時間である。前シフ

トの作業は次のシフトに引き継がれる（つまり、シフト開始時の立ち上げ効果はない）。

表 A.2　システムおよび操業コスト

項目	費用
第 1 種の誤りによる不必要な洗浄	$5／部品
単純なロボット	$1,200／週（各機械）
複雑なロボット	$5,000／週
部品保持費用	$0.75／システム内時間
検査員	$50／時間

評価のために依頼されたシナリオは、以下の 2 つの検査工程の自動化案である：

1. 4 台の「単純な」ロボットを部品タイプごとに設置する。各ロボットは 1 個ずつ部品を検査する。第 1 種の誤り確率は 0.5%、第 2 種の誤り確率は 0.1%である。検査時間は部品タイプに関係なく、三角分布（5.5 分、6.0 分、6.5 分）に従う。
2. すべての部品タイプを検査できる「複雑な」ロボットを 1 台設置する。このロボットも 1 個ずつ部品を検査する。第 1 種の誤り確率は 0.5%、第 2 種の誤り確率は 0.1%である。検査時間は部品タイプに関係なく、三角分布（1.0 分、2.0 分、2.5 分）に従う。

これらのロボットは、マシニングセンターと自動洗浄機を製造している同じ会社の製品であるため、同様の故障パターンを示すと考えられる。つまり、機械加工装置と洗浄機械ですでに示されている確率分布と同じ故障確率が適用される。費用分析に使用するデータは表 A.2 に示されている。ロボット導入の経済性分析はすでに完了しており、表 A.2 には購入費、設置費用、操業費、維持費などの関連費用を週単位で示している。

Simio シミュレーションモデルを構築し、最適な検査プロセスを分析・決定する。各シナリオの分析では、以下の項目を推定する：

・週当たりの平均総費用。
・週当たりの第 1 種および第 2 種の平均エラー数。
・週当たりの完了部品総数（システム離脱数）の平均。

以下の問いについて、Simio モデルによるシミュレーション実験の結果と分析に基づいて回答しなさい。

1. マネージャは仕様を確認後、「単純な」ロボットの検査平均時間が現行工程より著しく長いことに気づいた。そのため、検査工程の変更により全体の生産量（週当たりの完成部品数）が**減少**する可能性を懸念している。新しい検査方法は週当たりの完成部品数に影響を与えるだろうか。もしそうであれば、現行システムと比較してどの程度変化する（増加または減少する）のかを分析してほしい。

2.　3 つの検査方式の中で、コスト最小化の観点から推奨すべき方式はどれか。
3.　もし自動検査工程のいずれかまたは両方が手動検査と同等以上のコストとなった場合、週当たりの運用コストを手動検査より低く抑えるための方策を検討せよ。ただし、すべてのコストは固定値であり、調整不可能と仮定する。

A.1.2　サンプルモデルと結果

本項では、著者が作成したサンプルモデルの分析結果を提示する。なお、結果は仮定した数値に基づいているため、読者の結果と完全に一致しない可能性があることに注意されたい。サンプルモデルの Facility ビューを図 A.2 に示す。

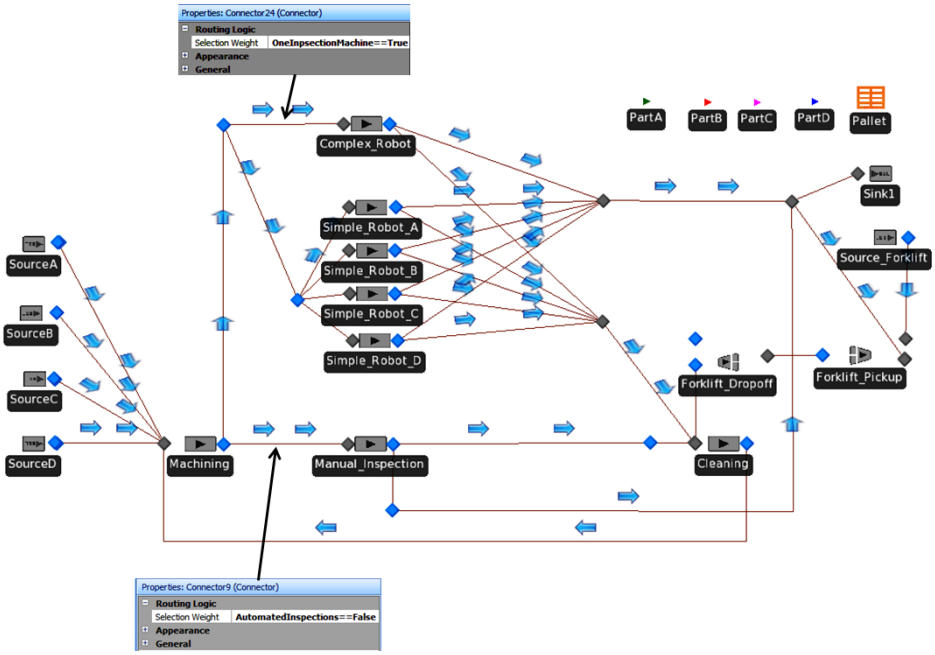


図 A.2　サンプル機械加工・検査モデルの Facility ビュー

実験では、以下のコントロールとパラメータを使用した：

・実行期間：192 時間
・ウォームアップ期間：96 時間
・反復実行回数：2,000
・実験コントロール：
　- AutomaticInspection（自動検査；論理演算子）
　- OneInspectionMachine（1 台の複雑な検査機械；論理演算子）
　- WeeklyMachineCost（週当たりの機械加工費用）
　- InspectionWorkerHours（検査作業者の時間）
　- Warm-up（ウォームアップ期間）

図 A.3 は、Simio Experiment を用いて異なるコントロールによる各シナリオのアクティビティと総費用を決定する方法を示している。Simio のシナリオ比較機能を活用するため、3 つのオプションをすべて単一のモデルに実装した。異な

るシナリオを同一実験内で比較するため、論理演算子の AutomaticInspection と OneInspectionMachine を使用した。各コントロールのチェックボックスを選択すると、対応するプロパティに True 値が設定される。モデルは検査プロセスを実行する前に、コネクタの Selection Weight プロパティでこれらの値を参照する。この例を図 A.2 に 2 つ示している。

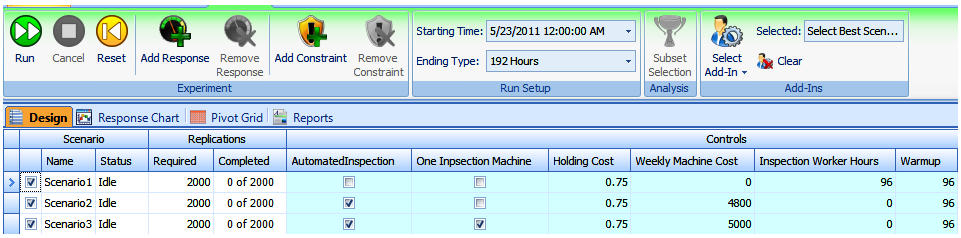


図 A.3　機械加工・検査の Experiment Design スナップショット

週当たりの総費用は以下の式で計算する：

```
TotalCost = WeeklyMachineCosts
+ WeeklyHoldingCost.Value
+ TotalTypeIErrors.Value * 5
+ InspectionWorkerHours * 50          (A.1)
```

式 A.1 の各項の計算方法は以下のとおりである：

1. **WeeklyMachineCosts**：A.1.1 項で示した費用。これは実験で制御される標準の数値プロパティである。
2. **WeeklyHoldingCost.Value**：モデル状態変数 HoldCost を参照する出力統計値。エンティティが Sink オブジェクトに入る（システムを離脱する）たびに、以下の式 A.2 を用いて状態変数 HoldCost が更新される：

   HoldCost=(TimeNow – ModelEntity.ArrToSys) * 0.75 + HoldCost　(A.2)

   式 A.2 では、エンティティのシステム内滞在時間に、A.1.1 項で定めた部品保持費用（0.75）を乗じて計算する。この値で状態変数を更新することで、出力統計量 WeeklyHoldingCost は全エンティティの保持費用合計となる。
3. **TotalTypeIErrors.Value * 5**：第 1 種の誤りに関連した費用で、A.1.1 項で示した$5 の費用を表す。TotalTypeIErrors は、モデル状態変数 TypeIError を参照する出力統計量である。検査工程で第 1 種の誤りが発生すると、この状態変数が増加する。
4. **InspectionWorkerHours * 50**：A.1.1 項で示した毎時$50 の費用を表す。InspectionWorkerHours は、実験で制御される標準の数値プロパティである。

式 A.1 と上記の実験パラメータを用いて、各シナリオの週当たりの平均費用を算出した。結果を表 A.3 に示す（すべての半幅は期待値の 95%信頼区間に基づく）。表 A.4 には、各検査工程における第 1 種および第 2 種の誤りの週当たりの平均合計を、表 A.5 には週当たりの完了部品（システム離脱）の平均総数を示す。

表 A.3　各シナリオの平均総費用

検査シナリオ	費用	半幅
現状の手作業	$7,303.31	$63.20
4 台の単純なロボットによる自動化	$7,268.96	$58.13
複雑なロボットによる自動化	$7,119.54	$56.37

表 A.4　各シナリオの第 1 種および第 2 種の誤り発生数の平均

検査シナリオ	第 1 種の誤り [95%信頼区間]	第 2 種の誤り [95%信頼区間]
現状の手作業	10.23 [10.09, 10.37]	69.93 [69.55, 70.31]
4 台の単純なロボットによる自動化	1.02 [0.97, 1.06]	2.26 [2.19, 2.32]
複雑なロボットによる自動化	1.04 [1.00, 1.09]	2.24 [2.17, 2.30]

表 A.5　各シナリオの週当たり平均処理部品数

検査シナリオ	完成部品数	半幅
現状の手作業	2,255.29	2.21
4 台の単純なロボットによる自動化	2,253.88	2.13
複雑なロボットによる自動化	2,255.95	2.18

A.1.2.1　問への対応

1. 新しい検査方法は週当たりの完成部品数に影響を与えるだろうか。もしそうであれば、現行システムと比較してどの程度変化する（増加または減少する）のかを分析してほしい。

この問題は簡単な待ち行列解析（定常システムでは流入と流出が等しく、部品の期待数は到着率×処理時間で表される）で解くことができるが、シミュレーション実験の結果も活用できる。図 A.4 の SMORE プロットによると、1 週当たりの平均処理部品数はシナリオ間でほとんど差がない。シナリオ 1（現在の手動検査）の週当たりの完成部品の平均数について、シナリオ 2（4 台の単純なロボットを用いる自動検査）とシナリオ 3（1 台の複雑なロボットを用いる自動検査）との間に**統計的に有意な**差がないことを確認するため、2 つの対応のあるt検定を実施した。これらの検定結果は図 A.5 と図 A.6 に示されている。

95%信頼区間$[-1.54, 4.38]$において平均の差がゼロを含んでいることから、シナリオ 1 と 2 の間で週当たりの処理部品数に統計的に有意な差は見られない。図 A.6 の比較から、シナリオ 1 と 3 についても同様の結論が得られる。具体的には、週当たりの平均処理部品数について、平均の差の 95%信頼区間$[-3.70, 2.38]$がゼ

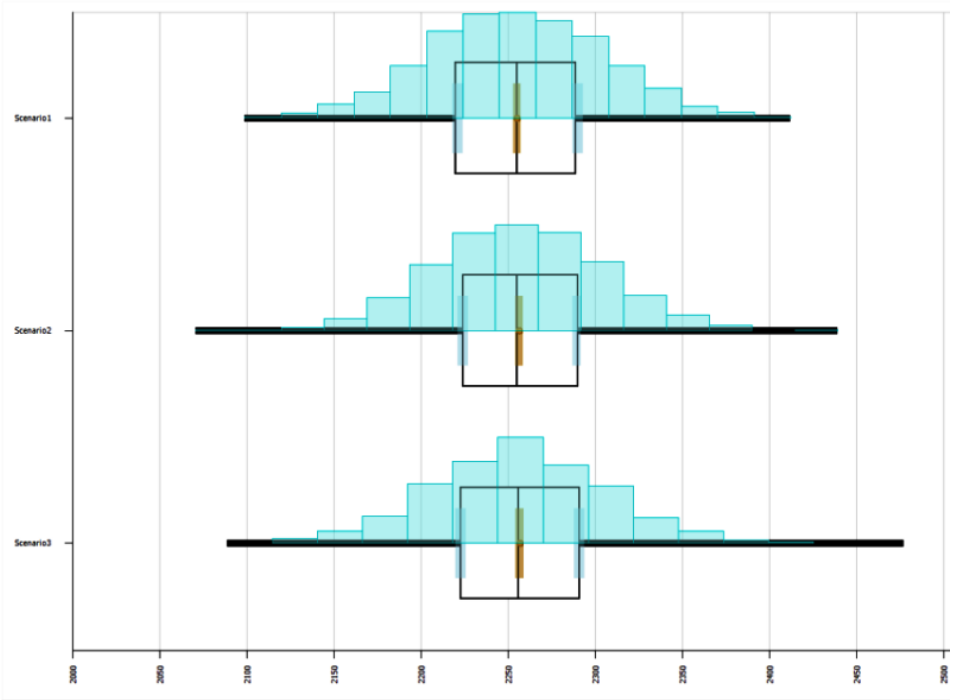


図 A.4　各シナリオにおける週当たり完成部品平均数の SMORE プロット

Paired T-Test and CI: scenario1, scenario2

```
Paired T for scenario1 - scenario2

                          N     Mean  StDev  SE Mean
scenario1              2000  2255.29  50.35     1.13
scenario2              2000  2253.88  48.54     1.09
Difference             2000     1.42  67.48     1.51


95% CI for mean difference: (-1.54, 4.38)
T-Test of mean difference = 0 (vs not = 0): T-Value = 0.94  P-Value = 0.348
```

図 A.5　シナリオ 1 と 2 の対応のあるt検定の結果（Minitab を使用）

Paired T-Test and CI: scenario1, scenario3

```
Paired T for scenario1 - scenario3

                          N     Mean  StDev  SE Mean
scenario1              2000  2255.29  50.35     1.13
scenario3              2000  2255.95  49.75     1.11
Difference             2000    -0.66  69.33     1.55


95% CI for mean difference: (-3.70, 2.38)
T-Test of mean difference = 0 (vs not = 0): T-Value = -0.43  P-Value = 0.671
```

図 A.6　シナリオ 1 と 3 の対応のあるt検定の結果（Minitab を使用）

ロを含んでいることから、シナリオ 1 と 3 の間にも統計的に有意な差はない。この統計的な証拠から、検査手法を変更しても週当たりの完成部品数は実質的に**変化しない**と結論づけられる。

