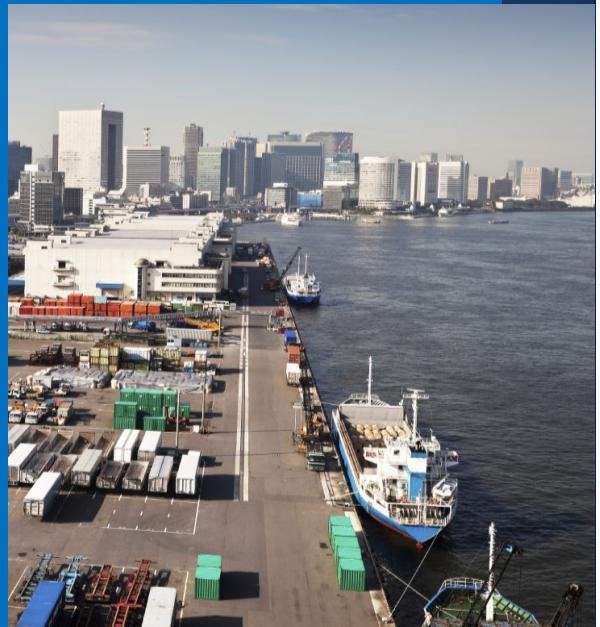


Simio と シミュレーション モデリング・解析・ 応用

第 6 版



**Jeffrey S. Smith
David T. Sturrock
Soemon Takakuwa**

Japanese Translation of:

**Simio and Simulation:
Modeling, Analysis,
Applications**

6th Edition



Simio とシミュレーション

—モデリング・解析・応用—

第 6 版

原著者

Jeffrey S. Smith
David. T. Sturrock
Soemon Takakuwa

訳

野村 淳一
三輪 冠奈
譚 奕飛
岳 理恵
楊 文賀

監修

高桑 宗右エ門

ISBN: 978-1-938207-05-1

第6版 まえがき

この第6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に、シミュレーションの用法について説明したものである。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る。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。近年のソフトウェアは、従前にもましていっそ利用しやすくなっている。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ある。

第6版は、Simio Version 14以降を用いて書かれている。第5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る。特に、第6版については、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た。また、多くの機能が追加されており、読者の皆さんからのご意見なども考慮されている。Monte Carlo、入力解析、出力解析の内容や、データ駆動型、データ生成モデリング手法といった新たな内容を取り上げた。そして、シミュレーションをサポートする AI およびニューラルネットワークに関する節を追加した。最後に、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、シミュレーションがデジタルツインやオペレーションスケジューリング・統制の生成や効果的なオペレーションに如何に寄与するのかについても説明を加えた。また、章末問題は、読者の意見なども反映させ拡充した。そして、全体の構成を再編成し、最近追加された新しい Simio の機能の多くについて、内容を更新した。

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る。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る。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かつ理解しやすいように工夫した。

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。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だろう。

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るだろう。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だろう。さらに、付録AおよびBの実社会のシミュレーション応用事例から、講義内容に応じて紹介されるとよい。

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るだろう。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とインダストリー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る。また、付録Aの内容は、応用研究や個別の研究課題への学生への課題として用いるのも良いだろう。

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していることを前提として執筆した。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いる。読者は、確率や統計学の専門家でなくとも良いが、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る。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た。

謝辞

ここに至るまで、多くのかたがたの助力を得た。初版の共著者である Dr. Alexander Verbraeck には、多大な貢献をしていただいた。Dr. C. Dennis Pegden は特にスケジューリングの章に重要な寄与をされた。The Simio LLC の技術スタッフである、Cory Crooks、Glenn Drake、Glen Wirth、Dave Takus、Renee Thiesing、Katie Prochaska、Christine Watson の各氏からは、Simio の機能を理解し最善の説明をするための助言を受け、また原稿のチェックをしていただいた。
(中略) ここに深甚なる謝意を表したい。

Jeffrey S. Smith
Technical Fellow, Simio LLC

David. T. Sturrock
Co-founder and Technical Fellow, Simio LLC

第6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カンファレンス（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6版の原稿準備には、第5版に引き続き、野村淳一（第1、2、3、6、10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新進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と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、隨時打合せや確認をしながらようやく刊行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。

2022年3月
高桑 宗右エ門

目次

第1章 シミュレーションとは.....	1
1.1 本書について	1
1.2 システムとモデル.....	3
1.3 ランダム性とシミュレーションプロセス.....	5
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	5
1.3.2 シミュレーションプロセス.....	7
1.4 シミュレーションを適用する場合（適用しない場合）	8
1.5 シミュレーションの成功に必要なスキル.....	9
1.5.1 プロジェクトの目的.....	10
1.5.2 仕様書	11
1.5.3 プロジェクトの反復.....	13
1.5.4 プロジェクト管理とアシリティ	14
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	15
第2章 待ち行列理論の基礎.....	19
2.1 待ち行列システムの構造と用語.....	20
2.2 リトルの法則とその他の関係式.....	23
2.3 複数窓口待ち行列ステーションの特性.....	24
2.3.1 $M/M/1$	24
2.3.2 $M/M/c$	24
2.3.3 $M/G/1$	25
2.3.4 $G/M/1$	26
2.4 待ち行列ネットワーク	28
2.5 待ち行列理論とシミュレーションの比較.....	30
2.6 問題.....	31
第3章 シミュレーションの種類.....	33
3.1 シミュレーションの分類.....	33
3.1.1 静的モデルと動的モデル.....	33
3.1.2 連続型動的モデルと離散型動的モデル.....	34
3.1.3 確定的モデルと確率的モデル.....	35
3.2 静的かつ確率的なモンテカルロ	36
3.2.1 Model 3-1：2つのサイコロ	36
3.2.2 Model 3-2：モンテカルロ積分	38
3.2.3 Model 3-3：単一期間在庫の収益	40
3.2.4 Model 3-4：新製品決定モデル	44
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	47
3.3.1 Model 3-5：マニュアル動的シミュレーション	47
3.3.2 Model 3-6：単一窓口待ち行列の待ち時間	53
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	56
3.4.1 汎用プログラミング言語.....	56
3.4.2 専用シミュレーションソフトウェア	57
3.5 問題.....	58

第4章 初めての Simio モデル	63
4.1 基本的な Simio ユーザインターフェース	64
4.1.1 リボン	65
4.1.2 Suooprt リボン	65
4.1.3 Project Model タブ	65
4.1.4 オブジェクトライブラリ	65
4.1.5 プロパティウィンドウ	67
4.1.6 ナビゲーションウィンドウ	67
4.1.7 SimBits	67
4.1.8 ウィンドウとタブの移動・構成	68
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	69
4.2.1 モデルの構築	69
4.2.2 初期の実験と分析	75
4.2.3 反復と出力の統計分析	77
4.2.4 定常状態と終結状態のシミュレーション	81
4.2.5 モデルの検証	83
4.3 Model 4-2 : Processes を用いたモデル	84
4.4 Model 4-3 : 現金自動振込機(ATM)	89
4.5 平均値を超えて : Simio MORE(SMORE)プロット	93
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	98
4.7 インタラクティブ・ログとダッシュボードレポート	100
4.8 モデルアニメーションの基礎	102
4.9 モデルのデバッグ	106
4.9.1 気づきにくい問題の発見	107
4.9.2 デバッグプロセス	107
4.9.3 デバッグツール	108
4.10 まとめ	110
4.11 問題	110
第5章 Simio の中級モデリング	113
5.1 Simio のフレームワーク	113
5.1.1 オブジェクトの概要	113
5.1.2 プロパティと状態変数	116
5.1.3 トークンとエンティティ	121
5.1.4 プロセス	123
5.1.5 リソースとしてのオブジェクト	125
5.1.6 データ範囲	127
5.1.7 式ビルダ	127
5.1.8 費用計算	129
5.2 Model 5-1 : PCB 組立	130
5.3 Model 5-2 : PCB 組立の拡張	134
5.3.1 再加工ステーションの追加	134
5.3.2 リンクの重みに対する式の利用	138
5.3.3 リソーススケジュール	139
5.3.4 機械の故障	145
5.3.5 Model 5-2 の検証	146

5.4 Model 5-3：工程選択のある PCB モデル	148
5.5 Model 5-4：複数の代替シナリオの比較	153
5.6 まとめ	157
5.7 問題	158
第 6 章 入力解析	161
6.1 単一変量の入力確率分布の特定	162
6.1.1 一般的アプローチ	162
6.1.2 観測された実データを使用する方法	162
6.1.3 確率分布の選択	163
6.1.4 観測された実データへの分布適合	166
6.1.5 適合度検定の詳細	171
6.1.6 分布の適合に関する問題	176
6.2 入力のタイプ	181
6.2.1 確定的と確率的	181
6.2.2 スカラーと多変量と確率過程	182
6.2.3 時変到着率	183
6.3 乱数生成法	184
6.4 確率変量および確率過程の生成	187
6.5 Simio Input Parameters の利用	190
6.5.1 応答感度	193
6.5.2 サンプルサイズ誤差推定	196
6.6 問題	197
第 7 章 モデルのデータ操作	199
7.1 データテーブル	199
7.1.1 テーブルの基本	199
7.1.2 Model 7-1：データテーブルを用いた ED モデル	200
7.1.3 シーケンステーブル	205
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	206
7.1.5 到着テーブルと Model 7-3	210
7.1.6 リレーションナルテーブル	212
7.1.7 テーブルのインポートとエクスポート	213
7.2 スケジュール	215
7.2.1 カレンダ作業スケジュール	216
7.2.2 手動スケジュール	217
7.3 レートテーブルと Model 7-4	218
7.4 Lookup テーブルと Model 7-5	220
7.5 リストとチェンジオーバー	222
7.6 状態変数配列	224
7.7 データ駆動型モデル	224
7.7.1 テーブルとリピートグループ	225
7.8 データ生成モデル	227
7.9 まとめ	229
7.10 問題	229

第8章 アニメーションとエンティティ移動	233
8.1 アニメーション	233
8.1.1 なぜアニメートするのか?	233
8.1.2 ナビゲーションと View のオプション	234
8.1.3 Drawing リボンによる背景アニメーション	236
8.1.4 Animation リボンの状態変数アニメーション	238
8.1.5 Symbols リボンによるシンボルの編集	240
8.1.6 よりリアルなエンティティアニメーション	242
8.1.7 Model 8-1: PCB 組立のアニメート	244
8.2 エンティティ移動	248
8.2.1 フリースペースを通るエンティティ移動	248
8.2.2 コネクタ、タイムパスおよびパスの利用	250
8.2.3 コンベアの利用	252
8.2.4 Model 8-2: コンベアがある PCB 組立	253
8.2.5 Worker の利用	255
8.2.6 Vehicle の利用	258
8.2.7 Model 8-3: 病院の職員に関する ED の拡張	258
8.3 まとめ	265
8.4 問題	265
第9章 Simio の高度なモデリング	267
9.1 Model 9-1: ED モデルの再考	267
9.1.1 OptQuest を用いた Model 9-1 の最適なリソースレベルの探索	276
9.1.2 サブセット選択と KN を用いた Model 9-1 における代替シナリオの ランキングと選択	283
9.2 Model 9-2: ピザのテイクアウトモデル	287
9.2.1 Model 9-2 の実験	295
9.3 Model 9-3: 固定容量バッファ	298
9.4 まとめ	303
9.5 問題	303
第10章 その他のモデリングトピック	307
10.1 Search ステップ	307
10.1.1 Model 10-1: ステーションのエンティティを検索・取り出す	307
10.1.2 Model 10-2: バッチ内の総処理時間の合計	309
10.2 Model 10-3: 退去と放棄	310
10.3 タスクシーケンス	311
10.4 イベントに基づく決定ロジック	314
10.5 その他のライブラリとリソース	317
10.5.1 Flow ライブラリ	317
10.5.2 Extras ライブラリ	318
10.5.3 Shared Items フォーラム	318
10.6 実験	319
10.6.1 並列処理	319
10.6.2 クラウド処理	320
10.7 AI とニューラルネットワーク	320

10.7.1 ニューラルネットワーク	321
10.8 まとめ	323
10.9 問題	323

第11章 カスタマイズと Simio の拡張..... 325

11.1 オブジェクト定義に関する基本的なコンセプト	325
11.1.1 モデルのロジック	326
11.1.2 外部ビュー	327
11.1.3 オブジェクト定義のサブクラス化	328
11.1.4 プロパティ、状態変数、イベント	330
11.2 Model 11-1：階層的オブジェクトの構築	330
11.2.1 モデルロジック	331
11.2.2 外部ビュー	332
11.3 Model 11-2：ベースオブジェクトの作成	334
11.3.1 モデルロジック	334
11.3.2 外部ビュー	337
11.4 Model 11-3：オブジェクトのサブクラス化	338
11.4.1 モデルロジック	338
11.4.2 外部ビュー	340
11.5 ユーザ拡張機能の利用	341
11.5.1 ユーザ拡張機能の作成・展開の手順	342
11.6 まとめ	343
11.7 問題	344

第12章 インダストリー4.0における

シミュレーションによるスケジューリング .. 347

12.1 時代を超えた産業革命	347
12.1.1 第1次産業革命：機械生産	348
12.1.2 第2次産業革命：大量生産	348
12.1.3 第3次産業革命：デジタル時代	348
12.2 第4次産業革命：スマートファクトリー	348
12.2.1 次は？	349
12.3 デジタルツインの必要性	350
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	351
12.5 シミュレーションによるスケジューリングの役割	353
12.5.1 デジタルツインとしてのシミュレーション	354
12.6 計画とスケジューリングにおける未解決の問題	355
12.7 シミュレーションによるスケジューリング	357
12.8 リスクに基づく計画とスケジューリング	360
12.9 Simio RPS による計画とスケジューリング	361
12.10 スケジューリングのインターフェース	363
12.11 Model 12-1：スケジューリングへのモデルファーストアプローチ	365
12.11.1 簡単なスケジューリングモデルの構築	365
12.11.2 より現実的なモデルの作成	366
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	368
12.11.4 その他の評価ツール	369

12.12 Model 12-2: スケジューリングへのデータファーストアプローチ	370
12.12.1 データインポートのためのモデル設定.....	371
12.12.2 データインポート	371
12.12.3 モデルの実行と分析	372
12.13 追加情報と例	373
12.13.1 Simio E-bookによる計画とスケジューリング	373
12.13.2 スケジューリング例	374
12.14 まとめ	375
12.15 問題	375
付録 A Simio を活用したケーススタディ	377
A.1 機械加工検査工程	377
A.1.1 問題記述	377
A.1.2 サンプルモデルと結果	379
A.2 遊園地	386
A.2.1 問題記述	386
A.2.2 サンプルモデルと結果	389
A.3 小さなレストラン	391
A.3.1 問題記述	391
A.4 小さな銀行支店	393
A.5 Vacation 市の空港	396
A.6 シンプルな最良の病院	400
付録 B Simio 学生コンペティションの問題	403
B.1 革新的なレンタカーサービス	403
B.2 Simio 掘削ロジスティクス社	404
B.3 Simio 救急治療センター	404
B.4 航空機生産の問題	404
B.5 ラテンアメリカのサプライチェーン	405
B.6 紙パルプ製造業の供給	405
B.7 グローバルな通貨取引	405
B.8 Sunrun ソーラーパネルの設置	406
B.9 種子生産	406
B.10 メガストア流通センター	406
B.11 地方空港の計画	407
B.12 ミーバック配送センター	407
B.13 小規模レストランの運営	407
B.14 DDMRP を用いたサプライチェーン計画	408
付録 C 本書の使い方	409
C.1 フォーマットガイドライン	409
C.2 Simio ソフトウェア	409
C.2.1 Mac で Simio を使うには	410
C.3 本書のファイルとリソース	410
C.3.1 モデルとデータファイル	410
C.3.2 MMC 待ち行列解析	410

C.3.3 Stat::Fit 入力解析	410
C.3.4 @Risk ソフトウェア	411
C.3.5 解答とその他のファイル（教員用）	411
参考文献	413

第Ⅰ章 シミュレーションとは

過去 40 年にわたってシミュレーションが利用されてきたが、ピークを過ぎたわけではない。むしろ、これから最盛期を迎えるようとしている。業界最大規模の ICT アドバイザリ企業であるガートナー (www.gartner.com) は最近の分析で、2010 年の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 位に挙げた (Gartner 2009)。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再び強調している：

分析は「ビジネスのエンジン」であるため、組織は時間的に厳しい場合であっても、ビジネス・インテリジェンスに投資している。ガートナーは、ビジネス・インテリジェンスの次の大きな段階として、より情報に基づいた意思決定を提供するために、さらにシミュレーションや推定に傾倒していくと予測している。

パフォーマンスの向上と低価格化によって、IT リーダーはビジネスのあらゆる行動について、分析とシミュレーションを実行できるようになりました。クラウド・ベースのアナリティクス・エンジンおよびビッグ・データ・リポジトリとモバイル・クライアントをリンクさせることによって、あらゆる場面で、いつでも最適化とシミュレーションを実行することが可能になりました。この新しいステップは、ビジネス・プロセスにおける行動のあらゆるタイミングと場所で、いっそう柔軟な意思決定を強力に支援するために、シミュレーション、予測、最適化および他のアナリティクスを提供します。

ハードウェアおよびソフトウェアにおけるシミュレーション関連の技術は、ここ 10 年で飛躍的に進展した。PC の高性能化により、数年前と比べても当初考えられなかつたほど高速に計算ができるようになっている。ユーザインターフェースと製品設計の進歩により、シミュレーションを効率的に活用するため必要な専門知識のハードルが下がり、大幅に簡潔な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ブレイクスルーにより、モデリングの柔軟性が大幅に改善され、非常に複雑なシステムの正確なモデリングが可能となった。ハードウェアやソフトウェア、そして誰もが利用できるシンボルを使うことで、初心者でも、あらゆる経験を持つ人達の円滑なコミュニケーションをサポートする、魅力的な 3D アニメーションを利用したシミュレーションを作成することができる。これらの技術革新などにより、シミュレーションが重要な技術としての地位に推し上げられた。

本書では、一般的なシミュレーション技術の基礎と成功するためのスキルだけでなく、最先端のシミュレーションパッケージに対する実用的な解説を行う。これにより、読者がシミュレーションの世界への扉を開くことを願っている。

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、ソフトウェア固有の概念によらない、基本的な技術を理解することから始める。Ⅰ章「シミュレーションとは」では、典型的なシミュレーションの応用例や、シミュレーションの適用を適切に判断する方法、およびシミュレーションプロ

2 第1章 シミュレーションとは

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やその長所・短所を紹介する。また、特にシミュレーションモデリングの要素を検証するために待ち行列理論をどのように用いるかを解説する。3章「シミュレーションの種類」では、シミュレーションの技術的側面と用語をいくつか紹介する。いくつかの観点からさまざまな種類のシミュレーションを分類し、例題を紐解くことにより解説する。

次に、Simio を使った多数の例を示しながら、より詳細なシミュレーションの概念を解説する。このような概念を解説するには、章ごとに技術的な要素（たとえば検証や出力分析など）を分ける方法もあるだろう。しかし、本書では、それぞれの例を小規模のプロジェクトとして捉え、各々の例でより多くの概念を紹介する。このアプローチは、初期の段階から総合的な実践によりスキルを習得する最善の方法であると考えられ、各プロジェクトでこれらのスキルを強化することができる。

4章「初めての Simio モデル」では、Simio の簡単な解説から始め、それから Simio で単一サーバ待ち行列モデルを構築する。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解説する。基本的なモデル構築と分析のプロセス自体は Simio に固有のものではないが、実装手段として Simio に焦点を当てる。このプロセスには、モデリングのスキルだけでなく、シミュレーションの出力結果に対する統計解析や実験、モデルの検証も含まれる。同じモデルは、別のモデリング手法を解説するため、また「カーテンの後ろ」で行われている動作に対する大きな洞察を与るために、低レベルのツールを使用して再現される。そして、3つ目の ATM モデルに続き、さらに Simio の革新的な SMORE プロットを使用した出力分析を紹介する。最後に、モデルにはびこる厄介な「バグ」を発見し、追跡する方法を解説して、本章を終える。

5章「Simio の中級モデリング」の目標は、前章で提示した Simio における基本的なモデリングおよび分析の概念に対する理解を深めることであり、そのためにより現実的なシステムのモデルを構築し、実験することから始める。まず、Simio の挙動の仕方と一般的なフレームワークについて、少し詳しく議論する。その後、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や条件分岐および合流などの機能を追加する。これらのモデルを開発する際、新しい Simio の機能を隨時紹介する。また、複数の代替シナリオを比較するというさまざまな状況に共通した目標を考慮して、統計的なシミュレーション実験を計画し、分析する方法を再び検討する。この章を終える時点で、Simio を用いて、ある程度の複雑なシステムをモデル化し分析する方法について、しっかりと理解できているはずである。

本書のこの時点で、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いるため、ここからはモデルを動作させる入力分布と入力過程に関する話題を取り扱う。6章「入力解析」では、シミュレーションに対する入力の種類の違いや、実世界で観測されたデータをシミュレーションプロジェクトに有用なデータに変換する方法、そしてほとんどのシミュレーションに必要となる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、視野を広げて、現実のシステムを表現するために必要とされる多くのデータの種類を検討する。ここでは、簡単な救命救急(ED)モデルを構築し、モデルの入力データ要件を満たすために Simio のデータテーブル構造を活用する方法を紹介する。続いて、シーケンステーブル、リレーションナルデータテーブル、到着テーブルの各概念、そしてデータテーブルのインポートおよびエクスポートを説明するために、モデルをより詳細に修正する。さらに、ED モデルを拡張し、ワークスケジュール、レートテーブル、および関数テーブルを解説する。最後に、リスト、配列、転換行列を簡単に紹介して、本章を終える。この章を終ると、モデルで頻繁に用いられるデータの種類を使いこなすことができるだろう。

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に 2D および

3D アニメーションを追加することにより、検証やコミュニケーションがさらに深まり、そして信頼性も高まる点について述べる。それから、背景アニメーション、カスタムシンボル、およびステータスオブジェクトなどの、さまざまなアニメーションツールを解説する。すでに作成したエレクトロニクス組立モデルを使って、新しいアニメーションのスキルを練習するだけでなく、さまざまな種類のリンクを解説し、作業の流れを処理するためのコンベヤを追加する。最後に、エンティティの移動を充実させる Simio の Vehicle および Worker オブジェクトを導入して、再び ED モデルを対象に、要員配置とアニメーションの充実を考える。

9 章は「**Simio の高度なモデリング**」である。ここでは、意思決定のためのモデルの使用方法、特にシミュレーションによる最適化を解説するために、ED モデルの初期バージョンを利用する。それから、ピザショップの例題を使って、いくつかの新しいモデルの構成要素を解説し、さらにこれまでに導入した概念をまとめた。組立ラインを対象とする 3 番目のモデルでは、スループットを最大化するバッファ容量の割り当てを検討する。

10 章「**その他のモデリングトピック**」は、第 4 版で追加した項目である。ここでは、Search ステップや、(待ち行列の) 退去と放棄、タスクシーケンスおよびイベントに基づく決定ロジックなどの強力なモデリング概念を解説する。また、フロー処理に対する Flow ライブラリや、クレーンやエレベータなどの設備に関する Extras ライブラリ、およびその他のツールのソースとなる Shared Items フォーラムについて紹介する。10 章の最後に、実験や多くの反復実行および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議論する。

11 章「**Simio のカスタマイズと拡張**」では、やや高度な内容を扱う。独自のカスタムオブジェクトやライブラリを構築する上で、ガイドとして機能するアドオンプロセスを使用する。ここでは、ベースオブジェクトから階層的にオブジェクトを構築する例や、標準ライブラリのオブジェクトをサブクラス化する例を挙げる。そして、Simio に独自のルール、コンポーネント、およびアドオンを実装して、Simio を拡張する方法を解説する。

12 章「**インダストリー4.0 におけるシミュレーションを用いたスケジューリング**」では、シミュレーションを計画およびスケジューリングのツールとして利用する方法を探求する。シミュレーションを用いた計画およびスケジューリングは長年議論され、利用されているが、シミュレーションソフトウェアの近年の進歩によって、実装および活用が大幅に容易となっている。また、インダストリー4.0 実装の一部として、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を議論する。Simio のリスクベースの計画およびスケジューリング技術 (RPS) の議論でこの章を終える。

付録 A「**Simio を用いたケーススタディ**」では、システムを分析するために Simio モデルを開発・利用する、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導入する。これらの問題は、比較的大規模で、これまでの章における宿題レベルの問題とは異なっており、より現実的な問題に対処するスキルを高める機会を提供する。

付録 B「**Simio 学生コンペティションの問題**」では、大規模になりつつある学生シミュレーションコンペティションで扱われた最近の問題を概観する。ここは、挑戦的なプロジェクトを探したり、自身のプロジェクトを始める際のアイディアを得る良い機会となるだろう。

最後に付録 C では、本書で参照されるファイルやその他の資料の在り方などを紹介する。

1.2 システムとモデル

システムは、ある目的に対してまとまって動作する構成要素の集合を指す、非常に広範な意味を持つ用語である。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、システムの範疇として考えられる。そのようなシステムが（既存であるか、あるいは計画中であるかに関わらず）、さまざまな構成や状況の下でどのように動作するかを理解しようとするることは自然であり、ときには不可欠な態度である。

システムシステムがすでに存在する場合は、注意深く観察することによって、洞察を得ることができるだろう。このアプローチの欠点の 1 つは、検討したい特定の条件を一度でも目にするため

4 第1章 シミュレーションとは

に、実際のシステムを長時間観察しなければならない可能性があることであり、信頼できる結論を得るためにじゅうぶんなサンプルを得ることについてはいわずもがなである。もちろん(たとえば、世界的な流通ネットワークのように)適度な詳細さでシステム全体を観測できる視点を見つけることが難しいシステムもある。

また、システムに対する変更を検討したい場合には別の問題が発生する。たとえば、変更の影響を調べるために、あるシフトに一時的に2人目の要員を加えるなど、簡単に実際のシステムを変更できるケースもある。しかし、多くの場合、この方法は実用的ではない。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利用するべきか評価したい場合、その投資額を考えてみるとよいだろう。さらに、実際のシステムがまだ存在しない場合、観察は不可能である。

とりわけこのような理由から、理解を深めるためにモデルを用いる。モデルには多くの種類があり、それぞれに利点と制約がある。自動車や飛行機のモデルのような物理モデルは、風洞試験で見られるように、現実感と物理環境との相互作用の双方を検討できる。数学的な表現を用いる解析モデルは、特定の問題領域において極めて有用であるが、その利用可能領域はたいてい限定されている。シミュレーションは、それよりも、はるかに広い適用性があるモデリングアプローチの1つである。

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するものであり、システムの動作に伴う結果を、通常は時間の経過と共に、適切な詳しさで描き出す。ほとんどの場合、シミュレーションモデルは、一般的なシステムの構成要素を表現し、それら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、多くの場合、既存のシステムに加える変更の影響を予測すると共に、新たなシステムのパフォーマンスを予測するためにも活用される。またシミュレーションは、システムの設計やエミュレーション、および運用にも頻繁に用いられている。

シミュレーションは、確率的あるいは確定的なものに分類される。もっとも一般的な確率的シミュレーションでは、ほとんどのシステムに見られる変動を表すためにランダム性が導入されている。たとえば、人に関連する活動の結果(タスクが完了するまでの時間や品質水準)は常に変動し、外部からの入力(顧客や材料)は異なり、例外(障害)も発生する。確定的モデルには変動が存在しない。これらは、設計に関する応用例ではまれであるが、スケジューリングやエミュレーションへの適用例など、モデルベースの意思決定支援ではより一般的な方法である。3.1.3項「確定的モデルと確率的モデル」において、これらの概念を詳しく解説する。

シミュレーションには、離散および連續の主要な分類がある。離散および連續という用語は、システムを記述するための状態変化の性質に由来する。離散的な時点(イベント時刻と呼ぶ)でのみ変化する状態がある(たとえば、待ち行列長や作業員の状態)。一方で、時間経過と共に連續的に変化する状態もある(たとえば、タンク内の圧力やオープン内の温度)。もっぱら離散的あるいは連續的なシステムもあるが、どちらの状態も混在するシステムもある。3.1.2項「連續型動的モデルと離散型動的モデル」でさらに検討し、連續型シミュレーションを例示する。

連續型のシステムは、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では、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、連續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、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をモデル化するために活用される場合が多い。

さらに、4つの離散型モデリングのパラダイムを紹介する。イベントは、システム状態が変化する(たとえば、顧客の到着もしくは出発)時点をモデル化する。プロセスは、時間経過にともな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、サービス時間の時間経過、作業者の解放などの活動)。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あり、多数の自律的な

意思決定を行うオブジェクト（たとえば、兵士、市場での企業、伝染病の感染者など）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には、複数のパラダイムを組み込むものもあることから、これらのパラダイム間の区別がやや曖昧になっている。Simio はマルチパラダイムのモデリングツールであり、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。ある1つのパラダイムを使用することもできるし、同じモデルで複数のパラダイムを組み合わせることもできる。Simio は、オブジェクトの使い勝手とプロセスの柔軟性を兼ね備えている。

シミュレーションは、さまざまな用途に応用されている。ここでは、システムの有効性を理解し、改善するためにシミュレーションが活用されたいくつかの分野の例を示す：

- ・ 空港：駐車場発のシャトルバス、チケットカウンタ、セキュリティ、ターミナル輸送、フードコート、手荷物取り扱い、ゲート割当て、航空機の着氷防止。
- ・ 病院：救急部門、防災計画、救急車の派遣、地域サービス戦略、資源配分。
- ・ 港湾：入港トラフィック、出港トラフィック、港湾管理、コンテナ保管、設備投資、クレーン操作。
- ・ 採掘：原材料輸送、労働者の輸送、機器の割当て、バルク材の混合。
- ・ 遊園地：ゲストの輸送、アトラクションの設計・立上げ、待ち行列、乗車人員配置、混雑の管理。
- ・ コールセンター：人員配置、スキルレベルの評価、サービスの改善、研修計画、スケジューリングアルゴリズム。
- ・ サプライチェーン：リスクの低減、発注点、生産と在庫の配置、輸送、拡張管理、緊急時対応計画。
- ・ 製造業：設備投資分析、ライン最適化、製品ミックスの変更、生産性の向上、輸送、労働力の減少。
- ・ 軍事：ロジスティクス、保守、戦闘、暴動対策、索敵、人道援助。
- ・ 通信：メッセージ転送、ルーティング、信頼性、停電や攻撃などへのネットワーク頑強性。
- ・ 刑事司法制度：保護観察または仮釈放の運用、刑務所の使用率とキャパシティ。
- ・ 緊急対応システム：応答時間、対策地点、設備の水準、要員配置。
- ・ 公共部門：投票区への投票機の配置。
- ・ カスタマーサービス：ダイレクトサービスの改善、バックオフィス業務、リソース配分、能力計画。

製造業のためのツールという見方を超えて、シミュレーションの対象および適用領域は、幅広く、事実上無限に広がっている。

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に含まれる典型的な手順を議論する。また、シミュレーションモデルに対する入力および出力における不確実性とランダム性の重要な役割について説明する。

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

いくつかのシミュレーションモデリングの例では確定的な値のみを用いることもあるが、シミュレーションモデルの大半は、モデル化するシステムにある種のランダム性が含まれるため、何らかのランダム性が組み込まれている。典型的なランダム性を持つ構成要素には、処理時間、サービス時間、顧客またはエンティティの到着時刻、移動時間、機械／リソースの故障および修理、および類似の事象がある。たとえば、深夜にファーストフードレストランのドライブスルーに向かう場合、

6 第1章 シミュレーションとは

何分で着くのか、到着したときに何人の顧客が待っているのか、あるいは何分で食事にありつけるのかについて、いくつかの変数を命名したとして、正確に知ることはできない。過去の経験や知識に基づいて、これらの値を推定することができるかもしれないが、確実に予測することはできない。モデルでこれらの確率的な値に対する確定的な推定値を使用すると、妥当性のない（一般に過度に楽観的な）性能予測に終わる。しかし、標準的な解析モデルでは、これらのランダムな要素を組み込むことが困難あるいは不可能な場合がある。一方、シミュレーションを用いることで、ランダムな要素を含めることが非常に容易になる。実際に、シミュレーションをモデリングおよび解析のツールとして人気のあるものにしている要因は、確率的なふるまいを簡単に組み込める能力による部分が大きい。これは本書全体を通じて、基本的なテーマになっている。

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である ((Ross 2010) および (nelson 1995) を参照されたい)。もっとも基本的な事項は、確率変数が実験の結果によって決定される値を返す関数ということである。つまり、私たちは実験が遂行された後にならないと値を知ることができない。シミュレーションの文脈では、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である。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。上式の右辺は、確率変数 X が値 x 以下の値をとる確率を表している。離散確率変数においては確率質量関数 $p(x_i)$ が考慮されなければならない、連続確率変数においては確率密度関数 $f(x)$ を評価する (表 1.1 参照)。いったん確率変数 X を分類すれば、期待値 ($E[X]$) や分散 ($\text{Var}[X]$)、あるいは分位や対称性／歪度などの確率分布の特徴を測ることができる。多くの場合、都合よく対応する母集団のパラメータを得られないため、標本平均 \bar{X} や標本分散 $S^2(X)$ などの標本統計量に依存することになる。これらの推定量に対する適切なサンプルサイズを決定することが重要である。他の多くの実験手法とは異なり、シミュレーション分析では、必要を満たすようなサンプルサイズを制御できることも多い。

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

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、シミュレーションによって生成された観測値に基づいて分布の特性（すなわち、平均、分散、パーセンタイルなど）を分析する。ここで、小規模な医療クリニックのモデルを考える。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、それらはすべて確率変数である (図 1.1 参照)。

システムをシミュレートするために、モデルへの入力として、これらの確率変数を理解し、その観測値を生成する必要がある。多くの場合、入力確率変数を特定するために利用できる実際のシステムから得られたデータが存在する。典型的な出力は、患者の待ち時間、システム内時間、介護士および空間の利用率などであろう。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する。シミュレーションモデルの実行を制御することによって、関心のある出力を特定するために生成された観測値を活用することができる。以下の節では、一般的なシミュレーションプロセスの文脈で、入力および出力の分析を説明する。

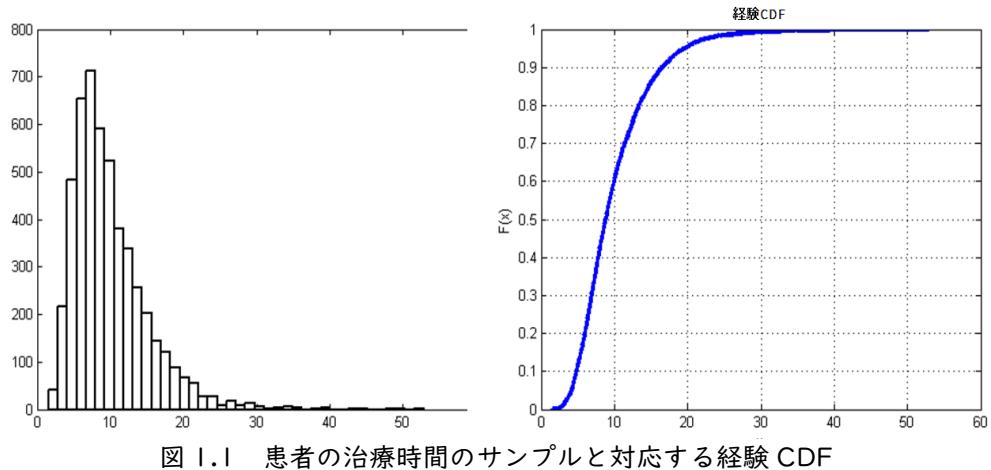


図 1.1 患者の治療時間のサンプルと対応する経験 CDF

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。プロセスが厳密にシーケンシャルではなく、たいてい反復的であることに留意してほしい。次項より、これらの概念を簡単に説明し、本書を通してより詳細な事柄を扱っていく。