これらのシナリオを比較するもう一つの方法として、Simio の **Scenario Subset Selection** 機能がある。表 A.6 のレスポンス目的関数に基づき「Subset Analysis」を実行すると、Simio は順位づけと選択ルールを用いて各レスポンスを「Possible Best（最適の可能性あり）」または「Reject（棄却）」に分類する。Experiment ウィンドウの Design ウィンドウでは、「Reject」の Response は薄い茶色で、「Possible Best」のシナリオは茶色で表示される。分析結果を図 A.7 に示す。

表 A.6　レスポンスと対応する目標

レスポンス	目標
Total processed	Maximize
Type I errors	Minimize
Type II errors	Minimize
Time in system (TIS)	Minimize
Total cost	Minimize
Holding cost per week	Minimize

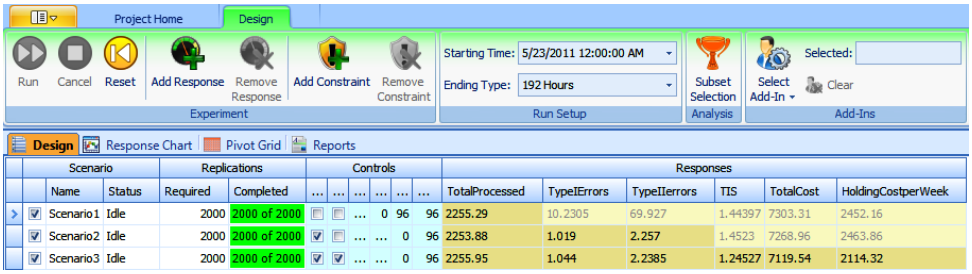


図 A.7　Simio の Subset Selection を用いたレスポンス分析の結果

Simio の Scenario Subset Selection 分析は、対応のある t 検定と同様の結果を示している。いずれのシナリオでもレスポンス TotalProcessed は薄い茶色で表示されていない。つまり、Simio の Subset Selection は、処理部品の週当たり平均数を最大化する「Possible Best（最適の可能性あり）」として、3 つのシナリオすべてを選定した。この分析結果は、週当たりの処理部品数に大きな差異がないという先の予測を裏づけている。

2.　3 つの検査方式の中で、コスト最小化の観点から推奨すべき方式はどれか。

SMORE プロットの図 A.8 からは、どの検査方法がもっとも費用効率がよいかを直接判断することは困難である。そこで、シナリオ間の週当たりの費用について統計的に有意な差があるかを分析するため、Simio の Scenario Subset Selection を再度使用した。図 A.7 の分析結果では、シナリオ 1 と 2 の総費用は「Reject（棄却）」を示す薄茶色で表示され、シナリオ 3 は「Possible Best（最適の可能性あり）」と判定された。この分析結果と、全体の生産量が維持されると

いう事実から、1 台の複雑なロボットによる検査工程への変更を推奨できる。

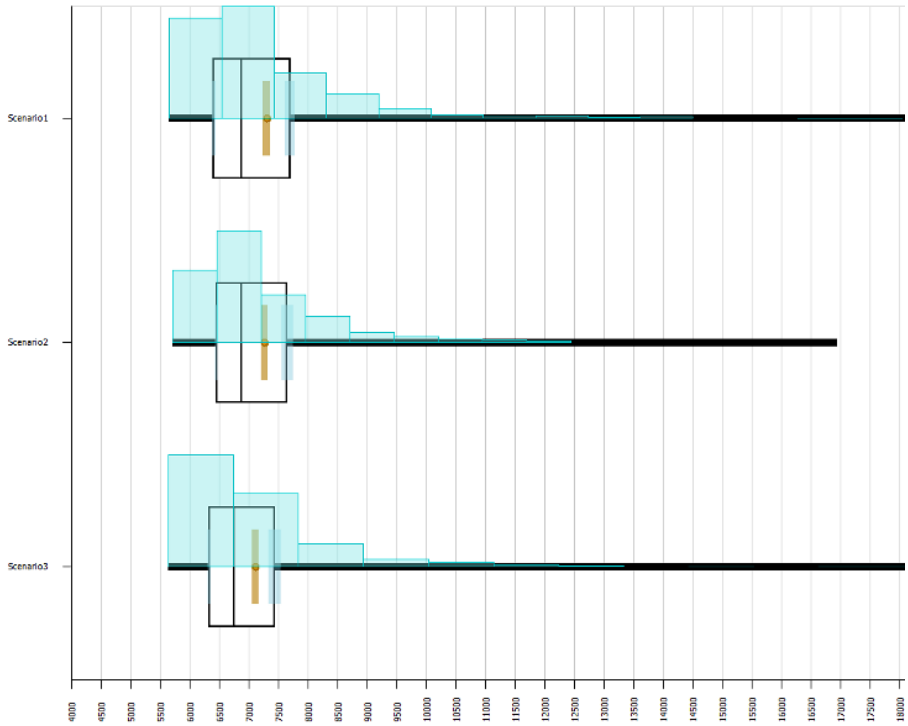


図 A.8　各シナリオにおける総費用の SMORE プロット

3.　もし自動検査工程のいずれかまたは両方が手動検査と同等以上のコストとなった場合、週当たりの運用コストを手動検査より低く抑えるための方策を検討せよ。ただし、すべてのコストは固定値であり、調整不可能と仮定する。

すでに説明したように、シナリオ 1 と 2 の間の週当たりの費用には統計的に有意な差がない。この検査工程の費用を下げる方法を探るため、式 A.1 の総費用を構成する要素を検討する。シナリオ 2 の総費用は、機械費用、第 1 種の誤りに関連する費用、および部品保持費用の 3 つから成る。機械費用は固定で調整できず、第 1 種の誤りの費用も週当たり約$5 と少額であるため、シナリオ 2 の総費用を下げるには部品保持費用の削減が必要となる。部品保持費用はシステム内の滞留時間に直接関係するため、4 台の単純機械の処理時間短縮が有効な解決策となり得る。この仮説は図 A.7 の結果からも裏づけられる。つまり、もっとも低い総費用のシナリオは、もっとも短いエンティティの平均システム内時間ともっとも低い週当たりの保持費用を示している。この仮説を検証するため、単純ロボットの処理時間を段階的に短縮する実験を設定する（表 A.7 参照）。

図 A.9 の実験結果から、単純なロボットの平均処理時間を減少させるにつれて総費用が低下することがわかる（表 A.7 に示すように、シナリオ 4 から 8 では単純なロボットの検査時間を 0.1 分ずつ短縮している）。

表 A.7　4 台の単純機械の処理時間の変化による効果を調べる新しい実験シナリオ

シナリオ	単純なロボットの検査時間
オリジナルシナリオ 1（手作業による検査）	N/A
オリジナルシナリオ 2	tria(5.5, 6.0, 6.5)
オリジナルシナリオ 3（複雑なロボットによる自動化）	N/A
シナリオ 4	tria(5.4, 5.9, 6.4)
シナリオ 5	tria(5.3, 5.8, 6.3)
シナリオ 6	tria(5.2, 5.7, 6.2)
シナリオ 7	tria(5.1, 5.6, 6.1)
シナリオ 8	tria(5.0, 5.5, 6.0)

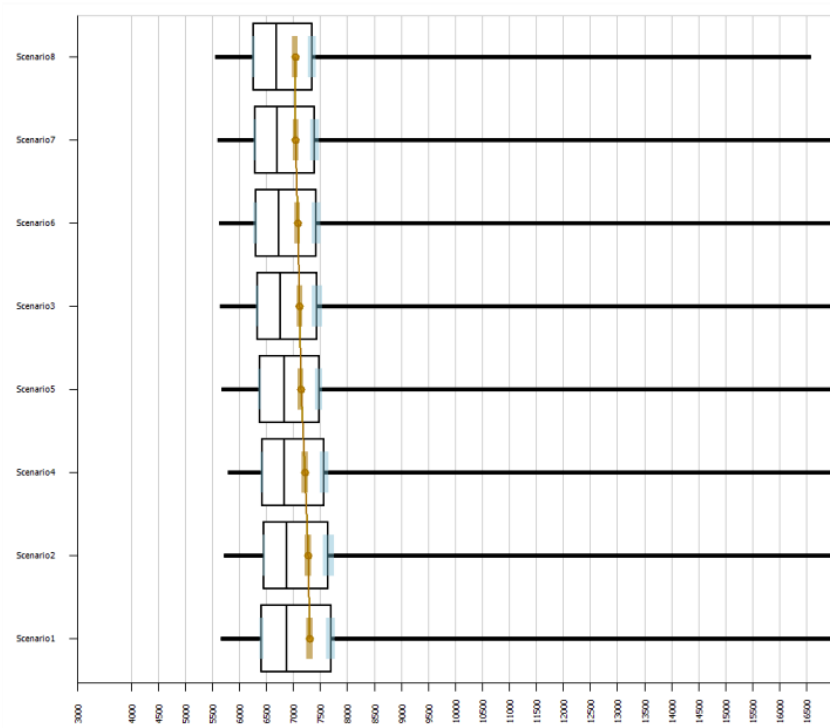


図 A.9　8 シナリオの総費用に対する SMORE プロット

Simio Scenario Subset Selection を用いて、単純なロボットの処理時間をさらに変更した効果を分析する。図 A.10 は、表 A.6 と同じレスポンス目標を設定した場合の Subset Selection の結果を示している。

図 A.10 に示されているように、シナリオ 1 の平均総費用はシナリオ 6 から 8 のものと統計的に有意な差がある。ロボットのメーカーが平均処理時間を 0.3 分短縮できる場合（三角分布(5.2, 5.7, 6.2)）、4 台の単純なロボットを用いる検査工程は手動検査よりも低い週当たりの費用を実現できる。図 A.10 では、4 つの「Possible Best」シナリオ（シナリオ 3、6、7、8）が示されている。これらの

シナリオについて反復回数を 2,000 から 2,500 に増やして Scenario Subset Selection を実施すると、シナリオ 3 は「Possible Best」から除外される（図 A.11）。この分析結果から、ロボットのメーカーが単純なロボットの平均処理時間を 0.3 分短縮できる場合（三角分布(5.2, 5.7, 6.2)の処理時間）、部品検査工程には 4 台の単純なロボットの導入を強く推奨できる。

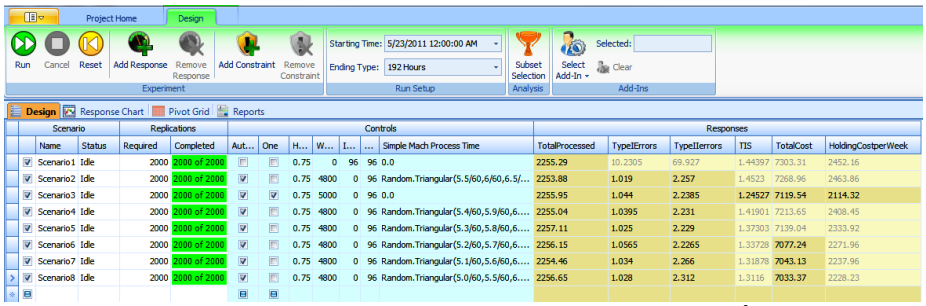


図 A.10　8 シナリオに Scenario Subset Selection を用いたレスポンス分析の結果

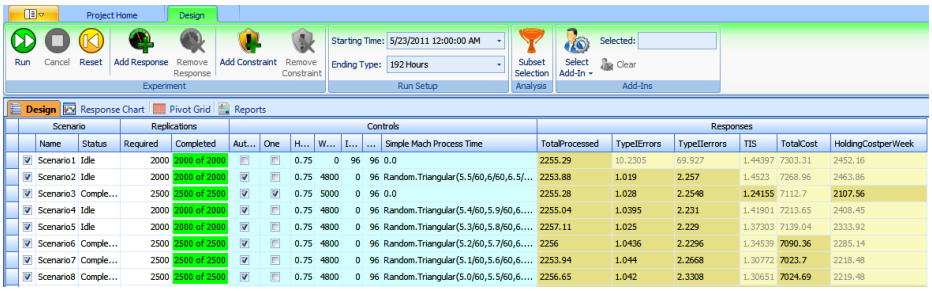


図 A.11　選択したシナリオで実行反復回数を 2,500 に増加した結果

A.2　遊園地

A.2.1　問題記述

ジェットコースター（Roller coaster）、観覧車（Ferris wheel）、アーケードゲーム（Arcade）、カーニバルゲーム（Carnival games）、売店（Concessions）、トイレエリア（Bathroom area）から構成される遊園地がある（遊園地の基本的なレイアウトは図 A.12 を参照）。営業時間は午前 10 時から午後 10 時までである。顧客は開園時刻の午前 10 時から到着し始め、午後 6 時まで続く。顧客の到着は非定常ポアソン過程に従い、表 A.8 に示された割合で発生する。

遊園地に入場した顧客には、ジェットコースターと観覧車で優先的に乗車できる「ファストパス」の購入機会が与えられる。遊園地のマネージャは、ファストパス導入後の待ち時間に関する顧客からの苦情を受け、ファストパスシステムの改善に向けた支援を求めている。マネージャの目標は、ファストパス利用者と一般客両方の待ち時間を適切に管理しながら、ファストパスによる収益を最大化することである。

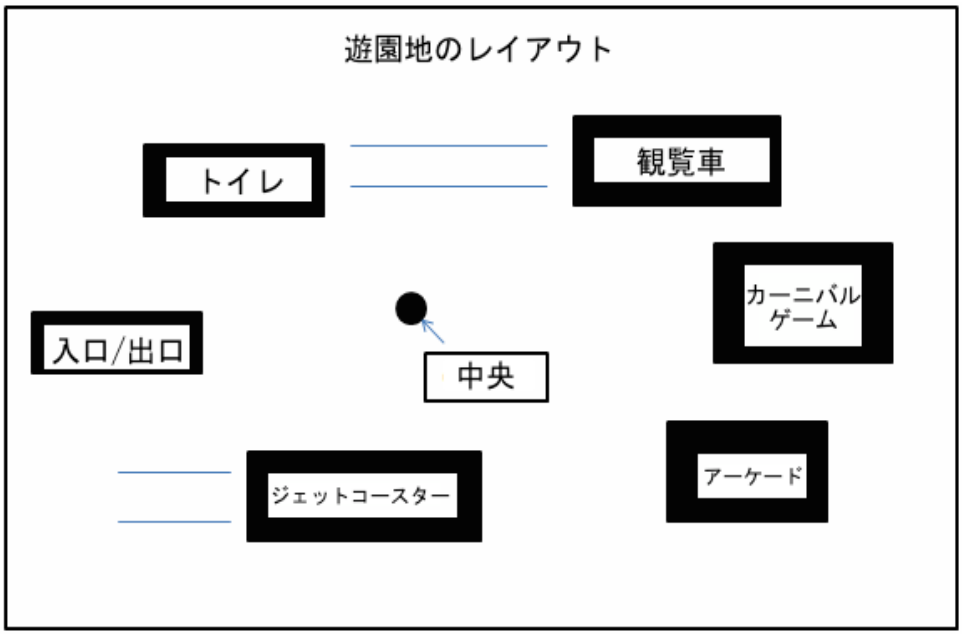


図 A.12　遊園地のレイアウト

表 A.8　遊園地の時間帯別来場者数

時間間隔			時間当たり顧客数
10:00 am	–	12:00 pm	75
12:00 pm	–	2:00 pm	150
2:00 pm	–	4:00 pm	100
4:00 pm	–	6:00 pm	75