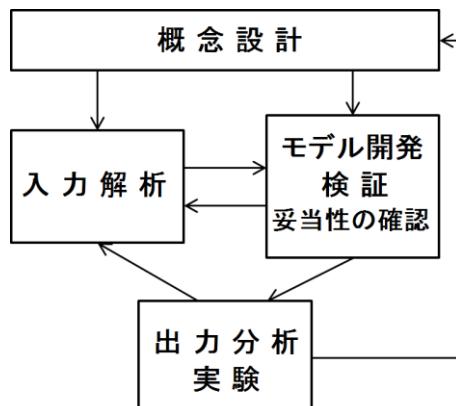


図 1.2 シミュレーションプロセス

概念設計では、モデル化されるシステムと、シミュレーションモデルの構築に対する基本的なモデリング手法を、詳細に理解しておく必要がある。概念設計は、紙とペン、あるいはホワイトボード、もしく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。それにより、利用するシミュレーションソフトウェアの制約に縛られないという効果がある。概念設計に対して明確に定義されたプロセスや方法論は理想的であるが、その通りには運ばないだろう。それどころか、プロジェクトを計画することは、問題やそれに対するモデリング手法の詳細を考察し、議論する非公式なプロセスである。それから、モデル開発者はモデリング手法の詳細を体系的に概説し、ソフトウェア固有の詳細まで適用を決定する。なお、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ているため、概念設計の重要な側面として、モデルが必要な疑問点に対する答えを出すことを保証することがある。一般に、シミュレーションの初心者（および新人モデル開発者）は、概念設計段階に費やす時間があまりにも少ない。得てして、一足飛びにモデル開発プロセスを開始する傾向にある。概念設計に充當する時間が少なすぎると、ほとんどの場合、プロジェクトの完了に必要な時間が増加する。

入力解析（6 章で詳しく述べる）には、システムへの入力を特定し、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を開発することが含まれる。事実上（Simio を含む）すべての商用シミュレーションソフトウェアは、入力観測値を生成するための機能を備えている。したがって、主要な入力解析の作業は、入力確率変数を特定することと、シ

8 第1章 シミュレーションとは

ミュレーションソフトウェアに対応する確率分布および確率過程を指定することになる。多くの場合、現実世界のデータとしてサンプル観測値が存在するため、シミュレーション中にサンプルを生成するために用いられる標準確率分布あるいは経験確率分布に、実データを適合させることが一般的なアプローチとなる（図1.1参照）。その他には、実際の観測データからランダムにサンプルを生成する方法もある。入力に関する実データが存在しない場合は、入力解析を支援する一般的な経験則と感度分析を活用する。これらの方針において、選択された入力に対するモデル出力の感度を分析することが重要である。6章では、入力パラメータの利用について議論し、入力解析への活用方法を解説する。

モデル開発は、概念モデルを実行形式のシミュレーションモデルに変換するコーディングプロセスである。「コーディング」という言葉で逃げ出さないでほしい。多くのシミュレーションパッケージは、モデルの開発／修正をサポートするため、洗練されたグラフィカルユーザインターフェースを備えている。そのため、コーディングは一般に、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入力するだけである。しかし、効果的なモデル開発には、一般的なシミュレーションの方法論や、特定のソフトウェアの操作方法などへの詳細な理解が必須となる。検証および妥当性の確認の段階では、モデルが正しいことを確認する。検証は、モデルが開発者の意図したと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、モデルがモデル化された実際のシステムに対して正確であることを保証するプロセスである。ただし、単純なモデルを除くすべてのモデルについて、正確さを証明することは不可能である点に留意してほしい。その代わり、モデル開発者（または顧客）が満足する証拠を集めることに焦点を当てる。このことはシミュレーションの初心者を混乱させるかもしれないが、それが現実である。モデル開発、検証、妥当性の確認に関する話題は、4章以降で扱う。

モデルが検証され、妥当性が確認されたら、次に元となったシステムに関する情報を収集するため、モデルを実行する。これには、出力分析および実験が含まれる。上記の例では、患者が介護士から診察を受けるまで待っていた時間の平均や、待合室の患者数の90パーセンタイルや、ドライブスルーで待っている車の平均数などのようなパフォーマンス指標を評価することに关心があるだろう。また、患者の平均待ち時間を30分以下にするような介護士数や、注文に対して平均5分以内で応じるためのキッチンスタッフ数などの意思決定を行うことにも興味があるだろう。シミュレーションモデルを用いてパフォーマンス指標を評価し、意思決定を行うことは、出力分析と実験に含まれる事柄である。出力分析は、シミュレーションによって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について推定する。実験では、モデルの入力と構造を体系的に変化させ、代替的なシステム構成を調査する。出力分析はモデリングを行う章（4、5、9章）にわたって扱う話題となる。

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、近年耳目を集めている。主な理由の1つは、「複雑」という言葉自身で体現されている。関心のあ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簡単であれば、シミュレーションは必要ではなく、また使用されるべきではない。正確な分析手法として、待ち行列理論や確率論、あるいは代数学や微積分のような手法を代わりに用いるべきである。正確に解析的な解を見つけることができるシンプルなシステムをシミュレートすることは、結果をより不正確にしてしまうノイズ（つまり不確実性）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あふれており、前記の非常に「簡単な」モデルの領域から抜け出すことになる。そして、複雑なシステムについて妥当なモデルをまとめて構築しようとするのであれば、そのモデル自体もおそらくかなり複雑となり、単純で解析的な分析にはなじまないであろう。正確な解析的な解を得るために能力を保つことを目標に、複雑なシステムについて簡単なモデルを構築したとしても、結果として得られるモデルは過度に簡単となる可能性があり、そのモデルがシステムを有効に表現できているかどうか疑念が生じる。簡単なモデルからは、洗練され、

疑う余地もなく、正確な、閉じた形の解析解を得ることができるが、解析的に扱いやすいモデルを得るためにには（現実的に極めて疑わしいかもしれない）簡略化のための多くの仮定を置いている可能性があり、実際に解を得られたと確信を持っていえるかどうかは疑わしい。それは確かにモデルに対する解であるが、そのモデルが現実をじゅうぶんに表現できていなければ、問題に対する解を得られたとはいえないだろう。

モデルがどの程度非現実的かを測定するのは難しい。そのような疑問を呈することの意味についても明らかではない。一方、最終的に解析的な解を得られるモデルを構築する必要がなければ、適切な方法でシステムを模擬するために、必要なだけ複雑でごちゃごちゃしたモデルとなっても構わない。しかし、簡潔な解析的に扱いやすいモデルが利用不可能となれば、たんに複雑なシステムを模擬して、シミュレーションを行う必要が生じる。そこでは、複雑な（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を行い、結果として何が起こるかを観察する必要がある。この部分については、物事を現実的にするために、一般にモデルへの入力値を確率的、すなわちランダムにし、固定した定数による入力値ではなく、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からの結果も確率的となり、不確実性を有するものとなる。

シミュレーションの出力における不確実性、あるいは精度の悪さは、明らかに否定的な面である。しかし、この不正確さの程度を測定することは難しくはなく、得られた解が好ましくない（すなわち、結果があまりにも不正確である）ならば、それに対処する方法はわかっており、たんにより多くの時間を割いて、コンピュータシミュレーションを実行すればよい。ほとんどの統計的サンプリングの実験とは異なり、シミュレーションではランダム性および反復実行回数について完全に制御可能であり、必要な精度を得るためにこの利点を活用することができる。シミュレーションの実用性に関しては、コンピュータの計算時間が真の障害である。しかし、容易に入手可能な高速でマルチプロセッサのコンピュータ上でソフトウェアを実行していれば、あるいはクラウドコンピューティングを活用していたとしても、不正確さを測定可能で許容可能なほど低くし、妥当な結果をもたらすように繰り返しシミュレーションを行うことが可能である。

これまで、シミュレーションはしばしば「最後の手段として」、つまり「他のすべての手段が失敗したとき」にのみ取りうるアプローチとして、却下されることもあった (Wagner 1969) pp. 887, 890)。上述のとおり、妥当で解析的に扱いやすいモデルが利用可能であれば、シミュレーションを用いるべきではないことには頷ける。しかし、多くの（おそらくほとんどの）場合、実際のシステムはあまりにも複雑であるため、すなわちルールに従わない場合があるため、解析的に扱いやすいモデルを、信頼できる妥当性を持って構築し分析することは難しい。本書の立場では、誤ったモデルを対象として正確な解析的な分析を行い、その誤差を定量化することもできない解を得ることよりも、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し、削減できる近似解を得るほうがよいとする。前者の分析は、不正確さよりも悪い状況に陥ると考えるものである。

さて、本書の例および図は、Simio のバージョン 14 で作成されている（新しいバージョンの Simio を利用している場合は、本書 Web サイトに補足情報を記載しておくので参照されたい）。Simio の各バージョンは、低レベルの挙動（同時に起こるイベントの処理順序など）に影響を与える恐れのある変更が含まれている可能性があるため、異なるバージョンでは、シミュレーション実行において、本書と異なる数値出力結果を生成する可能性がある。「どちらの結果が正しいのか？」と読者は疑問に思われることだろう。本書では、統計的に妥当な結果を得る方法や、その結果を理解する方法について解説する。バージョン間のまれなバグの場合を除いて、すべてのバージョンでは、それぞれの実行で数値が異なったとしても、同じモデルに対する統計的に等価な（妥当な）結果が生成される。

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を学び、その基礎技術を理解することだけでは、成功を保証することはできない。シミュレーションプロジェクトを成功に導くには、それよりもはるかに多くの

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は、シミュレーションで成功する方法を尋ねることが多い。答えは簡単で「ハードワークとすべてを完璧にこなす」ことである。しかし、それだけの答えではおそらく納得されないだろう。次項より、考慮すべき重要な事柄を確認してみよう。

1.5.1 プロジェクトの目的

多くのプロジェクトは、納期を確定することから始まるが、成果として必要な事柄に対してはおまかなかいわゆるアイディアしかなかったり、遂行する方法については漠然としたものであったりする。シミュレーションを示されたとき、最初に発せられる質問は「目的は何か？」であろう。簡単に答えられる白明な質問のように思えるかもしれないが、関係者は答えがわから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だろう。しかし、目的を決定する前に、利害関係者とのことをよく知っておかねばならない。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のことである。明らかに上司は利害関係者であろう（あなたが学生なら、指導教員は確実に利害関係者の1人である）。しかし、時には、主要な利害関係者を見出すことに時間を割かなければならない。利害関係者は、多くの場合、異なる（時には矛盾する）目的を持っているため、心を配らねばならないのである。

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希望の結果（製品処理量の増加、待ち時間の減少、メンテナンスの削減、など）を得られるかどうか評価してほしいと依頼されたとしよう。この場合、考えられる利害関係者と、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証明したいと思っているため、劇的なコスト削減や生産性向上を実証したいと考えている。また、社内にあなたの作成したモデルを売り込むことができるよう、見栄えのよい 3D アニメーションを期待している。
- ・ 生産管理部課長：彼は、新しいクレーンを購入することが生産目標を達成する唯一の道であると確信しており、あなたのモデルによりそれが証明でき、その情報をもって彼にとってのキーパーソンを説得しようと考えている。
- ・ 生産管理部長：彼は、長年の生産担当の経験により、「シミュレーション」のようなものが実際に利益をもたらすとの確信を持てずにいる。社内の駆け引きのため、この取組みへの期待はわずかであり、プロジェクトが失敗すると予想している（密かにそうなることを期待している）。
- ・ 財務部長：彼女は、クレーン購入に資金を費やすことを非常に懸念しており、導入後の生産性がじゅうぶんではないと疑念を抱いている。彼女は、実際に前回の幹部会議で、客観的な分析をするために、シミュレーション研究ではなく試運転を行うことを主張した1人である。
- ・ 生産ラインリーダー：彼女は 15 年間勤務しており、材料の運搬を担当している。彼女は、生産性を増加させる、より安価で効果的な方法を知っていて、この情報を誰かと共有したいと考えている。
- ・ 材料運搬作業員：勤務時間の多くが材料の運搬に充てられており、新しいクレーンが導入されるとレイオフされるのではないかと心配している。そのため、新しいクレーンの導入は現場を悪化させるとの見解をあなたに披露し、納得させるために全力をつくすつもりだ。
- ・ 技術部課長：彼のスタッフはすでに困惑しているので、できる限り関わり合いたくないと考えている。しかし、新しいクレーンが導入された場合、それをどのように構成して使用すべきかについて、非常に具体的なアイディアを持っている。

このシナリオは、いくつかの実例を組み合わせたものである。小規模なプロジェクトや中小企業では利害関係者の人数は少ないかもしれないが、基本原理は同様である。相反する目的や動機はまったく珍しくない。各々の利害関係者は、プロジェクトに対する貴重な情報を持っているが、彼らから提供された情報を評価する際には、プロジェクトに対するバイアスやプロジェクト参加の動機を考慮に入れることが重要となる。

どのような利害関係者が存在するかについて理解されたと思う。さて今度は、彼らがそれぞれ、プロジェクトの目的をどのように見ているか、どのように貢献しようとしているか、そしてどのように彼ら自身の目的達成の優先順位を高めようとしているのかを考えよう。主要な目的を確認するためには、以下のような事柄を尋ねる必要がある：

- ・ 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、近似すればよいものはどれか。
- ・ 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、提供されるのはいつなのか。
- ・ 何回くらいの実験を必要としているのか。最適解を求めることが必要なのか。
- ・ アニメーションをどのように利用するのか（妥当性を確認するためのアニメーションと、取締役会におけるプレゼンテーション用のアニメーションとは、まったく異なる意味を持つ）。
- ・ どのような形式の報告を求めるのか（口頭発表、詳細な数値、要約、グラフ、文書）。

明確な目的を立てるための有効な方法の一つに、最終報告書のモックアップを設計することが挙げられる。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を作成するとしたら、あなたのニーズを満たしているか」と確認することができる。最終報告書の形式と内容について、利害関係者の合意を形成できれば、内容の適切な詳細度を決めるのは後回しにして、他のモデリング上の課題に専念することができる。また、このプロセスでは、潜在的なモデリングの目的を引き出すこともできる。

プロジェクトに対して必要な明確さが存在しない場合もある。このような場合、成果物やリソース、日時など、プロジェクト全体をとにかく計画して先に進めることは、失敗に向かって自らを駆り立てているようなものである。プロジェクトに対する明確性の欠如は、プロジェクトを段階的に行うようにと告げる天の声である。小規模なプロトタイプの作成から始めると、大きな問題が明確になる場合が多い。プロトタイプ作成の経験に基づいて、後続フェーズに対して詳細な計画を立てる道筋をつけることができるだろう。このことについては、次項で詳しく取り扱うことにする。

1.5.2 仕様書

「行く先がわからなければ、そこに着く時間を見る術はない」

ある大工のアドバイス：「2回測って、一度に切る（念には念を入れて）」

前項のアドバイスに従えば、少なくとも基本的なプロジェクトの目的を共有できるはずだ。それでは、モデルを構築する準備が整ったといえるだろうか？実は時期尚早である。多くの場合、利害関係者はいくつかのコミットメントを求めているだろう。

- ・ 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、信頼性のある返答をする準備ができているだろうか？おそらくできないだろう。

もちろん、最悪の（しかし一般に起こりうる）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、あなたは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ドルを支払う」と提案された情景を思い浮かべてほしい。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週間後に一部完成しただけで、未検証のモデルが残るだけである。約束した報酬は、依頼したものを得られないのだから、支払われるはずもない。

顧客にとっては、これらの質問の内の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、それらの回答の内、少なくとも1つ、もしくは2つを調整する権利を担保する必要がある。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るかもしれない。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双方を倍にできるかもしれない（品質を調整するのはよい考えとはいえないことが多い）。

幸運なら、利害関係者は上記4つの間に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：

1. はじめに

- a) シミュレーションの目的：高レベルの目的に対する議論である。このプロジェクトに期待される結果は何か？
- b) 利害関係者の識別：このモデルから得られる結果に関わる主要な関係者は誰か？他の人たちにも影響が及ぶのか？モデルは誰に、どのようにして使用されるのか？彼ら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れらをモデル化するアプローチの概略である。次の構成要素を含む（ただしこれら以外のものも考えられる）：
 - a) 設備：それぞれの設備は、その挙動、段取り、スケジュール、信頼性、およびモデルに影響を与えるその他の側面など、詳細に記述すべきである。必要に応じて、データ一覧表や図を含める。データがない場合、別個に考慮する必要がある。
 - b) 製品種：どの製品を含めるのか？どのように異なるのか？相互関連はどうか？各製品あるいは各製品群に必要な詳細度はどの程度か？
 - c) 運用：それぞれの作業は、設備と同じく、その挙動、段取り、スケジュール、信頼性、およびモデルに影響を与えるその他の側面など、詳細に記述すべきである。必要に応じて、データ一覧表や図を含める。データがない場合、別個に考慮する必要がある。
 - d) 運搬：内部および外部の運搬は、じゅうぶん詳細に記述すべきである。
- 3. 入力データ：モデルへの入力として考慮すべきデータは何か？この情報を提供するのは誰か？いつ？どのような形式で？
- 4. データ：モデルから生成されるべきデータは何か？ここでは、最終報告書のモックアップが当事者すべての期待を明確にするのに役立つ。
- 5. プロジェクトの成果物：以前に合意されたプロジェクトのすべての成果を議論する。このリストが埋められた時点で、プロジェクトが完成したとみなされる。
 - a) 文書：どのようなモデルの資料や手引き、ユーザマニュアルが提供されるのか？どの程度の詳しさなのか？

- b) ソフトウェアとトレーニング：ユーザがモデルを直接操作したいと望んでいる場合には、何が必須のソフトウェアなのか、それはプロジェクトの見積価格に含まれるのか、どのようなカスタムインターフェースが必要なのか、などを議論する。また、どのようなプロジェクトや製品のトレーニングが推奨されるのか、あるいは提供されるのかも明記する。
 - c) アニメーション：成果物としてのアニメーションはどのようなもので、使用される目的は何か（モデルの妥当性の確認、利害関係者の買取、マーケティング）？2D なのか 3D なのか？既存のレイアウトやシンボルを利用可能なのか？可能な場合、どのような形式で利用できるのか？何が、誰から、いつ提供されるのか？
6. プロジェクトのフェーズ：プロジェクトの各フェーズと、それぞれに想定される工数、納期、費用を記載する。
7. 署名：主要な利害関係者が署名する箇所である。

プロジェクトの開始時に、モデリングを開始してしまう傾向は確かに存在する。時間の制約もあまりなく、アイディアも溢れる、興奮する瞬間だからである。それを思い止めて、仕様書を作るのは非常に難しい。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信じてほしい。本節冒頭の引用文を想起してほしい。目的地とそこに到達する方法を決めるために立ち止まることは、見当違いの努力と無駄な時間を節約することにつながる。

想定されるプロジェクト期間の最初の約 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。これが余計な時間であると考えてはならない。むしろ、報告書のデザインなど、大部分が計画段階で決まるような特定のタスクについて、プロジェクトの初期に片付けてしまう方がよい。たとえば、プロジェクトの完成に 20 日間かかることが見込まれる場合は、約 2 日をこの作業に費やすべきであることを意味する。結果的に、プロジェクトの完成に 40 日間必要となることが次第に明らかになった場合でも、そのときになって慌てて代替案（目的の優先順位の変更、モデル対象の縮小、リソースの追加など）を検討するよりも、前もってある程度考えておくほうがよい。

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行われることがベストである。最初のステップから繰り返すことさえある。現時点では、目的を決め、仕様書を作り、プロトタイプを作成するだけだろうと思われるかもしれない。しかし、仕様書を書いている間に、おそらく新しい目的を発見するだろう。そして、プロトタイプを作っている間には、仕様書に追加すべき、重要な新しい事柄を見つけるだろう。

プロジェクトに踏み込んでいくにつれ、反復アプローチはさらに重要になる。シミュレーションの初心者は、アイディアを思いつくとモデリングを開始し、完成するまでそのモデルに機能を付け加えていく。そこで、初めてモデルを実行する。しかし、熟練したモデル作成者でさえ、最高のツールを使っていたとしても、誤りを犯す。誤りが「モデルのどこかにある」ということしかわからなければ、それを探し出し修正することは極めて難しい。シミュレーション教育の過去の経験に基づけば、これは学生にとって大きな問題である。

経験豊富なモデル作成者は、一般にモデルの小さな部分を構築し、実行・テスト・デバッグして、作成者が意図したように動くかどうかを検証する。それから、このプロセスを別の小さなモデルに対して繰り返す。現実のシステムと比較できるモデルができればすぐに、小さなモデルが意図したシステムと合致するかを確認する。この反復プロセスを、モデルが完成するまで繰り返すのである。このプロセスの各ステップでは、問題は直近に追加された小部分にある可能性が非常に高いため、問題の発見および修正がはるかに容易となる。また、各ステップで異なる名前（たとえば、MyModelV1、MyModelV2。あるいは日付や時間をファイル名に付加する）で保存することにより、必要に応じて以前のバージョンに戻すこともできる。

この反復アプローチには、潜在的に大きな問題が早期に排除されるという、特に初心者にとって

14 第1章 シミュレーションとは

のメリットがある。たとえば、エンティティをグループ化する方法について、誤った前提の下でモデル全体を構築してしまい、その誤りにまさに最終局面で気づいたとしよう。その時点では、モデルの根本から変更するために、膨大な修正作業が必要となるだろう。しかし、反復的にモデルを作成していれば、グループ化するための要素を初めて導入した際に誤りを発見することができただろう。そして、その時点でよりよい方法を比較的簡単に採用することができただろう。

最後に、反復アプローチの極めて重要なメリット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について予測できることの1つに、想定よりも工期が延びる傾向が強いことがある。シミュレーションモデルの構築においても、同じ問題を共有している。反復アプローチを採らずにプロジェクトの期限がきてしまい、まだモデルが動作しない場合、検証や妥当性の確認はおろか、プロジェクト遂行に費やした努力を披露することすらできない。しかし、反復アプローチによれば、期限が来たとしても、完成し、検証や妥当性の確認が済み、利用に耐えうる部分的なモデルは存在する。各反復でもっとも優先順位の高いタスクに取り組んでいれば、完成している部分はプロジェクトの目的の大部分を満たすものとなろう（80 対 20 の法則、あるいはパレートの法則という名で知られている）。

プロジェクトや適用対象により多少異なる場合はあるが、シミュレーション研究の一般的な手順は次のとおりである：

1. 高レベルの目的を定義し、利害関係者を明確にする。
2. 詳細な目標、モデル範囲、詳細さの程度、モデリング手法、および出力尺度などを記載した仕様書を定義する。また、最終報告書を設計する。
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を繰り返す。
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。反復する。
5. モデルの入力データを収集し、組み込む。
6. モデルを検証し、妥当性の確認を行う。利害関係者を巻き込む。必要に応じて、手順4に戻る。
7. 実験計画を立てる。実験を遂行する。利害関係者を巻き込む。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、賞賛を受ける。

反復して取り組んでいる間には、利害関係者と定期的にコミュニケーションする機会を持つことが重要だ。利害関係者は、驚かされることを好みない。プロジェクトが事前の想定とは異なった結果を示そうとしている場合、その原因を共に検討する。また、スケジュールが遅れ気味になった際には、利害関係者に早めに知らせ、深刻な問題が生じることを避けるべきだ。利害関係者をクライアントと捉えてはいけないし、敵対者としてもならない。利害関係者はパートナーである。このプロジェクトから最良の結果を得るために、互いに助け合わなければならない。そして、最良の結果は、モデル化される実際のプロセスを明らかにするために必要な、システムの詳細調査からもたらされる場合が多い。実際には、多くのプロジェクトにおいて、シミュレーションの「結果」が生み出される前に、それらの値の大部分は判明している。モデル作成者による早期の説明や、利害関係者との頻繁なコラボレーションによってもたらされる知識のおかげである。

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さまざまだが、もっとも明らかなものの1つに、完了期限を満たすことがある。意思決定が行われてしまってから結果を知らされるプロジェクトには、ほとんど価値が

ない。その他、成功要因に関連するものは、コスト、リソース、工数である。予算を超過するプロジェクトは、完了前に中止させられる場合がある。そのため、納期とプロジェクトのコストにじゅうぶんな注意を払う必要がある。双方とも、日々のプロジェクトをどの程度詳細に管理するかにかかっている。

管理の行き届いたプロジェクトは、意思決定を左右する明確な目標と確かな仕様書を決定することから始まる。プロジェクト全体を通じて、次のような、大小の意思決定が行われる：

1. 特定の部分を、どの程度の詳細さでモデル化すべきか？
2. 入力データは、どの程度の量を収集する必要があるのか？
3. もっとも注意を払うべき出力データはどれか？
4. モデルの妥当性が必要とされるときはいつか？
5. アニメーションにどの程度の時間を費やすべきか？分析にはどうか？
6. 次にするべきことは何か？

ほとんどすべてのケースでは、仕様書の中に、直接的あるいは間接的に答えが隠されている。すでに、主要な利害関係者の目的を理解し、その優先順位をつけている。その情報が、ほとんどの意思決定の基礎となる。

優先しなければならないことは、仕様書の更新、すなわち利害関係者からの新たな要求であり、つまりは仕様変更である。極端な対応として、強硬路線を取り、「仕様書がない場合、モデルに含めることはできない」ということもできる。まれにこの返答が適切かつ必要なケースであることがあるが、ほとんどは不適切だろう。シミュレーションは、探索と学習のプロセスである。新しい領域を探求し、対象システムについての理解が進むにつれ、研究の新たな課題、アプローチ、領域が変化していくのは当然だ。これらに対処することを拒めば、シミュレーションの潜在的な価値（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）を大幅に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探ることである。つまり、利害関係者に何か新しい作業をしてほしいと依頼されたなら、正しいこととして仕事を進める態度である。この対応を採れば、利害関係者は短期的に幸せを感じるだろうが、おそらくほとんどのより長期的な結果が遅れたり、あるいは未完のプロジェクトに終わる可能性もある。そして、利害関係者は非常に腹を立てるだろう。もし最新のアイディアを常に追求している場合には、結果の値を生成するために必要な優先順位の高い作業を完成させる時間すらないかもしれない。

重要なポイントは、これらの機会を管理することだ。その管理とは、利害関係者とのオープンなコミュニケーションから始まり、仕様書の項目とその相対的な優先順位を再検討することである。何かがプロジェクトに追加されたときには、他のものを変更する必要がある。おそらく、新しい項目を追加すると、プロジェクトの期限を少し延長することが重要になってくる。延長できない場合、この新しい項目を、他の落としてもよい作業よりも重視しなければならない（他の作業は想定以上にプロジェクトの進捗が滞っている場合に開発する「ウィッシュリスト」に移動させる）。あるいは、この新しい項目自体を「ウィッシュリスト」に移動させるべきかもしれない。

本書でいうアジャリティの定義は、変化に対して迅速に、かつ適切に対応する能力のことである。アジャイルである能力は、シミュレーションでの成功に多大な貢献を果たすだろう。

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

それでは、利害関係者が、あなたが彼らのために行うこと、合理的な期待を持っていることを確認して、本章を終える（図 1.3）。次の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配慮しなければならない。しかし、これらの期待と共に、利害関係者はあなたにいくつかの責任を負わせる（図 1.4）。前もって双方の期待を議論することにより、コミュニケーションが促進され、プロジェクト成功の地固めができる。つまり、全員のニーズを満たす Win-Win

16 第1章 シミュレーションとは

の関係を築くのである。これらの「権利」をブログ『Success in Simulation』(Sturrock 2012b) (www.simio.com/blog) から許可を得て引用する。このブログには、多くの成功のためのヒントと興味深いトピックが投稿されているため、ぜひ熟読されることを勧めたい。

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。	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。
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。	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。
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。	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。
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。	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。
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。	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、適切な分析が加えられる。
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	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つて影響されない。

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同意する必要がある。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、適切に開与し、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、定期的に開与する必要がある。
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奪取られないよう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つてはならない。長時間労働の前提に基づいた
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、強制されなければならない。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、分析の客觀性を保つために、所望の結果が得られるよう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（すべてではないが）システムは、待ち行列システムの一種として表現できる。たとえば、患者がある病院に到着すると（すなわち、予約なしでランダムに訪れる）、おそらく少し列（つまり待ち行列）に並んだ後、まずは受付で診察申込書に記入する必要がある（図2.1参照）。

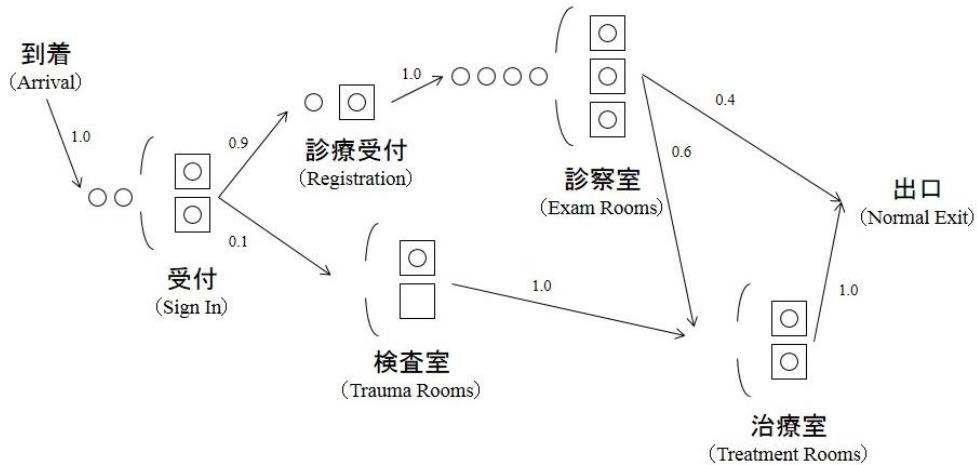


図2.1 病院を表す待ち行列システム

受付後、患者は診療受付、あるいは深刻な症状の場合、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、あるいは治療室に移動してから（おそらくそこで待ち行列に並んで）システムを離れる。検査室に向かった患者は、全員治療室に移動し（おそらくそこでも待ち行列に並び）、それからシステムを離れる。このような施設の設計および運営に対する問題としては、次のような事柄が考えられる：それぞれの診療科のスタッフは、いつ何人必要か。待合室の大きさはどの程度か。医師および看護師が患者の診察に費やす時間を増減すると、待合室で待っている患者にどのような影響が現れるか。訪れる患者が10%増加するとどうなるか。先着順ではなく、症状の深刻な順に患者を診察すると、どのような影響が現れるか。

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留めるが、この内容と用語に精通しておくことは、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。初步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おいても、有用であるだろう。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用いることもできる。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）シンプルなものである場合に限る（たとえば、シミュレートする実際のシステムでは正しいとはいえない指数確率分布を仮定する）。（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ことに確信を持てる。後でSimioのシミュレーションモデルを構築する場合に、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、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り、1909年に初めて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をどのように設計・運営すべきかに関する研究して以来、数学的に研究され続けている（Erlang 1909, Erlang 1917）。待ち行列理論については、それ自体をテーマとした書籍（たとえば Gross et al. 2008, Kleinrock 1975）も多いが、確率統計に関する書籍の章として書かれたもの（たとえば Ross 2010）や、製造業のような応用事例に関するもの

(たとえば Askin and Standridge 1993) など、広範囲に出版されている。また、「待ち行列理論 (queueing theory)」のキーワードでウェブ検索をすれば、140万件以上のページがヒットする。本章は、待ち行列理論について、完全な解説を加えようとするものではない。むしろ、用語を紹介し、基本的な結果を示して、待ち行列理論とシミュレーションを比較することで、そ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利用する特定の数式を理解してほしい。

本章では、読者が次のような確率の基礎を習得していることを前提としている：

- ・ 確率的な実験、サンプルサイズ、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：離散型に対する確率質量関数 (PMF)、連続型に対する確率密度関数 (PDF)、双方の型に対する累積分布関数 (CDF)
- ・ RV の期待値とその分布（たんに平均とも呼ばれる）、分散。RV がある区間に落ちる確率を計算するために、PMF や PDF、CDF を利用する方法
- ・ RV 間の独立性（あるいはその欠如）

これらの事項を習得していないければ、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。

本章および本書のあらゆるところで、指標、一様、三角などの、特定の確率分布を頻繁に用いる。確率の概念を用いている本では、多くの分布について、PMF および PDF の定義や、CDF、期待値あるいは分散の導出などの、分布の根柢について多くのページを割くことが通例であった。しかし、現在そのような情報は多くの情報源から容易に参照することができるため、本書では省略している（たとえば、Simio のマニュアルにも書かれているし（6.1.3 項参照）、オンラインでは https://en.wikipedia.org/wiki/List_of_probability_distributions があり、そこには約 100 種の一変量の確率分布へのリンクがある）。また、確率事典も編纂されている（たとえば Johnson et al. 1994、Johnson et al. 1995、Johnson et al. 2005、Evans et al. 2000）。確率分布に関してより詳しくは 6.1.3 項を参照してほしい。

2.1 節では、待ち行列システムに関して、一般的な構造、用語および表記法を解説する。2.2 節では、多くの待ち行列システムにおける評価指標間の重要な関係を述べる。2.3 節で、ある待ち行列システムの出力結果を示し、2.4 節で待ち行列のネットワークを扱うことについて簡単に述べる。最後に、2.5 節において、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し、双方が互いを補完するように長所と短所を持っていることを示す（バランスはとれているのだが、本書では多くの場合はシミュレーションの方がよいという立場である）。

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（たとえば、顧客、患者、仕事、メッセージなど）が到着し、単一のステーションあるいは複数のステーションで順番にサービスを受け、サービスを受けるために待機する可能性があり、それから離れるようなシステムである（システムを離れるシステムをオープン型と呼び、離れずにシステム内を巡回するものをクローズド型と呼ぶ）。

上記の病院（図 2.1）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があり（受付、診療受付、検査室、診察室、治療室）、それぞれは複数窓口の待ち行列ネットワークのノードと呼ばれる（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）。待ち行列ノードに複数の並列窓口が存在する場合（たとえば、診察室に 3 つ）、各窓口が別々の待ち行列を持つのではなく、1 つの待ち行列がエンティティをすべての窓口に送り出す。また一般に、各窓口は能力やサービス率の点で独立しているものと仮定する。図 2.1 では、弧（矢印）横の数値によって、その弧に患者が従う確率を示している。次に向かう地点について選択が行