A.2.1.1　追加情報

・ファストパスがまったく配布されない場合、顧客は通常約 20 分間待つ必要がある（これは現行システムの実績値である）。顧客は遊園地での待ち時間には慣れているものの、60 分を超えて待たされると著しい不満を示すようになる。
・一般客は「長過ぎる」待ち行列には加わらない傾向にあるが、ファストパス顧客は専用レーンがあるため待ち行列に躊躇なく並ぶ。観覧車の待ち行列が 80 人以上、ジェットコースターが 100 人以上になると、一般客は列に並ぶことを諦めて他のアトラクションに向かう。
・ファストパス顧客は 5～10 分程度の待ち時間は許容するが、10 分を超えると不満を感じ始める。
・顧客の 50%はアーケードゲームにまったく興味を示さない。
・顧客の 50%は 1 回、残りの 50%は 2 回トイレを利用する。トイレは 2 か所（各定員 5 名）のみで、1 日を通じて平均約 9 分の待ち時間が発生している。
・顧客は売店（フードコートを含む）を 2 回までしか利用しない。
・遊園地での滞在時間は顧客によって異なる。表 A.9 は滞在時間の確率質量関数

（PMF）を示しており、0.5、1、2、4、5（時間）の値をとる離散確率変数として表される。

表 A.9　利用者の遊園地滞在時間の確率質量関数

滞在時間	確率
0.5	0.01
1	0.14
2	0.25
4	0.4
5	0.2

A.2.1.2　顧客の移動時間とエリア選択

顧客は利用可能性と予想待ち時間に基づいて次に訪れる目的地を選択する。トイレと売店の利用は、前述のとおり 1 回か 2 回に制限される。遊園地の中央には「大きなボード」が設置されており、各アトラクションの現在の待ち時間予測が表示されている。すべての顧客は各エリアを訪れる前に中央エリアを経由し、この大きなボードを確認して次の目的地を決める。顧客は、その時点で利用可能なエリアの中から等確率で 1 つを選択する。たとえば、ジェットコースターに長蛇の列があり、食事も済ませ、アーケードゲームにも興味がない場合、顧客がカーニバルエリア、トイレ、観覧車のいずれかを選ぶ確率は同じになる。また、顧客の予定滞在時間を超過した場合は、出口への経路も選択肢に加わる。表 A.10 は、中央エリアから各エリアまでの顧客歩行時間を示している。表の**最小時間**は目的地までの最短所要時間を、**変動**は最小時間に追加される歩行時間（指数分布の平均値）を表す。

表 A.10　顧客の歩行時間（分）

エリア	最小時間	変動
ジェットコースター	1	1
観覧車	2	1
カーニバルゲーム	2	2
アーケードゲーム	3	2
トイレ	1	1
入口／出口	3	1

A.2.1.3　アトラクション

アトラクションの情報を表 A.11 に示す。ジェットコースターには 2 台の車両があり、1 台が運行中でも 2 台目に乗客が乗り込めるシステムとなっている。観覧車のゴンドラは 2 人乗りである。

A.2.1.4　その他のエリア

他の 4 つのエリアには、表 A.12 に示すように、それぞれ固有のサービス時間

（エリア滞在時間）と定員がある。最大待ち人数は、そのエリアでサービスを待つことができる最大顧客数を指す。たとえば、すべてのアーケードゲームが使用中の場合でも、最大 5 人まで待機できる。

表 A.11　アトラクションの情報（時間単位は分）

	車両／ゴンドラ数	定員	最大待ち人数	運行時間	乗車時間	降車時間
ジェットコースター	2	20	100	4	tria(5, 10, 13)	0.5
観覧車	30	60	80	10	1.2	0.5

表 A.12　その他エリアの情報（時間単位は分）

エリア	エリア滞在時間	定員	最大待ち人数
カーニバルゲーム	tria(5, 10, 30)	40	40
アーケードゲーム	tria(1, 5, 20)	30	5
トイレ	tria(3, 5, 8)	10	30
売店	tria(1, 4, 8)	4	25

A.2.1.5　追跡する統計値

・ジェットコースターと観覧車における、ファストパスと一般客それぞれの待ち行列の平均時間と最大時間。
・遊園地内のファストパス利用者と一般客の平均人数。
・アトラクション待ち行列におけるファストパス利用者と一般客の平均待ち時間。
・トイレの平均待ち人数と平均待ち時間。

マネージャはファストパスの運用に関心を持っている。遊園地ではファストパスの販売数を制限できると仮定する（つまり、来園者の一定割合のみがファストパスを購入できる。マネージャは、価格設定によってこの制限が可能だと確信している）。顧客満足度の低下や不満が生じない範囲で、ファストパス利用者の割合をどの程度に設定すべきか、マネージャに提言する必要がある。

A.2.2　サンプルモデルと結果

本項では、著者が構築したサンプルモデルの結果を示す。前のケーススタディと同様に、特定の結果はいくつかの前提に依存するため、読者の結果と必ずしも一致しないかもしれない。以下にモデル仕様を示す：

・反復実行期間：12 時間
・時間の基本単位：分
・100 回の反復実行

図 A.13 はサンプルモデルの Facility ビューを示している。遊園地では、ファストパス顧客数の増加に伴い、一般客とファストパス顧客の両方で待ち時間が増

加すると予想される。この仮説を検証するため、ファストパス顧客の割合を制御するプロパティを追加し、1%刻みで 0%から 25%まで変化させる実験を実施した。図 A.14 の SMORE プロットは、ファストパス顧客の割合に応じた一般客の待ち時間を示している。予想通り、ファストパス購入者の割合が増えるにつれて、待ち時間も増加している。

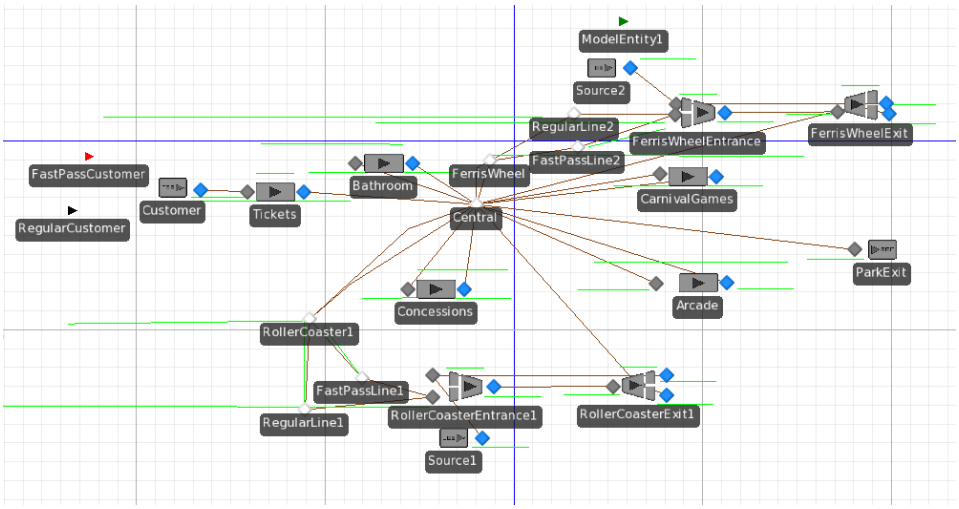


図 A.13　遊園地のサンプル Simio モデル

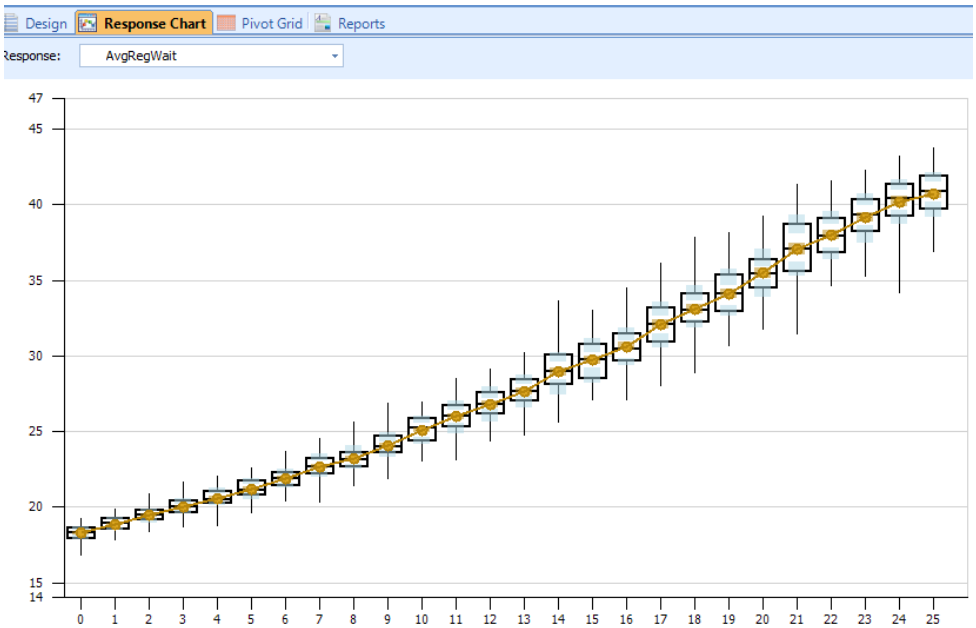


図 A.14　ファストパス顧客の割合ごとの一般客の平均待ち時間の SMORE プロット

同様に図 A.15 は、ファストパス顧客の割合ごとのファストパス利用者の待ち時間を SMORE プロットで示している。こちらも、ファストパス利用者の割合が増加するにつれて、待ち時間が長くなる。

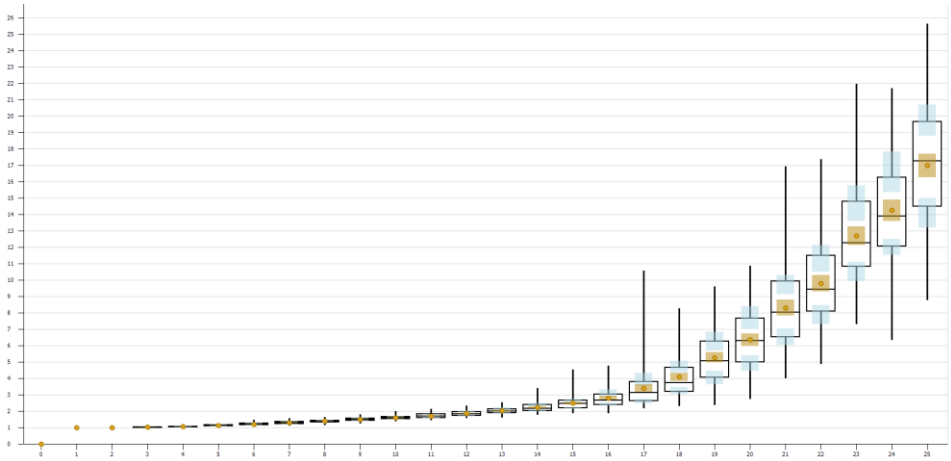


図 A.15　ファストパス顧客の割合ごとのファストパス顧客の平均待ち時間の SMORE プロット

ファストパス顧客の「最適な」割合を決定するためには、顧客の待ち時間に関する追加的なコスト情報が必要である。しかし、SMORE プロットは、マネージャの意思決定に役立つ相対的なパフォーマンス情報を提供している。たとえば、ファストパスオプションを利用する顧客が 15%の場合、一般客は 30～35 分の待ち時間となり、ファストパス顧客は約 5 分の待ち時間となることがわかる。ただし、これらの推定値は図 A.14 および図 A.15 の SMORE プロットを目視で確認した結果に基づいており、より正確な結果を得るためには、さらに詳細な追加実験が必要となるだろう。

A.3　小さなレストラン

A.3.1　問題記述

あるレストランのレイアウトが図 A.16 に示されている。

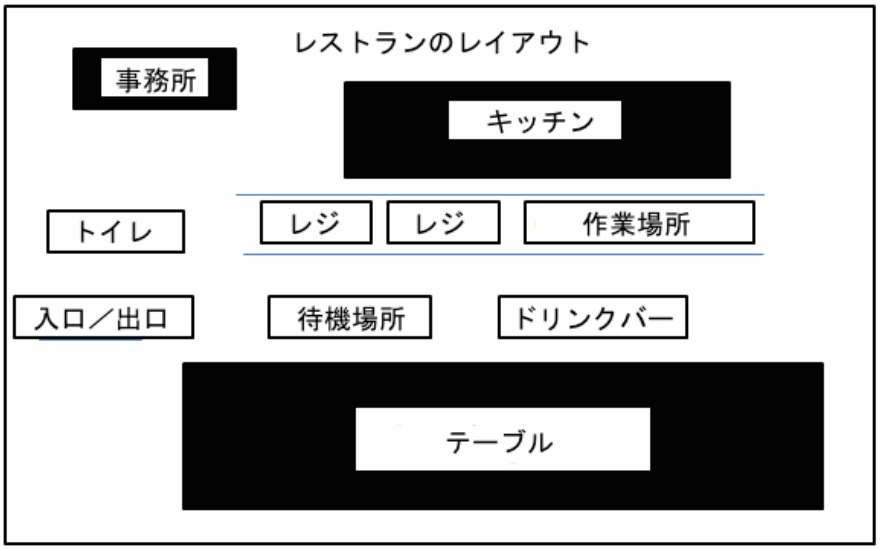


図 A.16　レストランのレイアウト

レストランは午前 8 時に開店し、午前 11 時まで朝食を、以降は午後 8 時まで昼食および夕食を提供している。午後 8 時に店内への入店は締め切るが、すでに入店している顧客は食事を終えるまで滞在できる。顧客は店内での食事か持ち帰り（To Go）を選択できる。過去の実績によると、10%の顧客が持ち帰りを選択しており、キッチンではこれらの注文を優先的に処理している。