われるステーション（受付および診察室）を出ようとするときには、これらの確率がわかっている必要がある。一方、すべての患者の次に向かう地点が同じ場合（到着、診療受付、検査室、治療室）では、明らかにその弧の確率は 1.0 であるが、表記を統一するために図に示しておいた。用語について、エンティティが待ち行列内に存在しているけれども、サービスを受けていない場合を待機中と呼ぶ。図 2.1 の診察室では、4 人の患者が待機中で、診察室のシステム内に 7 人の患者がいるという。また、3 人の患者が診察室でサービス中であるという。

窓口がエンティティに対するサービスを完了したとき、そのノードの待ち行列に他のエンティティが存在する場合、待ち行列内のどのエンティティを選んで、次にサービスを受けさせるかを決定する必要がある。これをサービス規律という。もっとも一般的で「公平である」と思われる規律は、**先着順**（もしくは先入先出（FIFO））であろう。しかし、後入先出（LIFO）のような別のサービス規律も考えられる。これは、たとえば、サラダバーに積まれている冷たい皿のような使い方である。つまり、きれいな皿が上に積まれ、顧客は上方から皿を取っていく様式である。その他にも、**最短ジョブ時間順**（SJF）あるいは**最短処理時間順**（SPT）とも呼ばれるような、待ち行列内のエンティティの違いに着目し、優先順位を利用するサービス規律もある。サービス規律 SJF では、待ち行列から選ばれる次のエンティティは、待ち行列内のエンティティ中でもっとも処理時間が短いものである（あらかじめ処理時間がわかっている、個々のエンティティに割り当てておく必要がある）。この規律を適用することで、短時間で済むジョブを、長時間かかるジョブの後ろに回すことなく、素早くサービスを提供することにより、すべてのエンティティに対する待ち行列での平均待ち時間の改善（削減）が期待できる。しかし、長時間かかるジョブは待ち行列の後ろに長い間押しやられる可能性もあるため、SJF が FIFO より「よい」かどうかは、平均滞在時間か、あるいは最大滞在時間か、どちらをより考慮するのかによる。ある種 SJF の反対ともいえる概念は、各エンティティにある値が割り当てられており（たとえば、サービス後の利益）、次のジョブとして、その値が最大のものを待ち行列から選ぶことだろう（まさに、時は金なり）。図 2.1 の病院システムにこれを当てはめると、最初に患者はいくつかの症状のレベルにトリアージされ、次に診察される患者は、もっとも深刻な症状の中から選ばれるようなサービス規律となる。同じ症状のレベルから患者を選ぶ必要がある際には、FIFO のようなタイブレイキングルールが適用される。

待ち行列システムで関心のある評価指標（あるいは出力指標）は、次のようなものである：

- ・ **待ち時間**：エンティティが待ち行列で費やした時間（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、各ステーションで別個に待ち時間を計ることもできるし、システムに到着してから去るまでに、各待ち行列で費やした待ち時間を患者ごとに積算することも可能である。
- ・ **滞在時間（システム内時間）**：待ち時間にサービス時間を加えた指標。待ち時間と同じく、待ち行列ネットワークでは、各ステーションで別々に、あるいは入口から出口までを包括的に、滞在時間を計測することができる。
- ・ **待ち行列内数（あるいは待ち行列長）**：待ち行列内のエンティティ数（サービス中のエンティティは含まない）。各ステーションで別個に、あるいはシステム全体で計測される。図 2.1 では、受付の待ち行列に 2 人、診療受付に 1 人、診察室に 4 人の患者が待っており、検査室および治療室には患者は待っていない。したがって、システム全体の待ち行列長は 7 となる。
- ・ **システム内人数**：待ち行列内数にサービス中のエンティティ数を加えた指標。同様に、各ステーションで別個に、あるいはシステム全体で計測される。図 2.1 では、受付に 4 人、診療受付に 2 人、診察室に 7 人、検査室に 1 人、治療室に 2 人の患者がおり、システム全体で 16 人の患者がいる。
- ・ **窓口（あるいは併存する別々の窓口群）の利用率**：稼働中（サービス提供中）である群内の個々の窓口（サーバ）の時間平均数を、群内の窓口総数で除算した指標。たとえば、診察室に 3 つの窓口があり、時刻 t において個々の窓口が稼働中（占有中）である数を $B_E(t)$ とする

と、利用率は、

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

となる。ここで、 h は運用中のシステムで観測される時間の長さ（あるいは範囲）である。

極めてまれな例外を除いて、待ち行列理論から得られる結果は、**定常状態**（あるいは**長時間実行、無限の範囲**）の条件下で導かれる。つまり、時間（実時間あるいはシミュレーション時間）が無限に続くという条件で、**定常状態の平均**としての結果が得られるのみである。ここで、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。各ステーションで別個に、あるいはシステム全体で計測される）
- 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。各ステーションで別個に、あるいはシステム全体で計測される）
- L_q = 定常状態における平均待ち行列長（各ステーションで別個に、あるいはシステム全体で計測される）。これは**時間平均**であり、通常の離散的な数値に対する平均とは異なるため、少し説明を加える必要があるだろう。 $L_q(t)$ 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 $L_q(t) \in \{0, 1, 2, \dots\}$ となる。時刻 t に対して $L_q(t)$ をプロットすると、待ち行列にエンティティが入る、あるいは出る瞬間に上下するように、0, 1, 2, …の水準で区別的に一定の曲線を描く。有限の時間軸 $[0, h]$ 上において、複数の待ち行列長の（時間）平均（单一の待ち行列を対象とする場合は、1つの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、各水準で $L_q(t)$ が費やす時間の割合で重みづけた、 $L_q(t)$ の水準0, 1, 2, …に対する**加重平均**である。よって、 $L_q = \lim_{h \rightarrow \infty} \overline{L_q}(h)$ となる。
- L = 定常状態における平均システム内人数（各ステーションで別個に、あるいはシステム全体で計測される）。 L_q と同様に、これも**時間平均**であり、 L_q と同様に定義される。 $L(t)$ は時刻 t における**システム内人数**（つまり待ち行列内数とサービス中のエンティティ数の和）として定義される。
- ρ = 単一の窓口あるいは併存する別々の窓口群の定常状態における利用率。一般にステーション毎に定義される。上述の診察室の例では、式 2.1 を $h \rightarrow \infty$ 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の多くは、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向けられており、本書で扱う範囲を超えている。ここでは、待ち行列理論の非常に典型的な特性である指数分布が、多くの応用や証明で重視されていることに触れておこう。たとえば、もっとも単純な待ち行列モデルでは、到着時間間隔（2つ連続したエンティティがシステムに到着する時間間隔）は、指数分布に従うものと仮定されており、これは個々のサービス時間についても同様である。より高度なモデルでは、これらの確率分布は指数分布から派生したもの（たとえばアーラン分布や、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）に一般化されたり、一般的な確率分布が用いられる。しかし、一般化することによって、結果の導出および適用がより複雑になる。

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することができる：

$$A/B/c/k$$

A は、到着プロセス、つまり到着時間間隔の分布を示す。 B は、サービス時間の RV である。 c は並列同一の窓口数である（図 2.1 で診察室は $c = 3$ 、診療受付は $c = 1$ である）。 k はシステムの容量である（すなわち上限であり、待ち行列と窓口の容量の和）。容量に制約がない場合（つまり $k = \infty$ ）、 $/\infty$ の記号は通常省略される。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される場合がある。 A と B に対するいくつかの選択肢を以下に示す。 M （マルコフ過程の頭文字）は、到着時間間隔およびサービス時間のいずれにおいても、指數分布に従うことを意味する。つまり、待ち行列モデル $M/M/1$ は、到着時間間隔およびサービス時間が指數分布に従うことがわかっている（ $M/M/1$ と表現できる。 m 個の IID の指數分布に従う確率分布の和から構成されるアーラン分布は、一般に E_m の記号が用いられる。したがって、待ち行列モデル $M/E_3/2/10$ は、到着時間間隔が指數分布、サービス時間がフェーズ 3 のアーラン分布、窓口が 2、一度にシステム内に存在できるエンティティの限度が 10（つまり待ち行列の容量は 8）の待ち行列システムを表している。記号 G は、到着時間間隔あるいはサービス時間に対して、それぞれ一般分布を用いたい場合に、 A あるいは B の部分に表記される。

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 の間にはいくつかの重要な関係式があり、いずれか 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して（あるいは推定して）求めることができる。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 の診察室だけ）を考える。さらに、次のいくつかの記号を導入する： $\lambda =$ 到着率（到着時間間隔分布の期待値の逆数）、 $\mu =$ サービス率（これは個別窓口のサービス率であり、複数窓口群に対するものではない。したがって、 $\mu = 1/E(S)$ である。ここに、 S は個別の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である）。

これらの関係式の中でもっとも重要なものは、リトルの法則である。この法則は、Little (1961) で初めて導かれ、徐々に一般化されてきた法則で（たとえば Stidham (1974)）、最近、50 周年を迎えて（Little 2011）。リトルの法則は、上述の記号を用いて、次のように簡潔に記述される：

$$L = \lambda W$$

ここでは、単一の複数窓口待ち行列ステーションについて考えるが、より一般的な状況でも適用できる。リトルの法則について注目すべきことは、時間平均（左辺の L ）とエンティティを基準に観測された平均（右辺の W ）が関係づけられている点である。

リトルの法則と同様に、ある待ち行列ステーションの待ち行列（窓口ではない）を次のように考えることができる：

$$L_q = \lambda W_q$$

リトルの法則よりも物理的に直感的な関係として、次の関係式が導かれる：

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

ここでは、少なくともサービス時間分布の期待値 $E(S)$ はわかっていると仮定している。これは、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すことができることを意味している。この関係式を、 $L = \lambda W$ および $L_q = \lambda W_q$ と共に用いれば、 W_q 、 W 、 L_q のいずれかがわかる。

れば、他の値を得ることができる。たとえば、 W_q がわかっている（あるいは推定できる）場合、代入して $L = \lambda(W_q + E(S))$ を得ることができる。同様に、 L がわかっていれば、少し変形して $W_q = L/\lambda - E(S)$ が得られる。

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 に示される待ち行列ネットワークのすべてではなく、診察室のようなステーションに関するもののみ）について、文献から公式を列挙する。その他の典型的な定常状態における平均評価指標を得るためにには、2.2 節に挙げたリトルの法則やその他の関係式を活用できることを想起してほしい。 $\rho = \lambda/(c\mu)$ は窓口群の利用率であり（前述のとおり、 c は待ち行列ステーションにおける個別窓口数である）、一般に ρ の値が大きいことはより高い混雑の予兆となる。これらの結果のすべてに対して、 $\rho < 1$ ($\rho \leq 1$ ではない) を仮定しなければならない（大部分の待ち行列理論に共通）。これらはすべて定常状態であり、長期間の実行でエンティティ数が限界なく増加し「爆発する」ことのないようなシステムとする必要がある。したがって、グループとしての窓口は、少なくともエンティティが到着する速度よりも速くサービスを提供できなければならない。

これから解説する 4 つの待ち行列モデルに加え、その他の待ち行列システムに関しても同様の式や結果が存在する。本章の初めに指摘したように、これらについては膨大な待ち行列理論に関する文献を参照してほしい。しかし、それらは極めて複雑な傾向にあり、クローズドな形の式を得られない場合もある。必要な評価指標が含まれない式の場合は、（実際に以下の 4 番目の例で行うように）様々な数値計算法や近似手法を用いて解く必要があるだろう。

2.3.1 M/M/1

おそらく、すべての待ち行列システムの中でもっともシンプルなものは、指數分布に従う到着時間間隔とサービス時間を持つ、単一の窓口のシステムである。

定常状態の平均システム内人数は $L = \rho/(1 - \rho)$ である。単一窓口の場合、 ρ は平均到着率 λ を平均サービス率 μ で除算したものである。 L は、次のように、別の形式で表現することができる：

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

L がわかっていれば、2.2 節の関係式を用いて、その他の評価指標 W_q 、 W および L_q を計算することができる。

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數確率分布に従うが、単一の待ち行列に対して並列同一の c 個の複数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。ここで本当に必要な唯一のものは、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、これは次のとおりである：

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

少し複雑であるが、 c は有限であるため、スプレッドシートでも計算できる数式であり、一般に極めて小さな値となる（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、 $0!$ は 1 と定義される）。たとえば、2.2 節の関係式から得られる他の評価指標

と $L = L_q + \lambda/\mu$ を用いることにより、次式が得られる：

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

$M/M/c$ についての上記の式はすべてクローズドな形である（すなわち、原則として数値を当てはめることができる）が、若干複雑である。特に $p(0)$ については、 c が非常に小さい場合を除いて、分母の総和は込み入ったものとなる。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 を参照）。このプログラムにより、到着率 λ 、サービス率 μ 、窓口数 c を指定することで、 L_q の値を計算することができる。`mmc.exe` を起動するには、まず Windows のコマンドプロンプトを開く必要がある（Windows の [スタート]-[Windows システムツール] フォルダ内、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は `C:¥>` と表示されるはずである。そこで、`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）。構文にしたがって `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を返す。返される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が何であるかは問わない（もちろん、 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が何かを明示する必要はない）。

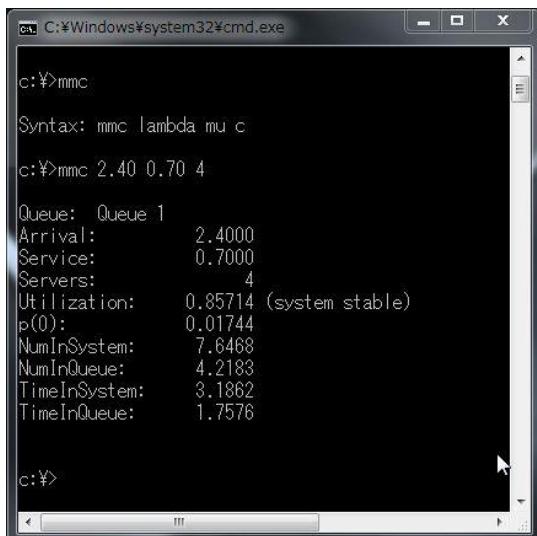


図 2.2 `mmc.exe` の利用例

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掲載されている。このモジュールは、単一窓口システムを解くことに加え、2.4 節で述べるジャクソンネットワークモデルを解くために使用することができる。

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であり、サービス時間の分布は一般分布である。これは、 $M/M/1$ モデルよりもより現実的なモデルである。 $M/M/1$ モデルは、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、これは現実には非常に極端な状況である。

ここで、 σ をサービス時間 RVS の標準偏差とすると、 σ の 2 乗はその分散となる。 $E(S) = 1/\mu$ であるので、次式を得る。

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

ポラチェック・ヒンチンの公式として知られる上式から、リトルの公式や2.2節の各関係式を用いて、 $L = \text{定常状態における平均待ち行列長}$ などの、各種評価指標を計算することができる。

ここで、 W_q はサービス時間分布の分散 σ^2 に依存し、分布の平均 $1/\mu$ に依存するわけではないことに注意が必要である。つまり、 σ^2 が大きくなると、 W_q も大きくなることを意味する。換言すると、時折、極めて長いサービス時間が発生すると、1つの窓口、つまりシステム全体を長時間占有する可能性がある。そのような状況下で新たにエンティティが到着すると、混雑が引き起こされることとなる（これは W_q やその他の評価指標から測定される）。サービス時間の分散が大きくなるほど、サービス時間分布の右裾が長くなることを意味する。したがって、上述の極めて長いサービス時間の状況が発生する機会が多くなるといえる。

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（もちろん、正の数しか知らない）、サービス時間が指数分布に従うものである。本書では、指数関数的なサービス時間は非常に極端な状況を生むとする立場を取る。しかしこの前提条件は、特にその無記憶性により、これから説明する結果を導いてしまうために、必要とされている。また、このモデルは単一窓口である。このモデルは、これまでの3つのモデルよりも複雑で分析が難しい。システム内人数を把握するだけでは、確率的に将来を予測するには不十分である。すなわち、指数関数以外（つまり無記憶性を持たない）の到着時間間隔は、直近の到着からどのくらいの時間が経過しているかも把握しておく必要があることを意味している。結果的に式は、クローズド系で、値を与えれば答えが出るというような状況でなければ、意味をなさないものとなる。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、連続分布の正の値だけをとることとする。またこれまで同様、 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ないし1の間の数とする。

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

以上より、次式を得る。

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

これまでのモデルと同様、上式から2.2節の関係式によりその他の評価指標を得られる。

ここで、式2.2を満たす z の値は何かという疑問を持たれることだろう。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・ラフソン法やその他の反復法による求解アルゴリズムに代表される数値計算法を用いる必要がある。そして、式2.2の右辺が求めるべき変数 z を含んだ積分であることにより、その計算過程は複雑なものとなる。

したがって、一般にこれは W の公式とはいえないだろうが、到着時間分布がある特定の場合には、なんとかできるかもしれない。たとえば、到着時間間隔が $[a, b]$ ($0 \leq a \leq b$) の連続一様分布の場合、確率密度関数は次のようになる。

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

これを式 2.2 に代入すると下式を得る。

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

若干計算すると、式 2.3 を得る。

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

式 2.3 は数値計算法により z について解くことができる。

Microsoft Excel のゴールシーク機能は、基本的に数値的な求解アルゴリズムであるため、うまく利用すれば解くことができる。Excel 2007/2010 のゴールシーク機能は、[データ] タブの [What-If 分析] の中にある。他のバージョンでは他の場所に配置されているかもしれない。図 2.3 に、Uniform_M_1_queue_numerical_solution.xls で Uniform/M/1 待ち行列についてゴールシーク機能を利用した様子を示す（このプログラムは、本書のウェブサイトの students セクションからダウンロードすることができる（付録 C 参照））。

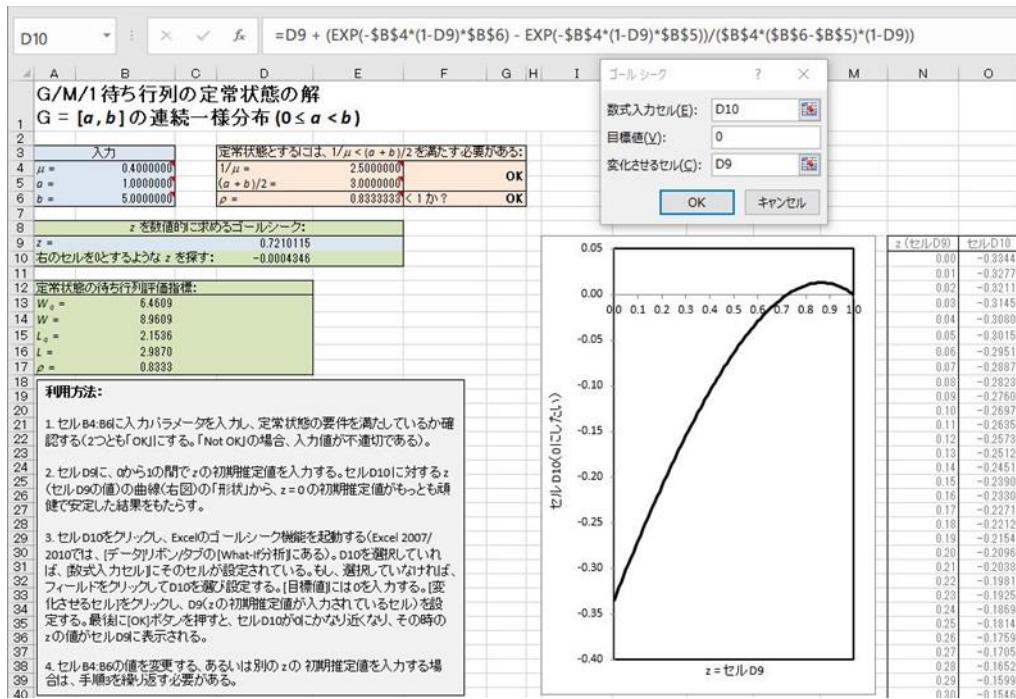


図 2.3 待ち行列評価指標を得るためにゴールシーク機能を用いる Excel シート

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用いて、セル D10 の値とセル D9 の値 (z の値) との差が 0 にもっとも近づく z の値を探す。 z の基底解として初期「推定値」を（0 から 1 の間で）設定する必要があり、セル D9 に入力しておく。実験はセル D9 の z の関数としてセル D10 に曲線の形状を示すため、セル D9 に $z = 0$ で初期化しておくことで、もっとも頑健で安定した結果をもたらす。図 2.3 では、設定後のゴールシークのダイアログと、探索完了後の結果が示されている。セル D10 が極めて 0 に近くなる近似解 z を得ると、標

準的な定常状態の待ち行列評価指標も計算される。図2.3の下部には、このスプレッドシートモデルのより詳しい利用方法が書かれている。

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意味するノード（図2.1の救急診療病院では診療受付や治療室）と、ノード間の経路（あるいはシステム外からノードへ、もしくはノードからシステム外への経路）を表す弧によって構成される。エンティティはシステム外からどのノードへも入ることができるが、この病院の例では受付ノードのみである。またエンティティは、どのノードからもシステム外へ出ることができるが、この病院例では診察室および治療室から離脱できる。システム内では、エンティティがあるノードを離れるとき、図2.1の分岐確率に基づいて次のノードに移る。たとえば、受付からは、確率0.9で診療受付に、確率0.1で検査室に移動する。すべてのエンティティが同じノードに向かう可能性もある。ここでは、診療受付および検査室ノードがそれに当たる。

ここで、システム外からのすべての到着過程は、互いに独立の指数分布に従うものとする（これはポアソン過程と呼ばれ、一定の時間間隔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箇所であり、そこからすべてのエンティティは受付ノードに向かう。さらに、すべてのサービス時間は互いに独立の指数分布に従い、入力到着過程からも独立であるものとする。また、すべての待ち行列の容量は無限である。最後に、各ノードの利用率（またはトラフィック密度） ρ は厳密に1よりも小さいものとし、定常にシステムは爆発しない。これらの前提により、これはジャクソン網と呼ばれ、多くの知見が得られている（Jackson 1957）。

病院の例で、受付ノードだけを見ると $M/M/2$ であり、所与の到着率を λ_{SignIn} と表す（つまり到着時間間隔は平均 $1/\lambda_{\text{SignIn}}$ のIIDの指数分布に従う）。注目すべきは、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 λ_{SignIn} ）と同率のポアソン過程となることである。換言すれば、窓口ステーションの出口を見ていると、その入口で目にすることと同じ確率的ふるまいを目の当たりにするだろう。さて、この例では、（偏りのある）コイントスを独立に行うことで、出力ストリームが分割される。つまり、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。したがって、2つの出力ストリームが存在することとなるが（診療受付に $0.9\lambda_{\text{SignIn}}$ 、検査室に $0.1\lambda_{\text{SignIn}}$ ）、それぞれは互いに独立したポアソン過程となる（これはポアソン過程の分流と呼ばれる）。そのため、診療受付は到着率 $0.9\lambda_{\text{SignIn}}$ の $M/M/1$ 待ち行列として分析でき、一方検査室は到着率 $0.1\lambda_{\text{SignIn}}$ の $M/M/2$ 待ち行列として分析できる。さらに、ここまで議論した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 λ_{SignIn} の $M/M/2$
 - ・ 診療受付：到着率 $0.9\lambda_{\text{SignIn}}$ の $M/M/1$
 - ・ 検査室：到着率 $0.1\lambda_{\text{SignIn}}$ の $M/M/2$
 - ・ 診察室：到着率 $0.9\lambda_{\text{SignIn}}$ の $M/M/3$
 - ・ 治療室：到着率 $(0.9)(0.6)\lambda_{\text{SignIn}} + 0.1\lambda_{\text{SignIn}} = 0.64\lambda_{\text{SignIn}}$ 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と呼ばれ、たんに到着率を合算すれば、全体の到着率を計算できる（なお合流した過程もポアソン過程となる）。

したがって、2.3節の $M/M/c$ の公式（あるいはmmc.exe）を用いることにより、定常状態の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、各ノード別々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。上述のリストより、各ノードの到着率は既知であるため、各ノードのそれぞれの窓口のサービス率がわかれば、その利用率を計算することができる。たとえば、診察室の3つの窓口のサービス率を μ_{Exam} としよう。すると、診察

室「ローカルの」トラフィック密度は、 $\rho_{\text{Exam}} = 0.9\lambda_{\text{SignIn}}/(3\mu_{\text{Exam}})$ となる。実際には、これらの利用率の計算は、（指数分布だけでなく）任意の到着間隔分布およびサービス時間分布において妥当である。ただし、到着あるいはサービス率は、それぞれ到着間隔およびサービス時間 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、病院の例では存在しなかったものの、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。その場合、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たんなる和となる。また、自己ループが存在するかもしれない。たとえば、治療室からの出力を例に取ると、80%の患者はシステムを離れるが、20%の患者は（おそらく再治療のために）治療室に戻る可能性もある。そのような場合、 $\lambda_{\text{TreatmentRooms}}$ について一次方程式を解く必要があるだろう。しかしながら、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 を下回っていることを確かめる必要がある。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) を決定することに関心がある。まず、各窓口への到着率を決定する。窓口のトラフィック密度が厳密に 1 より小さい場合、退去率は正確に到着率と同一となる。ここでは、すべての窓口についてこれが成り立つと仮定し、検証していくこととする。

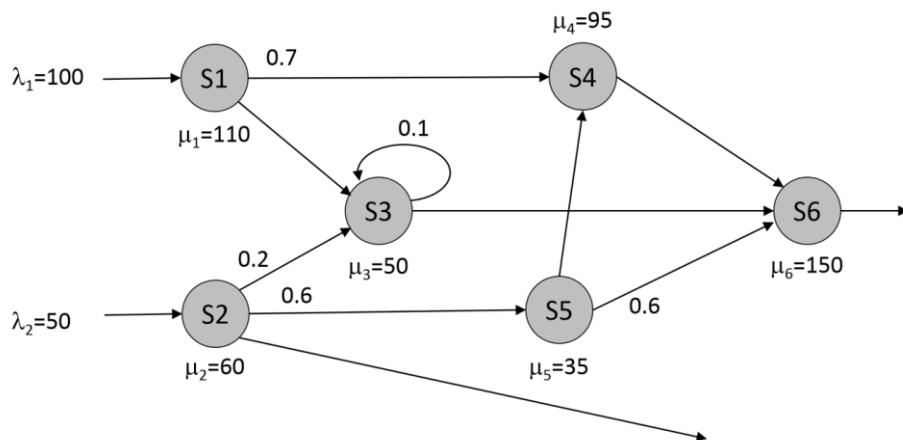


図 2.4 待ち行列ネットワークの例。すべての窓口で $c=1$ を仮定

到着率は次のように計算される：

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

これらの到着率と図 2.4 に示すサービス率、およびすべての窓口における $c=1$ の仮定について、2.3 節の式を適用して、各窓口 i に対する L_i 、 W_i および ρ_i を得る（表 2.1）。

表 2.1 図 2.4 のネットワークにおける個別の窓口に対する解

	1	2	3	4	5	6
λ_i	100.00	50.00	44.44	82.00	30.00	140.00
μ_i	110.00	60.00	50.00	95.00	35.00	150.00
ρ_i	0.91	0.83	0.89	0.86	0.86	0.93
L_i	10.00	5.00	8.00	6.31	6.00	14.00
W_i	0.10	0.10	0.18	0.08	0.20	0.10

ローカルトラフィック密度は厳密に 1 より小さいが、次のように、ネットワーク全体の平均システム内人数 L 、および平均滞在時間 W を求めることができる：

$$L = \sum_{i=1}^6 L_i$$

$$W = \frac{L}{\lambda_1 + \lambda_2}$$

本書で後ほど述べるとおり、この種の解析は、たとえ標準的な待ち行列理論の仮定が、シミュレートしようとする所与のシステムに当てはまらないとしても、モデルの検証には極めて有用である。

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に関する利点は、統計的な変動に左右されず、厳密であることである（ただし、2.3.4 項の $G/M/1$ 待ち行列のように、解を得るために数値計算が必要となる場合には、丸め誤差があるかもしれない）。後述するように、シミュレーションの結果は厳密ではなく、統計的な不確実性が存在する。そのため、その不確実性を適切に認識し、測定し、対処しなければならない。

しかし、待ち行列理論には別の短所があり、それは主に 2.3 節の数式を導出するために必要な前提条件にある。多くの現実的な状況では、これらの前提はおそらく当てはまらず、結果の正確性、ひいてはモデルの妥当性に与える影響がどの程度かを指摘することは非常に困難である。指数分布に従うサービス時間を仮定することは、指数分布のモードが 0 であることから、多くの状況で特に非現実的であるだろう。たとえば、空港の保安検査場で、サービス時間が、厳密に 0 より大きな特定の値ではなく、0 に非常に近いことが頻繁に起こると考えられるだろうか？上述したように、このような結果は定常状態の評価指標に関して成り立つものであり、短期の（あるいは有期の、もしくは過渡的な）評価に关心がある場合にはあまり役立たない。そして、近似手法が相当正確なものでない限り、待ち行列理論の結果は到着時間間隔およびサービス時間に利用されるあらゆる分布に対して、普遍的に利用できるものではない。

一方、シミュレーションは、短期（あるいは有限）の時間フレームをもっとも簡単に扱うことができる。実際に、シミュレーションでは、定常状態を扱うことは比較的難しい。というのは、非常に長い時間の実行が必要になることや、定常状態として相応しくない初期条件の影響による偏りに注意しなければならないからである。シミュレーションを用いる場合、研究対象の実システムに適合すると思われる任意の到着間隔およびサービス時間の分布を利用することが可能である（6.1 および 6.2 節参照）。特に、実データの分析から指数分布が適していると判断される場合を除き、すべての分布が指数分布であるとの前提を置く必要はない。このため、本書ではシミュレーションモデルは、現実のモデルをより現実的に、より妥当なものにする機会を提供するものであると考えている。特に、モデル構造が厳密な数理解析を行うことが困難なほど非常に複雑で、標準的な待ち行列モデルをはるかに超えているような場合に有効であろう。唯一の難点は、シミュレーション結果は統計的な推定値であり、正当で正確な結論を導くことができるように、適切な統計手法を用いて分析しなければならないことである。この点に関しては、本書でたびたび確認することにする。図

2.1 の病院例および本章を通して議論してきた事柄は、(適切な統計解析手法も交えて) Simio を用いて本書で実際にシミュレートすることにする。また、9 章ではより現実的な枠組みに発展させることにする。

2.6 問題

1. 平均到着時間間隔 1.25 分、平均サービス時間 1 分の $M/M/1$ 待ち行列について、 W_q 、 W 、 L_q 、 L および ρ を求めなさい。また、それぞれについて、言葉で説明しなさい。その際、すべての単位と時間を明記すること。
2. 問題 1 を発展させ、サービス時間に指数分布ではなく、 $a = 0.1$ ないし $b = 1.9$ の（連続）一様分布を仮定する。ここで、この一様分布の期待値は $(a+b)/2 = 1$ となり、問題 1 のサービス時間の期待値と同じである。5 つの評価指標を問題 1 の数値と比較し、サービス時間分布の変更の影響を説明しなさい（サービス時間の期待値は 1 のままである）。ヒント： a ないし b 間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 を発展させ、サービス時間として、最小値 $a = 0.1$ 、最大値 $b = 1.9$ 、最頻値 $m = 1.0$ の三角分布を仮定する。5 つの評価指標を問題 1 および問題 2 の数値と比較しなさい。ヒント：最頻値 m で a ないし b 間に分布する三角分布の期待値は $(a+m+b)/3$ 、標準偏差は $\sqrt{(a^2 + m^2 + b^2 - am - ab - bm)/18}$ である（ただし、 $a < m < b$ ）。
4. 問題 1、問題 2、問題 3 のそれぞれにおいて、到着率を徐々に増加させると何が起こるだろうか（たとえば、窓口 1 の理髪店で広告やクーポンによって事業を拡大したいというような状況を想像してほしい）。スプレッドシートやコンピュータプログラムを作成し、元の到着率の 1%、2%、3% と、システムが非定常 ($\rho \geq 1$) となるまで到着率を 1%ずつ増加させ、 W_q 、 W 、 L_q 、 L 、 ρ の 5 つの評価指標を再計算しなさい。到着率の増加率を横軸に 5 つの評価指標をプロットし、それをもとに議論しなさい。
5. 問題 1 を発展させ、それぞれの窓口の平均到着時間間隔 1.25 分、平均サービス時間 3 分の $M/M/3$ 待ち行列として、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コンピュータプログラムを作成する、もしくは mmc.exe を利用するとよい。
6. 問題 5 について、問題 4 と同様に到着率を増加させ、シナリオごとに W_q 、 W 、 L_q 、 L 、 ρ の 5 つの評価指標を求めなさい。到着率を 1%ずつ増やしながら、元の到着率を 100% 上回る（つまり 2 倍になる）まで続け、必要に応じて窓口を追加しなさい（ただし、定常性を確保するための最低限の窓口数とする）。
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、平均 6 分の指数分布に従う到着時間間隔で、患者が来院するとしよう。各ステーションの窓口数と分岐確率は図 2.1 の通りである。各ノードのサービス時間は指数分布に従い、平均はそれぞれ次の通りである（単位は分）：受付 3、診療受付 5、検査室 90、診察室 16、治療室 15。5 つのステーションのそれぞれについて、「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だろうか。つまり、外来患者の来院にうまく対処できるだろうか。運営の可否について、理由を述べなさい。窓口を 1 つ増やせるとしたら、5 つのステーションの内、どこに追加すればよいだろうか。理由と共に述べなさい。ヒント：スプレッドシートやコンピュータプログラム、あるいは mmc.exe を利用するとよい。
9. 問題 8 において、5 つの各ステーションについて、 W_q 、 W 、 L_q 、 L 、 ρ を計算し、文章で説明しなさい。問題 8 で任意のステーションに窓口を追加したが、この問題でも同じ意思決定をするか否か。理由と共に述べなさい（来院するのは重篤な症状の方を含む病気の方々であり、工場で加工される部品と同じように考えることはできない点に注意してほしい）。ヒント：スプレッドシートやコンピュータプログラムを利用することを強く勧める。次善の策として mmc.exe の利用もありうる。

32 第2章 待ち行列理論の基礎

14. 問題8および問題9（追加窓口を導入しない場合）において、流入率は毎時10であった（平均6分の到着時間間隔と同じ意味である）。病院全体の運営を可能にしながら、この流入率をどの程度高めることができるだろうか（つまり、どの程度まで外来患者の負荷に対処できるだろうか）。この問題に答えるために、必要以上の計算をしなくてもよい。正確な解を求めるには、求解アルゴリズムを利用したコンピュータプログラムを作成するか、Excelのゴールシーク機能を用いるのもよい。
15. 問題8および問題9（追加窓口を導入しない場合）において、予算削減を受け、現在10ある窓口の1つを閉じる必要に迫られている。ただし、各ステーションにつき、最低1つは窓口が必要である。病院全体の運営に与える影響がもっとも軽微で済むのは、どの窓口を削減することか。さらに閉鎖する窓口を2つに増やしても、病院の運営は継続できるか。それ以上の窓口を削減することについてはどうか。