グループが入店すると、推定待ち時間が伝えられ、店内で食事をする場合は滞在するか立ち去る（ボークする）かを決める。滞在を選択したグループは注文を行い、キッチンで食事が準備される。注文の品が用意されると、レストランの従業員が待機中のグループに渡す。To Go 注文のグループはそのまま退店し、店内飲食のグループはテーブルに案内される。食事を終えたグループが退店した後、レストランの従業員は次の利用のためにテーブルを清掃する。

顧客は表 A.13 に示される非定常ポアソン過程の割合に従って到着する。グループサイズは 1 人から 4 人で、その構成比は 1 人が 10%、2 人が 30%、3 人が 40%、4 人が 20%である。注文時間はグループサイズに比例する（例：3 人グループの注文時間は 1 人分の 3 倍）。各作業のサービス時間分布と担当従業員数は表 A.14 に示すとおりである。キッチンでは、グループごとに注文を取りまとめて一括で提供する。

表 A.13　時間帯別のレストラン来店グループの平均到着率

到着時間帯			到着グループ数
8:00 am	–	9:00 am	39
9:00 am	–	10:00 am	35
10:00 am	–	11:00 am	48
11:00 am	–	12:00 pm	44
12:00 pm	–	1:00 pm	49
1:00 pm	–	2:00 pm	39
2:00 pm	–	3:00 pm	26
3:00 pm	–	4:00 pm	20
4:00 pm	–	5:00 pm	39
5:00 pm	–	6:00 pm	48
6:00 pm	–	7:00 pm	37
7:00 pm	–	8:00 pm	31

表 A.14　サービス時間と従業員の情報（時間単位は分）

サービス	サービス時間の分布	従業員数
注文	expo(0.5)	1-2
調理	tria(2, 5, 9)	5-10
清掃	expo(0.5)	1-2
配膳	–	1-3

テーブルの待ち時間の期待値が 60 分を超える場合、または 5 グループ以上が待っている場合、新規グループは待つことを躊躇する。現在、レストランには 25 のテーブルがある。各グループの食事時間は三角分布(25, 45, 60)分に従う。

レストランには 3 種類の従業員がいる。調理人はキッチンで料理の準備と調理を担当する。一般従業員は受付（最大 2 レジスタ）、料理の配膳、テーブルの清掃を行う。マネージャは調理と配膳の補助を行うが、テーブル清掃と受付業務は担当しない。マネージャの主な役割はキッチンでの調理補助で、常時 1 名以上のマネージャが在席する。キッチン内の従業員が 10 名を超えると混雑状態となる。

注文から 30 分以上待たされた顧客は不満を感じ始める。また、料理は調理後 8 分で冷めてしまう。人件費はマネージャが時給$10.00、調理人が時給$7.50、一般従業員が時給$6.00 である。顧客 1 人当たりの平均支払額は$9.00 である。

忙しい日の現在のスタッフ配置は、マネージャ 2 人、調理人 8 人、一般従業員 5 人である。オーナーは新しい類似レストランのオープンを計画しており、現在のシステムについて以下の点を把握したいと考えている：

1. 長い待ち行列によって何グループが退店（ボーク）するか。テーブルが空くまでの平均待ち時間はどれくらいか。
2. 顧客満足（待ち時間と料理の適温提供）を維持しながら人件費を最小化するには、1 週間の各曜日にマネージャ、調理人、一般従業員を何名ずつ配置すべきか。また、選択したシナリオにおける総利益（売上－人件費）はいくらか。
3. 長時間の待ち時間を防ぎ、ボーク数を削減するために必要なテーブル数はいくつか。
4. 長時間の待ち時間によりボークが頻発しているため、レストランは実際の需要を把握できていない。前問で提案したテーブル数でシナリオを作成し、各従業員タイプの必要人数とテーブル待ち時間の平均を算出せよ。また、総利益はいくらか。
5. 到着率が 20%増加した場合、ボーク数を抑制するために推奨すべきテーブル数はいくつか。
6. 到着率が 30%減少した場合、各従業員タイプの推奨人数は何名か。

A.4　小さな銀行支店

小さな銀行支店について、以下のような設定と運営を検討する。支店内には ATM、窓口、およびマネージャが配置されている。営業時間は月曜から金曜までの午前 9 時から午後 5 時までである。午後 5 時以降は新規の来店を受け付けないが、すでに店内にいる顧客全員への対応が完了するまでは営業を継続する。図 A.17 に銀行のレイアウトを示す。

顧客は 4 つの独立した流れで来店し、それぞれは表 A.15 に示す時間帯別の非定常ポアソン過程に従う。来店客は目的に応じて、窓口（**窓口顧客**）、ATM（**ATM 顧客**）、またはマネージャ（**マネージャ顧客**）のいずれかを利用する。窓口と ATM

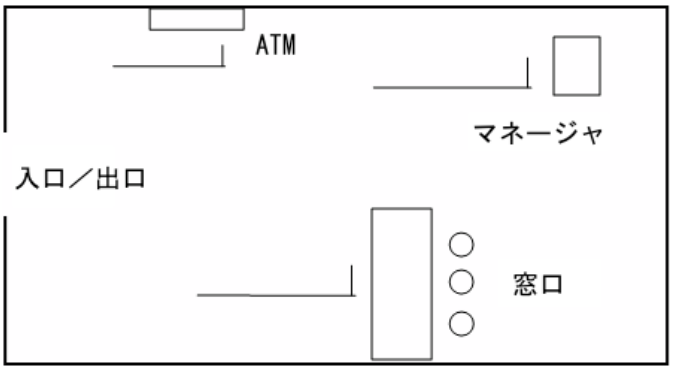


図 A.17　銀行のレイアウト

どちらでも可能な顧客（**窓口・ATM 顧客**）は、もっとも短い列を選択する。両方の列が同じ長さの場合は、ATM を選択する。ただし、最短の列でも 4 人以上待っている場合、70%の確率で入店を諦めて帰る（ボークする）。**ボークしたすべての顧客は、「不満足な顧客」として扱われる。**

表 A.15　到着率

時間帯	到着率（時間当たり顧客数）			
	窓口	ATM	窓口・ATM	マネージャ
9:00 － 10:00	20	20	30	4
10:00 － 11:30	40	60	50	6
11:30 － 1:30	25	30	40	3
1:30 － 5:00	35	50	40	7

ATM 顧客が到着した際、ATM の列が 3 人以上かつ窓口の列が 3 人未満の場合、その顧客は 85%の確率で窓口サービスに切り替える。ただし、窓口の待ち行列が 4 人以上の場合、これらの ATM 顧客の 80%は離れてしまう。また、窓口顧客が到着時に窓口の列が 4 人以上の場合、90%の確率で帰ってしまう。銀行の分析によると、このような「不満足な顧客」が発生するたびに、約 100 ドルの損失が生じている。

大多数の顧客は取引完了後に退店するが、ATM 利用者の 2%と窓口利用者の 10%は、サービス完了後にマネージャとの面談が必要となる。また、窓口の待ち時間が 5 分を超えた顧客は、窓口での用事が終わった後、50%の確率でマネージャとの面談を希望する。さらに、マネージャの待ち行列で 15 分以上待たされた顧客の 70%が不満を表明する。これらの待ち時間による不満顧客もすべて「不満足な顧客」として計上される。銀行内の各エリア間の移動時間は表 A.16 に示されており、これらはすべて固定値として扱われる。

ATM は一度に 1 人の顧客にしかサービスできず、ランダムに発生する詰まりによって取引に遅延が生じることがある。この詰まりは機械のハードリセットによる修理が必要となる。最近、マネージャの 1 人がインターンに依頼し、ATM の詰まりの頻度とハードリセット完了までの所要時間を把握するためのデータを収集さ

せた。これらのデータは本書のウェブサイトの students にある ATMdata_bank.xls ファイルに収録されている（付録 C 参照）。アップタイム（ハードリセット完了から次の詰まりまでの時間）とリセットに要する時間のデータは、それぞれ図 A.18 と図 A.19 のヒストグラムで示されている。

表 A.16　移動時間（秒）

	入口・出口	ATM	窓口	マネージャ
入口・出口	–	10	15	20
ATM	10	–	8	5
窓口	15	8	–	9
マネージャ	20	5	9	–

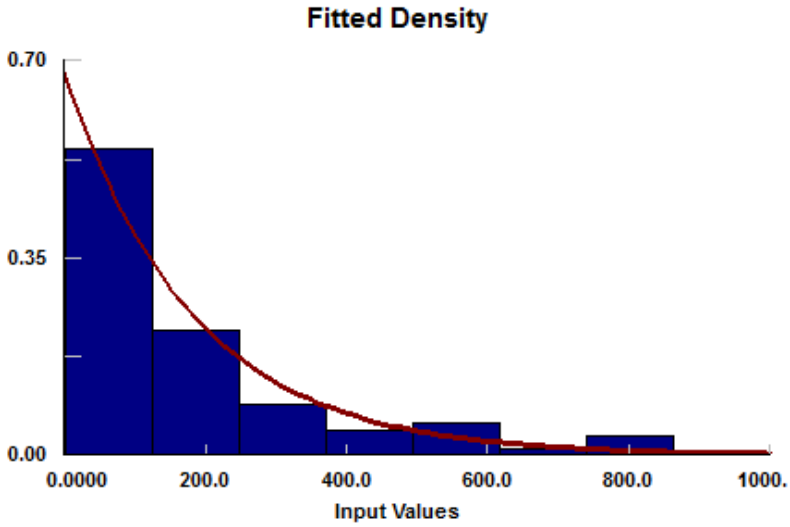


図 A.18　ATM のアップタイムとして収集されたデータのヒストグラム

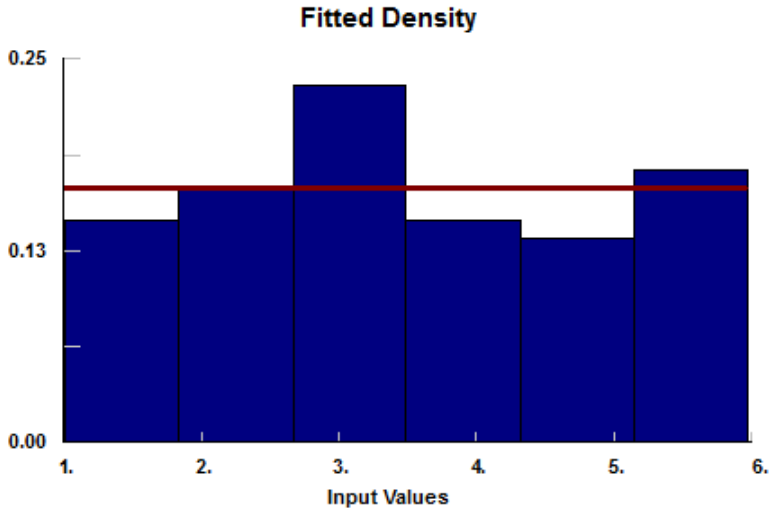


図 A.19　詰まった ATM のリセット時間として収集されたデータのヒストグラム

マネージャは一度に 1 人の顧客に対応でき、営業時間中は常に 1 名配置されている。窓口担当者には「**ベテラン**（experienced）」と「**新人**（inexperienced）」の 2 種類がいる。窓口には 1 つの共通待ち行列があり、各窓口は一度に 1 人の顧客のみ対応できる。窓口担当者の 1 時間当たりの人件費は、ベテランが$65、新人が$45 である。この費用には保険、福利厚生、401K プラン、休暇手当などの間接費と時給が含まれる。顧客のサービス時間は表 A.17 に示す分布に従う。なお、マネージャの対応時間は、その顧客が事前に ATM または窓口を利用したかどうかに関係なく独立している。

表 A.17　銀行支店のサービス時間（分）

サーバ	サービス時間の分布
ATM	tria(0.50, 0.83, 1.33)
ベテラン窓口	tria(2, 3, 4)
新人窓口	tria(4, 5, 6)
マネージャ	expo(5)

慣例により、銀行では 1 日を 4 つの時間帯に区分している。銀行のマネージャは、運営費を最小化するため、各時間帯に配置すべき窓口数についてあなたのアドバイスを求めている（分析では提供される費用のみに焦点を当てること）。窓口エリアには 3 つのステーションがあるため、稼働可能な窓口は各時間帯で最大 3 つまでである。なお、窓口担当者はスケジュールされた時間帯中、休憩なしで勤務する：

・時間帯 1：午前 9 時　－ 午前 11 時
・時間帯 2：午前 11 時－ 午後 1 時
・時間帯 3：午後 1 時　－ 午後 3 時
・時間帯 4：午後 3 時　－ 午後 5 時

また、銀行のマネージャは追加の ATM の設置を検討している。ATM 業者は 2 つの購入オプションを提案している。1 つは新型機、もう 1 つは大手銀行で使用されていた中古 ATM である。業者の見積もりによると、1 日当たりの費用は新型機が 500 ドル、中古機が 350 ドルである。中古 ATM は現在設置されている ATM と同じ頻度で詰まり、ハードリセットが必要となる。新型機も同様の頻度で詰まるが、20 秒で自動リセットができる機能を備えている。銀行マネージャは、これらの情報を踏まえ、追加の ATM を購入すべきか、また購入する場合はどちらを選ぶべきかについて、あなたの提言を求めている。

上述の銀行支店を Simio でモデル化する。モデルに加えて、銀行マネージャから要求された以下の項目にも回答する必要がある。提言は、本節の冒頭で示された費用測定基準（「不満足な顧客」への対応費用、追加 ATM の導入費用、窓口費用）に基づき、銀行の運営費用の最小化を目指すべきである。また、すべての回答は実験結果の分析によって裏づけられなければならない。

1.　銀行は追加の ATM を導入すべきか。また、導入する場合は新型機と中古機

のどちらを選択すべきか。

2. 銀行の窓口について、4 つの時間帯それぞれに最適な配置数を決定しなさい。
3. 費用測定基準に基づく銀行の 1 日当たりの予想運営費用はいくらか。
4. 「不満足な顧客」への対応にかかる 1 日当たりの予想費用はいくらか。
5. ATM の故障頻度と修理所要時間はどの程度か（**これらの質問には、既習の手法を用いて回答すること**）。
6. 銀行の 1 日当たりの平均対応顧客数はいくらか。ここでは、銀行に入店し取引を行ったすべての顧客を含める（ただし、長時間の待ち行列を理由に入店直後に退出した顧客は除く）。
7. 全来店顧客のうち、「不満足な顧客」の予想割合はどの程度か。