第3章 シミュレーションの種類

シミュレーションには非常に幅広く、多様なトピックがある。また、シミュレーションには多くの種類があり、遂行する方法も多岐にわたる。本章では、いくつかを紹介し、用語を解説する。

3.1 節では、いくつかの観点からシミュレーションの種類を分類する。3.2 節では、シミュレーションの種類の 1 つである静的かつ確率的（別名、モンテカルロ）シミュレーションについて 3 つの例を挙げる。3.3 節では、Simio のような専用の動的シミュレーションソフトウェアを用いずに、動的シミュレーションを行う 2 つの方法（手作業およびスプレッドシート）を紹介する。続く 3.4 節では、動的シミュレーションを行うソフトウェアの選択肢について述べる。3.4.1 項では、C++ や Java、Matlab などの汎用プログラミング言語で、3.3.1 項のシミュレーションロジックを遂行する方法を簡単に述べる。最後に 3.4.2 項にて、Simio などの専用シミュレーションソフトウェアについて、簡単に述べる。

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、活動および手法を包含する、素晴らしい言葉である。しかし、この多様さはときに「シミュレーション」を漠然とした、ほとんど無意味な言葉にしてしまう可能性もある。本節では、モデル構造によりシミュレーションの種類を分け、それぞれの例を紹介する。シミュレーションの分類には様々な方法があるが、ここでは静的／動的、連続変化／離散変化（動的のみ）、確定的／確率的の 3 つの分類法を採用する。本書では、主に動的で、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は、モデルの動作および実行に関して、時間の経過が意味のある役割を果たさないモデルのことである。たとえば、乱数ジェネレータを使用して、ギャンブルやロトをシミュレートしたり、積分の値や行列の逆行列を推定したり、あるいは会計上の損益計算書を評価することが挙げられる。モデル内に時間の概念は存在するが、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、モデルは静的である。たとえば、3.2 節で扱うスプレッドシートを用いた单一期間の在庫シミュレーションなどが例として挙げられる。

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および動作の本質的で明示的な役割を果たす。そこでは、シミュレーション上の時間経過を表現すること抜きでは、モデルの構築や実行は不可能である。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が起こるように時間を表現する必要があるため、ほぼ常に動的である。在庫シミュレーションでも、時間と何か、たとえば在庫水準や在庫補充方策など、との間に、論理的な関連がある場合は、動的になりうる。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、時間にわたって出発や移動、到着を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。それは、シミュレーション上の時刻の現在値を表す変数であり、シミュレーションの進行に伴って増加し、モデル全体からアクセスできるグローバル変数である。もちろん、シミュレーションクロックの時間単位について、モデル内で一貫性を保たなければならない。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）であり、物事が発生した瞬間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正確さで表現できるといえる（たとえ完璧でなくとも極めて正確である）。しかし、古いシミュレーションソフトウェアの一部や、戦闘シミュレータのような特殊なシミュレータの一部では、モデル作成者が、たとえば 1 分など、決められた小さな時間ステップ（時間スライスあるいは時間増分）を選択しなければならない場合もある。

このような場合、シミュレーションクロックは、その時間ステップでのみ（たとえば1分毎に）時刻を先に進め、その時点でモデル変数の更新が必要な事柄が発生したかどうかを評価することになる。これには2つの問題がある。第一に、計算時間の多くが、毎分ごとに何か発生したかどうかをチェックするためだけに浪費され、もし何も起こらなかった場合にはこの確認はまったく無駄になることである。第二に、一般に現実では物事は連續した時間のあらゆる時点で発生する可能性があり、前述の方法では1分の範囲に発生を強制するため、モデルに時間の誤差が生じる。この誤差は、時間ステップとして秒あるいはミリ秒、ナノ秒を選択することによって削減できるが、シミュレーションの実行時間の問題を悪化させるかもしれない。したがって、時間ステップ型のモデル、つまり整数型のクロックには、重大な欠点があり、ほとんどの状況においてメリットはないといわざるをえない。

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る。おそらく@RISKやCrystal Ballのような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連携して用いられるだろう。しかし、動的シミュレーションには、一般にその用途向けに作成された、より強力なソフトウェアが必要とされる。3.3.1項では、本書がもっとも重要だと考える種類のシミュレーション、すなわち動的離散型シミュレーションのロジックに関して、スプレッドシートを用いてデータを追跡することにより説明する。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えるが、それ以上のモデルは不可能であろう。長年にわたり、動的シミュレーションモデルを構築するために汎用プログラミング言語を用いてきたが、このアプローチは非常に退屈で、面倒で、時間を浪費するばかりであり、構築するシミュレーションモデルが大規模で複雑な場合はエラーを招きやすい（ただし、汎用プログラミング言語は、非常に複雑で特別なロジックを含むシミュレーションモデルの一部を構築する目的で、現在でも利用されることがある）。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ソフトウェアで構築することが非常に適している。

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、これらの状態変数は待ち行列長や待ち行列に顧客が到着する回数、窓口の状態（遊休、稼働、使用不能）などになるだろう。在庫シミュレーションでは、在庫水準や発注量などが状態変数に相当する。

シミュレーションモデル内の状態変数が連続的な時間に対して継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、タンク内の水の量は、水が上部から注がれ、下部から流れるにつれ、連続的な時間の任意の時点で、連続的に変化している（すなわち、微小に増加／減少している）。しばしば、このようなモデルは、状態が経時的に変化する割合と、その状態変数の水準や、他の状態変数の水準あるいは割合との間の関係を決める微分方程式で記述される。ときには、個々の車両に基づく交通流のように、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、交通を液体のように捉えている）。厳密にいうと連続的にモデル化されている離散的な状態の例として、有名な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、 $x(t)$ および $y(t)$ は、戦闘における時刻 t の両軍の戦闘力（兵員数）を表す。ランチェスター方程式には多くのバリエーションが存在するが、もっともシンプルなものは近代戦モデルと呼ばれる。これによれば、以下の式のように、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：

$$\begin{aligned}\frac{dx(t)}{dt} &= -ay(t) \\ \frac{dy(t)}{dt} &= -bx(t)\end{aligned}\tag{3.1}$$

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）である。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。もちろん式 3.1 の 2 つ目の方程式は、同様に y 軍の損耗率を表しており、定数 b は x 軍の戦闘力である。ここで紹介したランチエ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必要ない。しかし、より複雑なランチエスター方程式は解析的に解くことができず、シミュレーションが必要とされるだろう。

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はこの種類に該当する。待ち行列長や窓口の状態を記述する状態変数は、顧客の到着や顧客に対するサービスの完了、窓口の休憩、故障などのような、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においてのみ成立する。つまり、連続するイベント間に発生することを（実際何も起こらないため）目にすることはないのである。したがって、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ジャンプする。これらの各時点において、シミュレーションモデルは、発生する事柄と、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の変数）に対して加えるべき変更を評価しなければならない。

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ついて、各部品のサービス時間が一定で、部品間の到着時間間隔も一定（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、モデルを何度も実行しても同じ結果が得られる。

このようなモデルは、ありえ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、（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ような）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。これらの入力確率分布や確率過程を特定することは、まさにモデル構築の一部であり、6 章で議論する。そこでは、シミュレーションにおける確率分布や確率過程からの無作為抽選の方法を扱う。確率モデルは、根底では一種の乱数ジェネレータにより駆動しているため、モデルを一度実行すると、出力としてたまたま発生した 1 つのサンプルしか得られない。つまり、現実の生産ラインの 1 日の稼働により生み出される当日の生産量を観測すること、あるいはサイコロを一度振って、そこから 4 の目が出るのを目にする同じことと同じである。生産ラインのある 1 日の稼働を観察しただけで、その日に起きたことが毎日発生すると結論づけることは明らかにできないだろうし、サイコロを一度振って 4 が出たからといってサイコロの他の面も 4 であるとはいえない。したがって、結果のふるまいについて何らかの洞察（たとえば、生産量が目標に達しているか、あるいはサイコロのイカサマの有無など）を得たいのであれば、生産ラインを何度も稼働させる（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の結果のふるまいから何らかの洞察（たとえば、重要な評価指標の平均や標準偏差、範囲など）を導きたいのであれば、モデルを何度も実行する必要がある。これは、本書の 4 章および 5 章で取り扱う話題である。

確率的シミュレーションから得られる出力におけるランダム性あるいは不確実性を厄介なものとみなして、入力確率分布をその平均値で置き換えることにより不確実性を排除するように誘惑されるかもしれない。これは、平均値分析と呼ばれる不合理ではないアイディアである。これは、確かにシミュレーション出力を非ランダムで確定的なものにするが、同時に誤った解釈をもたらすこ

とが非常に多い。多くの場合、不確実性は現実の一部であり、出力パフォーマンス指標には不確実性が含まれる。しかし、それ以上に、モデル入力における変動性は、多くの場合、出力指標の平均値に大きな影響を与える。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純な単一窓口待ち行列を考える。待ち行列理論（2章参照）によれば、定常状態（すなわち長時間の実行）であれば、サービス開始前の待ち時間の期待値は8.1分である。しかし、指数分布に従う到着時間間隔およびサービス時間を平均値（それぞれ正確に1分および0.9分）に置き換えると、明らかに待ち行列には何も並ばなくなる。したがって、待ち行列での待ち時間の平均は0となり、正解の8.1分と比べて非常に誤ったものとなる。直感的にいえば、到着時間間隔およびサービス時間における変動は、多くの短い到着時間間隔を連続的に生成したり、長いサービス時間を連続的に生成したり（あるいはただ1つの非常に長いサービス時間を生成したり）する可能性がある。いずれも混雑をもたらし、待ち行列での待ち時間を長くするだろう。これは、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）。

さて、シミュレーションソフトウェアを利用して確率的モデルを実行すると、実際には毎回まったく同じ数値結果が得られるのであるが、この事実は初心者を大いに戸惑わせる。乱数ジェネレータを用いてモデルが実行されているため、このようなことは起こり得ないと思われることだろう。しかし、6章で議論するように、多くの乱数ジェネレータは真にランダムではなく、毎回同じ「無作為な」数列を生成する。したがって、同一のシミュレーション実験において、モデルを複数回反復実行する必要がある。これは、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ことを意味する。それにより、各反復実行において異なる（無作為な）出力が得られ、結果として不確実性に関する重要な情報がもたらされる。シミュレーションで利用される基本的な乱数の再現性は、デバッグ用として、あるいは分散減少法により出力結果の精度を高めるために必要とされている。この点に関しては、Banks et al. (2005) や Law (2015) を参照してほしい。

3.2 静的かつ確率的なモンテカルロ

おそらく、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、時間の経過がない、つまり時間の概念を持たないシステムのモデルに取り組むことであった。これらのモデルは、モンテカルロあるいは確率的モンテカルロと呼ばれる。地中海のギャンブル天国へのオマージュとして名づけられている。本節では、4つの簡単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である。2つ目（Model 3-2）はモンテカルロシミュレーションを用いて、解析的に解くことが難しい積分の値を推定する。3つ目（Model 3-3）は伝統的な単一期間の生鮮品在庫問題のモデルである。最後に、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、本書では「Model x-y」の表記は、x章のy番目のモデルであることを表している。付録Cに記載した本書のウェブサイトの「students」からダウンロード可能な関連ファイルは、「Model_x_y.拡張子」と命名されている。拡張子としては、Excel ファイルでは.xls であり、Simio プロジェクトファイルでは.spfx である。したがって、これから議論に対応するファイルは、Model_03_01.xls である。

本節の例題では、普及度と習熟度を鑑みて Microsoft Excel®を用いるが、この種のシミュレーションはC++やJava、Matlabのような汎用プログラミング言語においても簡単に構築できる。

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、それぞれ確率1/6で整数1, 2, …, 6の目を出すため、合計は整数2, 3, …, 12のいずれかになる。和の確率分布を見つけることは簡単で、一般に初級の確率論の書籍で扱われているため、本来はこのゲームをシミュレートする必要はないだろうが、静的モンテカルロの概念を説明するために、とにかくシミュレートして

みよう。

Model_03_01.xls (図 3.1 参照) で、Excel によりこのシミュレーションを構築した。ここでは、2 つの偏りのないサイコロを 50 回投げるシミュレーションを行い、50 回分の和を記録した。出る可能性のある目の値がセル C4:C9 に入力されている。各目の確率がセル A4:A9 に入力されていて、サイコロがその目の値より小さくなる確率がセル B4:B9 に入力されている (B4:B9 の確率は各サイコロをシミュレートする際に非常に便利である)。列 E はサイコロを振った回数であり、2 つのサイコロの出目が列 F および列 G に表示され、その合計が列 H に表示される。セル J4:L4 には、50 回分の和に対する基本的な要約統計量が計算される。スプレッドシート Model_03_01.xls を開くと、図 3.1 とは異なる結果が示されることに気づくだろう。これは、Excel の乱数ジェネレータ (後述) がいわゆる撃発性を持っており、スプレッドシートが再計算される度に「新しい」乱数を再生成するためである (これはファイルを開く際にも行われる)。

A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	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												
2	P(目 = i)	P(目 < i)	i	回数	サイコロ1	サイコロ2	和		和の平均	和の最小	和の最大		
3	0.1667	0.0000	1	1	1	6	7						
4	0.1667	0.1667	2	2	4	2	6						
5	0.1667	0.3333	3	3	3	4	7						
6	0.1667	0.5000	4	4	2	5	7						
7	0.1667	0.6667	5	5	1	3	4						
8	0.1667	0.8333	6	6	5	5	10						
9	0.1667	1.0000 ← 検算	7	7	6	3	9						
10			8	8	6	4	10						
11			9	9	4	2	6						
12			10	10	6	2	8						
13			11	11	3	6	9						
14			12	12	5	1	6						
15			13	13	2	5	7						
16			14	14	2	1	3						
17			15	15	2	1	3						
18			16	16	2	1	3						
19			17	17	2	5	7						
20			18	18	1	3	4						
21			19	19	2	2	4						
22			20	20	5	2	7						
23			21	21	1	6	7						
24			22	22	5	6	11						
25			23	23	3	4	7						
26			24	24	3	5	8						
27			25	25	5	1	6						
28			26	26	1	4	5						
29			27	27	3	5	8						
30			28	28	6	6	12						
31			29	29	2	4	6						
32			30	30	5	6	11						
33			31	31	4	6	10						
34			32	32	4	3	7						
35			33	33	5	1	6						
36			34	34	2	5	7						
37			35	35	2	2	4						
38			36	36	1	3	4						
39			37	37	3	2	5						
40			38	38	1	2	3						
41			39	39	4	1	5						
42			40	40	1	2	3						
43			41	41	2	4	6						
44			42	42	6	5	11						
45			43	43	3	2	5						
46			44	44	3	2	5						
47			45	45	6	5	11						
48			46	46	3	4	7						
49			47	47	1	2	3						
50			48	48	3	4	7						
51			49	49	6	6	12						
52			50	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心臓部は、セル F4:G53 に各サイコロの目を生成、あるいは「シミュレート」することにある。これを 2 つのステップで議論したい。最初のステップは、Excel の乱数ジェネレータである RAND() を用いていることであり、0 ないし 1 の連続一様分布に従って個々の観測値を生成しているであろうことを意味する。「であろう」としたのは、RAND() のアルゴリズムが「真」の無作為性で乱数を生成するのではなく、0 ないし 1 の間に一様に分布するように「見える」観測値を生成するための、特別な数学的手順に基づいているからである（もちろん公式な統計的推定の検定はクリアしている）。これらは、擬似乱数と呼ばれるが、以下では単に乱数と記すこととする。乱数ジェネレータの処理内容については 6 章で詳説するが、ここでは確かな数学研究に基づい

た複雑なアルゴリズムであると理解しておけばじゅうぶんだろう。Excel の RAND() が用いている手法には批判もある。ここでは例示の目的で使用するだけであるが、Palisade Corporation の @RISK® (www.palisade.com) に代表される Excel の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などで利用される乱数ジェネレータの方がより安心かもしない。

第2のステップは、連続的な観測値、すなわち RAND() から得られる [0,1] で連続一様に分布する U を、サイコロに偏りがないことから、各整数 $1, 2, \dots, 6$ の内の 1 つに変換することである。確率変数からの観測値（ときに確率変量と呼ばれる）の生成に関する一般的な考え方は 6 章で議論するが、ここでの目的に対して直感的なスキームを考え出すのは簡単である。連続区間 $[0, 1]$ を、各幅を $1/6$ とする 6 つの連続部分区間 $([0, 1/6], [1/6, 2/6], \dots, [5/6, 1])$ 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終端が閉じているか開いているかは重要ではない。すなわち、 U は確率 0 の場合のみ境界上に位置するため、このことを心配する必要はない。 U が $[0, 1]$ にわたって連続一様に分布していることを想起すれば、各部分区間の幅 $1/6$ に等しい確率でいずれかの部分区間に分類されるため、1 番目の部分区間にいる場合は値 1 が生成され（あるいは返され）、2 番目の部分区間にいる場合は値 2 が、というように 6 つの部分区間で同じことが起こる。これは、セル F4:G53 に次の式を入力することで実装される（すべてのセルには同じ式が入る）：

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

第4引数に TRUE を設定した Excel 関数 VLOOKUP は、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、その行に対応した B4:C9 の 2 番目の列（第3引数が 2 であるため）から値を返す ($\$B\$4:\$C\9 の \$ は、列 F および列 G でこの式をコピーした際に、参照元を特定のセル範囲に固定させる)。したがって、RAND() が B4 と B5 の間の場合、つまり 0 ないし 0.1667 の間であり、約 $1/6$ の幅の 1 番目の部分区間である場合、希望通り C4 (すなわち 1) を得る。RAND() がこの 1 番目の区間にない場合は、VLOOKUP が 2 番目の部分区間である B5 および B6、すなわち 0.1667 ないし 0.3333 の間に移る。そして、RAND() がこれら 2 つの値の間にあるならば C5 (つまり 2) を返し、それはおおよそ確率 $0.3333 - 0.1667 = 1/6$ で起こる。この動作が必要に応じて最後の区間（すなわち、RAND() が 0.8333 ないし 1 の間であり、幅が約 $1/6$ の区間）まで続き、その際には C9 (つまり 6) を得る。VLOOKUP 関数に詳しくなければ、インターネットで検索するか、Excel のオンラインヘルプを調べてほしい。

和の範囲が 2 ないし 12 であり、和の真の期待値が 7 であることがわかっているので、J4:L4 の要約統計量は妥当である。Excel の RAND() 関数は揮発性であるため、そのすべてのインスタンスは、F9 キーを押すなど（他にも再計算の方法はあるが）スプレッドシートが再計算される度に、新しい乱数を再生成する。したがって、何度か試行し、個々のサイコロの目の変化や、J4 に表れる和の平均（あるいは K4 および L4 の和の最小および最大の値）の変化に注目してほしい。

このアイディアは、取り得る値が有限（ここでは 6 つの値）である限り、どんな離散確率分布にも拡張できるし、不等確率分布（たとえばイカサマのサイコロで、6 つの面の出る確率が等しくないケース）にも応用可能だろう。このサイコロ投げシミュレーションに関する応用例は、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、その形状から $h(x)$ の不定積分を導くのが難しく（あるいは不可能で）、計算する方法がないものとしよう。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、それが真でない場合でも機能するものとしよう。これにアプローチする通常の方法は **数値解析** である。すなわち、範囲 $[a, b]$ 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、あるいは手の込んだ式

によって) 近似解を求めるにより面積を計算し、最後にすべての面積を合計する。区間を狭めれば狭めるほど、この数値解析は、積分の真の値の近似値としてより正確になる。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作成することも可能である。

この方法を行うには、まず a ないし b の間に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 章で議論するが、ここでは直感的に式を導くことにする。 $U = \text{RAND}()$ は 0 ないし 1 の間に連続一様に分布するため、 $b - a$ を掛けて U の範囲を「拡張」すれば、分布の一様性を歪めずに、0 ないし $b - a$ の間に一様に分布するように範囲を「拡張」できるだろう(「拡張」という表現は、 $b - a > 1$ を念頭に置いていることを暗に意味しているが、 $b - a < 1$ の条件下でも同様である。後者の場合は「拡張」ではなく「圧縮」という表現が適切であろう)。それから範囲に a を加えれば、幅は $b - a$ のまま、希望する $[a, b]$ へと右側にシフトすることができる。また、この変換は線形であるため、シフト後の範囲も一様に分布することになる(ここでは $a > 0$ を前提としているため「右側にシフト」という表現を使ったが、 $a < 0$ の場合には、 $a < 0$ を範囲に加えて「左側にシフト」させる)。

次に、前段落でしたように、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 を評価し、最後に $b - a$ を掛けて確率変量 $Y = (b - a)h(X)$ を求める。 h 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 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、長方形の面積を求ることと同等になる。しかし、仮に h 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 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 $h(x)$ 下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 $\phi_{\mu, \sigma}(x)$ と表記され、次式で与えられる:

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

ここで x は実数である。 $\phi_{\mu, \sigma}(x)$ 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れが理由である)。したがって、これはモンテカルロ積分についての適切なテストケースであり、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 参照)で、Excel によってこのシミュレーションを作成した。 $a = 4.5$ ないし $b = 6.7$ の間に連続一様に分布する X ($i = 1, 2, \dots, 50$ について X_i と表記する) の 50 のサンプルをシミュレートし(列 E)、列 F に $(6.7 - 4.5)\phi_{5.8, 2.3}(X_i)$ の値を 50 個得る。F 列では、密度関数 $\phi_{5.8, 2.3}(X_i)$ を評価するために、Excel 関数 NORMDIST を用いている。列 E および列 F に入力された式の詳細は、読者自身で確かめてほしい。列 F の 50 個の値の平均値はセル H4 に計算されており、

セルI4はNORMDISTを用いた正確なベンチマークの値である。F9キーを何度か押すことによって、シミュレーション結果における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方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次元の積分においては、モンテカルロ積分が用いられることはめったにない。より高い次元の積分や不規則な形状の積分に関する推定において、モンテカルロ積分が用いられることが多い。この例題のモンテカルロ積分に関する分析および拡張は、問題3および問題4を参照してほしい。



図3.2 正規密度関数の静的モンテカルロ積分

3.2.3 Model 3-3：単一期間在庫の収益

いま、あなたがスポーツ用品店を経営しており、地元のスポーツチームがリーグ優勝した場合、チームのロゴが入った帽子をいくつ発注するべきか考えてみよう。過去の経験に基づき、来月（ピーク期）の需要 D の推定値は、 $a = 1000$ と $b = 5000$ の間であろう。それは一様分布を仮定しているが、帽子は離散的な品目であるため、 $a = 1000, 1001, 1002, \dots, 5000 = b$ の帽子がそれぞれ売れる事象は、確率 $1/(b + 1 - a) = 1/(5000 + 1 - 1000) = 1/4001 \approx 0.000249938$ となる。あなたはサプライヤから卸売単価 $w = \$11.00$ で帽子を仕入れ、ピーク期に小売価格 $r = \$16.95$ で販売する。ピーク期に完売した場合、再発注の機会はないため、追加売上（および利益）を逃すことになる。一方、ピーク期後に帽子が残っている場合、在庫処分価格 $c = \$6.95$ の値をつけて販売するため損失が発生する。ここで、残った帽子はこの在庫処分価格ですべて売り尽くせるものとする。以上の条件から、利益を最大にするにはどれだけの帽子を発注すればよいだろうか。

ピーク期に帽子を1つ売ると、 $r - w = \$16.95 - \$11.00 = \$5.95$ の利益を得るため、ピーク期の需要をすべて満たすように発注を行いたい。しかし、ピーク期を逃すと帽子1つにつき $w - c =$

$\$11.00 - \$6.95 = \$4.05$ の損失が生じるため、過剰あるいは過少に発注することは避けたい。需要が実際にわかるまで待って、その数の帽子を注文したいのはやまやまであるが、生産が間に合わず顧客は待ってくれないため、需要が実現する前に注文しなければならない。これは傷みやすい在庫の分析に関する古典的な問題であり、一般に新聞売り子問題として知られている。発刊当日を過ぎるとほとんど価値を無くす日刊新聞を対象とした問題であり、生鮮食品や園芸店、血液バンクなどの応用例がある。多くの研究は、特定の特別なケースにおける解析的解を調べるものであるが、ここではより一般的に適用できる簡単なスプレッドシートシミュレーションモデルを構築するため、別の問題に対して簡単に設定を変更することができる。

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 $\{a = 1000, 1001, 1002, \dots, b = 5000\}$ 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。これらの式を確認するためには、 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 $(\text{総収入}) - (\text{総費用})$ として表現され、以下のとおりとなる：

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

利益 $Z(h)$ は、式の中に確率変数 D を含んでいるため自身も確率変数であること、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ておこう。ここでは、3.2.1 項でも用いた Excel の乱数ジェネレータ RAND() から始めることにしよう。その関数を用いることで、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 $\{a, a+1, \dots, b\}$ 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、これはたん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：

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

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、これは確率 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、 $i \leq a + (b+1-a) < i+1$ であり、移項すると次式を得る：

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

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：

$$= \text{INT}(a + (b + 1 - a) * \text{RAND}()) \quad (3.5)$$

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。

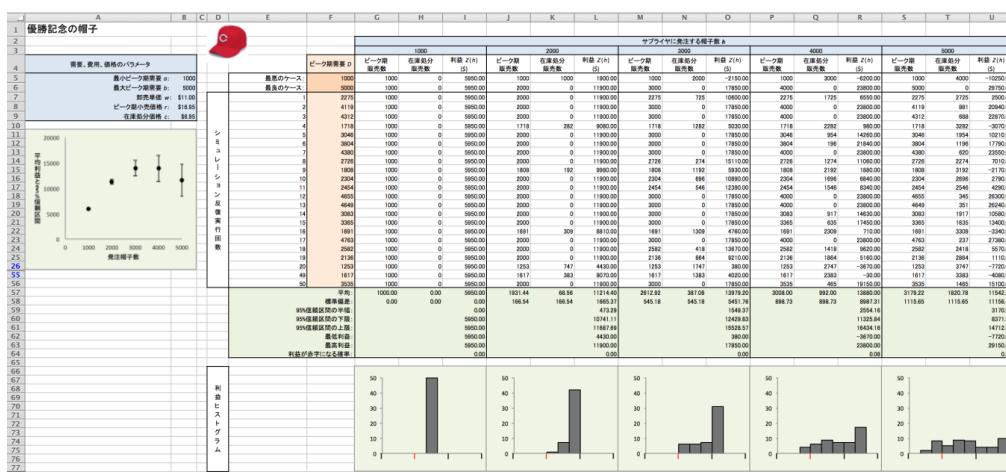


図 3.3 単一期間在庫のスプレッドシートシミュレーション

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入れられている。入力パラメータをこのように分けて設定し、数式で参照することにより、後にこれらの値を変更した影響を分析する「What-if」実験を容易に行うことができる。すなわち、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くなるわけである。今回は、恣意的ではあるが、50 回の IID の反復実行を行うこととし（各行が 1 回の反復実行に相当する）、列 E に反復実行回数を示した。反復実行は行 7 から始まり、列 F のオレンジのセルには、50 回分のピーク期需要 D が示されている。それらのセルには、式 3.5 に基づく数式が下記のように入力され、需要 D が生成される：

$$= \text{INT}(\$B\$5 + (\$B\$6 + 1 - \$B\$5) * \text{RAND}())$$

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、残りの反復にコピーできるように、\$を活用し絶対参照とした。これらの式の上（行範囲 5:6）に、ピーク期需要 D としてもっとも悪いケース ($a = 1000$) ともっともよいケース ($b = 5000$) に対して、発注帽子数の各試行値における収益を示している。このワークシートを開いたとき、図 3.3 に掲載された数値結果と異なるものが表示されるだろう。これは、Excel 関数 RAND() が、シートで再計算が行われるたびに、各乱数を再生成するためである。強制的に再計算させるにはキーボードの F9 キーを押せばよい。それにより、数値（およびグラフ）が変化する様子を目にするだろう。ここでは 50 回

の反復実行を行なっているが、F9 キーを押して得られる 50 回の反復実行ごとに、かなりのバラツキが生じている。これは、適切な実験計画とシミュレーション実験の分析の重要性、およびじゅうぶんなサンプルサイズの把握の必要性を示唆している。

前提条件の下、ピーク期需要 D の範囲からいくつかの η (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 η の各値において、 η にその値を代入したとして、ピーク期需要の 50 回の反復実行について収益を計算し、数値および図により統計値を要約した。

$\eta = 1000$ 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、1000 個の帽子はピーク期の価格で売り切れる。したがって、収益は毎回必ず $(r - w)D = (\$16.95 - \$11.00) \times 1000 = \$5,950$ となる。そして、在庫処分価格で販売する帽子が残る可能性はなく (列 H には 0 が並ぶ)、この収益よりも低くなるリスクもない。しかし、もちろん $D > 1000$ のケースならば得られる \$5,950 よりも多くの収益をあげる機会もない。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、 $\eta = 1000$ という小さく、保守的で、リスク回避的な選択は、みすみす得られる利益を自ら捨てているといえる。

スプレッドシートの右側に目を向けると、より大きい発注量 η のケースが示されており、 $\eta = 3000$ あるいは 4000 の辺りで平均収益 (行 57) は最大に達し、 $\eta = 5000$ に至ると、ピーク期後に売れ残る帽子の数が比較的多くなるため、収益は減少する。これは、列 A のプロットにも描かれている。そのプロットでは、点が標本平均を表し、垂直のひげが母平均の 95% 信頼区間を示している。 η が増えるにつれて、ピーク期の販売量および在庫処分の販売量の双方の変動が大きくなるため (ただし双方の和は常に η である)、収益の変動も大きくなる (行 58 の標準偏差を参照)。このため、期待収益の 95% 信頼区間もより大きくなり (行範囲 59:61 および列 A のプロットを参照)、また収益のヒストグラムもより裾野が広がることとなる (ヒストグラムのスケールは統一されている)。これらの結果からは、非常に大きな収益を得られる機会があるともいえるし、反面、収益が減少する悪いケースも想定される (行 64 およびヒストグラムにおいて赤色の記号で示された損益分岐点より左の領域を見てほしい)。以上の結果から、収益増加あるいは減少の両面のリスクについて考えてみてほしい。

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であるが、Palisade Corporation の@RISK シミュレーションアドインを利用したものである。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されるとよいだろう。

このシートを実行する (あるいは正しく表示する) ためには、まず@RISK を Excel に読み込んでおく必要がある。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 の機能だけで作成したものとの違いを挙げるに留める：

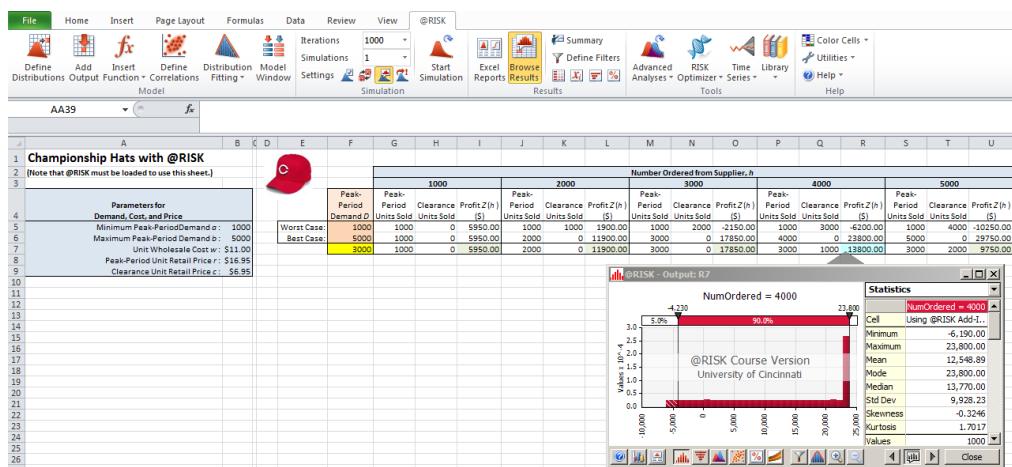


図 3.4 @RISK を用いた單一期間在庫のスプレッドシートシミュレーション

1. 基本的なモデルを作成する必要があり、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な部分である。
2. この1行のシミュレーションを、それが反復実行を意味する多くの行にコピーする必要はない。
3. @RISKには、このモデルでピーク期の需要として必要な離散一様分布を含め、多くの入力確率分布が用意されている。入力分布の設定は、セルF7に@RISK関数=RiskIntUniform(1000, 5000)と入力すればよい。これには、まずセルをクリックした後、@RISKリボンのDefine Distributionsボタンを押して必要な分布を選択し、入力パラメータを設定する（このモデルでは確率分布は1つでよい）。
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に入力する（もしくはプルダウンから選択する）だけでよい。今回は1000を選んだが、これはExcelの機能だけで行うオリジナルのシミュレーションで1000行を準備するのと同じである。
5. 調べたい出力を表すセルを指定する必要があり、ここでは収益のセルI7、L7、O7、R7、U7である。この設定は、これらのセルを1つずつクリックし、@RISKリボンのAdd Outputボタンを押すだけでよい。オプションとして、出力のラベルをNameで変更することもできる。シミュレーションファイルでは、これらのセルに薄緑色をつけておいた。
6. シミュレーションを実行するには、@RISKリボンのIterationsフィールドの右隣にあるStart Simulationボタンをクリックする。
7. @RISKが出力（平均や信頼区間、ヒストグラムなど）を収集し、要約してくれるため、これらについて準備する必要はない。
8. @RISKの出力には、多くの表示方法がある。おそらくもっとも有用なものは、注釈つきのヒストグラムである。これを（図3.4のように）表示させるには、出力セル（図ではn=4000の収益を表すセルR7）をクリックし、それから@RISKのBrowse Resultsボタンをクリックすればよい。垂直の分割バーを左右にドラッグすることで、それらの観測値の割合を確認できる（図では損失の割合が15.6%）。また、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が表示される。

@RISKのようなアドインを使えば、スプレッドシートにおいて静的シミュレーションを実施する作業は大幅に省力化できる。もう1つの利点は、それらのアドインが、Excel関数RAND()よりもはるかに優れた乱数ジェネレータを提供していることである。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）などがある。

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に関するモンテカルロモデルを解説する。このモデルに対して2つの実装を紹介する。1つは前述したモデルによく似たExcelモデルであり、もう1つはPythonによって実行するプログラミングによるアプローチである。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¹。オリジナルの問題は、新製品の製造およびマーケティングを考察するもので、製品から得られる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引くことで求められ、次の式で表される：

¹ 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、残念ながら現在はアクセスできないが、参考のため記す。

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

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である。上式では、モデル簡略化のために貨幣の時間的価値などの要因を無視しているが、改善してモデルの現実感を高めることは容易だろう。企業は将来、その製品を生産するにあたって起こることを予測したいが、利益の式の構成要素には不確実性がある。特に、Q、P、V は、次の確率分布に基づく確率変数であると仮定される：

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

なお、F は \$150,000 で一定である。答たい究極の問いは、その新製品の生産・販売プロジェクトを始めるべきか否かである。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。企業の意思決定者を支援するため、この確率変数に対する確率分布を推定したい。それから、この分布を用いて、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推定する。