A.5　Vacation 市の空港

Vacation 市の空港（VCA）は、人気の観光地に近接する大規模な国際空港だ。季節により到着便数が大きく変動するこの空港で、VCA 経営陣は航空会社の顧客サービスレベル（CSL）調査を実施し、ターミナル 2 の総システム内時間が不十分であることを認識した。経営陣は、コストを考慮しながら CSL の向上を目指している。特に、適切な人員配置の決定と、新規ゲートの増設か新規滑走路の建設かの判断に向けた客観的な分析を求めている。主要な評価指標は、大型機・小型機それぞれのシステム内時間、ゲート稼働率、各業務における作業者の稼働率である。

VCA には 2 つの国際ターミナルがある（図 A.20）。

図 A.20　Vacation 市の空港の航空写真

両方のターミナルで共通の 3 つの滑走路があり、各滑走路の到着率は異なる。航空機が滑走路に着陸すると、誘導路を通ってターミナルへ向かう。その後、所定の時間分布に従って駐機レーンとゲートに誘導される。駐機後は、荷物と乗客の降機（同時進行）、清掃、燃料補給、ケータリング（順次実行）、そして荷物と乗客の搭乗（同時進行）といった一連のサービスが必要となる。各サービスには専用のチームが配置され、それぞれ異なるリソースを使用する。すべてのサービスが完了すると、航空機は誘導路を通って国際線用の滑走路に向かう。このプロセスは

図 A.21 に示されている。既存システムで利用可能なチームの規模と数は表 A.18 に示すとおりである。

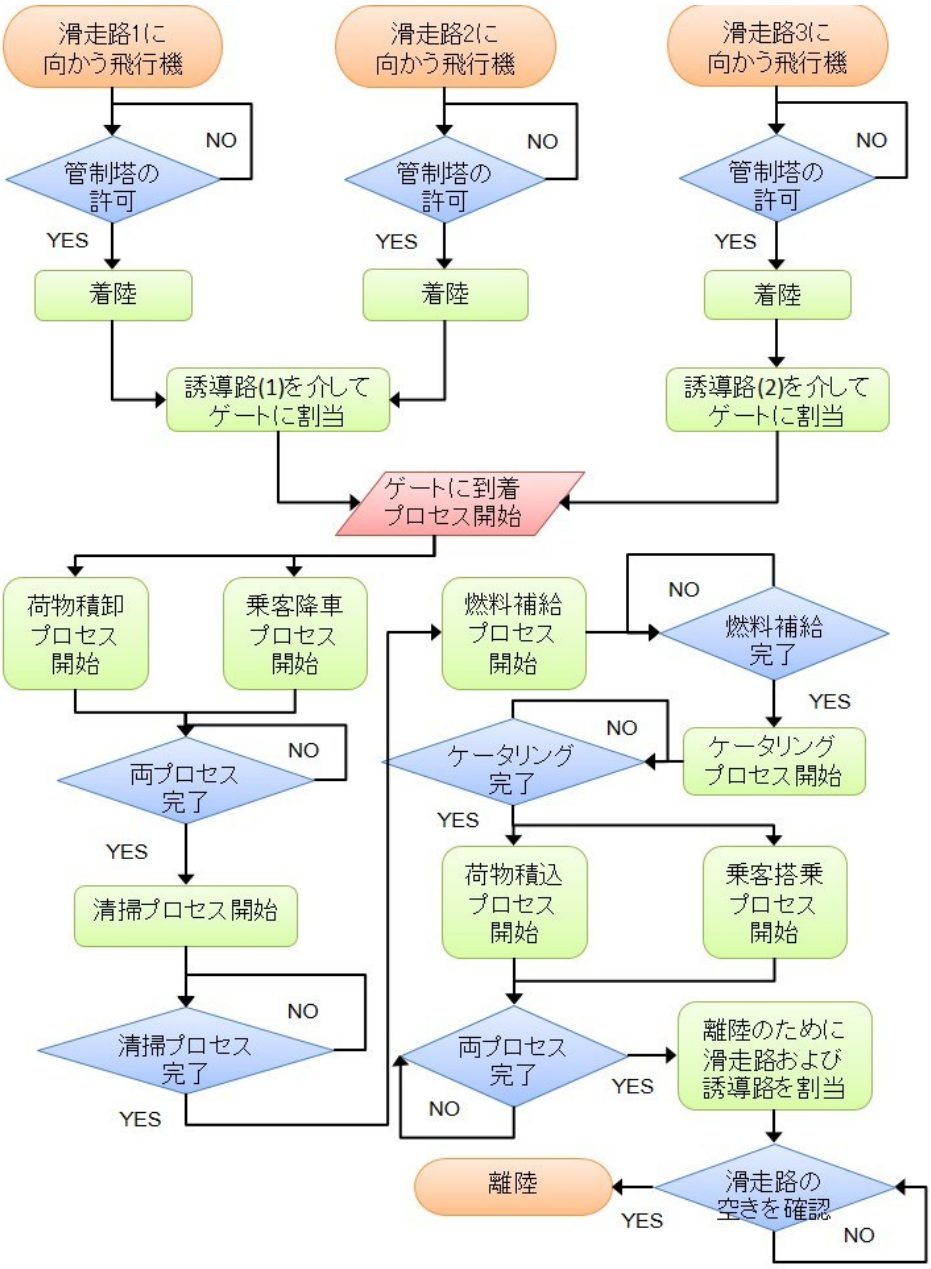


図 A.21　VCA オペレーションの既存システムフローチャート

航空機の到着は時間帯によって変動し、正午頃にピークを迎える（表 A.19）。空港経営陣は航空機を大型機（120 席以上）と小型機（120 席未満）の 2 種類に分類している。すべてのゲートは両方のサイズの航空機に対応可能である。計画期間中、ターミナル 2 での到着便の配分は、滑走路 1 が 20%、滑走路 2 が 20%、滑走路 3 が 30%となっており、残りの便はターミナル 1 で調整される。

表 A.18　VCA ターミナル 2 における作業要員の現状

プロセス	チーム内の作業員数	現在のチーム数
手荷物	2	6
乗客（搭乗等）	1	4
清掃	2	2
燃料補給	1	2
ケータリング	2	1

表 A.19　1 時間ごとの各滑走路における VCA の航空機到着率

時刻	滑走路 1	滑走路 2	滑走路 3
24 – 1	2	5	3
1 – 2	3	6	2
2 – 3	1	2	2
3 – 4	3	1	1
4 – 5	3	3	2
5 – 6	2	2	3
6 – 7	4	4	4
7 – 8	7	6	6
8 – 9	9	8	8
9 – 10	4	7	5
10 – 11	8	6	7
11 – 12	6	10	8
12 – 13	9	11	11
13 – 14	9	10	10
14 – 15	12	8	9
15 – 16	7	8	7
16 – 17	5	8	6
17 – 18	5	5	3
18 – 19	4	9	7
19 – 20	9	6	6
20 – 21	7	4	5
21 – 22	11	10	9
22 – 23	7	8	8
23 – 24	6	5	4

新しい滑走路（11,000 フィート）の建設費用は 1 リニアフィート当たり $2,200、新しいゲート（21,000 平方フィート）の建設費用は 1 平方フィート当たり $28 である。各作業チームの 1 時間当たりの費用（関連設備費用および按分費用を含む）は表 A.20 に示される。

表 A.20　VCA ターミナル 2 の作業チーム別時間当たり費用

プロセス	費用/チーム
手荷物	$142
乗客（搭乗等）	$40
清掃	$108
燃料補給	$150
ケータリング	$145

VCA の経営陣は、上記の各オペレーションにおける典型的な処理時間（分）について 100 個のサンプルを提供した。これらのデータは教科書サイトの students にある `AirportCaseStudyInputData.zip` ファイルで確認できる。

謝辞：Vacation 市の空港は、実際の空港の課題解決のための Hazal Karaman と Cagatay Mekiker によるプロジェクトに基づく。このプロジェクトはピッツバーグ大学の大学院における初級シミュレーション講義用に開発された。

A.6　シンプルな最良の病院

「シンプルな最良の病院（STBH）」は、救命救急科（ED）にポッドシステムを導入しようとしている大規模病院である。ポッドとは 7 つの診察室に囲まれたエリアで、専任スタッフが配置される。病院は 6 つのポッドを設置し、各ポッドに 1～2 名の看護師を配置する計画である。STBH の経営者は、以下の 2 点について評価を求めている：

- 患者の待ち時間を最小限に抑えつつ、看護師の効率的な稼働率を維持するために、各ポッドに何名の看護師を配置すべきか。
- 医師が血液検査を指示してから検査室に到着するまでの所要時間はどれくらいか。なお、病院の方針では検査時間を 15 分未満に維持する必要がある。

STBH の ED には 2 つの入口がある（図 A.22）。1 つは徒歩で来院する患者用、もう 1 つは救急車で搬送される患者用である。徒歩の患者は、まず受付で書類を記入し、その後トリアージエリア（重症度判定エリア）へ移動する。トリアージエリアでは、看護師が症状の重症度に基づいて処置の優先度（1～5、5 が最重症）を決定する。

図 A.22　STBH の施設レイアウト

優先度が決まると、トリアージ看護師は患者情報をコンピュータシステムに入力し、ポッドと看護師の空き状況（ポッドに空きがあり、担当看護師が他の患者対応中でない状態）に応じて患者を割り当てる。その後、トリアージ看護師は患者を空き部屋のあるポッド入口まで案内する。ポッド看護師は入口で患者を引き受け、診察室へ誘導し、患者が落ち着けるよう補助しながら必要な設備を準備する。

患者がポッドに到着後、医師が診察を行い、必要に応じて血液検査（全患者の75%が対象）を指示する。看護師は時間的余裕がある際に採血を実施し、患者名をラベリングして病院事務職員（HUC）に渡す。HUC は血液検体を受け取り、必要な検査項目をラベリングし、シュートを通じて検査室へ送る。なお、HUC のデスクでの処理時間は三角分布(3, 4, 5)分である。

教科書サイト（付録 C 参照）の students にある STBH_ArrivalData.xls ファイルには、処置優先度レベル別の患者到着データが収録されている。過去のデータによると、優先度 4 の患者の 80%、優先度 5 の患者の 95%が救急車で搬送され、その他の患者はすべて直接来院する。各処理時間は三角分布（最小値、最頻値、最大値）と仮定され、表 A.21 に示されている。

表 A.21　STBH における優先度別の処理時間（三角分布、単位：分）

重症度	受付時間	トリアージ時間	処置時間
1	3, 5, 8	4, 5, 7	15, 85, 125
2	3, 5, 8	4, 5, 7	20, 101, 135
3	3, 5, 8	4, 5, 7	35, 150, 210
4	1, 2, 4	1.5, 2, 4	65, 205, 350
5	1, 2, 4	1.5, 2, 4	90, 500, 800

トリアージでは、患者の血圧と体温を測定し、一般的な病状を記録する。重篤な症状を訴える患者については、トリアージを迅速に行い、速やかに処置室へ案内する。患者の部屋の割当は以下の方針で行う。最初の患者はポッド 1 に案内し、ポッド 1 が満室（7 室すべて使用中）の場合は、次の患者をポッド 2 へ案内する。これをポッド 2 が満室になるまで続ける。この過程で、もしポッド 1 に空室が出た場合、次に入室する患者はポッド 1 に案内される。常に番号の小さいポッドの空室を優先的に使用するこの方式は、病院の運営費とスタッフ配置の効率化に寄与している。なお、ED での必要な処置を終えた患者はシステムから退出したものとみなす。

STBH の経営者は以下の主要な測定基準を重視している：

・ATWA：徒歩来院患者の待合所における平均待ち時間（全優先度レベルの平均）。
・NBT：実施された血液検査の総数。
・ATB：血液検査に要する平均時間。
・A1TS, …, A5TS：優先度レベル 1 から 5 までの患者それぞれのシステム内滞在時間。

STBH の経営者は、以下の 6 つのシナリオについて特に関心を持っている（こ

れ以外のシナリオも検討される可能性がある）：

1. 通常より少ないスタッフ配置（看護師 10 人、通常 12 人）で、列車衝突や飛行機墜落などの大規模事故により、優先度 4・5 の患者約 54 人が ED に搬送される 1 日間のモデル化。
2. 全看護師が勤務している状況で、優先度 1・2・3 の患者約 72 人が来院する 2 日半のモデル化。このシナリオは、軽度の伝染病発生により、多数の患者が咽喉炎、頭痛、胃の不調を訴える状況を想定している。
3. 通常の週と、低スタッフ配置（1 ポッドにつき看護師 1 人）かつ少ない患者数（週 200 人以下）の週との比較。
4. すべての患者の処置時間が 10%増加した場合と通常状況との 1 週間の比較。
5. 通常の平日から、各ポッドに看護師 1 人配置の週末への移行のモデル化。
6. ウイルス蔓延により、優先度 1・2 の患者 120 人が 3 日間で ED を受診する状況のモデル化。スタッフ配置は通常より少なく、3 つのポッドには看護師 2 人ずつ、残り 3 つのポッドには 1 人ずつと仮定する。ウイルス関連のため、患者の 90%が血液検査を必要とする。

謝辞：STBH は、実際の病院の課題解決のための Karun Alaganan と Daniel Marquez によるプロジェクトに基づく。このプロジェクトはピッツバーグ大学の大学院における初級シミュレーション講義用に開発された。

付録 B　Simio アカデミックケーススタディ・学生コンペティション課題

Simio は半年ごとに、同社の顧客が直面する実際の課題に基づいたアカデミックケーススタディを公開している。このケーススタディプログラムは、2023 年 8 月から従来の学生コンペティションに替わるものとして導入された。

https://www.simio.com/academic-case-studies/

B.1　Simio Fresh Mex

注文を受けてから調理するメキシコ料理レストラン「Simio Fresh Mex」のコンサルタントとして、総支配人のレストラン運営改善を支援する任務を担当する。総支配人は飲食業界での経験は豊富だが、技術的な知識やモデリングの経験がないため、運営とスタッフ配置のルール確立に苦心しており、顧客サービスの改善を目指している。

Simio Fresh Mex は店内注文とオンライン注文の両方に対応し、5 つのステーションで構成される調理ラインですべてのメニューを調理する。この課題では、レストラン運営の Simio モデルを 1 つまたは複数開発し、それらを活用してリソース要件と顧客サービスに基づいたシステムの最適化を図る。具体的には、スタッフ配置、タスク割当、その他の運営パラメータを決定する。モデルの開発と実験のための過去のデータも提供する。

B.2　Simio Warehouse

このケーススタディは、高度な映像処理、AI、暗号資産アプリケーション用のコンピュータビデオカードを扱う e コマースサイトの倉庫を分析するためのモデル開発について扱う。この施設では 3 種類のカード（A、B、C）を取り扱っており、Simio Warehouse は 3 社のサプライヤから未包装カードをバルクパック容器で仕入れている。倉庫には、注文履行前にビデオカードのテストと包装を行うパッケージングラインを備えている。

このケーススタディでは、倉庫業務の Simio モデルを開発し、そのモデルを用いて運用戦略のパラメータとピッキング・パッキング作業に必要な人員数を決定することが課題となっている。

B.3　革新的なレンタカーサービス

空港における新しいスタートアップビジネスを調査している。このサービスは、旅客が空港駐車場に自家用車を預け、その車を他の旅客にレンタルすることで、駐車料金が無料になり、さらにレンタル収入も得られる仕組みである。以下が基本的なコンセプトである：

・事前登録した出発旅客は、自家用車をレンタカーとして提供することで、最長 3 週間まで駐車場を無料で利用できる。

・提供された車は、まず所有者負担で清掃される。レンタル終了時には無償で再度洗車されるため、貸借者と所有者の双方がきれいな車で空港を離れることができる。
・到着旅客は、スマートフォンアプリで利用可能な車を選択できる。
・レンタカーは車種と年式に基づいて 4 種類に分類され、種類ごとにレンタル価格が設定される。所有者にはレンタル収入の 50%が支払われる。

車の稼働率が高いため、実際の駐車スペース以上の顧客にサービスを提供することが可能である。

このビジネスモデルは、まず中規模空港で試験運用し、収益性を確認してから他の空港へ展開する計画である。潜在的な投資家からの支持を得るために、試験運用システムを最適な形で設計することが重要である。そのため、システムのレイアウトを含む重要な設計判断には、シミュレーションを活用する。

B.4　Simio 掘削ロジスティクス社

Simio 掘削ロジスティクス社（SDL）は、複数のオフショア船をチャーターし、オフショアの掘削拠点間で原材料を輸送している。現行システムでは、少数の船でオフショア拠点へのサービスを提供している。原材料不足による掘削拠点の遊休コストが極めて高いため、SDL は掘削効率の最大化を目指して船を調達する傾向にある。しかし、データによると港とオフショア拠点の双方で相当な待機時間が発生している。SDL は船舶のスケジューリングを改善し、業務効率化を通じてオフショア船の規模を適正化したいと考えている。

この課題ではまず、各掘削拠点専用の船舶サイズで現行システムをモデル化する。このモデルでは、さまざまな船舶サイズを考慮し、オフショア船による複数種類の貨物輸送をシミュレートする。システムの複雑性として、船舶への積載作業、輸送、荷降ろし時の気象条件や波高の影響を組み込む。この現行モデルは、システム改善戦略を評価するためのベースラインとなる。

このプロジェクトの次なる重要課題は、複数の代替戦略を構築し評価することである。たとえば、高水準のサービスを維持しながら総コストを削減する改善システムの構築に向けて、全拠点へのサービスを提供する共通船団の導入を検討したい。プロジェクトの評価は、シミュレーションモデルの品質と、提案された改善システムの総合的な効率性およびコストの両面から行われる。

B.5　Simio 救急治療センター

Simio 救急治療センター（UCCS）は、複数の救急治療センターを運営し、通常の診療時間に加え、夜間や週末も外来患者の治療を提供している。UCCS では、感染症、インフルエンザ、軽度のケガや骨折など、一次医療で一般的な症状を治療し、看護師、内科医、助手、整形外科専門医などの医療専門家を配置している。

施設内には受付、待合室、治療エリア、および各種検査・処置室を備えている。患者の需要は時間帯や曜日、さらにセンターごとに変動する。UCCS は最適なレ

イアウト、スタッフ配置、部屋の配置、その他の運営方法を決定したいと考えている。また、検査や軽度の症状に関する予約システムのインセンティブを強化することで、必要なスタッフ数の削減が可能かについても評価を行う予定である。

複数のセンターの最適化に同一モデルを活用するため、患者の動態、施設やスタッフの構成、最適化実験に関する設定を容易に修正できるデータ駆動型のシミュレーションモデルが必要とされている。提案は UCCS の経営陣向けに行われることから、3D アニメーションと結果の透明性が重要視される。

B.6　航空機の生産システム

ある航空機メーカーが、最終組立システムの計画的な変更について評価している。生産ラインには複数のワークステーションがあり、各ステーションは 10～20 の作業タスクを担当している。各タスクには、作業者、ツール、マテリアルハンドリングに関する固有の要件がある。システムはスケジュールに基づいて稼働し、ライン休止時までに完了していない作業は後工程へ送られない。この作業の搬送方針は、本プロジェクトの評価項目の一つとなっている。生産プロセスは急勾配の学習曲線を持つ労働集約型で、仕掛品が高額なため、労働生産性の向上と仕掛品の削減が重要課題となっている。

現在の生産プロセスでは 2 製品を製造しているが、同社は新製品の追加を計画している。第 3 の生産ラインの追加により、生産計画の詳細な評価が必要となっている。

同社は、現状の 2 本および将来の 2～3 本の生産ラインについて、最適な生産計画と新しいスループット率の評価を望んでいる。そのため、需要プロファイル、ライン割当戦略、リソース配分戦略、作業搬送方針など、さまざまな観点からの評価が必要とされている。

B.7　ラテンアメリカのサプライチェーン

ラテンアメリカを拠点とする小売店が、カリブ海地域と南米への事業拡大を計画している。同社は現在、アジアで製品を生産しているが、市場の需要は地域によって大きく異なり、かつ変動が激しいため、柔軟かつ迅速な対応が不可欠である。そのため同社は、複数の物流センター建設について検討を進めている。

地域別の物流センターへの投資か、アジアの工場からの直接輸送かを検討する必要がある。物流センターを設置する場合は、将来の事業拡大を見据えた最適な立地を選定しなければならない。この際、各国の市場成長率とともに、港湾からのアクセス性も重要な判断基準となる。

同社は、物流センターにおける SKU（最小管理単位）別の発注方針と戦略も評価したいと考えている。この評価では、工場の生産リードタイムと生産能力の制約に加え、港湾や船舶の制約も考慮が必要である。また、輸送費、在庫保管費、機会損失、発注処理費などのコストが、重要な意思決定要因となる。

B.8　製紙・パルプ産業における原材料供給

紙パルプ製造業のアメリカの業界団体は、木材の原材料調達における非効率性を認識している。各社が独立して操業しているため、材木運搬トラックは目的地の製材所に向かう途中で、他の製材所を頻繁に通り過ぎることになる。これにより、本来避けられるはずの輸送コストが発生している。業界団体は、欧州のコンソーシアムを参考に、木材の輸送を一元管理することで物流コストの最小化を目指している。

彼らは、製材所への木材輸送を管理する新しいコンソーシアムの設立を検討している。目標は輸送コストの最小化である。対象地域には 3 つの製材所が連続的に操業しており、各製材所は 1 日当たり 4～6 千トンの木材を消費する。製材所ごとに独自の需要と最大在庫量が設定されており、一部の製材所は離れた場所に「ドロップヤード」と呼ばれる追加の在庫スペースを保有している。木材は地域の小規模伐採業者によって供給され、各業者は季節により変動する独自の伐採能力と輸送能力を持っている。

コンソーシアムは、現行システムと提案システムの両方について、必要な投資額、最適な運用パラメータ、想定される費用削減額の評価を求めている。

B.9　グローバル通貨取引

世界の金融市場において主要な保管銀行は、両替サービスを提供している。銀行は各取引において顧客から徴収する手数料により収益を得ている。毎日数千件の予測不可能な取引が発生するため、銀行は各種通貨の現金を保有する必要がある。各営業日の終わりに、銀行は CLS 銀行（多通貨同時決済銀行）との間で決済を行い、各通貨の残高をリセットする。CLS はこの決済に手数料を課す。この仕組みにより、銀行のリスクは一日に限定される（日中流動性リスクとして知られている）。特定の通貨の残高が不足した場合、銀行は取引先とスワップ取引を行うことができる。これは CLS が提供するサービスと本質的に同じだが、営業時間中いつでも実行できる。スワップ取引はリスク軽減に役立つものの、CLS と比べてコストが高い。

銀行には法的にリスク管理が義務づけられている。顧客取引に対応できない事態が発生すれば、世界の金融市場に重大な混乱をもたらす可能性がある。そのため銀行は、さまざまなストレスシナリオにおいてもじゅうぶんな現金を保有していることを政府に証明しなければならない。各通貨の適切な保有残高を決定することは複雑な課題である。許容可能なリスク範囲内で収益を最大化するため、取引、スワップ、決済などのキャッシュフローをシミュレートする必要がある。このシミュレーションには、収益、コスト、リスクの分析を含め、複数のシナリオで実施する。

B.10　Sunrun のソーラーパネル設置

Sunrun は、米国最大の住宅用太陽光発電会社である。同社は住宅所有者のニーズに応え、設置に関するすべての工程を担当し、ソーラーシステムの調整を行う。

Sunrun は 21 州で事業を展開し、年々成長を続けている。

このプロジェクトでは、ソーラーパネル設置における「ラストワンマイル」の複雑さについて検討する。このプロセスには、機器の搬入、購入者とその家族への説明、屋根へのソーラーパネルの設置が含まれる。現在、標準的な設置プロセスを見直し、予期せぬ遅延や工期の延長に対してより柔軟に対応できる新しい配送プロセスへの移行が提案されている。シニアマネージャは、現行プロセスの評価、両シナリオの比較、および設置プロセス改善のための提言を求めている。

B.11　種子生産

種子生産プロセスには、特定の交配種の畑への植え付け、栽培期間中の管理、収穫、そして生産施設での種子処理など、多様な工程が含まれる。北米での収穫期間は 8 月中旬から 10 月上旬までの 8～10 週間である。収穫後、種子の処理と包装のための専用エリアが生産用地として確保される。種子生産施設は 9 月中旬から翌年 4 月まで稼働する。

複数の種子生産施設では、それぞれ異なる製品ミックスを扱っているため、各施設の生産計画はさまざまな制約を考慮して立てる必要がある。適切な種子を適切な時期に顧客へ届けるため、製品ポートフォリオ、複雑な部品表、施設内での製品の経路、物理的なスペース、マテリアルハンドリングシステム、設備の構成と効率性、そして労働力などの制約を考慮しなければならない。

B.12　メガストア物流センター

S.I.M.I.O.（統合オペレーションによる超集中型メガストア）は、最新のテクノロジーと高度な分析を活用してコストを最小化している。これは同社の物流センター運営の高度な洗練性に表れている。意思決定には常にリアルタイムデータが活用され、検証、最適化、自動化が日常業務に組み込まれている。しかし、オペレーションの複雑さゆえに、新規投資の効果を予測することは容易ではない。特に、人件費削減の観点から、注文の優先順位づけと配車をどのように最適化すべきかが課題となっている。このようなシステム性能に関する課題に対しては、詳細なデジタルツインを構築し、活動をシミュレートして結果を定量的に分析することがもっとも効果的である。

労働力はコストの主要な要因であり、限られたリソースである。システムは収容（車両制約あり）、選択、補充という 3 つの作業領域に分類される。本課題では「収容」に焦点を当てる。時間当たりのパレット処理数、設備稼働率、コスト、移動距離などの指標を改善するため、さまざまな配車方式を評価することが主目的である。増加するインバウンド需要に対応するための追加設備投資を評価し、目標達成に向けた方策を検討する。このモデルは、パラメータの柔軟な調整を可能にするため、データ駆動型アプローチで実装する必要がある。

B.13　地方空港の改善計画

ある地方空港では、周辺地域の発展や経済成長に伴い、利用客が増加している。

この増加により、空港は頻繁に混雑に見舞われ、地域の経済成長を妨げる可能性が出てきた。そこで空港では、混雑緩和と乗客の利便性向上を目指した改善プロジェクトの評価を進めている。

空港内の移動時間（到着、チェックイン、手荷物預け入れ、セキュリティ、ゲート到着）に焦点を当て、混雑緩和策を検討している。滞在時間短縮のための選択肢は複数あるものの、コストと滞在時間のばらつきを考慮すると、最適な方法を特定することは難しい。現在の改善案には、チェックインとセキュリティの機能向上、ターミナル内の乗客運搬車の増設などが含まれる。空港は、これらの選択肢を評価し、もっとも費用対効果の高い方法で旅客の滞在時間を短縮する必要がある。

空港内の混雑に加え、ターミナルと周辺道路の交通渋滞も課題となっている。交通局は混雑緩和策として新規バス路線の追加を提案し、さらにバス利用促進のため、バス利用者専用の保安検査場の設置も検討している。ただし、空港はこれらの提案実施による具体的な影響を把握できていない。

現在の空港運営のモデルを開発し、改善案の影響を比較するためのベースラインを確立する。そして、提案内容を評価し（必要に応じて新たな提案も行い）、もっとも費用対効果の高い運用のための包括的な提言をまとめなさい。

B.14　ミーバック配送センター

ミーバック配送センター（MDC）は、フットウェアやアパレル商品を保管し顧客に配送する倉庫である。注文は、消費者による単品購入（シングル）から複数アイテムの注文まで多岐にわたる。MDC では、注文数の増加とともに、注文の種類も多様化している。倉庫のプランナーやマネージャは、顧客配送の目標を達成するため、ピッキング作業の戦略改善を目指している。また、ピッキングの下流工程で働くチームも考慮に入れ、倉庫システム全体の作業負荷を平準化する方法を決定する必要がある。このプロジェクトは、商品棚からのピッキングから、注文品の梱包・出荷までを対象としている。

注文増加に対応するため、MDC は保管場所からの商品回収にウェーブピッキング管理手法（ウェービング）を導入している。このウェービングでは、注文をグループ化し、そのユニットを倉庫内の保管ゾーンごとに分類する。作業員が各ゾーンでピッキングを行い、回収した商品をステージングエリアへ移動させる。全商品がステージングエリアに集まったら、梱包・出荷工程へと進む。最終的に、ウェーブは再び個別の注文単位に整理され出荷される。

MDC は、労働力の最適活用と顧客納品目標の確実な達成を目指し、最適なウェービングとピッキング戦略の策定を進めている。ウェーブピッキング戦略では、注文の特性、商品の保管位置、使用可能なピッキング容器に基づいて理想的なウェーブサイズを設定する必要がある。また、同一ウェーブへの注文グループ化基準も検討が必要だ。さらに、MDC はウェービングの効率を高めるための倉庫レイアウトの見直しも検討している。これらの取り組みを通じ、今後のウェーブ計画に活用できる実用的なルール作りを目指している。

B.15　小規模レストランの運営

Simio BBQ Smoke Pit は、地元で注目を集める新進気鋭のレストランである。BBQ スモークミートと定番のサイドメニューを提供する、賑やかなダウンタウンの古い建物を改装したテイクアウト専門店だ。しかし、人気の高まりとともに、商品の欠品や人手不足により、顧客の待ち時間が長期化している。このため、需要に対応すべく、サービスのボトルネックを解消する新たな運営方針の策定が必要となっている。