「実際」の各確率分布から得られた 250,000 サンプルによるヒストグラムを利用して、図 3.5 に Q、P、V の分布を示す。ここで、「これらの分布はどこから得られたのか？」と疑問に思われるだろう。確率分布を適合させるための時系列データを企業が持っていることもあるだろうし（6.1 節参照）、分布を推定するための専門的な経験を持つ人物やグループがいる可能性も高いだろう。今回は、販売量について、じゅうぶんな不確実性をモデル化したい。したがって一様分布を用いるが、何も情報がなく、誰かが「10,000 から 15,000 の間のどこかではないか」といった状況である。販売価格の多くは市場によって決定されるため、ここでも不確実性を持たせたい。しかし、いくぶん中心傾向を持つようにし、製品が大ヒットする（高い販売価格を望める）もしくはみじめな失敗に終わる（選んだ対数正規分布の最小値である 1 に近い価格を付けざるを得ない）可能性を持たせた。最後に、企業は変動費をうまく管理できると確信しているが、原材料費や労務費などの要因に対して、若干の不確実性がある。ここで用いる対数正規およびワイブル分布は、一般的な形状をとる特徴がある。

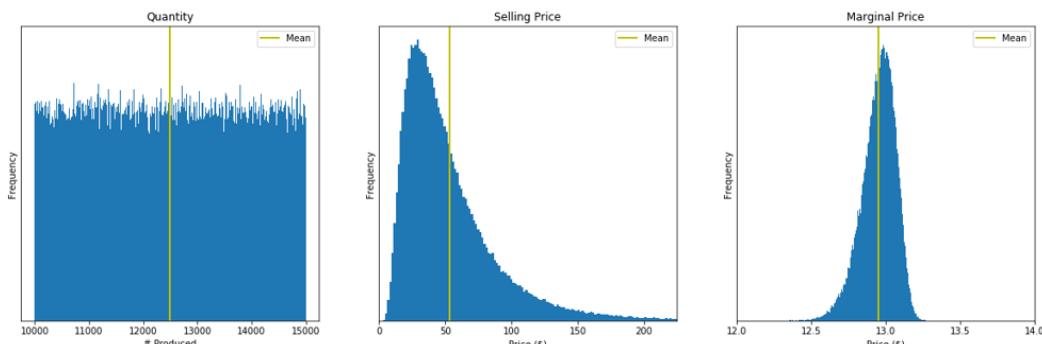


図 3.5 Model 3-4 の Python バージョンで生成された Q、P、V の分布からの標本

前述のモンテカルロの例題と同様、まず各確率変数の観測値をサンプリングし、それから式 3.6 を用いて総利益の確率変数のサンプルを計算して、その後このプロセスを繰り返す。ここでのモンテカルロモデルに対する疑似コードを、アルゴリズム 1 に示す（計算を簡単にするために、式 3.6 を並べ替えたことに留意）。

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

```

(対応するコード／モデルの実行時、TPはサンプリングされた総利益の観測値のリスト／配列とされ、その値を確率分布のパラメータの推定に用いる)

Q、P、Vの観測値のサンプリングはアルゴリズムIの重要な部分であり、確率変数から観測値を生成できる任意のツール／言語を用いて、そのアルゴリズムを容易に実装できるだろう(6.4節で述べるが、このような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およびPythonバージョンでは、表3.1に示す数式／関数を用いることとする(これらの数式の妥当性については、6.3節および6.4節を参照してほしい)。

表3.1 Model 3-4 の Q、P、V の観測値を生成する数式／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)+l
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 : 対数正規確率変数に用いる正規分布の平均

σ : 対数正規確率変数に用いる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 : 生成される観測値の数

図3.6に、Model 3-4のPythonバージョンにおける計算に関わる部分と、250,000回の反復により生成された総利益(TP)のヒストグラムを示す。また、図3.7に、Model 3-4のExcelバージョンの一部(100,000回の反復結果)を示す。どちらのバージョンのモデルも、アルゴリズムIで記した総利益の観測値に対する配列を生成し、それから要約および記述統計量を計算して、確率分布を推定するためにヒストグラムを生成する。