バーベキュー肉の燻製工程とサイドメニューの調理には長時間を要するため、調理済み食品の在庫管理が課題となっている。在庫不足は顧客満足度を低下させ、頻繁な品切れは待ち時間の増加を招き、顧客離れと機会損失につながる。一方、過剰在庫はコスト増加の要因となる。特に高価な肉は、余った場合その日のうちに廃棄しなければならない。

当面の課題は、利益と顧客満足度を最大化するため、スタッフ配置と調理量の最適なバランスを見出すことである。そのためレストランは、顧客の来店・注文パターン、必要なリソース、調理能力の詳細な分析を計画している。Simio BBQ Smoke Pit は、各メニュー項目に対する効果的な補充戦略の確立を目指している。この戦略には、調理のタイミングと量の決定に加え、調理場と保管設備間での中間在庫の効率的な配分も含まれる。また、人員体制の見直しや設備投資による改善策について、その投資対効果の検証も行う予定である。

B.16　DDMRP によるサプライチェーン計画

Simio Shelving Shop は、デザインバリエーション豊富なシェルビングユニットを専門に製造・卸売する企業である。現在、注文の充足率が低下しており、収益に影響が出ている。そこで経営陣は外部コンサルタントを起用し、在庫とバッファリングシステムの見直しを依頼した。コンサルタントは、在庫切れ問題への対策として、需要主導型資材所要量計画（DDMRP）という新しい在庫管理手法を提案した。

現在、工場では静的なバッファレベルを採用しており、各工程に一定量の原材料を確保し、最終製品も一定数を保持している。しかし、この方式では工場とサプライヤ全体のリードタイムや品質のばらつきにより、完成品と原材料の在庫切れが発生している。経営陣は、DDMRP バッファリングソリューションの導入を決定した。このダイナミックバッファリング方式により、主要業績評価指標（KPI）、特に充足率と平均在庫コストの改善が期待できる。さらに、DDMRP バッファプロファイルは、経営陣が求めるバッファレベルのリスク監視機能も提供する。

現在の課題は、工場への動的 DDMRP バッファの導入である。まず営業、品質、機械、サプライヤの既存データを活用してバッファの配置を決定し、その後パラメータを調整して KPI の向上を図る。バッファ導入後は、工場の KPI 変化を予測する必要がある。また経営陣は、リードタイムと製品品質にばらつきがある潜在的サプライヤの再評価も計画している。この評価では、各サプライヤに最適なバッファを設定し、充足率と平均在庫コストへの影響を分析する。

B.17　診療所と救急医療センター

Simio 診療所・救急医療センター（SCUC）は、健康診断、軽度の外傷治療、臨床検査、整形外科ケアなど、多岐にわたる医療サービスを提供する医療機関である。施設では、予約なしでの治療を必要とする患者と、予約制のプライマリケアを必要とする患者の両方を受け入れている。予約患者は事前の約束で来院する一方、予約外の患者は医療が必要な際に随時来院する。これらの予約外の患者は緊急治療を必要としないが、診察までに数日から数週間待つことはできない状況にある。

SCUC 運営部は新施設への拡張を計画しており、センターの建設と開設準備を支援するチームを募集している。運営部は、新施設での従業員採用に向けた適切な人員配置とスケジュールの決定を目指している。また、診察室の数と配置に関する指針も必要としている。これらを決定するため、チームは過去の活動記録から患者の来院パターン、治療時間、患者層を分析する必要がある。

SCUC は、患者満足度と運営コストのバランスを評価する人員配置モデルの提案を求めている。患者満足度は、施設での待ち時間や未治療での帰宅者数などで測定される。運営部としては、適切な患者満足度を維持しながら、過剰な人員配置や診察室の空室によるコスト増加を避けたい。また、新規医療機器への投資も検討しているが、導入前にそれらが主要指標に与える影響を見極めたい。

B.18　洋上補給物流

Simio Supply Logistics（SSL）は、さまざまな洋上掘削場所との間で船舶をチャーターする組織である。SSL は月単位でレンタル可能な多様な船舶から選択できる。プラットフォーム補給船団（PSV）は港から選定・派遣される。これらの船舶は複数の深海掘削装置にサービスを提供している。現在のシステムは、高コスト、未充足の需要、過剰な待機時間により精査の対象となっている。SSL の経営陣は、チームに対して船舶のスケジューリングを改善し、発生コストを最小限に抑えながら掘削会社の要求をよりよく満たす方法を見出すことを望んでいる。

SSL は現在の船舶運用の分析を要請している。データ管理チームが収集した過去のデータを用いて、現行システムの性能と貨物需要の充足状況をモデル化する。チームは、船舶の積載・派遣計画を立てる際、異なる船舶タイプの仕様と貨物の容量制限を考慮する必要がある。さらに、掘削装置が位置する北海は、航行する船舶にとって複雑な海域となり得る。波高や一般的な気象条件などの要因が、各船舶の港から掘削装置への貨物輸送能力、そして掘削装置での荷降ろし能力に大きく影響する。現行システムのモデル化により、将来の改善戦略を評価するための基準が得られる。

経営陣は、チームに代替的なビジネス戦略と手順の開発・評価を要請している。新戦略では、生産性と稼働率を向上させつつ、運用コストの全体的な削減を実現する必要がある。

B.19　サッカースタジアム運営

Simio サッカー委員会（SSC）は、Estadio Simio で開催される親善試合におけ

るファン体験の向上を支援するために新たに設立された組織である。会場の運営チームは現在、収益の低さ、過度な待ち時間、ファンの満足度の低さにより精査の対象となっている。委員会は、スタジアム運営の現状を理解し、今後の改善に向けた提言を求めるためにシミュレーションチームを招聘した。

SSC は Estadio Simio で最近開催された試合のデータを収集し、問題の発生状況とその原因を理解するためにこの情報の活用を要請している。チームは、チケット販売、警備、売店、グッズ販売、座席選択などの場面でのファンとの接点を検証する必要がある。これらの領域は、ファンの調査に基づいて運営が不適切であるとの指摘を頻繁に受けている。不適切な人員配置、時代遅れで最小限の設備、待ち行列制御方針の欠如といった要因が、スタジアムの効果的な運営能力に大きく影響している。また、試合のスコア、天候条件、試合中の時間帯といった要因もファンの行動に大きな影響を与える。収益、待ち時間、ファンの満足度が現状まで低下した理由を理解する必要がある。

SSC は、チームに新しい試合日運営戦略の策定を依頼している。この戦略は、スタジアム内のさまざまな場所での待ち時間を短縮しながら、会場の収益性を向上させるものでなければならない。

B.20　外国為替取引

BankOfSimio は、顧客向けに外国為替取引サービスを提供するグローバルカストディ銀行である。たとえば、顧客が英国ポンド（GBP）で事業を行う英国企業の株式を購入したいが、米ドル（USD）で支払いたい場合、BankOfSimio は USD を受け取り GBP を提供することで、顧客が目的の株式を購入できるようにする。BankOfSimio は提供する各取引において手数料を徴収する。顧客は取引の遅延を避けるため、迅速な両替を期待している。そのため、BankOfSimio は取引対象となる各通貨の現金を手元に保持しておく必要がある。

営業日終了時に、BankOfSimio がある通貨で純増、別の通貨で純減となった場合、CLS（多通貨同時決済銀行）との決済を行う。CLS は少額の手数料で、BankOfSimio が余剰に持つ通貨を受け取り、不足している通貨を同等額で返却する。この決済により、BankOfSimio のリスクは 1 日以内（日中流動性リスク）に限定される。営業日終了前に特定の通貨が取引支援のために必要となった場合、取引相手行との日中スワップを実施できる。CLS と同様のサービスだが、より高額な手数料が発生し、取引に遅延が生じて顧客の不満につながる可能性がある。

グローバル金融における主要銀行として、BankOfSimio にはリスク管理が法的に義務づけられている。短時間でも顧客取引に対応できない場合、金融市場に混乱をもたらす可能性がある。そのため、政府は BankOfSimio に対して、さまざまなストレステストにおいて長期の遅延を回避できるじゅうぶんな資金を保有していることの証明を求めている。各通貨の保有量の決定は困難な課題である。学生には、許容可能なリスクの範囲内で利益を最大化する数量を見出すため、システム活動（取引、スワップ、決済からのキャッシュフローなど）のシミュレーションが求められる。

付録 C　本書の使い方

この付録では、フォーマットの意味、適切な Simio ソフトウェアの選び方、および本書で参照するファイルやリソースの入手方法について、ヒントを紹介する。

C.1　フォーマットガイドライン

本書では、用語の意味を明確に伝えるため、以下の書式を統一して使用する：

- 新しい用語を初めて紹介するときは、**太字**で表記し、その定義に注意を促す。例：「**SimBits** は小さくてわかりやすく編集されたモデルである」
- 重要な用語や単語は**太字**で強調し、読者の注意を喚起する。例：「カスタムロジックの指定には Simio **プロセス**を使用する」「オブジェクトはライブラリから**配置**し、Navigation ウィンドウで**定義・編集**する」
- 入力値や選択肢は別フォントで表示する。このフォントはファイル名の表示にも使用する。例：「Server Name プロパティには、`Server1` を入力する」「Ranking Rule には、`First in First Out` を選択する」。
- 要素名、ステップ名、ロケーション、ボタン、プロパティ名は、大文字と小文字を使い分けて表記し、特別なフォントは使用しない（例：Material、Process Type、Project Library、Facility Window）。

C.2　Simio ソフトウェア

本書は、Simio 17 以降向けに執筆した。より新しいバージョンのソフトウェアでは、細かな違いやモデリングのより簡単な方法が見つかる可能性があるが、基本的な概念は変わらない。

Simio Trial Mode（トライアル版）は、ライセンスのない Simio ソフトウェアのデフォルトモードだ。トライアル版は製品評価用の短期間（30 日間）のアクセスを提供するものである。Simio の基礎を独学する場合には適しているが、モデルの提出が必要な授業での使用は**推奨しない**。これは、ライセンス版の Simio ではトライアル版で保存したモデルを開けないためである。

Simio Academic Version（アカデミック版）が本書での使用に推奨するソフトウェアである。アカデミック版は、Simio の商用ソフトウェア（Simio RPS）と機能的に同等だが、非商用利用に限定される。すべてのアカデミック版には、OptQuest 最適化ソフトウェアも含まれる。Simio アカデミック版は毎年 7 月に更新し、原著第 7 版出版時点（2024 年 7 月）では、バージョン 17.261.39190 となっている。

Simio アカデミック版の無償利用を希望する教授/講師は、`https://www.simio.com/academic-licensing/` で詳細と申請手順を確認できる。無償ライセンスは、教員、ティーチングアシスタント、および学生用ソフトウェアラボでの使用を許可している。1 回の申請で、講師や学部が必要とするライセンス数を取得できる。

Simio Student Edition（学生版）ソフトウェアは、上述の Simio アカデミック版と機能的に同等だが、授業に登録している学生、または教員/講師の指導下で研

究やプロジェクトを行う学生個人での使用向けにライセンスしている。

Simio のアカデミック製品の詳細は、https://www.simio.com/academics/で確認できる。

C.2.1　Mac で Simio を使うには

Simio は Windows アプリケーションだが、多くのユーザが Mac 上で Simio を実行している。他の Windows 用アプリケーションと同様に、まず Mac 上で Windows を動作させる環境が必要だ。一般的な方法として、Parallels、Bootcamp、Oracle VM VirtualBox、VMware Fusion などがある。いずれの方法でも、Windows ライセンスと Mac にインストールするソフトウェアが必要になる。学校によっては Microsoft Windows の無料版や割引版を提供していることがあるので、詳細は学校の管理者に確認してほしい。それ以外の場合は、Windows を購入すれば、Windows のウェブサイトから Windows 10 の ISO ファイルをダウンロードできる。

これらのプログラムにはそれぞれ特徴があり、学校を通じて、または学生割引で利用できる他のオプションもあるかもしれない。すべての選択肢を検討して、自分に最適な方法を見つけることをお勧めする。

Simio のインストールと操作に関する詳しい手順は、こちら（https://cdn.simio.com/SimioLicenses/MacInstructions.pdf）を参照してほしい。

C.3　本書のファイルとリソース

C.3.1　モデルとデータファイル

本書で参照されるスプレッドシート、サンプルモデル、データファイルは、以下の zip ファイルにまとめられている：https://textbook.simio.com/SASMAA7/files/SASMAA_StudentFiles_7e.zip

C.3.2　MMC 待ち行列解析

2 章で使用した MMC 待ち行列プログラムは、以下の URL からダウンロードできる：https://textbook.simio.com/SASMAA7/files/MMC_calculator.zip

mmc.exe を実行するには、まず Microsoft Windows のコマンドプロンプトを起動する。次に mmc.exe ファイルをシステムのルートドライブ（通常は C:）に移動し、コマンドプロンプトで cd C:¥と入力する。コマンドプロンプトに C:¥>と表示されたら、mmc と入力すると、プログラムのパラメータの構文が表示される。

C.3.3　Stat::Fit 入力解析

Stat::Fit は、分布を適合させるための分析ソフトウェアだ。商用バージョンは https://www.geerms.com/で入手できる。同じ Web サイトでは、50 データポイントまで利用可能な無料の学生版も提供している。6 章では、100 データポイントまで利用できる教科書版 Stat::Fit を使用している。これは少し古いバージョンだが、より多くのデータを分析できる。教科書版は、https://textbook.simio.com/SASMAA7/files/StatFitTextbookVersion.zip からダウンロードできる。

教科書版 Stat::Fit を使用するには、上記の zip ファイルをダウンロードし、アクセス可能な任意のフォルダに解凍する。解凍後の statfit.exe をダブルクリックすれば起動する。より便利に使用するために、デスクトップにショートカットを作成できる。なお、教科書版には一般的なソフトウェアのようなインストーラは含まれていない。

C.3.4　@Risk ソフトウェア

@Risk ソフトウェアは本書の中核的な内容ではないが、3.2.3 項で概説している。現在、Lumivero は商用ソフトウェア@Risk Industrial の 15 日間トライアル版を https://palisade.lumivero.com/trials/risk/で提供している。このトライアル版は学生も利用できるが、利用期間は 15 日間に限られる。Lumivero のアカデミック向け製品についての詳細は https://palisade.lumivero.com/academic-offerings/を参照してほしい。

C.3.5　講師用の解答とその他のファイル

PowerPoint スライドと厳選された問題の解答は、認証済みの講師のみ利用できる。講師・教員がこれらのリソースへのアクセスを希望する場合は、academic@simio.com に申請してほしい。また、授業での活用に最適なケーススタディのウェビナーは https://www.simio.com/academic-case-studies/と https://www.simio.com/solution-series/で閲覧できる。

C.4　オンラインブックツールバーの使い方

原著の英語版にはオンラインブックが用意されている（https://textbook.simio.com/SASMAA7/）。HTML ドキュメントの上部に表示されるツールバーは、GitBook が提供するツールキットの一部で、**GitBook ツールバー**と呼ばれている（図 C.1）。見落としがちだが、便利な機能を多数備えている。

図 C.1　GitBook ツールバー

GitBook ツールバーの一番左のボタンで、サイドバー（目次など）の表示/非表示を切り替える。キーボードの s キーでも同様の操作が可能だ。GitBook は設定を記憶するため、サイドバーを閉じた状態で終了すると、次回も閉じたままで起動する。これは検索キーワードやフォント設定なども同様だ。

2 番目のボタンは検索機能だ。ブラウザの検索機能とは異なり、現在のページだけでなく**本全体**を検索できる点が特徴である。キーボードショートカットは f（Find）である。クリックするとサイドバー上部に検索ボックスが現れ、入力したキーワードに一致するセクションのみが表示される。検索結果のハイライトには少々時間がかかることがあるが、矢印キーで前後の検索結果に移動できる。検索ボタンを再度クリックするか、検索ボックス外で f キーを押すと、検索がクリアされる。

3 番目のボタンではフォントとテーマを設定できる。フォントサイズ、書体（明朝体・ゴシック体）、テーマ（ホワイト・セピア・ナイト）を、読みやすさに応じて調整できる。

4 番目のボタンでは、オンライン教科書の PDF 版をダウンロードでき、必要に応じて印刷も可能である。

5 番目の情報（i）ボタンでは、ドキュメント内の移動に使えるキーボードショートカット一覧を確認できる。

ツールバー右側には、X（旧 Twitter）や Facebook、LinkedIn などの SNS で章のリンクを共有するためのボタンがある。本書の内容は、役立つと思われる方々と自由に共有できる。なお、Simio がユーザのソーシャルメディアアカウントや連絡先情報を収集することはない。

参考文献

[1] S.C. Albright, W.L. Winston, and C.J. Zappe. Data Analysis and Decision Making With Microsoft Excel. South-Western Cengage Learning, Mason, Ohio, revised third edition, 2009.