```

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

```

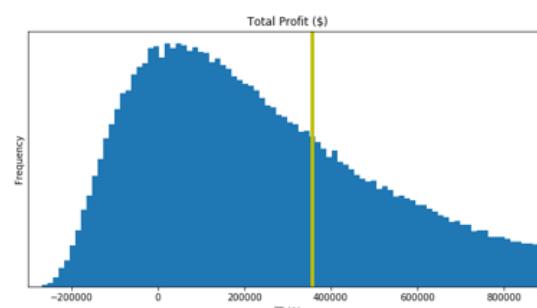


図3.6 Model 3-4 の Python バージョンの計算部分と総利益のヒストグラム

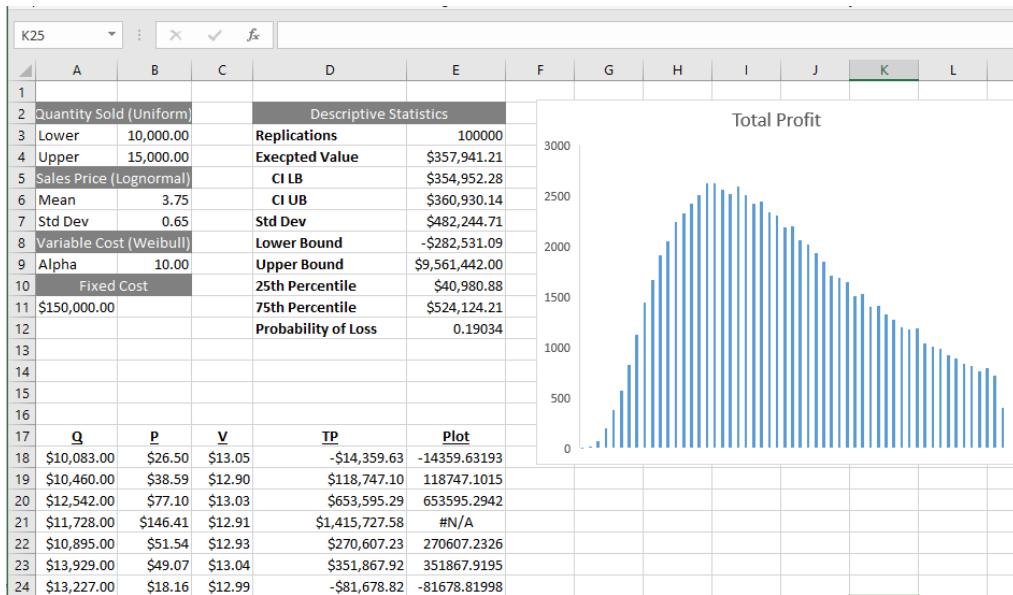


図 3.7 Model 3-4 の Excel バージョン

Model 3-4 の実行結果により、元々の問い合わせ、すなわちプロジェクトを開始するべきか、に答える準備ができた。記述統計量およびヒストグラム（図 3.7 と図 3.6）が有用な情報を提供している。期待利益は約\$360,000 であるが、損失が生じる可能性もおよそ 19% である。さらに、標準偏差（\$482,000）や第 1 および第 3 四分位点（それぞれ \$41,000 と \$524,000）から、プロジェクト開始のリスクに関する情報も得られる。4 章では、この文脈におけるリスクと誤差の概念について、より詳しく述べる。もちろん、プロジェクト開始の是非に対する実際の意思決定は、企業のリスク許容度や期待資本利益率、その他の機会などにも依存する。しかし、モデルによって得られる情報が、意思決定者に対して有用なものであることは明らかだろう。

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 のような動的シミュレーション専用のソフトウェアの利点を用いずに、動的シミュレーションを行う方法を考えたい。3.3.1 項では、簡単な単一窓口待ち行列のシミュレーションを手作業で（つまり、コンピュータを用いずに）行う。記録と計算のために Excel スプレッドシートの助けを借りるが、全体的なマニュアルシミュレーションの「モデル」には何ら影響を及ぼさない。3.3.2 項では、非常に特殊な関連性を利用して、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開発する。ただし、その特殊な関係は、より複雑な動的待ち行列シミュレーションでは活用できない点に注意してほしい。

3.3.1 Model 3-5：マニュアル動的シミュレーション

Simio のような高水準シミュレーションソフトウェアを利用すると目にすることはないが、すべての動的離散型シミュレーションは、背後ではほとんど同じように動作している。本項では、シミュレーションの動作の仕組みを解説する。Simio のような高水準ソフトウェアであれば、実際にはこのようにシミュレーションを遂行する必要はないが、その概念を理解しておくことはやはり重要である。なお、そのロジックは、C++ や Java、Matlab のような汎用プログラミング言語を用いて動的離散型シミュレーションをコーディングする際に必要である。

ここで例とするシステムは、簡単な単一窓口待ち行列である。エンティティは、システム外から到着時刻にシステム内に現れ（最初の到着は時刻 0 とする）、所与のサービス時間を要求する。すべての時間は分であり、待ち行列ルールは先入先出（FIFO）である。なお、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ものである。シ

ミュレーションは、エンティティが待ち行列内にあっても、あるいはサービス中であっても、時刻8分で終了するものとする。出力結果として、窓口の利用率（稼働中の時間の割合）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、のそれについて、平均、最小、最大を計算する。(a)および(b)の平均は対象エンティティの単純平均であるが、(c)および(d)の平均は時間平均である。つまり、経時に後者の変数を追跡した曲線下の総面積を、最終シミュレーションクロックの時刻8で除したものとなる。たとえば、 $L_q(t)$ を時刻tにおける待ち行列長とすると、待ち行列長の時間平均は $\int_0^8 L_q(t)dt/8$ となる。これは、曲線の値が0,1,2,…であった時間の割合を重みとする、待ち行列長の値(0,1,2,...)の加重平均である。同様に、 $B(t)$ を時刻tにおける窓口の稼働状態を表す関数（遊休は0、稼働は1）とし、 $L(t)$ を時刻tにおけるシステム内数（待ち行列内とサービス中のエンティティの和）とすると、窓口の利用率は $\int_0^8 B(t)dt/8$ となり、システム内数の時間平均は $\int_0^8 L(t)dt/8$ となる。

これから、図3.8に示すModel_03_05.xlsを用いて、システムの挙動を追跡し、計算する（このファイルは付録Cに記した方法でダウンロードできる）。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、到着時刻とサービス時刻の入力として、このモデル固有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して機能する、一般的なモデルではない。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、所与の到着時刻を利用した数式が入力されており、到着時間間隔が計算されている。ある行の到着時間間隔は、当該行のエンティティの到着時刻と、1つ前の行のエンティティの到着時刻の時間間隔である（このため到着時刻0であるエンティティ1の到着時間間隔は存在しない）。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
单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																
5	1	0.000000	0.486165													
6	2	0.985171	0.354917													
7	3	2.279418	0.354954	0.354917												
8	4	2.909957	0.634539	0.563653												
9	5	4.033663	1.123406	0.051975												
10	6	5.136229	1.102866	2.652734												
11	7	6.104247	0.420508	0.420508												
12	8	6.744513	0.203508	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	0.330272	
13	9	7.848616	1.01103	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	2.988174	
14	10	8.330407	0.484791	0.290218												
15																
シミュレーション実行																
16																
17																
18	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	待ち行列内での番目	2番目	次の到着	次の離脱	終了	システム内時間	エンティティごとの離脱	曲線下の面積	
19	0.000000	初期化	0	0	0	0	0.000000		0.000000	0.000000			0.000000	0.000000	0.000000	
20	0.000000	到着	1	1	0	0	1.965834	0.486165	0.000000	0.000000			0.000000	0.000000	0.000000	
21	0.486165	離脱	1	0	0	0	1.965834	0.486165	0.000000	0.000000			0.486165	0.486165	0.486165	
22	1.965834	到着	2	1	1	0	1.965834	2.275418	2.275418	2.275418	2.275418	0.000000	0.000000	0.000000	0.000000	
23	2.275418	離脱	2	1	0	0	1.965834	2.275418	2.275418	2.275418	2.275418	0.000000	0.000000	0.309584	0.309584	
24	2.630855	離脱	3	1	0	0	2.275418	2.909957	2.909957	2.909957	2.909957	0.000000	0.000000	0.355537	0.711074	0.355537
25	2.909957	到着	4	1	2	1	2.275418	4.033663	4.033663	4.033663	4.033663	0.000000	0.000000	0.279002	0.279002	0.000000
26	3.988174	離脱	3	1	0	0	2.909957	4.033663	4.033663	4.033663	4.033663	0.000000	0.000000	0.079715	0.151830	0.079715
27	3.549525	離脱	4	0	0	0	4.033663	5.136229	5.136229	5.136229	5.136229	0.000000	0.000000	0.563853	0.563853	0.000000
28	4.033663	到着	5	1	0	0	5.136229	6.104247	6.104247	6.104247	6.104247	0.000000	0.000000	0.000000	0.000000	
29	4.084938	離脱	5	0	0	0	6.104247	6.744513	6.744513	6.744513	6.744513	0.000000	0.000000	0.051375	0.051375	0.000000
30	5.136229	到着	6	1	1	0	5.136229	6.744513	6.744513	6.744513	6.744513	0.000000	0.000000	0.778893	0.778893	0.000000
31	6.539427	到着	7	1	2	1	6.744513	6.845616	6.845616	6.845616	6.845616	0.000000	0.000000	1.403198	1.403198	0.000000
32	6.744513	到着	8	1	3	2	6.845616	6.845616	6.845616	6.845616	6.845616	0.000000	0.000000	1.044450	3.133050	2.088900
33	7.848616	到着	9	1	3	2	6.845616	6.845616	6.845616	6.845616	6.845616	0.000000	0.000000	0.056653	0.113308	0.056653
34	7.848616	到着	9	1	3	2	6.845616	6.845616	6.845616	6.845616	6.845616	0.000000	0.000000	0.154384	0.462152	0.303768
35	8.000000	終了	1	3	2	6.845616	6.845616	6.845616	6.845616	6.845616	0.000000	0.000000	0.8676	0.2401	0.6232	
36													0.0095	0.3864		
37													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。	
38													2.6527	1.2495		

図3.8 単一窓口待ち行列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においては、まず、シミュレーション時刻のある時点に発生し、システム状態を変化させるイベントを特定する。例題の单一窓口待ち行列では、イベントとして、エンティティの到着、エンティティのサービスの終了およびシステムからの離脱、そしてシミュレーションの終了がある（シミュレーションの終了は「人工的な」イベントのように思われるだろうが、一定のシミュレーション時間（ここでは8分）を実行するシミュレーションを停止するための1つの方法である）。より複雑なシミュレーションでは、機械の故障や修理の完了、患者の症状の回復、従業員の休憩、車両の出発および目的地への到着、などのイベントが考えられる。複雑なモデルにおいてイベントを特定することは困難な場合があり、別の方を考える必要があるかもしれない。たとえば、单一窓口待ち行列では、エンティティがサービスを受け始めるときをイベントとして考慮することもできる。しかし、この「サービス開始」イベントは、あるエンティティがサービスを終えたときに別のエンティティが待ち行列に存在する場合にのみ発生する。あるいは、あるエンティティが到着時に窓口が遊休状態であり、他のエンティティが待ち行列に並んでいない場合に、直

接サービスを受けられるときに発生する。したがって、この「サービス開始」イベントは、到着および離脱イベントにすでに含まれているため、不必要的イベントである。

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に、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、これから起こるイベントの情報が収められた一種のデータ構造である。データ構造の形式と型はさまざまであるが、イベントカレンダには、将来のイベントが起こるシミュレーション時刻、イベントの種類、および関連するエンティティについての情報が含まれる。シミュレーションは、イベントカレンダ上の次の（つまり直近の）イベントを特定し、その時刻までシミュレーションクロック変数を進め、それからシステム状態変数や、必要な出力を得るために必要な統計的変数に影響を与えるイベントロジックを遂行し、必要に応じてイベントカレンダを更新する、という手順で進められる。

イベントの種類ごとに、遂行されるロジックを特定する必要がある。ロジックは、その瞬間のシステム状態に依存することもある。この例題モデルにおいて、それぞれのイベントが発生したときに遂行すべき事項を以下に示す：

- **到着**：まず、現在時刻（列 A のシミュレーションクロックの値）に所与のリストにある次の到着時間間隔を加えて、イベントカレンダの次の到着スポット（図 3.8 の Model_03_05.xls では列 J）を更新して、次の到着イベントを予定する。直前のイベントと現在時刻との間の時間に対して、曲線 $B(t)$ 、 $L(t)$ および $L_q(t)$ に関する連続時間出力結果の面積を計算する。計算には、直前のイベントから今回のイベントが起こる前までの値が用いられる。そして、列範囲 O:Q に計算結果を記録する。次に何をするかは、窓口の状態に依存する：
 - 窓口が遊休中の場合、到着するエンティティを即座にサービス中とし（列 G の対応行にサービスを受けるエンティティの到着時刻を入力する）、このエンティティのサービス完了時刻を、現在時刻と所与のリストの次のサービス時間を加算することで予定する。さらに、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、到着するエンティティの到着時刻を、サービスを待っているエンティティの待ち行列を表すリスト、すなわち列範囲 H:I の対応行に記入する（例題のシミュレーションではこの待ち行列長は 2 以上にならないため、待機中のエンティティの到着時刻を保存するために 2 つの列があればよいが、待ち行列がそれよりも長くなる他のケースでは、当然より多くの列が必要になる）。この到着時刻は、待ち時間およびシステム内時間を計算するために、後で必要となる。この時刻は、このエンティティの待ち時間の開始時点を表しており、それが待ち行列を出てサービスを受け始める時刻は現時点ではわからないため、このエンティティに対応する列 N に待ち時間を記録することはできない。
- **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。また、列 M の対応行に離脱するエンティティのシステム内時間を計算し記録する（シミュレーションクロックの現在時刻から、列 G に記録されているエンティティの到着時刻を引く）。次に何をするかは、待ち行列に待機中のエンティティが存在するかどうかに依存する：
 - 待ち行列で他のエンティティが待っている場合、待ち行列の先頭のエンティティを取り出し、サービス中にする（列 H の到着時刻を列 G の次の行へとコピーする）。それから、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を予定する（列 K）。さらに、このエンティティの待ち時間を計算し（列 A のシミュレーションクロックの現在値から、列 G にコピーしたばかりの到着時刻を引く）、列 N の対応行に記録する。
 - 待ち行列に待機中のエンティティが存在しない場合、窓口の状態変数（列 D の 18 行目以降）を、遊休中を表す 0 とし、次の離脱を表すイベントカレンダのエントリ（列 K）

を空白とする。

- ・ **シミュレーションの終了**（シミュレーションロック時刻 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、さらに最終評価尺度を行 36:38 に計算する。

イベントを処理するには計算のためのコンピュータ時間が必要とされるが、イベントの遂行中にはシミュレーション時間は経過しない、つまりシミュレーションロックの時刻は止まっていることを理解してほしい。そして、このマニュアルシミュレーションの実践中に、到着時間間隔やサービス時間をチェックするので、リストを隅々まで見渡すことで、それらの時間を使用しなかったり、あるいは再使用することがないことを確認できるだろう。

図 3.8 の Model_03_05.xls の行範囲 19:35 では、時刻 0 の初期化（行 19）から時刻 8 分に終了するまで（行 35）、各イベントの遂行が 1 行ずつ追跡されている。列 A はイベントのシミュレーション時刻 t 、列 B はイベントの種類、列 C にはエンティティ番号が示されている。すべての行のセル内容は、当該行のイベントが遂行された後の値を表している。列範囲 D:F にはそれぞれ、窓口の状態 $B(t)$ （遊休中は 0、稼働中は 1）、システム内エンティティ数、待ち行列長（この例題では常にシステム内数 - 窓口状態に等しい）が入る。列 G はサービス中のエンティティの到着時刻が記録され、列範囲 H:I には待ち行列の先頭から 2 つのエンティティに対応する到着時刻が入力されている（この例題では、待ち行列長は 2 より大きくならない）。

列範囲 J:L はイベントカレンダであり、その 3 つの値から最小値を選ぶことによって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次のような別のデータ構造も考えられる。すなわち、[エンティティ番号、イベント時刻、イベント種類]の形式のレコード（行）のリスト（あるいはツリーやヒープといったより洗練されたデータ構造）とし、新しいイベントを予定する際にはイベント時刻の昇順でリストへと追加し、次のイベントが常にリストの先頭になるようにすることである。後者の方法は、実際にシミュレーションソフトウェアが動作する方法に近く、同じ種類の複数のイベントを予定できることや、（特にヒープ構造を用いると）より動作が速いことなどの利点がある。これは、多くの種類のイベントが存在し、さらに長いイベントリストとなる大規模なシミュレーションの実行速度に関して、大きな影響を与える。ただし、この例題においては、列範囲 J:L の簡単な構造でじゅうぶんである。

システム内時間あるいは待ち時間が観測されたとき（それぞれ離脱あるいはサービス開始時に起る）、それらの値が計算され（列 A のシミュレーションロックの現在値 - 到着時刻）、列 M あるいは列 N に記録される。最後に、列範囲 O:Q に直近のイベント時刻と当該行のイベント時刻の間の面積が、これらの曲線の時間平均を計算するために記録される。

シミュレーションは、時刻 0 の行 19 で初期化することから始まる。ここで、L19 のシミュレーション終了イベント時刻は時刻 8 に設定され、次の離脱時刻は空白のままとされる（Excel は空白セルを大きな数値として解釈するため、A20 の関数 MIN はうまく動作する）。

A20 の式 =MIN(J19:L19) は、1 つ上の行のイベントカレンダから最も小さな（最も早い）値、すなわち 0 を返す。これは、時刻 0 でエンティティ 1 が到着することを意味している。D20 の窓口の状態を、稼働中を表す 1 とし、E20 のシステム内数を 1 とする。F20 の待ち行列長は 0 のままである。また G20 を 0 として、このエンティティの到着時刻を記録する。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であり、H20:I20 には何も記録されない。J20:L20 のイベントカレンダにおいては、次の到着時刻として現在時刻(0)に次の到着時間間隔(1.965834)を加えた値に、また次の離脱時刻として現在時刻 (0) に次のサービス時間 (0.486165) を加えた値に更新される。シミュレーション終了イベントの時刻は変化せず、上の行の値をコピーするだけである。これは到着イベントであるためシステム内時間の終了は観測されず、M20 は空白となる。しかし、今回到着したエンティティは直接サービスを受けられるため、N20 の待ち時間に 0 を記録できる。セル範囲 O20:Q20 は、直前のイベント以来の曲線 $B(t)$ 、 $L(t)$ および $L_q(t)$ 下の面積を

計算するものであるが、時刻は 0 のままであるため、この場合すべて 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 (A21) と現在時刻 (A20) 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のため、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が、読者自身でスプレッドシート、特に数式の入力されたセルを確認してほしい。

それから、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空ではない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 からコピーする)。この記録は、エンティティの待ち時間およびシステム内時間を計算するため、後で使用される。ここでは、待ち時間もシステム内時間も完了を観測されないため、M23 あるいは 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。ここで、エンティティ 3 は待ち行列の先頭から窓口に進むため、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 にエンティティ 3 の待ち時間が計算される (=A24-G24 = 2.630955 - 2.275418 = 0.355537)。

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になるまで、このようにしてシミュレーションが進行する (予定されている時刻 8.330407 のエンティティ 9 の到着、あるいは時刻 8.168503 のエンティティ 7 の離脱は発生しない)。時刻 8 に行なうことは、セル範囲 O35:Q35 に終了時の曲線下の面積を計算することだけである。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 を使って、評価尺度を計算している。セル範囲 O36:Q36 の時間平均は、曲線下の面積の総和を、A35 にある最終シミュレーションロック時刻の 8 で除算したものである。例題における評価尺度は、窓口利用率は 62.32% (O36)、平均システム内時間および平均待ち時間はそれぞれ 0.8676 分と 0.2401 分 (M36 および N36)、そして平均待ち行列長は 0.3864 (Q36) である。4 章では、Simio でこれと同じモデルを構築することになるため、評価尺度もすべて同一となることが確認できるだろう。

ここで、Model_03_03.xls の平均収益を行なったのと同様に、列 M および N において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一緒に計算しないのかと、疑問を持たれることだろう (列範囲 O:Q のエンティティについては、時間経過によって重みづける必要がないため、同じように計算する意味はないが)。結局のところ、出力のバラツキについては、列 M および N にあるシステム内時間や待ち時間の観測値の変動を見るだけでよいだろう。実際には、標準偏差を計算するために行を挿入することができるが、それは正しいことであるかを確かめずに、可能であるから実行するという悪い例となるだろう。それでは、この例において標準偏差を計算することの問題点とは何であろうか。統計学の基礎クラスの内容であるが、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) は、連続するエンティティが互いの物理的な状況に依存しているため、互いに相関関係 (おそらく正の相関) にある (このような順序に基づく相関を自己相関、あるいは系列相関と呼ぶ)。たとえば、単一窓口の銀行において、あなたが不運にもバケツ一杯もの小銭を預けようとしている顧客の後ろに並んでしまった場合、その顧客には長いシステム内時間がかかり、それが原因であなたの待ち時間

は非常に長くなる。正確にいうと、これは正の相関である。このようなケースでは、標本標準偏差は容易に計算できるけれども、独立したサンプルから導出したものと比べると、母集団の標準偏差を過小評価する傾向にある。つまり、標本標準偏差が低く偏ることとなり、結果の真の変動性も低く見積もってしまうこととなる。問題はそれぞれの観測値の相関の程度であり、相関が強いほど、標本分散の推定量における下方バイアスもより深刻なものとなる。これ（自己相関）と同じ理由のために、システム内時間や待ち行列内時間のヒストグラムを作成することを控えてきた（ただし、Model_03_03.xls の月間収益は独立同一分布であるため作成した）。すなわち、正の相関を持つ観測時間は、実行期間が確率分布のすべての範囲から適切に「サンプリング」されるほどじゅうぶんに長いものであった場合を除いて、ヒストグラムが本来表すものよりも左もしくは右にシフトされるリスクがある。

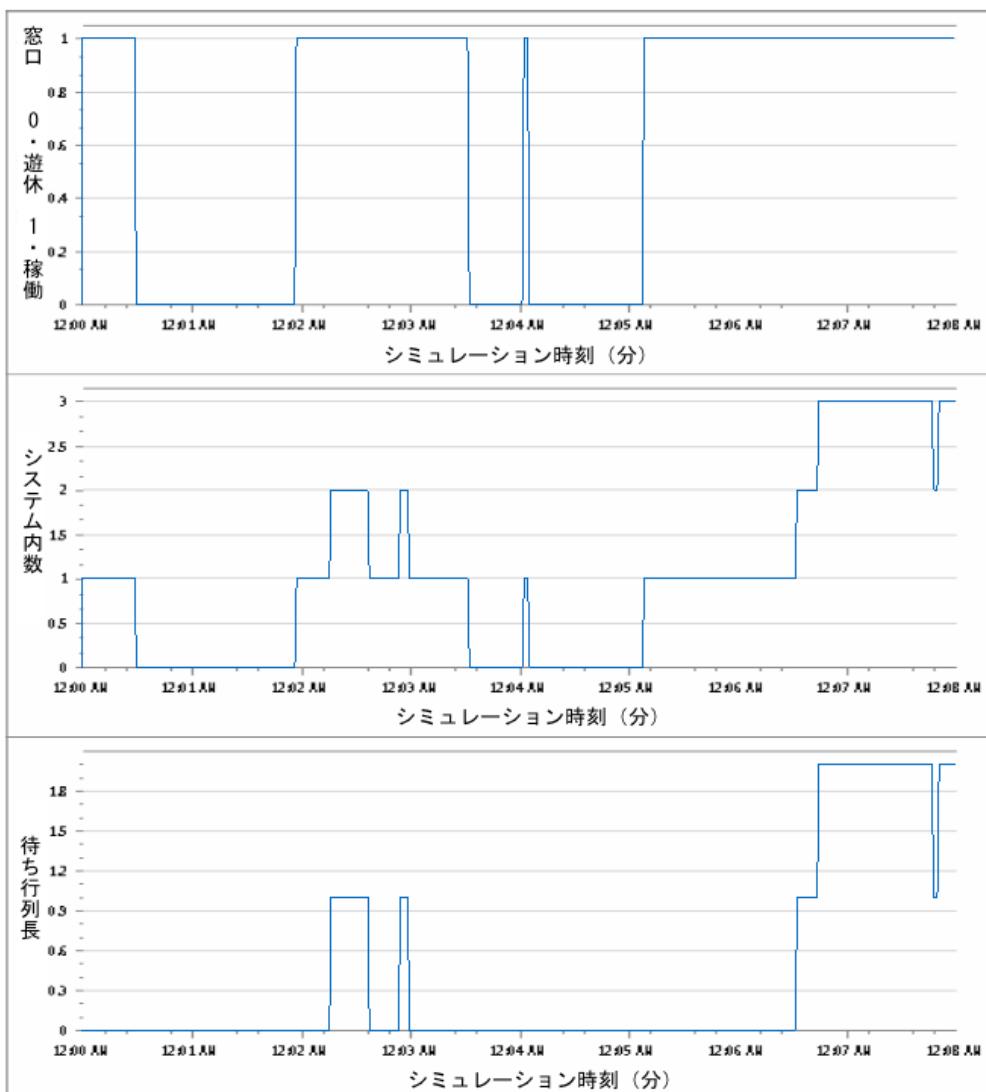


図 3.9 マニュアルシミュレーションにおける曲線 $B(t)$ 、 $L(t)$ 、 $L_q(t)$ のプロット

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、系内時間および待ち時間の観測された個別の値、そして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、1つ実用的なコメントをしておこう。この小規模な例題では、行範囲 36:38 に Excel 関数を用いるだけで、要約出力パフォーマンス評価尺度を計算することができたため、この方法でうまくいった。しかし、例題の 17 行をはるかに超える数千行にわたるような長い期間のシミュレーションや、何度も反復実行するようなシミュレーションでは、評価尺度を得るために個々の値をすべて保存し計算に利用しなければならないため、極めて非効率な方法となるだろう。

シミュレーションソフトウェア（あるいは汎用プログラミング言語を用いたシミュレーション）で実際に行われている方法は、統計アキュムレータ変数を設定する方法であり、観測された順に各値をその中に加えることにより、個々の値を保存し、後で呼び出す必要はなくなる。また、各評価尺度の現時点の最大および最小値を保存し、新しい観測値と保存されている値を比較して極値を判断するロジックを加えることもある。ストレージの有効活用の観点から、計算上はるかに効率的であることは自明であるだろう。

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してほしい。つまり、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必ずしも正確な結果を示さなくなるのである。この理由は、多くのセル（ピンクのセル）に入力されている数式が、現在入力されているイベントおよび状況においてのみ有効な点にある（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している）。実際、スプレッドシートは動的離散型シミュレーションを遂行するには適しておらず、実践で利用されるケースはほとんどない。ここでスプレッドシートを活用したのは、データの保存方法や計算方法を解説するためであり、信頼性があり柔軟なモデルを構築するためではない。3.4.1 項において、動的離散型シミュレーションに対して汎用プログラミング言語を利用することについて簡単に解説するが、複雑なモデルには Simio のようなシミュレーションソフトウェアが実質的に必須であり、本書の大部分は後者の解説に充てられている。

3.3.2 Model 3-6：単一窓口待ち行列の待ち時間

スプレッドシート（表計算ソフトウェア）は広く普及し、誰もが少なくとも何ができるか多少は知っているだろう。このため、スプレッドシート内で完全な、一般的で、柔軟な動的シミュレーションモデルを構築できるのか、いぶかしがられるだろう。3.2 節で静的かつ確率的なモンテカルロシミュレーションに対してスプレッドシートを利用したが、プログラミング言語を用いるとより効率的に行えただろう。3.3.1 項では、マニュアル動的シミュレーションについて、数値を記録し、計算を行うため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内の数式は特定の入力データ（到着時間間隔およびサービス時間）に依存していたため、完全性、一般性、あるいは柔軟性を要する実モデルとはいえないものだった。一般的に、非常にシンプルな動的シミュレーションをスプレッドシートで表現することは、実質的に不可能ではないけれども難しい。しかし、3.2 節のような静的シミュレーションに対しては、広範囲に用いられている（静的および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 項の單一期間在庫問題で時間の経過を説明したが（ピーク期は 1 ヶ月で、それが過ぎると在庫処分期となる）、そのモデルにおいて時間は明示的な役割を果たしていなかった。ピーク期を一瞬で終わる期間として捉え、瞬時に在庫処分期に変わると理解することもできるかもしれない。したがって、時間の経過が果たす役割の観点から、そのモデルは真に静的なモデルであった。本項では、時間の経過がモデル動作の中心部を担う、非常に簡単な動的単一窓口待ち行列モデルをシミュレートする。しかし、スプレッドシートの範囲内でこれを行うことが可能なのは、以降のケースでは、エンティティの待ち時間（サービス時間を除く）の出力処理のみに特化してシミュレートし、簡単な漸化式でそれを表現しているという事実に大きく依存している。3.3.1 項のマニュアルシミュレーションとは異なり、ここではその他の出力を直接計算することはできない。

時刻 0 において空かつ遊休で始まり、時刻 0 に最初のエンティティが到着する単一窓口待ち行列に対して、 A_i をエンティティ $i-1$ とエンティティ i ($i = 2, 3, \dots$) 間の到着時間間隔とする。さらに、 S_i をエンティティ i ($i = 1, 2, \dots$) に必要なサービス時間とする。 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ことに注意してほしい。しかし、2 章とは異なり、解析解を導出しようとしているのではなく、シミュレートしようとしているため、これらの分布は任意である。関心のある出力処理は、 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、窓口での時間は含まず、行列で（ムダに）

待っている時間のみを対象とする。最初のエンティティは、システム内に他のエンティティが存在せず、窓口が遊休中に（時刻0で）到着するため、その幸運なエンティティの待ち時間は明らかに0である。したがって、必ず $W_{q,1} = 0$ となる。しかし、後続の待ち時間 $W_{q,2}, W_{q,3}, \dots$ については、 A_i および S_i の値に依存するため、どのような値になるかはわからない。ただし、このケースでは、到着時間間隔およびサービス時間がいったん確率分布から得られれば、連続する $W_{q,i}$ 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）：

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

ここで、 $W_{q,1} = 0$ である。そこで本項では、各エンティティにつき1行を割り当て、到着時間間隔およびサービス時間の確率分布から観測値が生成されるように、スプレッドシートに表現する。

双方の分布は指数分布とし、到着時間間隔の平均は1.25分、サービス時間の平均は1分とする。さらに、すべての観測値および分布は互いに独立しているものとする。したがって、次に必要なことは、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節で議論するように、連続確率分布に対してこれを行う（少なくとも行おうとする）一般的な方法は、 $U = F(X)$ とする関数を用いることである。ここに、 U は0と1の間で連続一様に分布する乱数であり、 F は確率変数 X の累積分布関数である。そして、この式を X について解くと、所与の生成された乱数 U について、 X の観測値を生成する式が導出される。平均 β の指数分布の場合、累積分布関数は次のようになる：

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

$U = F(X)$ を式3.8に代入し、 X について解くと、下式を得る：

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

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、さらにExcelの乱数ジェネレータ RAND()を用いることにより、到着時間間隔およびサービス時間が指数分布に従う单一窓口待ち行列の待ち時間のプロセスを、Excelでシミュレートする準備が整った。図3.10に、行範囲26:101を非表示としたModel_03_06.xlsを示す（すべての行を見るにはファイルを開いてほしい。もっとも重要な部分はセル内の数式である）。青色のセル範囲A4:B6にはモデルの入力パラメータが入力されており、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に計算されている（M/M/1待ち行列に対するこれらの式は2章を参照のこと）。D列にエンティティ番号 $i = 1, 2, \dots, 100$ が示されており（ここではエンティティの待ち時間を100回観測することを恣意的に決めた）、各行には当該エンティティの待ち時間が対応する。E列には、セルB5に入力されている平均到着時間間隔に基づく数式 = $-\$B\$5 + \text{LN}(1 - \text{RAND}())$ 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も同様にして、サービス時間が生成されている。G列では、リンドレーの漸化式（式3.7）が実装されている。たとえば、セルG20（エンティティ番号 $i = 17$ ）には、 $W_{q,17}$ の計算のために数式 = $\text{MAX}(G19 + F19 - E20, 0)$ が入力されており、 $W_{q,16}$ 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によって対応セルを変化させたいため、これらのセル参照に \$ を使っていないことに注意してほしい。これは、待ち時間の相互依存関係（相関関係）を示しており、シミュレーションの出力に対して適切な統計解析手法を用いなければならないことを示唆している（つまり、古典的な統計分析とは異なり、シミュレーションの出力における連続する観測値を独立したものとして扱えないことを意味する。これについては、3.3.1項の最後に議論した）。



図 3.10 Excel による単一窓口待ち行列の待ち時間

列範囲 E:G は右側に 4 回コピーされ、100 回分のエンティティの待ち時間のシミュレーションに対する合計 5 回の独立した反復実行の準備が整う。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用いていること以外、数式の内容などはすべて変わらない。列 T には、 $i = 1, 2, \dots, 100$ の各々について、5 回の反復実行から得られる 5 つの $W_{q,i}$ の平均値が計算されている。すなわち、この列には、5 回の反復実行において 1, 2, …, 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の平均が示されている。そして、行 104 には、 A_i および S_i の列平均が計算されている。これらはセル B5 および B6 に入力された到着時間間隔およびサービス時間の（推定）平均値をただ確認するだけである。ただし、 $W_{q,i}$ の列平均については、5 回の反復実行のそれぞれを対象とする、待ち時間の反復実行内の平均値となる（3.3.1 項で述べたとおり、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について標準偏差を計算することは控える。同様に、これらのヒストグラムも示さない）。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の平均値（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）が示されている。

RAND() の変動性を見るため F9 キーを何度か押し、到着時間間隔、サービス時間、待ち時間の変化の様子と、グラフの移り変わりを確認してほしい（F9 を押し、乱数が再生成されるたびに、最大の待ち時間に対応してグラフの縦軸目盛りは自動的に更新される）。注意点を以下に列挙する：

- すべてのシミュレーションによる曲線は左端が 0 に固定されており、最初に到着する幸運なエンティティの待ち時間が毎回 0 であることに対応している。しかし一般に、時間が経過するにつれて（横軸のエンティティ番号が増加するにつれて）、待ち時間が徐々に増加する傾向にある。ただし、到着時間間隔およびサービス時間におけるランダムノイズにより、単調に増加することはない。
- 太く濃い曲線は、薄い曲線ほど上下に変動しない。これは、太い曲線が薄い曲線の平均であり、各エンティティ番号の待ち時間の大小はそれぞれ相殺される傾向にあるためである。
- すべての曲線は、蛇行する特徴を示す。つまり、曲線は確かにランダムであるが、右側に進むにつれ、新しい値は前の値と極端に離れず、少なくともゆるやかな繋がりを持っている。これにより、反復実行内において待ち時間が（正の）自己相関にあることがわかる。この特徴は、式 3.7 のリンドレーの漸化式で確認でき、また 3.3.1 項の最後でも議論した。
- F9 を押していると、待ち時間が定常状態における期待値（水平の破線）へと収束したり、しなかったりする。ここから、次の 2 つが示唆される：(a) システムに多くのノイズがある。(b) 長期間の実行における定常状態の条件に近づくために、100 より多くのエンティティが必要である。

B5 および B6 の平均到着時間間隔と平均サービス時間を様々に変化させる際に、平均到着時間を常に平均サービス時間よりも大きくしなければならないことに注意してほしい。そうしないと、定常状態における待ち時間の期待値 W_q の式 B9 が妥当性を欠くこととなる（ただし、その場合でもシミュレーションの妥当性は保たれる）。また、上記に注意して変化を確認してほしいが、特に平均到着時間間隔および平均サービス時間の値が近い場合を試してほしい（B8 のトラフィック密度 ρ が極めて 1 に近くなり、 W_q が非常に大きくなるだろう）。

最後に、スプレッドシートでこの**特殊な動的シミュレーション**が可能なのは、式 3.7 のリンドレーの漸化式を活用できることに完全に依存していることを強調しておきたい。また、このケースは待ち時間の出力プロセスに特化しており、この簡単なシステムでさえ、それ以外の指標（たとえば、3.3.1 項のマニュアルシミュレーションで行った待ち行列長や窓口の利用率など）を対象に構築できていないことにも注意してほしい。一般に、複雑な動的システムに対してリンドレーの漸化式を有効に活用できることはほとんどないため、スプレッドシートで現実的な動的システムをシミュレートすることは、まったく非現実的であり、ほとんどの場合は不可能である。これが、Simio のような動的シミュレーションソフトウェアが必要とされる大きな理由である。

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ために、2 つの異なる方法（マニュアルおよびスプレッドシート）を苦労して説明した。しかし、どちらも実用上受け入れられるものではなかった。そこで本節では、動的シミュレーションを行うための、2 つより実践的な方法の概要を述べる。

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++ や Java、Matlab のような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により、さらに効率的で一般的に表現できる。歴史的にいえば、これはシミュレーション専用のソフトウェアが誕生する前には、すべてのシミュレーションにおいて行われていたことである。モデルが比較的単純であったり、コンピュータの計算速度に対する利点がある場合など、特定の状況においては今日でも行われている方法である。しかし、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が、非常に手間がかかり、開発に時間がかかり、そしてエラーが発生しやすい方法でもあるともいえる。反面、高水準の専用シミュレーションソフトウェアを利用する開発者では想定もできないような極めて突飛なふるまいをモデル化できる、非常に柔軟な能力を獲得できるという利点も持ち合わせている。

動的離散型シミュレーションに対して汎用シミュレーション言語を用いる場合においても、イベントを特定し、各イベントのロジックを定義する必要がある。次に、各種イベントの活動を実行するためにサブプログラム（あるいは関数）を書く（3.3.1 項の到着および離脱イベントを解説した箇条書きの箇所が、サブプログラムの構造化に際して参考になるだろう）。イベントサブプログラムは、3.3.1 項のイベントカレンダの更新のような、マニュアルシミュレーションで行ったことのすべてを担当するメインプログラムあるいは関数と連携して、次のイベントを決定したら直接対応するイベントサブプログラムが実行されることとなる。終了時には、適切な要約統計量を計算し出力するようなプログラムも必要となる。また、（確率的シミュレーションにおいては）乱数ジェネレータを用いて入力確率分布から観測値を生成する方法や、ある種のデータ構造によりモデルの状態変数を追跡する方法なども考慮しなければならない。推察できるように、モデルが複雑になれば行わなければならない作業や分析の時間も極めて多くなり、エラーの検出にも非常な困難が予想される。

それにもかかわらず、大規模で特殊な状況や、あるいはより標準的なシミュレーションを目指して、汎用シミュレーション言語による方法を追求している研究もある。これらに関しては、たとえば 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、様々な種類の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。それらは、シミュレーションで行われることを追跡・演算するためにスプレッドシートを利用したマニュアルシミュレーションや、汎用プログラミング言語でマニュアルシミュレーションのロジックに沿ってシミュレーションをプログラムする方法、シミュレーション専用のアドインを活用し、スプレッドシートで一般的なシミュレーションモデルを構築することであった。

また、これらのアプローチにはすべて、シミュレーションを遂行するための欠点が異なる観点から存在することも述べた。マニュアルシミュレーションは、明らかに実際上の応用にはまったく不向きである。シミュレーションに汎用プログラミング言語を用いる人たちもあり、彼らは完全なモデリングの柔軟性と実行速度の速さを売りにしている。しかし、ゼロから複雑な動的シミュレーションをプログラミングすることは、構築に必要な時間が膨大で、しかもプログラミングエラーが発生する可能性が高いため、広く用いるには実用的ではない。スプレッドシートは、特に適切なシミュレーションアドインを利用して、おそらく静的シミュレーションに対して今後も広く用いられるだろう。しかし、大規模な動的シミュレーションに関するロジックやデータ構造、その他の一般的なニーズに対応するには、シンプルすぎるだろう。

実際、比較的簡単に利用できる高水準な動的シミュレーションソフトウェアに対するニーズ、特に離散確率型動的シミュレーションに関しては、かなり以前に(スプレッドシートが誕生する前に)実現されている。1960年代にさかのぼると、GPSS (Schriber 1974, Schriber 1991) や 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、いくつかは Simio と同じくオブジェクト指向型の初期の例となっている(ただし、当時はそのような概念はまだ存在しなかった)。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、代表例として GASP (Pritsker 1974) がある。これらは、よりモダンで自己完結的なシミュレーション言語、たとえば SLAM (Pritsker 1995) や SIMAN (Pegden et al. 1995) などの開発につながった。しかし、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、使用あるいは結果の解釈に際して、ユーザフレンドリもしくは直感的とはいえないかった。その後、グラフィカルユーザインターフェースおよびアニメーション機能を備えたソフトウェアが開発され、多くのユーザに門戸が開かれた。これらのソフトウェアには、Arena (Kelton et al. 2015) や AnyLogic (www.anylogic.com)、ExtendSim (www.extendsim.com)、Promodel (www.pmccorp.com)、Flexsim (www.flexsim.com)、Simul8 (www.simul8.com) などがある。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)、特に広範囲のレビューには Nance and Sargent (2002) を参照されたい。また、Swain (2015) は隔年行われるシミュレーションソフトウェアのサーベイの最新版(第10版)である。

Simio は比較的新しい製品であり、ソフトウェア開発およびモデリング能力に関する最新の知見に基づいて開発されている。新しいけれども、Simio は SIMAN および Arena の開発者と同じ人物が開発しているため、背後には多くの経験が生かされている。知的オブジェクトを中心に据え、オブジェクトの構築および利用方法に画期的な変化をもたらす、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しない、簡単でグラフィカルなプロセスフローを用いて作成される。これにより、より簡単に、独自のモデリングオブジェクトや目的に特化したライブラリを構築することができる。また、Simio の柔軟性とリアルタイムデータとの広範な統合は、インダストリー4.0 に必要なデジタルツインの作成に適している。本書の後半の大部分は、Simio を利用して、効果的に実践的な問題を解決するシミュレーションの利用方法を述べることとなる。