[2] R.G. Askin and C.R. Standridge. Modeling and Analysis of Manufacturing Systems. Wiley, New York, 1993.

[3] J. Banks, J.S. Carson II, B.L. Nelson, and D.M. Nicol. Discrete-Event System Simulation. Pearson Prentice Hall, Upper Saddle River, New Jersey, fourth edition, 2005.

[4] B. Biller and B.L. Nelson. Fitting time series input processes for simulation. Operations Research, 53:549-559, 2005.

[5] G.M. Birtwistle, O.-J. Dahl, B. Myhraug, and K. Nygaard. Simula BEGIN. Auerbach, Philadelphia, Pennsylvania, 1973.

[6] J. Boesel, B.L. Nelson, and S. Kim. Using ranking and selection to "clean up" after simulation optimization. Operations Research, 51:814-825, 2003.

[7] P. Box G.E, G.M. Jenkins, and G.C. Reinsel. Time Series Analysis: Forecasting and Control. Prentice Hall, Englewood Cliffs, New Jersey, third edition, 1994.

[8] R. Conway, W. Maxwell, J.O. McClain, and L.J. Thomas. The role of work-in-process inventory in serial production lines. Operations Research, 35(2):229-241, 1988.

[9] L. Devroye. Non-Uniform Random Variate Generation. Springer-Verlag, New York, 1986.

[10] A.K. Erlang. The theory of probabilities and telephone conversation. Nyt Tidsskrift for Matematik, 20, 1909.

[11] A.K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. Elektrotkeknikeren, 13, 1917.

[12] M. Evans, N. Hastings, and B. Peacock. Statistical Distributions. Wiley, New York, second edition, 2000.

[13] Gartner. Gartner identifies the top 10 strategic technologies for 2010. Gartner Symposium/ITxpo, 2009.

[14] Gartner. Gartner identifies the top 10 strategic technology trends for 2013. Gartner Symposium/ITxpo, 2012.

[15] L.J. Gleser. Exact power of goodness-of-t tests of kolmogorov type for discontinuous distributions. Journal of the American Statistical Society, 80:954-958, 1985.

[16] F. Glover, J.P. Kelly, and M. Laguna. New advances for wedding optimization and simulation. Proceedings of the 1999 Winter Simulation Conference. 255-260, 1999.

[17] F. Glover and G.A. Kochenberger, editors. Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

[18] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, http://www.deeplearningbook.org, 2016.

[19] D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris. Fundamentals of Queueing Theory. Wiley, Hoboken, New Jersey, fourth edition, 2008.

[20] S. Harrod and W.D. Kelton. Numerical methods for realizing nonstationary poisson processes with piecewise-constant instantaneous-rate functions. Simulation: Transactions of The Society for Modeling and Simulation International, 82:147-157, 2006.

[21] W.J. Hopp and M.L. Spreaman. Factory Physics. McGraw Hill, New York, New York, 2008.

[22] ISA-95.com. Business to manufacturing markup language (B2MML). https://isa95.com/b2mml/, 2016.

[23] J.R. Jackson. Networks of waiting lines. Operations Research, 5:518-521, 1957.

[24] S. James and A. Duncan. Over 100 data and analytics predictions through 2028. Gartner Research, 2023.

[25] N.L. Johnson, A.W. Kemp, and S. Kotz. Univariate Discrete Distributions. Wiley, New York, third edition, 2005.

[26] N.L. Johnson, S. Kotz, and N. Balakrishnan. Continuous Univariate Distributions, volume 1. Wiley, New York, second edition, 1994.

[27] N.L. Johnson, S. Kotz, and N. Balakrishnan. Continuous Univariate Distributions, volume 2. Wiley, New York, second edition, 1995.

[28] W.D. Kelton. Implementing representations of uncertainty. In S.G. Henderson and B.L. Nelson, editors, Handbook in Operations Research and Management Science, Vol. 13: Simulation, 181-191. Elsevier North-Holland, Amsterdam, The Netherlands, 2006.

[29] W.D. Kelton. Representing and generating uncertainty effectively. Proceedings of the 2009 Winter Simulation Conference, 40-44, 2009.

[30] W.D. Kelton, R.P. Sadowski, and N.B. Zupick. Simulation With Arena. McGraw-Hill, New York, sixth edition, 2015.（高桑宗右ヱ門監訳、野村淳一訳（2007）『シミュレーション―Arena 活用した総合的アプローチ』コロナ社、第４版）

[31] S. Kim and B.L. Nelson. A fully sequential procedure for indifference-zone selection in simulation. ACM Transactions on Modeling and Computer Simulation, 11:251-273, 2001.

[32] P.J. Kiviat, R. Villanueva, and H.M. Markowitz. The SIMSCRIPT II Programming Language. Prentice Hall, Englwood Cliffs, New Jersey, 1969.

[33] L. Kleinrock. Queueing Systems: Volume I - Theory. Wiley, New York, 1975.

[34] M. Kuhl, S.G. Sumant, and J.R. Wilson. An automated multiresolution procedure for modeling complex arrival processes. INFORMS Journal on Computing, 18:3-18, 2006.

[35] A.M. Law. Simulation Modeling and Analysis. McGraw-Hill, New York, fifth edition, 2015.

[36] P. L'Ecuyer. Uniform random number generation. In S.G. Henderson and B.L. Nelson, editors, Handbook in Operations Research and Management Science, Vol. 13: Simulation, 55-81. Elsevier North-Holland, Amsterdam, The Netherlands, 2006.

[37] P. L'Ecuyer and R. Simard. Testu01: A c library for empirical testing of random number generators. ACM Transactions on Mathematical Software, 337: Article 22, 2007.

[38] L. Leemis. Nonparametric estimation of the intensity function for a nonhomogeneous poisson process. Management Science, 37:886-900, 1991.
[39] D.H. Lehmer. Mathematical methods in large-scale computing units. Annals of the Computing Laboratory of Harvard University, 26:141-146, 1951.
[40] D.V. Lindley. The theory of queues with a single server. Proceedings of the Cambridge Philosophical Society, 48:277-289, 1952.
[41] J.D.C. Little. A proof for the queuing formula l = λw. Operations Research, 9:383-387, 1961.
[42] J.D.C. Little. Little's law as viewed on its 50th anniversary. Operations Research, 59:536-549, 2011.
[43] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation, 8:3-30, 1998.
[44] MESA International. Business to manufacturing markup language (B2MML). http://www.mesa.org/en/B2MML.asp, 2018.
[45] L. Mueller. Norwood re-department simulation models: Present and future. Master's thesis, Department of Quantitative Analysis and Operations Management, University of Cincinnati, 2009.
[46] R.E. Nance and R.G. Sargent. Perspectives on the evolution of simulation. Operations Research, 50:161-172, 2002.
[47] B.L. Nelson. Stochastic Modeling, Analysis and Simulation. McGraw-Hill, New York, first edition, 1995.
[48] B.L Nelson. The more plot: Displaying measures of risk & error from simulation output. Proceedings of the 2008 Winter Simulation Conference, 413-416, 2008.
[49] B.L. Nelson. Personal communication. 2013.
[50] B.L. Nelson. Personal communication. 2016.
[51] C.D. Pegden. Deliver On Your Promise How Simulation-Based Scheduling Will Change Your Business. Simio LLC, Sewickley, PA, 2017.
[52] C.D. Pegden, R.E. Shannon, and R.P. Sadowski. Introduction to Simulation Using SIMAN. McGraw-Hill, New York, second edition, 1995.（高桑宗右ヱ門訳（1993）『生産システム・シミュレーション―アプローチと SIMAN』コロナ社）
[53] C.D Pegden and D.T. Sturrock. Rapid Modeling Solutions: Introduction to Simulation and Simio. Simio LLC, Pittsburgh, Pennsylvania, 2013.
[54] A.A.B. Pritsker. The GASP IV Simulation Language. Wiley, New York, 1974.
[55] A.A.B. Pritsker. Introduction to Simulation and SLAM II. Wiley, New York, fourth edition, 1995.
[56] D. Robb. Gartner taps predictive analytics as next big business intelligence trend. Enterprise Apps Today, 2012.
[57] S.M. Ross. Introduction to Probability Models. Academic Press, Burlington, Massachusetts, tenth edition, 2010.
[58] S.L. Savage. The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty. Wiley, Hoboken, New Jersey, 2009.
[59] T.J. Schriber. Simulation Using GPSS. Wiley, New York, 1974.
[60] T.J. Schriber. An Introduction to Simulation Using GPSS/H. Wiley, New York,

1991.

[61] K. Schwab. The Fourth Industrial Revolution. Random House USA Inc., 2017.（クラウス・シュワブ著、世界経済フォーラム訳（2016）『第四次産業革命―ダボス会議が予測する未来』日本経済新聞社）

[62] A.F. Seila, V. Ceric, and P. Tadikamalla. Applied Simulation Modeling. Thomson Brooks/Cole, Belmont, California, 2003.

[63] E. Song and B.L. Nelson. Quickly assessing contributions to input uncertainty. IIE Transactions, 47:1-17, 2015.

[64] E. Song and B.L. Nelson, and C.D. Pegden. Advanced tutorial: Input uncertainty quantification. Proceedings of the 2014 Winter Simulation Conference, 162-176, 2014.

[65] W.S. Stidham. A last word on l = λw. Operations Research, 22:417-421, 1974.

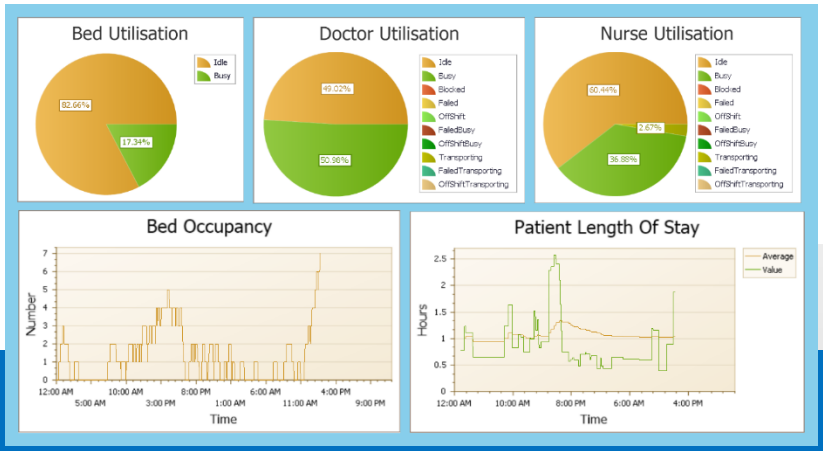
[66] D.T. Sturrock. New solutions for production dilemmas. Industrial Engineer Magazine, 44:47-52, 2012a.

[67] D.T. Sturrock. Simulationist bill of rights. Success in Simulation Blog, 2012b.

[68] J.J. Swain. Simulation: Necessary tool for problem solving. OR/MS Today, 2021

[69] J.W. Tukey. Exploratory Data Analysis. Addison-Wesley, Reading, Massachusetts, 1977.

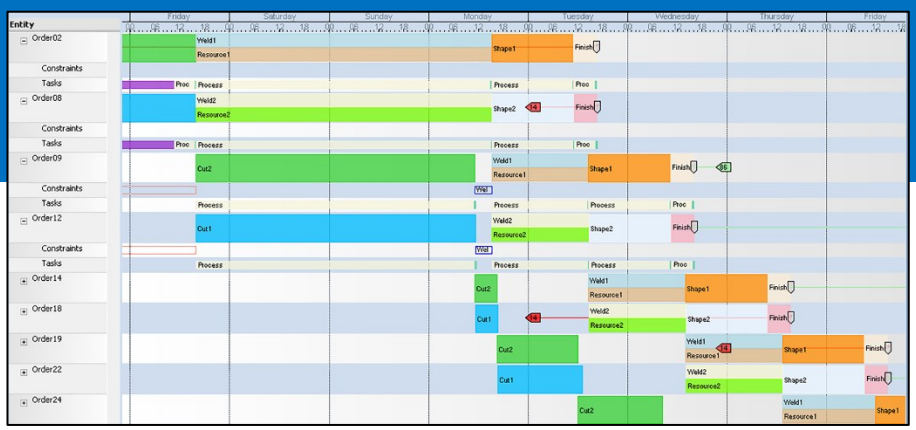
[70] H.M. Wagner. Principles of Operations Research, With Applications to Managerial Decisions. Prentice Hall, Englewood Cliffs, New Jersey, 1969.



Simio

Forward Thinking

www.simio.com



ISBN 978-1-938207-08-2