3.5 問題

1. 3.2.1 項のサイコロシミュレーションを、次のように発展させなさい（ただし、一度に 1 つずつ）。
 - ・ 50 回投げる代わりに、500 回投げるよう Spreadsheet シートを拡張し、結果を比較しなさい。F9 キーを押すと乱数の組が「新しく」なり、結果も更新される。
 - ・ サイコロの各目の出る確率を $1/6$ で一定から、他の確率に変えなさい。ただし、すべての確率の和は 1 となるように注意すること。
 - ・ @RISK (3.2.3 項参照) あるいは、静的モンテカルロ Spreadsheet シミュレーション用の Excel アドインを用いて、偏りのないサイコロ 2 個を 10,000 回投げた結果を、各目の和 $2, 3, \dots, 12$ から得られる真の確率、つまり真の期待値 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輪がそれぞれ $1, 2, \dots, 12$ となる確率を（初級の確率論から）導きなさい。それらの確率を用いて、2 つのサイコロの目の和について（真の）期待値および標準偏差を求め、セル J4 の（標本）平均と比較しなさい。さらに、Spreadsheet シートに 2 つのサイコロの目の和に対する（標本）標準偏差の計算を追加し、標準偏差について正確な解析的な結果と比較しなさい。なお、サイコロを投げる回数は 50 回のままでする。
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 $E(Y) = E[(b-a)h(X)] = (b-a)E[h(X)]$ と変形することから始め、それから確率変数関数の期待値の定義を用いる。最後に、 X は $[a, b]$ で連続一様に分布するため、 $a \leq x \leq b$ について密度関数 $f(x) = 1/(b-a)$ となることを利用する。
4. @RISK または静的 Spreadsheet シミュレーション用の別の Excel アドインを利用し、3.2.2 項の例題について、図 3.2 の Model_03_02.xls では 50 個の値だった X_i を、10,000 個の値となるように拡張しなさい。Model_03_02.xls および（ほぼ）正確な数値積分の結果と、この拡張で得られた結果を比較しなさい。
5. 3.2.2 項のモンテカルロ積分において、Spreadsheet シートに 50 個の値に対する標準偏差の計算を追加し、セル H4 にすでにある平均値と共に、セル I4 にある正確な積分値に関する 95% 信頼区間を計算しなさい。得られた信頼区間は、正確な積分値を含むだろうか？どのくらいの頻度で含まれるか（繰り返し F9 キーを押し、手書きでメモをとる）？90% 信頼区間および 99% 信頼区間にについても、同様の手順で検討しなさい。
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% 減少させることによりサービス提供速度を向上させなさい。3.2 節と同じ評価指標にどのような影響があるか述べ、議論しなさい。
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遅くしなさい。同様の指標への影響を述べ、議論しなさい。
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、窓口が遊休となつ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）。これには、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のため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。これを実現するには、次の 2 つの方法が考えられる：(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。その後、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、結果は同じであるが、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. 3.2.3 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 においては、恣意的に 50 回の反復実行（50 行）を行うことを決めたため、平均収益の期待値には、サンプルサイズ 50 に対する何らかの信頼区間の半幅（行 59）が付随していた。Model_03_03.xls を 200 回の反復実行を行うように変更すると（反復実行は 4 倍となる）、想像通り、半幅はより小さくなるが、どの程度小さくなるだろうか。評価尺度からこれを測ることができ（Excel の乱数ジェネレータの変動性のため、計測は少し困難であるが）。標準的な統計学の書籍を開いて信頼区間の半幅に関する計算式（平均に対するもので、一般に正規分布およびt分布を用いる方法がある）を調べ、その計算式をふまえて実証結果を解釈しなさい。半幅を（元の 50 回の反復実行時の結果と比べて）10 分の 1 に削減する（つまり平均の推定値の統計上の精度を一桁増やす）には、およそ何回の反復実行を行えばよいだろうか。
10. 3.2.3 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 においては、ある反復実行（行）について F 列の同じピーク期需要の実現値が、5つすべての発注量%の試行値に対して収益の計算に使用されていた。代わりに、それぞれの%の値に対して、独立した需要の列を設けることもできる。Model_03_03.xls をこれが行えるように変更し、妥当性を確認しなさい。本当に関心のある事柄は、異なる%の値に対する収益の違いであるが、この新しい方法はよりよいといえるだろうか。平均収益の推定値には「ノイズ」（統計的変動性）が存在することを想起してほしい。ノイズが少なければ、推定値の正確性がより高まる。それは、所与の%に対する収益の値自体にも、異なる%間の収益の差異についても、どちらにもいえることである。
11. 3.2.3 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 において、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 へと削減するか、あるいは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 へと上げるか、どちらかを選択できるとしよう（どちらのケースでも在庫処分価格は \$6.95 のまととする）。どちらがよいだろうか。2つの代替案について、もっともよい（最高収益を記録する）%の値を基にして比較しなさい。なお、結果について何らかの統計分析を加えた上で、比較すること。
12. 問題 11 において、広告代理店に価格を上げるための宣伝ではなく、ピーク期需要の最大を 5,000 から 6,000 に増加させるような宣伝を依頼することにしよう。仕入れ値、ピーク期の小売価格、および在庫処分価格は、それぞれ \$11.00、\$16.95、\$6.95 のまととする。広告代理店に関する 2 つの代替案について、もっともよい（最高収益を記録する）%の値を基にして比較しなさい。
13. 3.2.3 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 において、ピーク期後に残った帽子を（損失として）売る代わりに、慈善団体に寄付することで、寄付分の仕入れ額が法人税課税対象額から控除されるものとしよう。また、法人税率は 34% である。これは最終的な収益の足しになるだろうか。2つの代替案について、もっともよい（最高収益を記録する）%の値を基にして比較しなさい。
14. 3.2.4 項の Model 3-4 のオリジナルバージョンは、入力確率変数に対して、次の確率分布を用いていた： $Q \sim \text{Uniform}(8000, 12000)$ ； $P \sim \text{Normal}(10, 3)$ 、2.0 で左側を切り捨て； $V \sim \text{Normal}(7, 2)$ 、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 の値に範囲ができる。これらのオリジナルの分布を用いたモデルを構築し、そのモデルにより、3.2.4 節と同様の分析を行いなさい。
15. 3.3.2 項の待ち行列スプレッドシート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検討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、説明しなさい。
16. 3.3.2 項の待ち行列スプレッドシート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分へと減少させることにより到着率を増加させ、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、それから1.01へと減少させ、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双方に関して、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、0.99、0.95、0.88、最後に0.80へと変化させなさい。何が起こるか。2章で議論した「安定性」の観点から、結果は「有意義である」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回の反復実行を分けて、「有意義」かどうかについて考察すると良いだろう。
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、ポンド単位で各\$1.05、\$3.17、\$1.99、\$0.95の卸売価格で購入し、ポンド当たりの小売価格\$1.29、\$3.76、\$2.23、\$1.65でそれぞれを販売する。各日の需要量(ポンド)は、最小はすべての商品で0であるが、最大はそれぞれ10、8、14、11である。なお、彼はポンド単位でしか販売しないことにしている。各商品の需要量は、上記の範囲の離散一様分布に従うものとし、ワルサーはすべての需要を満たすじゅうぶんな在庫量を常に保有するものとする。夏商戦は90日間で、各日の需要量は他の日の需要量から独立している。各日およびシーズン全体のワルサーの総費用、総収入、総利益をシミュレートする、スプレッドシートシミュレーションを作成しなさい。
19. ある新車に\$22,000の店頭価格(希望小売価格)がつけられている。しかし、顧客が選べるオプション(AT、オーディオ、フォグラント、防眩バックミラー、ムーンルーフ)もあり、選んだオプションに応じて車の店頭価格に、\$0ないし\$2,000が上乗せされる。すべてのオプションの総費用は、その範囲に連続一様に分布する。また、多くの顧客は車の価格について、値下げ交渉を行う。この値下げ金額は、\$0ないし\$3,000の間で連続一様に分布し、オプションとは無関係であるとしよう。したがって、車の販売価格は、\$22,000にオプションの総費用を加え、さらに値下げ金額を控除した金額となる。
- a) 100台分の実際の販売価格を計算するExcelスプレッドシートシミュレーションを作成しなさい。ただし、Excelに備わっている機能だけを用いること。つまり、@RISKや他のサードパーティ製アドインは利用しない。100台分の販売価格について、標本平均、標準偏差、最小値、最大値を求めなさい。Excelの機能を用いて、(100台のみではなく)将来にわたって期待される販売価格の95%信頼区間を作成しなさい(ヒント:Excel関数TINVを用いて、入力パラメータを検討するとよい)。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の方法をまねするか、Excelのデータ分析ツールの「ヒストグラム」機能を用いなさい(データ分析ツールを利用するには、次の方法でインストールする必要がある。[ファイル]-[オプション]-[アドイン]から、画面の一番下の[管理]の「Excelアドイン」を選び[設定]を押す。それから「分析ツール」をチェックして[OK]を押す)。すべてをExcelで行うこと。つまり電卓や統計表は必要ない。
- b) 同じExcelファイルの2番目のシートで、a)を繰り返しなさい。ただし、今度は@RISKもしくは他の静的スプレッドシートシミュレーション用Excelアドインを活用し、必要な一様確率変量を生成するために、独自の数式ではなく、確率変数生成機能を用いること。なお、販売価格の期待値に対する信頼区間を計算する必要はない。100台ではなく、10,000台の車について、良好な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のヒストグラムを用いればよい)。また、すべてをExcelで行うこと(@RISKは除く)。
20. 問題19をより詳細に検討するために、5つのオプションの価格と、各オプションを顧客が選択する確率を表3.2に示す(たとえば、ある顧客がATを選択する確率は0.6である、のように読み取る)。顧客は、他のオプションを選んだかどうかに関わらず、各オプションを独立して選択すると仮定する。問題19のa)を繰り返しなさい。ただし、今回は表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か決める。なお、値下げ交渉は問題19と同じだが、

顧客のオプションの選択とは無関係である。すべてを Excel で行いなさい。つまり、@RISK は用いない。

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 と評価する。

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のプロセスを紹介することである。シミュレーションモデルを構築すると同時に、シミュレーション出力結果の統計分析も行う。このようにすれば、モデルを構築すると同時に、モデル化されたシステムでどのような有効な推論ができるかを理解できる。本章では、完結する Simio モデルを 1 つだけ構築する。そして、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紹介する。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、ここでは実行手段として Simio を用いることにする。

本章で使用される初步的なモデルはきわめて簡単であり、実行期間を除くと、3.3.1 節の手動で行われた Model 3-4 や 3.3.2 項で紹介されたスプレッドシートモデル Model 3-5 と基本的に同じである。このモデルの馴染みやすさとわかりやすさから、モデルよりもむしろその構築プロセスと基本的な Simio 概念を中心に理解することができる。そして、初步的なモデルにいくつかの簡単な変更を加えることで、追加的な Simio の概念を解説する。そして、次章では、より包括的なシステムをサポートするために、追加的な Simio の特徴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さらに拡張する。このモデルは、単一サーバ待ち行列システムであり、到着率 $\lambda = 60$ エンティティ/時間、サービス率 $\mu = 48$ エンティティ/時間である（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、または救急センターのトリアージナース、その他のさまざまなことを表すことが想定される。本章の目的においては、少なくとも当分の間は、何がモデル化されているかについてはさほど重要ではない。はじめに、到着過程がポアソン分布（すなわち、到着時間間隔は、指数分布に従い、互いに独立している）に従い、サービス時間は指数分布に従い、（互いに、そしてサービス開始時間間隔とは）独立しており、待ち行列は無限の容量があり、待ち行列のルールは先入先出（FIFO）であると仮定する。ここでは、待ち行列におけるエンティティの数（平均と最大の両方）、エンティティが待ち行列で待機する時間（再び平均と最大）、サーバの稼働率などの典型的な待ち行列関連の測定基準に关心がある。その关心が長期の実行または定常状態の挙動におけるものであるなら、このシステムは標準的な待ち行列解析法を使用することで容易に分析することができる（2 章で解説）。しかしここでは、Simio を使用してこのシステムをモデル化することに关心がある。

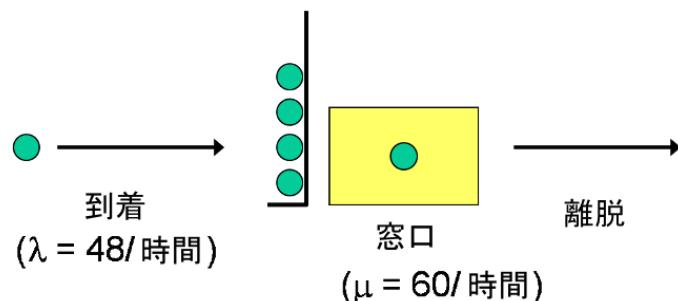


図 4.1 例題の単一サーバ待ち行列システム

本章は実際に Simio を使用して待ち行列システムをモデル化するための 2 つの方法を説明する。最初の手法は、Standard ライブラリ（4.2 節）から Facility ウィンドウと Simio オブジェクトを用いる。2 番目の手法は、下位レベルでモデルを構成するために、Simio Processes（4.3 節）を用いる。これは、適切またはさらに詳細なものをモデル化するのによく必要とされる。これらの 2 つの手法は、完全に別々な方法であるというわけではない。Standard ライブラリオブジェクトは、実際には Processes を用いて構築されている。あらかじめ構築された Standard ライブラリオブジェクトは、モデル構築のための上位レベルの、そしてより自然なインターフェースを提供しており、

オブジェクトに関する基本機能にアニメーションが備えられている。カスタマイズのための Processes は低レベルのインターフェースを Simio に提供しており、通常、特別な機能やより速い実行を必要とするモデルに用いられる。また、Simio では Standard ライブラリオブジェクトを構成する Processes にアクセスできるが、これについては後の章で取り上げる。

本章では、4.1 節の Simio ウィンドウとユーザインタフェースから解説する。上述したように、4.2 節は Standard ライブラリオブジェクトを用いてどのように Facility ウィンドウでシステムのモデルを構築するかを解説する。次に、このモデルをいくらか実験し、統計的に独立している反復や、ウォーミングアップ、定常状態対終結シミュレーション、そしてモデルの検証などの重要な概念を紹介する。4.3 節では、最初のモデルをオブジェクトではなく、Simio の Processes を用いて再構築する。4.4 節は、モデルの背景を初期モデルに追加して、到着時間間隔とサービス時間の分布を変更する。4.5 節と 4.6 節は、シミュレーション出力データの有効な統計分析について、Simio で活用可能な革新的なアプローチを利用する方法を示している。4.8 節は、基本的な Simio アニメーション機能について説明し、アニメーションをモデルに追加する。モデルを実行すれば、当初は気づかなかった挙動を発見でき、より興味深さが増すだろう。本章最後の 4.9 節では、モデルの問題点を検索し修正する手順を解説することにする。本章でモデル化されるシステムはかなり簡単であるが、この例題を終えると、どのように Simio のモデルを構築するかだけではなく、どのようにモデルを利用するかについても理解することが可能である。

4.1 基本的な Simio ユーザインタフェース

Simio モデルの構築を開始する前に、この節では、様々なモデリングの構成要素のうちで何を利用することができますか、どのように操作するかを紹介し、Simio のユーザインタフェースを概観することにする。

Simio を起動すると、新規 Simio モデルが表示される（デフォルト）。また、File ページの Load most recent project at startup のチェックボックスがチェックされていれば、もっとも直近に開いたモデルが表示される。新規 Simio モデルの画面を図 4.2 に示す。モデルの構築にすぐに取り掛かりたいだろうが、その前に、インターフェースや Support リボン（下記参照）で提供する Simio に関連する重要な詳細情報を探求されることを推奨する。遠回りのようだが結局は、これらの情報は時間の節約に役に立つ。

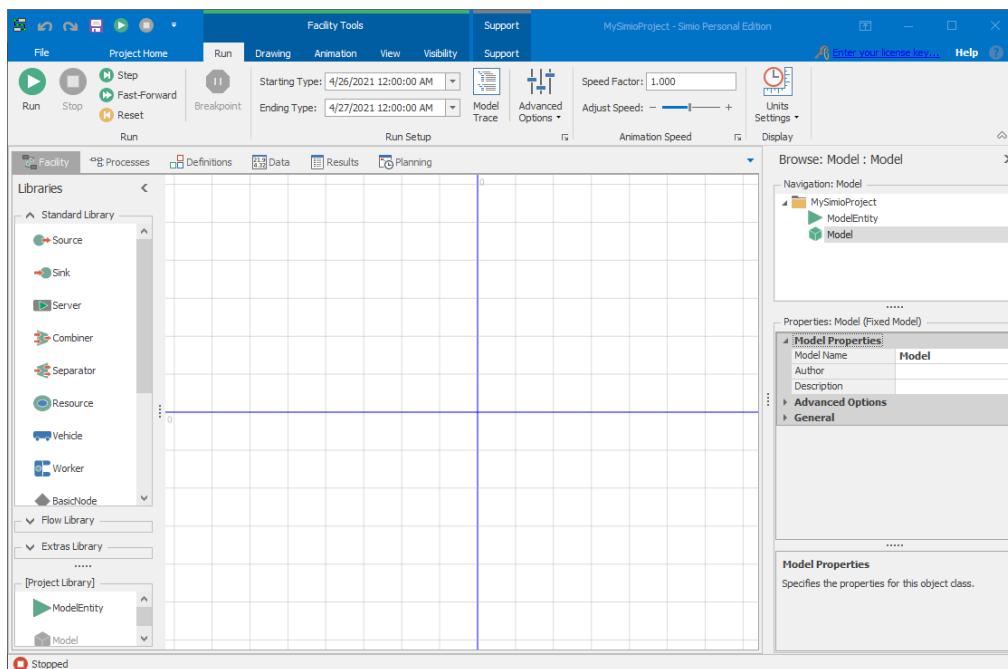


図 4.2 新規モデルの Facility ウィンドウ

4.1.1 リボン

リボン (Ribbon) は、Microsoft™ Office 2007 で取り入れられ、伝統的なスタイルであるメニューとツールバーから置き換えられた革新的なインターフェースである。リボンは、コンテンツを直感的に構成したり、自動調整をしたりすることで、迅速にタスクを完成させるのに役立つものである。コマンドはロジックグループで構成されており、そのグループはタブの下にまとめて集められている。各タブは、モデルの実行やシンボルの描写のように、アクティビティの種類によって関連づけられている。タブを自動的に表示させたり、または、作業の状況によって前面へ表示させたりすることもできる。たとえば、シンボルを作成中のときは Symbols タブが目立つようになる。どのリボンが表示されるかは、プロジェクトの場面に大きく依存することに注意してほしい（すなわち、インターフェースの要素のうち、どの項目が選択されているのかということに依存する）。

4.1.2 Support リボン

図 4.3 に示す Simio の Support リボンは、Simio を最大限に活用する情報が含まれている。また、モデルについてのアイディア、質問および問題点について、Simio のサポート技術者への連絡方法も紹介されている。それ以外の役立つ情報については、Simio Technical Support (<http://www.simio.com/resources/technical-support/>) を利用してほしい。このリンクから、技術支援のポリシーに関する説明や、Simio User Forum および Simio に関連するその他のグループへのリンクにアクセスできる。Simio のバージョン情報やライセンス情報も Support リボンから見ることができる。サポートが必要なときには、Support リボンから得られる情報が特に重要である。

Simio ウィンドウの右上にある?アイコンや F1 キーで、Simio の包括的なオンラインヘルプを閲覧することができる。印刷可能なバージョンを利用する場合は、Simio Reference Guide (pdf ファイル) を参照してほしい。ヘルプとレファレンスガイドでは、Simio の特徴について基本と応用にわたって解説している。索引つきの情報を提供しており、検索も可能である。Support パネルには、さらにトレーニングするための機会も提供されており、トレーニングビデオやその他のオンラインリソースへのリンクもある。また、サンプルプロジェクトや Simbits (後述) を開くリンクや、Simio 関連の書籍やリリース情報、Simio ユーザフォーラムなどへのリンクも提供されている。

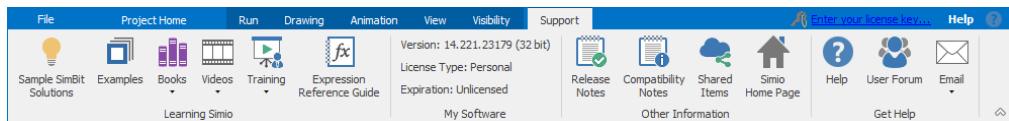


図 4.3 Simio の Support リボン

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、リボンのすぐ下に 2 つのタブのセットがある。これらは、アクティブモデルや実験と関連した複数のウィンドウを選択するために利用する Project Model タブである。利用可能なウィンドウは、選択されたモデルのオブジェクトのクラスに依存するが、通常、Facility、Processes、Definitions、Data および Results である。RPS ライセンスを利用している場合、Planning タブもある。後で詳細にそれぞれのウィンドウについて説明するが、はじめにモデルの構築、実験および実行の大部分を担う Facility ウィンドウについて時間をかけて解説する。

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、特に一般的なモデリング領域とテーマを中心としたオブジェクト定義の集まりである。ここでは、ライブラリについての簡単な紹介に留めるが、5.1.1.1 では、オブジェクト、ライブラリ、モデルとそれらの関係について詳細に解説することにする。ライ

ライブラリは、Facility ウィンドウの左側に表示されている。Standard ライブラリ、Flow ライブラリ、および Extras ライブラリはデフォルトで配置されており、Project ライブラリはプロジェクトに欠かせないものである。Standard、Flow、および Extras ライブラリを利用するには、ライブラリウィンドウの下方にあるそれぞれの名称をクリックする（ただし、同時に開けない）。Project ライブラリを開き続けるには、「…」で表されるセパレータをドラッグして、長さを調整してほしい。他のライブラリは、Project Home リボンにある Load Library ボタンから追加することができる。

Facility ウィンドウの左側の **Standard Object** ライブラリは、Simio の標準オブジェクトの汎用セットである。それぞれのオブジェクトは、モデル化される対象物、装置、またはアイテムを表す。多くの場合、Standard ライブラリからオブジェクトをドラッグして、Facility ウィンドウにそれらをドロップすることによって、モデルの大部分が構築される。表 4.1 は Simio Standard ライブラリのオブジェクトを記載している。

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、到着パターンのエンティティオブジェクトを生成する。
Sink	モデルでプロセスを完了したエンティティを破棄する。
Server	機械またはサービスオペレーションなどのような能力を与えたプロセスを表す。
Combiner	親エンティティと共に複数のエンティティを結合する（たとえば、パレット）。
Separator	エンティティのバッチグループを分割するか、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、解放される一般的なオブジェクトである。
Vehicle	固定ルートに従うか、またはオンデマンドで積み降ろしを実行できるトランスポータである。
Worker	ヒトに関連したアクティビティをモデル化する。可動オブジェクトとして、またはトランスポータとして用いることができ、シフト表に従うことができる。
BasicNode	複数のリンク間の簡単な交点をモデル化する。
TransferNode	目的地と移動方法を変更できる複雑な交点をモデル化する。
Connector	2つのノード間のゼロ時間のリンクである。
Path	エンティティが自身のスピードで主体的に動くことが可能なリンクである。
TimePath	すべてのエンティティについて、指定された時間で移動するリンクである。
Conveyor	蓄積型や非蓄積型のコンベヤ装置をモデル化するリンクである。

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る。したがって、プロジェクトで作成された新しいオブジェクト定義は、そのプロジェクトの Project ライブラリに表示されることになる。Project ライブラリ内のオブジェクトは、Navigation ウィンドウ（後述）経由で定義・更新され、Project ライブラリを介して（Facility ウィンドウに設置されることで）利用される。モデリングを簡素化するために、Project ライブラリには ModelEntity オ

プロジェクトがあらかじめ用意されている。Flow ライブラリは、フロータイプの処理システムをモーデリングするためのオブジェクトの集合である。Extras ライブラリには、マテリアルハンドリングや倉庫に関するオブジェクトが含まれている。これらのライブラリに関する情報は Simio ヘルプを参照してほしい。また、その他の分野に特化したライブラリについては、Support リボンにある Shared Items ボタンをクリックするか、Simio User Forum へアクセスしてほしい。なお、独自オブジェクトの作成方法については、11 章で述べることにする。

4.1.5 プロパティウィンドウ

右下の Properties ウィンドウは、現在選択されているオブジェクトやアイテムのプロパティ（特性）を表示する。たとえば、Server が Facility ウィンドウに置かれたとして、それが選択されているとき、Properties ウィンドウでそのプロパティを表示して、変更することができる（図 4.4 参照）。グレーのバーは同類のプロパティのカテゴリかグループを示している。初期設定により、利用頻度の高いカテゴリは、すべてのプロパティを見る能够性を擴張される。初期設定で利用頻度の少ないカテゴリは折りたたまれているが、「+」印を左クリックすることによって、それらを擴張することができる。プロパティ値を変更すると太字で表示され、デフォルト値からの差異の認識を簡単にするために、そのカテゴリは擴張される。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、Reset を選択すればよい。

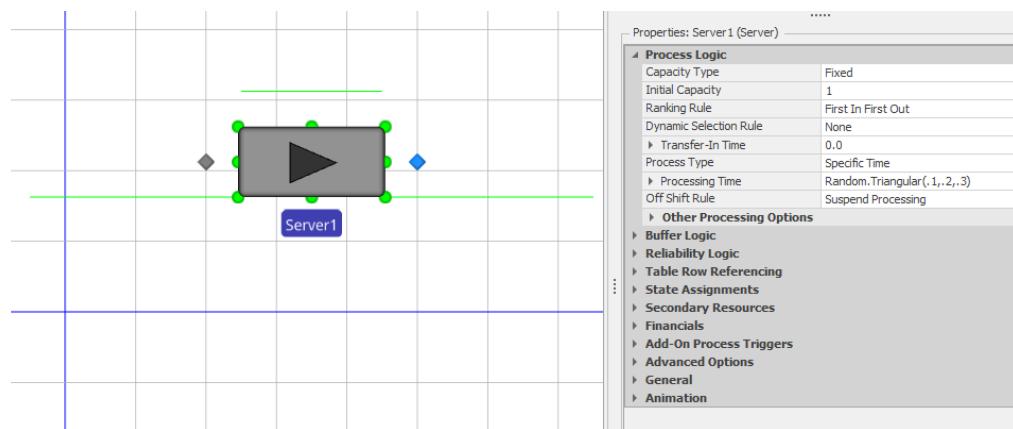


図 4.4 Server オブジェクトのプロパティ

4.1.6 ナビゲーションウィンドウ

Simio プロジェクトは、シンボルや実験のような他のコンポーネントと同様に、1 つ以上のモデルやオブジェクトで構成される。コンポーネント間は、右上の Navigation ウィンドウを用いて操作できる。新しいコンポーネントを選択する都度、それに従ってタブとリボンが変更される。たとえば、Navigation ウィンドウで ModelEntity を選択すると、少し異なるプロジェクトモデルのタブセットが利用可能であることに気づくだろう。そして、エンティティオブジェクトの状態を加えるための Definitions タブを選択することができる。そして、メインのモデルを編集するために、Navigation ウィンドウにおけるオリジナルの Model オブジェクトを選択する。どのオブジェクトを操作しているかわからないときは、Navigation ウィンドウの上部のタイトルバーか、Navigation ウィンドウのハイライトしているバーを確認する。オブジェクトがライブラリに表示され、かつ Navigation ウィンドウにも表示されることは、初心者ユーザにとって混乱されやすい点である。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ことに留意してほしい。その他の詳細情報は 5.1.1.1 で述べることにする。

4.1.7 SimBits

SimBits は、ユーザが確実に利用したくなる機能の 1 つである。SimBits は小さくてわかりやす

く編集されたモデルである。それぞれ、モデル化の概念を解説しており、あるいは一般的な問題を解決する方法を説明している。付属の pdf ファイルやオンラインヘルプでそれぞれの完全な解説を参照できる。メニュー項目 Open から直接それらを読み込むことができる（現在開かれているモデルが置き換える）が、有用な SimBit を検索するもっともよい方法は、Support リボンにある SimBit ボタンで探すことである。このボタンからターゲットページに進むと、興味がある SimBits を迅速に検索して読み込むことができるフィルタリングメカニズムにより、すべての SimBits の分類リストを見つけることができる（この場合、別の Simio ウィンドウが起動し、構築中のワークスペースは保持される）。SimBit は、モデリング技術、オブジェクト、および構成要素に関して、新しい知識を習得するために役立つツールである。

4.1.8 ウィンドウとタブの移動・構成

上述の内容では、初期設定のウィンドウ位置で言及してきたが、容易にいくつかのウィンドウの位置を変更できる。右クリックするかまたはドラッグすることによって、設計時や実験時のウィンドウとタブ（たとえば、Process ウィンドウ、あるいは個々のデータテーブルタブ）をそれらの初期位置から変更することができる。ドラッグしている間、レイアウトターゲットと呼ばれる 2 セットの矢印が表示される：1 つはウィンドウの中心部に表示され、もう 1 つはウィンドウの外部に表示される。たとえば、図 4.5 では、テーブルのタブをドラッグし始めた場合のレイアウトターゲットを示している。テーブルタブをいずれかの矢にドロップすると、テーブルはその位置で新しい画面で表示されることになる。

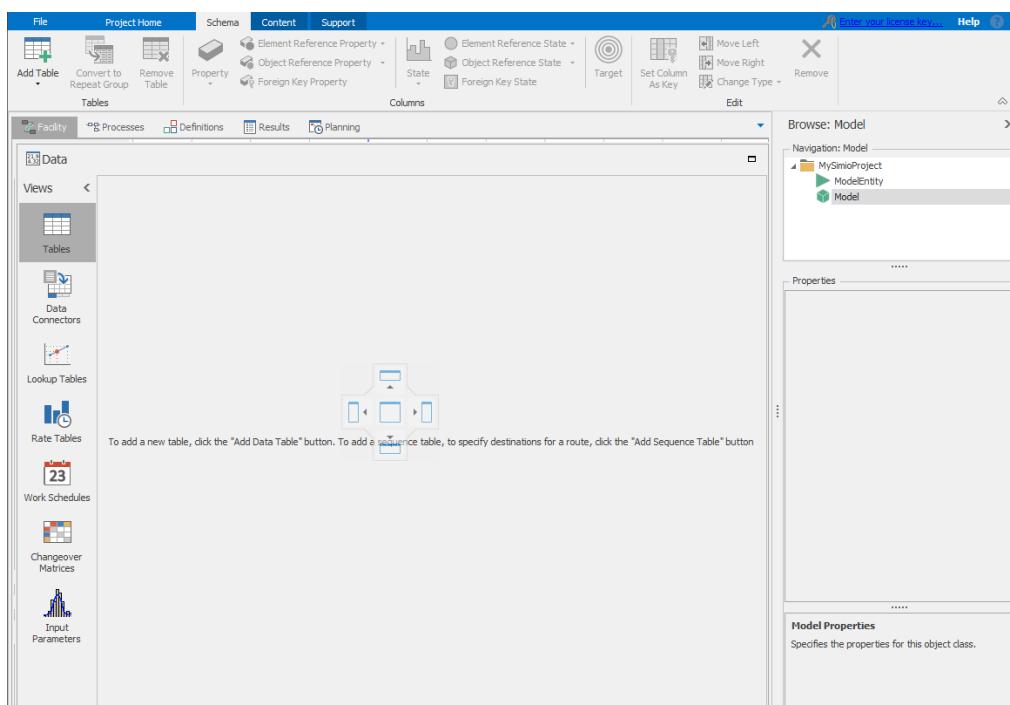


図 4.5 タブ化したウィンドウをドラッグし新しい表示位置へと移動する様子

タブを右クリックし、適切なオプションを選択することにより、水平・垂直のタブグループにウィンドウ配置を調整できる。また、スクリーンを最大限に利用するために、Simio アプリケーションの外部や別のモニタにいくつかのウィンドウ（Search、Watch、Trace、Errors、オブジェクトの Consoles）をドラッグすることができる。ユーザ自身によるカスタマイズしたウィンドウ配置を止めたい場合やウィンドウの場所がわからなくなった場合は、ウィンドウをデフォルトの配置に戻すため、Project Home リボンにある Reset ボタンをクリックする。

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、何回かの実験と分析をする：4.2.1 項は Simio を用いてどのようにモデルを構築するかについて述べる。Standard ライブラリを利用した Facility ビューにおいて、1 回実行し、結果を概観する。次に、4.2.2 項は標準の待ち行列理論での期待値と比較するために、はじめにシステムについて初期の非公式な実験を実施する。4.2.3 項は、シミュレーションの出力結果の分析や統計的な繰り返しに関する概念と、Simio ヘルプをどのように利用するかについて紹介する。4.2.4 項では、長時間の実行と短時間の実行についての議論を展開する。長時間の実行により、モデルの物事がどのように挙動するかに关心があるならば、モデルのウォームアップが必要となることがわかるだろう。4.2.5 項は、4.2.2 項で取り上げられた、いくつかの同じ問題を再考する。特に、モデルの妥当性を検証する。しかし、すでに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のような、よりよいツールを身につけているので、ここでの議論は、興味ある単独のシナリオ（システム構成）のみを考えることとする。5.5 節および 9.1.1 項で代替シナリオを比較するという、より一般的な目標について議論し、そこでいくつかの追加的な統計手段を紹介する。

4.2.1 モデルの構築

Simio モデルは、Standard ライブラリオブジェクトを用いる方法がもっとも一般的な方法である。あらかじめ用意されたこれらのオブジェクトは、多くの一般的なタイプのモデルに適応する。図 4.6 は、Simio の Facility ビューを利用した、待ち行列システムの完成モデルを示している（Facility タブが、Project Model タブエリアでハイライトされていることに注意する）。次の段落から、段階的にモデルを構築する方法について解説する。

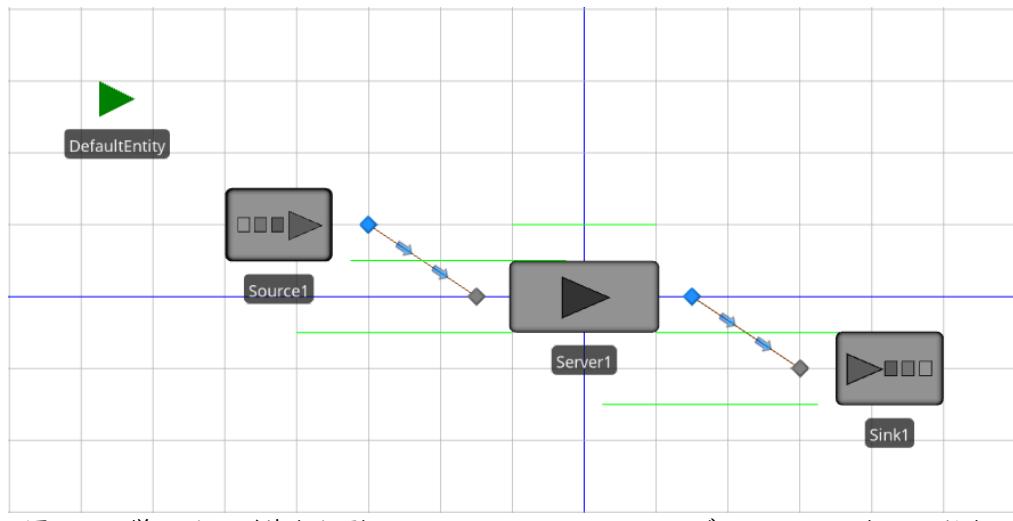


図 4.6 単一サーバ待ち行列システム(Model 4-1)のモデル(Facility ウィンドウ)

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、および離脱プロセスを含む。Facility ビューでは、Source、Server、および Sink オブジェクトを用いて、これらのプロセスをモデル化することができる。モデル作成を開始するには、Simio アプリケーションを起動させる。そして、必要に応じて、File ページにある New をクリックする（File リボンからアクセス可能）。デフォルトの新規モデルがいったん開くと、Facility タブをクリックすることによって Facility ビューが開き、左の Libraries バーの Standard Library セクションの見出しをクリックすることで Standard Library が表示される。図 4.2 は、上述の画面を示している。最初に、ProjectLibrary パネルにある ModelEntity オブジェクトをクリックして、ModelEntity オブジェクトを追加する。次に、それを Facility ウィンドウへドラッグアンドドロップする（オブジェクト

は ProjectLibrary パネルで定義しているので、実際には、そのインスタンスをドラッグして、ドロップしていることになる)。次に、Standard Library にある Source オブジェクトをクリックして、それを Facility ウィンドウへドラッグアンドドロップする。同様に、Server オブジェクトと Sink オブジェクトのインスタンスを Facility ウィンドウにドラッグアンドドロップする。次のステップでは、モデルの Source、Server、および Sink オブジェクトを接続する。この例では、標準の Connector オブジェクトを用いる。それは、ノード間のエンティティの搬送が、ゼロシミュレーション時間であることを表す。このオブジェクトを用いるには、Standard Library にある Connector オブジェクトをクリックする。Connector を選択すると、カーソルは十字線に変化する。新しいカーソルで、Source オブジェクトの Output Node (右側) をクリックする。そして、次に Server オブジェクトの Input Node (左側) をクリックする。この操作は、Source オブジェクトから Server オブジェクトへエンティティが(即座に、すなわちゼロシミュレーション時間で) 流動することを Simio に設定していることになる。同じ操作で、Connector を Server オブジェクトの Output Node と Sink オブジェクトの Input Node の間に追加する。図 4.7 は Source と Server オブジェクトの間にコネクタを接続したモデルを示している。

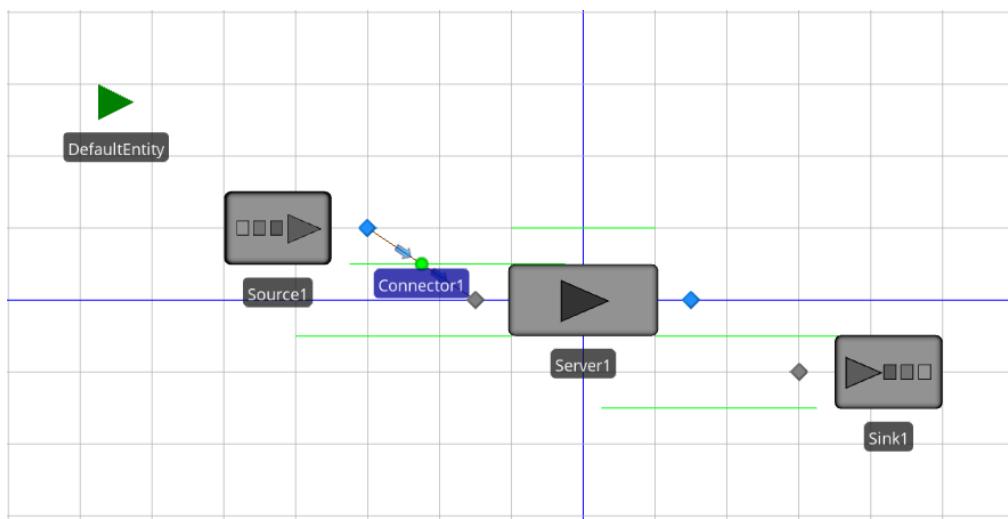


図 4.7 Connector オブジェクトで Source と Server がリンクされた Model 4-1

ところで、今、モデルを保存するよいタイミングであるだろう(よく知られているように「早めに保存して、こまめに保存する」ことがよい)。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ためのファイル名の拡張子である)。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利用可能である(付録 C 参照)。

モデルの構築を続ける前に、デフォルトで待ち行列を含んでいるいくつかの Standard ライブラリオブジェクトについて言及する必要がある。これらの待ち行列は、図 4.7 で示したように水平の緑の線によって表される。エンティティが待つ(すなわち、シミュレーション時刻のある期間、モデル上の同じ論理的な場所に留まる)可能性のあるところで、Simio は待ち行列を用いる。なお、技術的には、エンティティではなくトークンが Simio の待ち行列で待っているが、この問題についての詳細は 5 章で議論する。ここでは、アニメーションに表示させるため、待ち行列で待機しているエンティティと考えてほしい。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離脱を待つエンティティを蓄積する。
- Server1|InputBuffer.Contents : Server オブジェクトへの進入を待つエンティティを蓄積する。

- Server1 Processing.Contents : 現時点での Server オブジェクトによって処理されているエンティティを蓄積する。
- Server1 OutputBuffer.Contents : Server オブジェクトからの離脱を待つエンティティを蓄積する。
- Sink1 InputBuffer.Contents : Sink オブジェクトへの進入を待つエンティティを蓄積する。

図 4.1 の簡単な単一サーバ待ち行列システムでは、ただ 1 つの待ち行列を示している。そして、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、任意の時点でのシミュレーション時刻で処理中のエンティティを蓄積する。他の待ち行列は、このモデルで利用されていない（実際には、ゼロシミュレーション時間であり、エンティティはこれらの待ち行列で即座に移動する）。

この時点で、モデルの基本的構造を完成させたので、オブジェクトにモデルパラメータを追加しよう。この簡単なモデルへ、到着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うエンティティの到着を生成する。毎時 $\lambda=48$ 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が指数関数の平均 1.25 分に従うとして、Simio に設定する（エンティティの到着時間間隔 = $60/48$ は毎時 48 の到着率と同じである）。正式な Simio オブジェクトモデルにおいて、到着時間間隔は Source オブジェクトのプロパティの 1 つである。オブジェクトのプロパティは、Properties ウィンドウで設定および編集される。Source オブジェクトを選択（オブジェクトをクリック）し、右のパネル上に Properties ウィンドウを表示する（図 4.8 参照）。

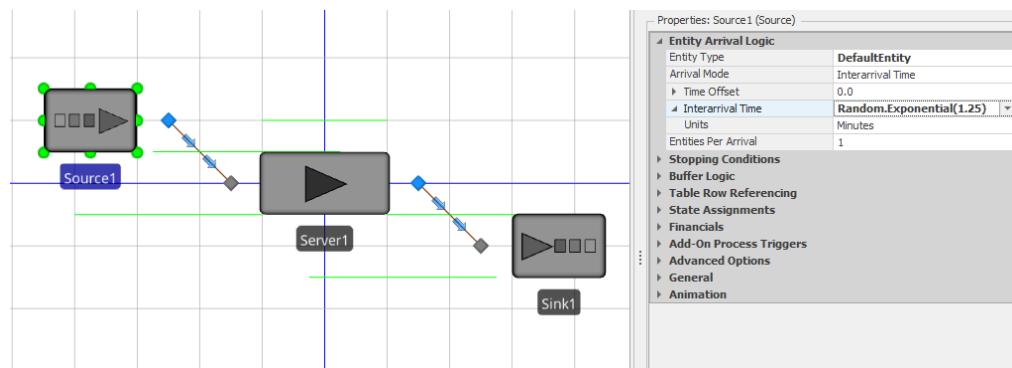


図 4.8 Source オブジェクトの到着時間間隔分布の設定

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)、そして、Units プロパティを Minutes と設定する。ここで、Units プロパティを展開するには、Interarrival Time の左側の矢印をクリックする。そして、右側のプルダウンを利用して Minutes を選択する。これは、エンティティを生成する時間であり、平均 1.25 で指数分布からランダムな値を抽出して、そして次のエンティティの到着率は $\lambda = 60 \times (1/1.25) = 48$ エンティティ/時として生成されるということを、Simio に設定している。4.4 節で、より詳しく Random というキーワードから、利用可能な乱数の機能について議論する。Time Offset プロパティ（通常、0 に設定）は、最初のエンティティが作成される時間を決定する。現在のところ、Arrival Logic に関連している他のプロパティは、デフォルトのままにする。これらのパラメータで、シミュレーション実行時間中、エンティティは反復的に生成される。

デフォルト・オブジェクト名（最初のソースオブジェクトでは Source1）は、変更可能である。選択されているオブジェクトの下のネームタブをダブルクリックするか、または、General プロパティセクションの Name プロパティで変更する。または、F2 キーを利用すれば、Simio のほとんどのアイテムの名前を変更することができる。また、General セクションには、Description プロ

パーティを含んでいることにも注意してほしい。モデルの説明にとても役立つ場合があり、各モデルのオブジェクトにおいて重要な記述を加える習慣を持つとよい。そこに入力されたすべての内容は、オブジェクト上にマウスをかざしたとき、ツール・チップ・ポップアップ・ノートに表示されるからである。

モデルの待ち行列のロジックを完成するために、Server オブジェクトに関する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 プロパティは、エンティティの処理時間を指定するために用いられる。このプロパティは、Units プロパティを Minutes とし、Random.Exponential(1) に設定する。ここで変更されたのは Processing Time プロパティであり、Process Type プロパティではないことに留意してほしい (Process Type はデフォルトの Specific Time のままでする (他のオプションについては、10.3 節で説明する))。最後のステップは、モデルを 10 時間実行するように Simio に設定することである。この設定をするには、Run リボン/タブをクリックし、そして、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、モデルの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明確に制御するのに用いられる。Speed Factor を 50 に変更する (Run リボンの Speed Factor フィールドに入力する) と、このモデルにとって、より視覚的に魅力的な速度まで加速する。対話型実行での最適な Speed Factor は、コンピュータの速度と同様に、モデルやオブジェクトパラメータ、および個々の選択に依存している。したがって、各モデルによって Speed Factor を調整してほしい (技術的に言えば、Speed Factor は、秒単位で表示するアニメーションフレーム間のシミュレーション時間の量である)。

ここで、実際に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 10 (設定した実行期間) に達すると、モデル実行は自動的に止まる。

対話型では、モデルを停止したり一時停止したりすることで、またはタブバーの Results タブをクリックすることで、モデルの結果を隨時確認できる。10 時間に達するまでモデルを実行し、現在の結果を確認してみよう。Simio は基本的なモデル結果を見るために、いくつかの方法を提供している。Pivot Grid と Reports である (左のパネルの上にオプションがあるので、対応するアイコンをクリックすると、画面を切り換えることができる)。図 4.9 は、10 時間実行して停止した Model 4-1 の Pivot Grid を示している。

「はじめに」で言及したように、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。その結果、対話型で例題を実行して得た出力値は、ここに示した出力数値 (この本を執筆したとき得た出力数値) に合っていない可能性がある。1.4 節の終わりに述べたとおり、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をしないことを強調している。これは、この本で繰り返して指摘しているポイントである。

The screenshot shows the Project Home ribbon with tabs for File, Project Home, Pivot Grid, and Support. Below the ribbon is a toolbar with icons for Run, Stop, Fast-Forward, Reset, Export Results, Change View, Add View, Manage Views, Units Settings, and Display. A navigation bar at the top includes Facility, Processes, Definitions, Data, Results, and Planning. On the left, a sidebar lists various report types: Pivot Grid, Reports, Dashboard Reports, Table Reports, Resource Gantt, Entity Gantt, and Logs. The main area displays a Pivot Grid report titled 'Average' for 'ModelEntity'. The report has columns for Object Type (DefaultEntity), Data Source ([Population]), Category (Content, FlowTime, Throughput), Data Item (NumberInSystem, TimeInSystem, NumberCreated, NumberDestroyed), Statistic (Average, Maximum, Total), and Average Total. The data shows values like 2.8790 for Average Total of NumberInSystem under Content. The report is filtered by 'Server' and 'Server1' and uses '[Resource]' as the Data Source. The bottom of the screen shows a progress bar at 100% and the date/time as (10.00 Hours) Monday, April 26, 2021 10:00:00 AM.

図 4.9 Model 4-1 の Pivot Grid レポート

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だろうが、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もできる。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もできる。

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) にある TimeInSystem (Data Item) の Average (Statistic) は、0.0613 時間 (0.0613 時間×60 分/時=3.6780 分) である。Pivot Grid の時間、長さ、および割合に関する単位を Pivot Grid リボンの Time Units、Length Units、および Rate Units のアイテムで設定できることに注意する。Minutes に設定すると、TimeInSystem の Average は 3.6753 となるので、手計算で得られた 3.6780 分の値は、少し丸め誤差がある。さらに、TimeInSystem (Data Item) は、FlowTime (Category) に属し、その値はエンティティ (動的オブジェクト) に基づいているので、Data Source は Dynamic Objects のセットになっている。

画面で Pivot Grid をスクロールすると、このような小さなモデルさえも、多くの出力パフォーマンス指標が確認できる。たとえば、0.0613 時間の TimeInSystem の Average のちょうど 3 つ

下の行では、Throughput (Category)において合計 470 のエンティティが作成された（すなわち、Source オブジェクトを通してモデルに進入している）ことが確認できる。そして、Throughput (Category) の下の行では、合計 470 のエンティティが破棄された（すなわち、Sink オブジェクトを通過してモデルから離脱した）ことがわかる。いつも当てはまるわけではないが、このモデルの実行で 10 時間に到着した 470 のエンティティのすべては、離脱したことになる。つまり、シミュレーションの終わりにエンティティは 1 つもない。それは、10 時間実行した最後で一時停止をし、アニメーションを見ることで確認できる（たとえば、9 時間に実行時間を変更し、次に、11 時間に変更してみると、いつもこのように終わるわけではないことがわかる。また、Pivot Grid の Throughput (Category) で NumberCreated や Number Destroyed を確認することでもわかる）。10 時間の実行において、平均システム時間の出力値である 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における単純平均である。

シミュレーション実行期間を様々に変更する際、8 分に変更してみてほしい。そして、図 3.8 で示している 3.3.1 節での手動のシミュレーション結果と Pivot Grid 結果を比べてほしい。実は、手動シミュレーションの「不思議な」到着時間間隔とサービス時間は、この Simio 実行より発生され、Model Trace 機能を用いて数値を記録したのである。

Pivot Grid は 3 種類の基本型のデータ操作をサポートしている：

- ・ グルーピング (Grouping)：異なる相対的位置に列見出しをドラッグすると、データのグループ化が変化する。
- ・ ソーティング (Sorting)：個々の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を支援するダイアログが表示される。フィルタを適用した列は、漏斗アイコンが表示される（マウスを上に置く必要はない）。データをフィルタリングすると、出力のデータ量にかかわらず、興味のある特定のデータを閲覧することが可能である。

Pivot Grids では、Pivot Grid において行ったフィルタ、ソート、分類に関して、複数の画面（ビュー）を保存できる。これらのビューは、パフォーマンス評価尺度を監視することに役立つだろう。Simio ドキュメンテーションセクションでは、Pivot Grid について、具体的な機能をどのように利用するかについての詳細を説明している。Pivot Grid 形式は、出力の列が多いときに情報を検索する際には非常に有用な形式である。

Reports 形式は対話型実行の結果を提供している。それは形式化されており、詳細なレポート形式や、印刷、他のファイル形式への出力、または電子メール送信に対応している（形式、印刷、および出力オプションは Ribbon の上の Print Preview タブから利用可能である）。図 4.10 は、リボンの Print Preview タブが開いた状態の、Reports 書式を示している。左の見出しが TimeInSystem - Average (Hours) までスクロールすると、0.06126 の値が見える。この値は、図 4.9 の Pivot Grid での出力パフォーマンス指標と同じ（四捨五入による）である。

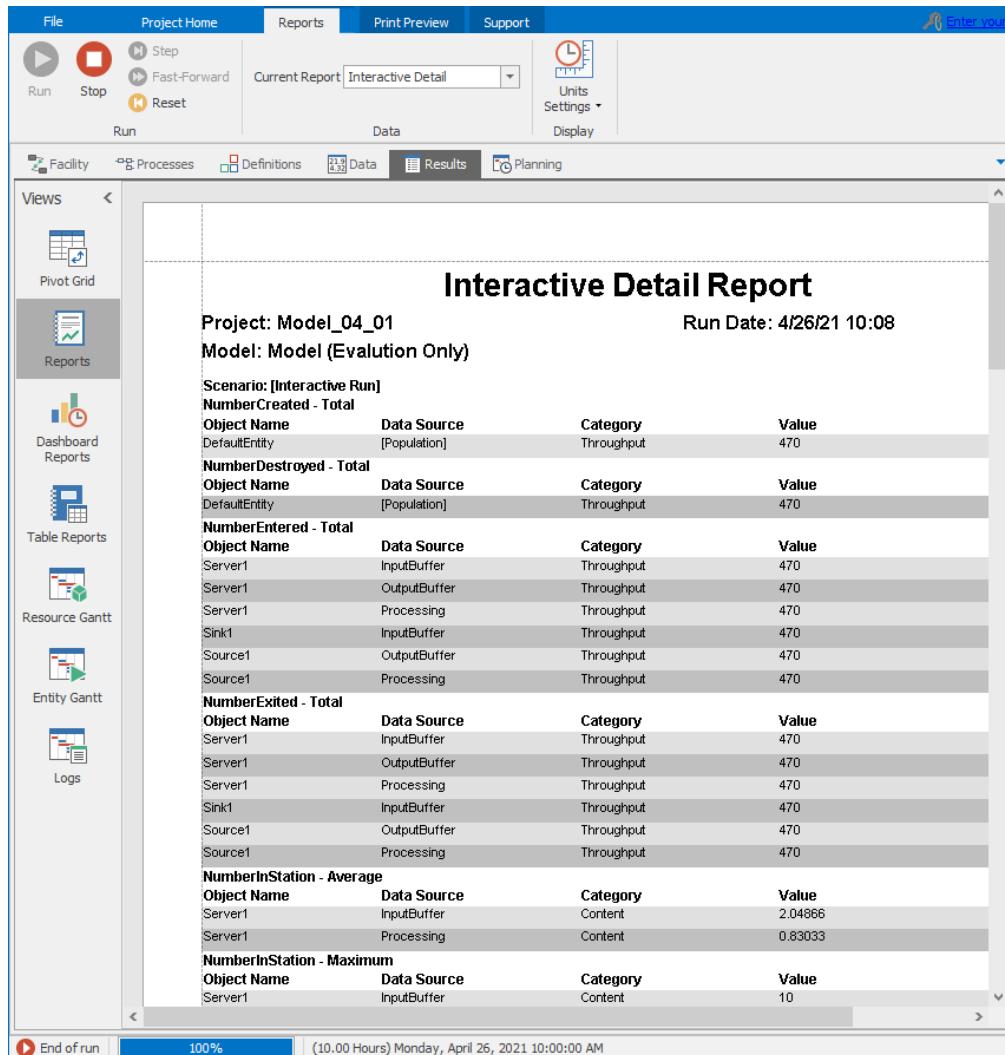


図 4.10 Model 4-1 の標準レポートビュー

4.2.2 初期の実験と分析

最初の Simio モデルを完成させたので、待ち行列システムを理解するために、最初の非公式な実験と分析を行う。すでに言及したように、システムにおける長時間の実行や定常状態のパフォーマンスについては、待ち行列解析を利用して分析することができる（詳細に関しては 2 章を参照）。このタイプの正確な解析は、簡単なモデル以外には適用できないことに注意する（この事実が、シミュレーションを用いる理由である）。表 4.2 は、定常状態の待ち行列の結果と図 4.9 の Pivot Table から得られたシミュレーションの結果を示している。

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	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

すぐに待ち行列の列における数値とモデルの列における数値が等しくないことに気づくだろう。その違いの理由を議論する前に、より重要であり関連する問題について議論する必要がある。Facility ウィンドウに戻り（リボンのすぐ下の Facility タブをクリックする）、モデルをリセット

して (Run リボンで Reset アイコンをクリックする)、モデルを再実行する。それを 10 時間、停止するまで実行させ、Pivot Grid を見ると、結果が前回の実行結果 (図 4.9) と同じであることに気づくだろう。このプロセスを何度も繰り返しても、いつも同じ出力値を得られる。3.1.3 項で言及したようにモデルのエンティティ到着時間間隔とサービス時間に乱数を用いているなら、シミュレーションに慣れていない多くの人にとって、これは妙であると思うだろう。これについては次に、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 項で指摘し、6.3 節で議論するように、使用される乱数は、予測できないという点から見ると真のランダムではない。この文脈では、**疑似乱数**であり、生成される数値の正確な順序が決定的であることを意味している。
2. 6.4 節で乱数発生法の過程について議論することにするが、いくつかのシミュレーションソフトウェアでは擬似乱数生成を制御でき、その制御を利点として利用できる。

「推定の乱数」が実際には予測できるという概念は、新たにシミュレーションを習得する人にとっては、確かに不安感を感じさせるのである。しかし、シミュレーションにとってこの予測は実際にはよいことである。それは、シミュレーション課題の採点を簡単にする（著者にとっては重要である）だけではなく、(より重要な) モデルのデバッグにおいても有用である。たとえば、シミュレーション出力において予測可能な影響があるようなモデルに変更を加えるとき、同じ目的のシミュレーションで同じ「乱数」入力を用いることができることは、非常に便利である。出力のあらゆる変化（または、不足）は、異なる乱数を得たことではなく、直接モデルの変更に起因したことがわかる。モデリングに踏み込むにつれ、モデルをデバッグするのにかなりの時間を費やしていることに気づくので、このことが有用であることがわかるだろう (4.9 節でデバッグプロセスや Simio のデバッグツールを中心に解説する)。さらに、この予測性は、**分散減少** (variance reduction) と呼ばれるさまざまな手法で、シミュレーションの実行時間を短縮することが可能である。このことは、一般的なシミュレーションのテキスト (たとえば、Banks et al. (2005) や Law (2015)) で議論されている。Simio のデフォルトの挙動では、モデルが実行されるたびに、同じ系列の乱数 (到着時間間隔やサービス時間のようなモデルを駆動する入力に関する観測値) を利用している。その結果、モデルを実行して、リセットして、再実行すると、別のある方法で動くように明確にコード化されないかぎり、そのモデルは同じ結果をもたらす。

ここで、なぜ初期のシミュレーションモデル結果と待ち行列の結果 (表 4.2) が等しくないかについての疑問に戻る。このミスマッチには、3 つの解釈が考えられる：

1. Simio モデルが間違っている。すなわち、モデル自体のどこかに誤った設定をしている。
2. 期待値が間違っている。すなわち、シミュレーションの結果が待ち行列理論による結果に合うべきであるという仮説が間違っている。
3. サンプリング誤差がある。すなわち、シミュレーションモデル結果が確率的な意味における期待値に合っていても、じゅうぶん長い時間、またはじゅうぶんな反復 (同じ状態から始めるが、別々の乱数を用いることでそれぞれ独立している実行) だけモデルを実行していないか、または結果を誤って解釈しているかである。

実際、期待値とシミュレーションの結果を見比べて結果が等しくないなら、モデルにかかわらず、いつもそれらのうちのどれかである可能性がある。このケースでは、期待値が間違いであり、じゅうぶんに長い間モデルを実行していない。待ち行列理論の結果は、長い時間の実行であり、定常状態のものであることを思い出してほしい。すなわち、モデルが本質的には無限の時間実行しているシステム/モデルの後に得られる結果であるが、このモデルでは、明らかにじゅうぶんでない実行時間であり、ほんの 10 時間しか実行していない。またじゅうぶんな反復をしていないのがわかる

(上の項目 2 と 3)。期待値を開発すること、期待値とシミュレーションモデルの結果を比較すること、そしてこれらが収束するまで繰り返すことは、**モデル検証と妥当性の確認**においては、非常に重要な要素である(4.2.5 項で再議論する)。

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**は、入力確率変数と乱数において、特定かつ別個であり、重複のない順列を用い、固定した開始および終了の組合せによってモデルが実行される(このケースでは、到着時間間隔とサービス時間は指數関数に従う)。ひとまず、開始と終了の状態は、開始と終了のシミュレーション時間によって、命令されると仮定しよう(後述するが、他にも設定可能な開始と終了における状態の種類がある)。そして、空かつ遊休状態でモデルを開始させて、10 時間実行すると、1 回の反復は成立する。モデルをリセットして再実行すると、同じ反復実行が再び成立する。上述したように、同じ乱数が入力されて、その結果として確率変数が入力されるので、同じ結果がもたらされる。**異なる反復**を実行するためには、異なる、別々の、そして重なっていない乱数と確率変数を入力設定する必要がある。幸い、Simio では、ユーザに意識させないようにこの過程を扱うが、対話型においては複数の反復実行をすることができない。代わりに、Simio Experiment を作成して、実行しなければならない。

Simio Experiments は、ユーザに指定された反復回数でモデルを実行させることができる。基本的に乱数は反復では重ならないため、Simio で生成した**確率変数**が、反復において互いに統計的に独立していることを保証している。必要とする統計分析において、独立性を保証することは重要である。実験を設定するために、Project ホームリボンを選択し、そして、New Experiment アイコンをクリックする。新しい実験を生成するために、上方の Replications Required と右側の Default Replications をデフォルトの 10 から 5 へ変更すると、図 4.11 に示されている Experiment Design ビューのようになる。

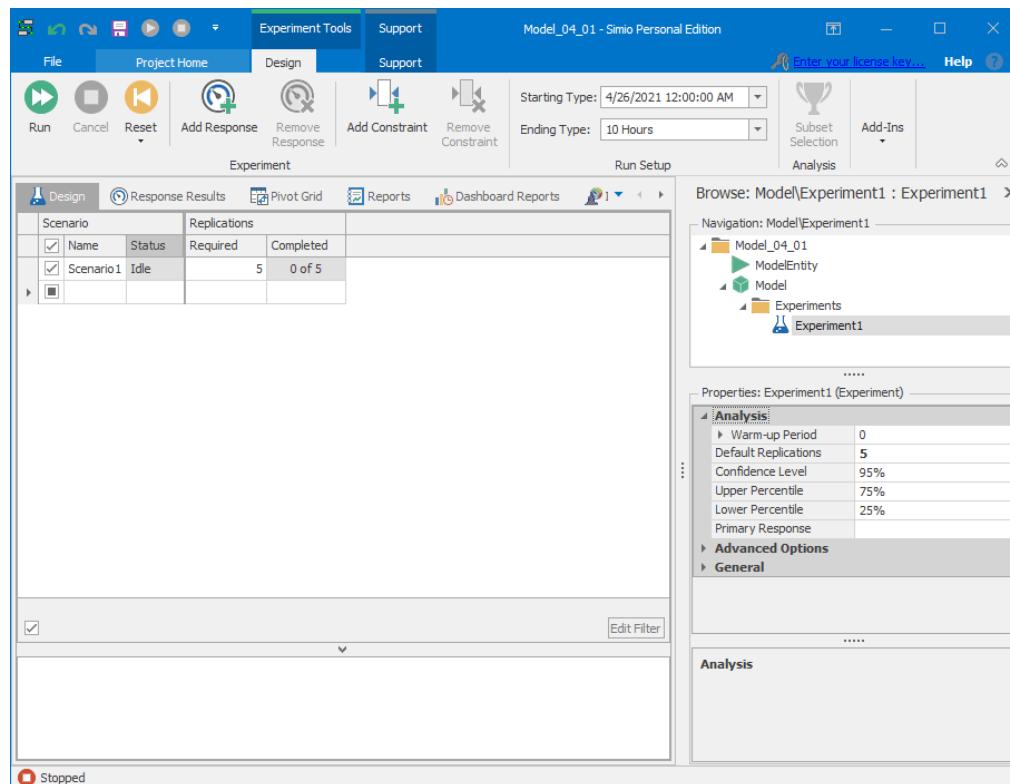


図 4.11 5 回の反復実行の初期実験計画

実験を実行するために、Scenario1(デフォルト名)の列を選択し、そして、Run アイコン(Model

ウィンドウにある 1 つの白い右矢印の Run アイコンではなく、Experiment ウィンドウにある 2 つの白い右矢印の Run アイコン) をクリックする。Simio が 5 回の反復を実行した後、Pivot Grid レポート (図 4.12 で示す) を選択する。

The screenshot shows a Pivot Grid report titled "Scenario 1". The columns are labeled "Object Type", "Object Name", "Data Source", "Category", "Data Item", and "Statistic". The rows show various metrics for ModelEntity and Server objects over time. The data includes Average, Minimum, Maximum, and Half Width values for metrics like NumberInSystem, TimeInSystem, and Throughput.

Object Type	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	Average (Hour)	0.0762	0.0489	0.1306	0.0395
					Maximum (Hour)	0.2888	0.2018	0.5096	0.1601
					Minimum (Hour)	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
					Total	464.4000	46.0000	479.0000	15.5208
Server	Server1	[Resource]	Capacity	ScheduledUtilization	Percent	78.7339	76.3323	83.0333	3.2689
					Total	465.2000	47.0000	480.0000	15.3912
				UnitsAllocated	Average	1.0000	1.0000	1.0000	0.0000
					Maximum	1.0000	1.0000	1.0000	0.0000
					Average	0.7873	0.7633	0.8303	0.0327
			ResourceState	TimeProcessing	Maximum	1.0000	1.0000	1.0000	0.0000
					Average (Hour)	0.0794	0.0578	0.0933	0.0166
					Occurrences	101.4000	89.0000	132.0000	21.7982
				TimeStarved	Percent	78.7339	76.3323	83.0333	3.2689
					Total (Hours)	7.8734	7.6332	8.3033	0.3269
			InputBuffer	Content	Average (Hour)	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
				HoldingTime	Total (Hours)	2.1266	1.6967	2.3668	0.3269
					Average	2.8718	1.5325	5.9590	2.2020
					Maximum	14.8000	10.0000	27.0000	8.6636
					Minimum (Hour)	0.2668	0.1500	0.5058	0.1764
					Half Width	0.0000	0.0000	0.0000	0.0000

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

対話型で実行している間に見た Pivot Grid (図 4.9) と比べると、Minimum (最小値)、Maximum (最大値)、および Half Width (半幅。期待値の 95% 信頼区間であり、これは Experiment Design プロパティの Confidence Level で編集可能である) の追加された結果の列を確認でき、それぞれの出力統計値として 5 つの独立した観測値があるという事実を反映している。

これらの反復実行間の出力統計が何であるかを理解することが重要である。エンティティの TimeInSystem 値の 3-5 行に注目してみると、たとえば：

- TimeInSystem の平均 (時間) の平均 (二度「平均」している) である 0.0762 は、それが反復の平均時間である 5 つの数値の平均である。5 回のうち最初の数は、図 4.9 の 1 回だけ実行した Pivot Grid からの 0.0613 であることがわかる。95% 信頼区間は、0.0762 ± 0.0395、または [0.0367, 0.1157] (時間) であり、95% の信頼度における反復内でのシステムの平均時間の期待値が含まれる。これは、このモデルのそれぞれ 10 時間の (5 回ではなく) 無限の反復実行を行い、反復実行内でのシステム平均時間のすべてを平均した結果であると考えることができる。この信頼区間がカバーしているものは、反復実行内の平均システム内時間として表されている、シミュレーションによって出力される確率変数に対する確率分布の期待値として捉えることができる (表 4.4 の説明で、出力における信頼区間についてさらに議論する)。
- Average(Hours) 行では、0.1306 はシステム内時間の反復実行内の 5 つの平均値の最大を表している (それぞれの最大値の平均ではない)。言い換えると、5 回の反復に対する 5 つの Average TimeInSystem 値でもっとも大きい値が 0.1306 であり、それが平均の最大となる。
- 次の行 Maximum(Hours) における 0.2888 は、5 回の反復内におけるシステム内のエンテ

イティ時間の最大の平均である。そして、95%の信頼区間 0.2888 ± 0.1601 によって、システム内時間の最大値の期待値をカバーしようとしている。すなわち、たった 5 回ではなく無数の反復実行を平均した、システム内時間の最大値をカバーしようとしている。

- おそらく、本当に最悪の場合のシステム内時間としてより重要であるのは、システム内時間に関して、5 回の反復実行中の最大である 0.5096 時間だろう。

表 4.3 は、5 回の反復実行における待ち行列の測定基準を示しており、それぞれ 5 回の反復実行における標本平均と標本標準偏差も示している。それぞれの反復出力値にアクセスするためには、Pivot Grid リボンで Export Details アイコンをクリックする。Pivot Grid の Export Summaries をクリックして、平均値や標準偏差などの反復実行間の結果を得る。出力データファイルは CSV 形式であり、Excel などのさまざまなアプリケーションで読み込むことが可能である。この表で最初に気づくことは、反復の間でかなり値が異なる（特に L と L_q ）ことである。このバラツキこそが、たった 1 回の反復結果から推論を得ることができない理由である。

表 4.3 初期モデルの 5 回の反復データ

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

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のそれぞれを $\hat{\rho}, \hat{L}, \hat{L}_q, \hat{W}, \hat{W}_q$ とする) は確率変数である。待ち行列解析では、正確な定常状態の値として ρ, L, L_q, W, W_q を与えている。どのように反復実行するか（同じモデルだが、別々の独立した確率変数を入力する）に基づいて、各反復はそれぞれの $\hat{\rho}, \hat{L}, \hat{L}_q, \hat{W}, \hat{W}_q$ の 1 つの観測値を生成する。統計用語でいうと、 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を利用して、確率変数の平均値を推計することができる。つまり、表 4.3 の平均列における値は、対応する確率変数の期待値の推定値である。この表から分からることは、推定値がどれくらい良いかということである。しかし、標本平均は一致推定量 (n と共にその分散は減少する) であり、**大数の強法則** ($n \rightarrow \infty$ につれ、反復による標本平均はそれぞれの確率変数の期待値に確率 1 で収束する) が成り立つことから、反復数を増加させるに従って推定はよくなることがわかっている。

表 4.4 は 50 回と 5 回の反復実行結果を比較している。より多くの反復実行によって、推定がよくなることを期待されるが、平均では、これらの推定値の品質（または、精度）についての特定の情報は少しも示していない。必要なことは、サンプリング誤差（平均は点推定である）に関する洞察を与えることになる区間推定が必要である。1 列は区間推定を示している。これらの列は通常の正規分布アプローチ（統計テキストの入門書にあるように、標本標準偏差と自由度 $n - 1$ のスチュードントの t 分布を用いる）から構成された平均値における 95% の信頼区間の半幅値を示している。

表 4.4 5 回の反復と 50 回の反復の比較

推定された 測定基準	5 回反復		50 回反復	
	平均	\pm	平均	\pm
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% の信頼区間 (CI) について反復実行が 5 回と 50 回について考えてみよう。

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 $1.456 \sim 5.862$ であると 95% で信頼することになる。一方、50 回の反復実行に基づくと、真の平均が $3.361 \sim 4.227$ であると 95% で信頼されることになる（厳密にいうと、この解釈は繰り返してこのように形成された 95% の信頼区間が、未知の真の平均を含むということである）。つまり、出力統計値の平均値の信頼区間によって、サンプリング誤差の基準と、確率変数の真の平均の推計値に対する品質（精度）が提供されている。反復実行（サンプル）数を増加させることによって、半幅を任意に小さくすることができる。たとえば、250 回の反復実行によって、 $[3.788, 4.165]$ となる CI の結果を得られる。明らかに、5 回の反復実行平均値の推定値よりも、250 回の反復実行に基づく推定値のほうに満足するだろう。独立した反復実行をさせる場合では、つまり、真の平均の正確な推定値を得たい場合、何回の反復実行をすべきであるかは信頼区間の半幅によって与えられる。信頼区間の半幅における公式の分母は \sqrt{n} であるため、初期の反復実行数と現在のサイズと比較して、信頼区間の半幅を半分にするためには、約 4 倍の反復実行をする必要がある。そして、現在のサイズから間隔を $1/10$ にするためには、約 100 倍の反復実行が必要である。残念ながら、「どれだけ狭めればじゅうぶんか？」（すなわち、与えられるシミュレーションモデルと決定状況（プロジェクトに関係しているアナリストかクライアントがしなければならない判断）において、 \pm の値がいくつであれば許容できるか）について、特定の規則はない。コンピュータ実行時間とサンプリング誤差を減少させることには、明確なトレードオフがある。上述したように、じゅうぶんな反復を実行することによって、 \pm を任意に小さくすることができるが、コンピュータ実行時間には費用がかかる。より多くの反復を実行するかどうか決めるとき、次の 2 つの問題が重要である：

1. サンプリング誤差による不正確な決定をした場合、費用はどれだけかかるか？
2. より多くの反復を実行する時間はあるか？

よって、表 4.2 に示されていたシミュレーションの結果が、待ち行列の結果に合っていない理由に関する最初の答えは、モデルの单一反復実行からの結果を用いていたということである。これは、サイコロを転がして 4（または、他の値）を観測し、その値が多く転がりを経て得た期待値であると宣言していることと同様である。明らかに、これは個々の転がりに関係ない、不じゅうぶんな推定値であるだろう。残念ながら、重大なリスクがあるにもかかわらず、シミュレーションを習得する初心者は、たった 1 回の反復実行から結果を用いることがよくある。前進するための一般的なアプローチは、複数の反復を実行し、出力統計の平均値を推計値として標本平均を用いるようにする。そして、真の平均の推定に关心があるのならば、適切な反復実行数を決定するのに役立つ 95% の信頼区間の半幅を用いることである。そして、結果を分析するとき、単に平均（推定値）を用い

ることの代わりに、信頼区間（区間推定）も用いる。実験（図 4.12 参照）の標準 Simio Pivot Grid レポートは、標本平均と 95% の信頼区間の半幅をすべての出力統計に提供することで、自動的にこのアプローチを支援している。

期待値とモデル結果間のミスマッチとなる 2 つ目の理由は、もう少し難解であり、モデルのウォームアップ期間の必要性にかかる。次節は、これを中心に詳しく議論することにする。

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の実行を開始するとき、モデルは、「空かつ遊休」と呼ばれる状態で開始する。それは、システムにはエンティティが全くなく、すべてのサーバが稼働していないことを意味している。簡単な単一サーバの待ち行列モデルで考えてみよう。到着する最初のエンティティは、サーバを決して待つ必要はない。同様に、(2 番目のエンティティの前に唯一あるエンティティは最初のエンティティなので、) 2 番目の到着エンティティはおそらく、100 番目に到着するエンティティより、待ち行列の（平均）待ち時間は少ないだろう。モデル化されるシステムの特性（このケースは、サーバ稼働率の期待値）に依存しており、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**の待ち時間の分布や期待値とかなり異なる場合がある。ここで、定常状態とは、空かつ遊休の**初期状態**の影響が事実上消えてしまうほどじゅうぶん長い時間の実行後の状態である。実行の開始時とモデルが「実質的に」定常状態に至る時点との間の時間を**初期過渡期間**と呼び、それについて議論する。

表 4.2 で結果を得るために利用した基本的な待ち行列解析（2 章参照）は、**定常状態**でシステムの正確な期待値の結果を与えている。これまで議論したように、事実上、定常状態に至る前に、待ち行列ネットワークにかかるほとんどのシミュレーションモデルは、初期過渡期間を経験している。初期過渡期間にモデル統計を記録し、反復要約統計表でこれらの観測を用いて、**初期のバイアス**となることがある。すなわち、 $E(\hat{L})$ は L と等しくないかもしれない。例として、モデルに 2、5、10、20、および 30 時間として実行期間を設定した 5 つの実験において、それぞれ 500 回の反復実行をする。 L （もちろん 95% の信頼区間と共に）の推定値の結果は以下の通りであった。

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 時間の実行に関しては、推定値が定常状態（定常状態値は $L=4.000$ である）に対してまだ下向きに偏っていることが、明確にわかる。30 時間は信頼区間が 4.000 を含んでいるが、平均値が少し低いのでまだ確信できない。より多くの反復実行を行うことで、信頼区間の幅がおそらく減少するだろう。そして、それらの反復実行により 4.000 が信頼区間の外側になり、30 時間の実行にはまだ偏りがあると結論を下せるかもしれないが、現時点では明確ではない。また、追加的な反復実行が、開始時の偏りが大きな影響を与える（これは統計的サンプリングの性質であるが）ということを証明しないこともありうる。幸運にも、多くの科学的あるいは社会学的な実験とは異なり、反復と実行期間の全体を管理し、満足するまで実験することができる（または、コンピュータ時間もしくは人間の忍耐のどちらかを使い果たすまで実験することができる）。先に進む前に、開始時の偏りは「複製」できないことを指摘しなければならない。過渡期間はシステムの特性であり、偶発性による人工物やサンプリング誤差の結果ではない。

各実行の中で完全な演算を通して、開始時の偏りを流すことができるくらい長い間モデルを実行することの代わりに、ウォームアップ期間を用いることができる。ここでは、モデル実行期間が分

割され、統計は初期（ウォームアップ）期間には集められないが、単に「見る」だけではなく、この期間もモデルは普通に実行している。ウォームアップ期間の後に、統計はいつものように収集される。統計の記録を開始させるとき、モデルの状態は定常状態に近くなる。それで、この単純モデルでは、最初のエンティティがウォームアップ期間後に到着するとき、待ち行列におけるエンティティの期待数は 3.2（定常状態における $L_q = 3.2$ ）であるだろう。例として、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つの追加実験を行った（Simio では、 n が希望するウォームアップ期間の長さであるときは、Experiment の Warm-up Period プロパティに設定する）。 $L = 4.000$ であると推定しているとき、結果は以下の通りである。

(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が、すべての場合における開始時の偏りを減少させ、または排除するのを「助け」、30 時間を超えて総合的な実行時間を増加させていないようである。つまり、ウォームアップ期間を用いることによってコンピュータの要件を増加させずに、推定値を改良させた。ここで生じる質問は「ウォームアップ期間はどれくらい長くすべきか？」ということである。一般に、モデルがどれくらいで定常状態に至るかを決定するのは、簡単ではない。1つのヒューリスティックであり直接的なアプローチは、Simio モデルの Facility ウィンドウ（モデルの Facility ウィンドウであり、Facility Tools 下の Animation リボンを選択する。アニメーションに関する詳細な情報については 8 章を参照）にダイナミックに動く Status Plots を挿入し、系統的に傾くのが止まるように見えたときに判断を下すことである。しかしながら、アニメーションの間は、一度に 1 つだけの反復を表現するので、これらはかなりノイズの多い（すなわち、可変な）ものである。ウォームアップ期間を指定することに関して、以下のことが推測される：

- ・ ウォームアップ期間が短過ぎると、結果として、まだ開始時の偏りがある（これは潜在的によくない）。
- ・ ウォームアップ期間が長過ぎると、サンプリング誤差は必要以上に高くなる（ウォームアップ期間の長さを増加するに従って、実際に記録するデータ量は減少する）。

その結果として、「もっとも安全な」アプローチは、合格水準のサンプリング誤差（信頼区間の半幅で測定される）を達成するために全体のウォームアップ期間を長くし、実行時間を伸ばし、反復回数を増加させることである。このアプローチを用いると、相対的に必要となるより少しコンピュータ時間を費やすかもしれないが、最近のコンピュータ時間は早い（実際にはそれを測定できないので、偏りは知らない間に危険となる）。

もちろん、前段落におけるウォームアップの議論では、定常状態値を実際に必要としていると仮定している。しかし、そうではないケースも多いだろう。代わりに、過渡期間中の「短い実行期間」のシステム挙動に興味を持つことは、確かに可能であり一般的である。たとえば、スポーツ競技において当日券販売が（空かつ遊休のシステムで）開き、ある予定された時間に止まるので、「定常状態」がない。これらのケースは、多くの場合、終結シミュレーションと呼ばれる。実験では単にウォームアップ期間を無視し（すなわち、デフォルトを 0 とする）、そして、シミュレーションがもたらすものは、関心のある期間のシステムの挙動についての偏りのない視点、そしてモデルの初期状態に関するものである。

通常、定常状態の目標か終結状態の目標のどちらが適切であるかという選択は、モデル構造が何であるかより、研究の意図が何であるかに関する問題である。もっとも、終結シミュレーションは設定や実行、そして分析がしやすいといえるだろう。それは、各反復に関する開始と停止の規則が

ただアナリストの判断ではなく、モデルの一部であるからである。唯一の実際における問題は、結果において許容できる統計的な精度を達成するために何回の反復が必要になるのかである。

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、反復問題および可能なウォームアップ期間についてはすでに記述したので、アップデートしたシミュレーションモデルの結果（30 時間の実行期間と 20 時間のウォームアップ期間としたモデルの 500 回の反復）とオリジナルの待ち行列解析結果の比較を再考する。表 4.5 は両方の結果を示している。表 4.2 に示されていた結果と比べて、モデルは「正しい」とさらに確信できる。換言すると、モデルが検証されたというかなり有力な証拠がある（すなわち、期待するようにそれは反応する）。モデルは、実証可能な方法での検証が可能ではないことに注意してほしい。代わりに、誤りを見つけるか、またはモデルが正しいと確信するまで、証拠を集めることができるだけである。

表 4.5 待ち行列解析と最終実験の比較

推定された測定基準	待ち行列	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

行ってきた過程を要約すると、以下の通りである：

1. モデル結果（待ち行列解析）に関して一連の期待値を得た。
2. モデルを構築して実行し、モデル結果と期待値（表 4.2）を比較した。
3. 結果が合致しなかったので、3 つの考えられる解釈を考えた：
 - a) Simio モデルが間違っている（すなわち、モデル自体のどこかに誤りを持っている）。これは考察を省略した。
 - b) 期待値が間違っている（すなわち、シミュレーションの結果が待ち行列結果に合うべきであるという仮定が間違っている）。開始時の偏りを排除し、定常状態に近づけるためにモデルのウォームアップが必要であったのがわかった（すなわち、過渡期間を含む分析が、定常状態の結果に合うべきであるという予測は間違っていた）。ウォームアップ期間を加えることで修正された。
 - c) サンプリング誤差（すなわち、シミュレーションモデル結果が確率的な意味における期待値に合っているが、じゅうぶん長い間モデルを実行していないか、または不当に結果を解釈している）。適切にモデル出力における偶発性を説明するためにモデルを反復実行し、実行期間を増加させることが必要であったことがわかった。
4. 最終的に、モデルが正しいと判断した。

すべてのシミュレーションプロジェクトに関して、この基本的な検証過程に従うことはよいことである。一般に、私たちが求めている厳密な結果を計算できないが（そうでなければ、シミュレーションがなぜ必要であるだろうか？）、たとえモデル化されるシステムの抽象的な見解に基づいていても、いつもいくつかの期待値を得られる。そして、非常に自信があるモデル（そして、期待値のセット）に収束するように、これまでに概説されたこれらの期待値と過程を用いることができる。Simio でモデル検証と実験の基本内容をカバーしたので、本章の残りでは、いくつか Simio モデル化概念について議論する。しかしながら、最終的には本書全体でこれらの基本的問題を再考する。

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 ライブラリからなど)で正確にモデル化することは、非常に直感的であり、多くの人々にとって(たいていは)簡単で、素早くモデル化することが可能である。しかし、低レベルの Simio Processes を用いたくなる状況がよくある。それは、モデルを構成するためや、既存の Simio オブジェクトを増強するため、オブジェクトで適応できない詳細で専門的なモデリングをするため、あるいは実行時間に問題があり実行スピードを改善するためである。Simio Processes を用いるには、Simio のかなり詳細な理解と一般的な離散事象シミュレーションの理解を必要とする。本節は、例題の単一サーバ待ち行列システムについて、非常に簡単ではあるが、基本的な Simio Process モデルのデモをするのみとする。次章で、モデリングの状況によって求められる Simio Processes に関して、詳細について説明する。

エンティティがサービス(簡単な待ち行列システムのサーバなど)に関して競合する有限容量リソースを含んでいるシステムをモデル化するために、Simio は **Seize-Delay-Release** モデルを用いる。これは標準の離散事象シミュレーションのアプローチである。そして、他の多くのシミュレーションツールでも、同一もしくは同様のモデルを用いる。この基本型について完全に理解をすることは、Simio Processes を有効に用いるために不可欠である。モデルは以下の通りに動作する：

- ・ 容量 c として、リソースを定義する。これは、シミュレーション時刻の任意な時点で、同時に 1 つ以上のエンティティを割り当てることができるリソースの任意のユニットの容量が 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任意のユニット数 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 ユニットの容量があるなら、すぐに s ユニット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 s ユニットがエンティティに割り当てたままで残っている間、サービスタイムを表す時間進行(遅延)を始める。さもなければ、エンティティは自動的に必要な容量が有効になるまで待っている待ち行列に置かれる。
- ・ エンティティのサービス時間の進行が完了したときに、エンティティはリソースの容量の s ユニットを解放して、プロセスにおける次のステップに進む。リソース待ち行列で待つエンティティがあり、リソースの利用可能な容量(直前に離脱したエンティティによって解放されたユニットを含む)がじゅうぶんであれば、待ち行列から最初のエンティティを取り除き、すぐに必要なユニットの容量をそのエンティティに割り当てる。そして、そのエンティティは時間進行を始める。

モデリングの視点からは、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はエンティティのリソース容量の割当てとエンティティ待ち行列を管理している。さらに、モデルを実行すると、Simio を含むほとんどのシミュレーションソフトウェアは、自動的に待ち行列、リソース、およびエンティティに関する統計を記録する。基本的な Seize-Delay-Release プロセスを図 4.13 に示す。この図において、「到着時間間隔」は連続するエンティティ間の時間間隔であり、「処理時間」は処理のためにエンティティが時間進行(遅延)する時間である。システム内数はシミュレーション時間の任意の時点におけるシステム内のエンティティ数を追跡する。そして、到着時刻を記憶・記録することによって、すべてのエンティティがシステム内で費やした時間をシステム内時間として追跡する。

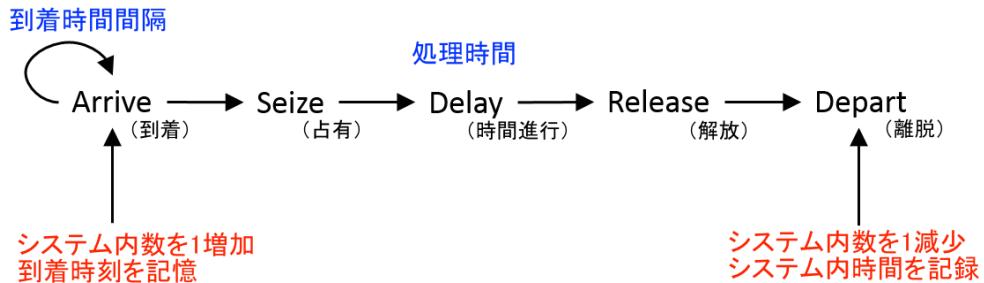


図 4.13 Seize-Delay-Release モデルの基本プロセス

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ついて) 単に $c = 1$ および $s = 1$ と設定する。図 4.13 に示すように、単一サーバモデルは単なる基本的な Seize-Delay-Release ロジックの実装である。Processes を用いてこのモデルを構築することは、Standard ライブラリオブジェクトを用いて構築するよりほんの少し複雑であるが、モデル開発に直面し、ユーザ定義の統計を集めるためにそのメカニズムを理解することは、有益である。Simio のこのモデルを実行するためのステップは以下の通りである(図 4.14 を参照)：

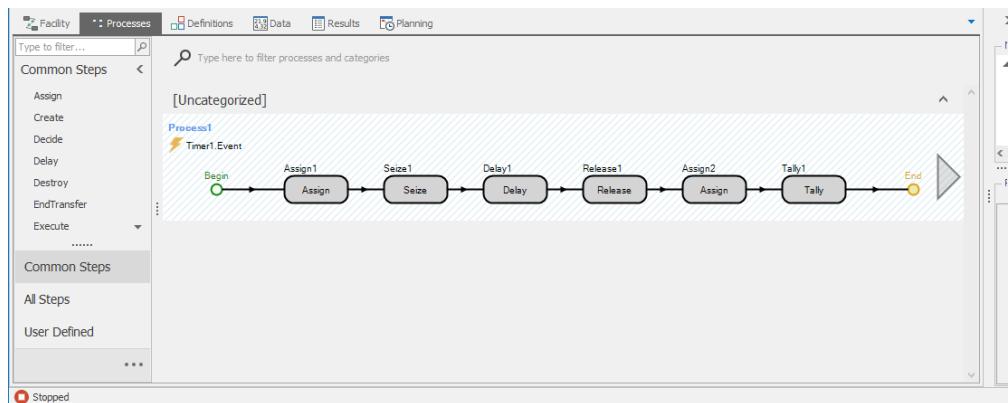


図 4.14 Model 4-2 の Process ビュー

1. Simio を開き、新しいモデルを構築する。
2. Standard Library から Facility ウィンドウに Resource オブジェクトをドラッグすることによって、Resource オブジェクトを作成する。オブジェクトのプロパティの Process Logic セクションで、Initial Capacity Type が Fixed であり、Capacity が 1 であることを確かめる(これらはデフォルトである)。General セクションのオブジェクト Name(デフォルトは Resource1)に注意する。
3. Model が Navigation パネルでハイライトされていることを確認し、Definitions タブをクリックすることで Definitions ウィンドウに切り替え、そして左のパネルアイコンをクリックして States セクションを選ぶ。これは、状態変数をモデルに追加するための準備である。
4. States リボンの Discrete セクションで Integer アイコンをクリックすることによって、新しい離散的な状態変数(States)を作成する。IntegerState1 となっているデフォルト Name プロパティを WIP へ変更する。離散的な状態変数は、数値を記録するのに用いられる。この場合、モデルに Integer Discrete State を作成することによって、モデルのエンティティの現在数を蓄積する場所を作成している(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、Elements セクションに切り替え、そして、Elements リボンで 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 を参照)。Timer エレメントは、エンティティ到着のトリガーとして用いられる(ループバックについて図 4.13 を参照)。 $\lambda = 48$ エンティティ/時間の割合でのポアソン到着、あるいは同等の平均値

$1/0.8=1.25$ 分の指数到着時間間隔にするために、Time Interval プロパティを Random.Exponential(1.25)に設定し、Units が Minutes に設定されていることを確認する。

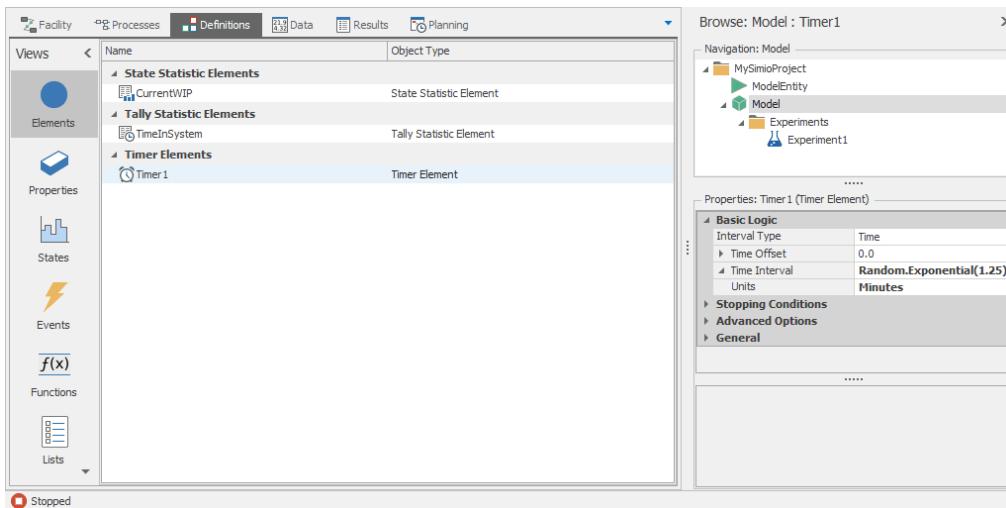


図 4.15 Model 4-2 の Timer エレメント

6. Elements リボンの Statistics セクションで StateStatistic アイコンをクリックすることによって、StateVariable を作成する。State Variable Name プロパティを WIP に設定する（以前にモデルで Discrete State を定義したので、プルダウンに現れる）。そして、Name プロパティを CurrentWIP に設定する。時間の経過で状態の値を追跡して、時間依存する統計値をこの値に記録するように Simio に設定している。
7. Elements リボンの Statistics セクションで Tally Statistic アイコンをクリックすることによって、TallyStatistic を作成する。Name プロパティを TimeInSystem に設定する。そして、Unit Type プロパティを Time に設定する。Tally Statistics は、観測統計量（すなわち、離散時間型の統計量）を記録するのに用いられる。
8. Processes タブをクリックして、Process ウィンドウに切り替える。そして、Process リボンで Create Process アイコンをクリックして、新しい Process を作成する。
9. Triggering Event プロパティに新たに作成されたタイマーイベントを設定する（図 4.16 を参照）。これは、タイマーが切れるたびに、そのプロセスを実行するように設定している。
10. Common Steps パネルから Assign ステップをドラッグして、プロセスの Begin インジケータの右に Assign ステップを配置する。State Variable Name プロパティを WIP、New Value プロパティを WIP+1 と設定する。これは、イベントが起こるときに、エンティティがシステムに到着したという事実を反映するために状態変数の値を増加したいためである（図 4.13 の「1 増加」を参照）。
11. 次に、プロセスの Assign ステップのちょうど右に Seize ステップを追加する。Resource1 は到着エンティティによって占有されるべきであることを示すために、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 ボタンをクリックする。そのとき、特定のオブジェクト Resource1 が占有されるように設定する（図 4.17 を参照）。

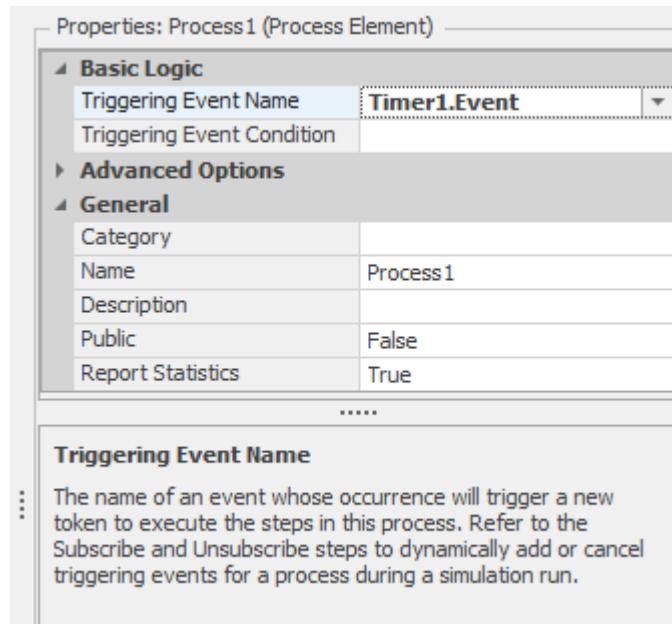


図 4.16 プロセスにおけるトリガーアイベントの設定

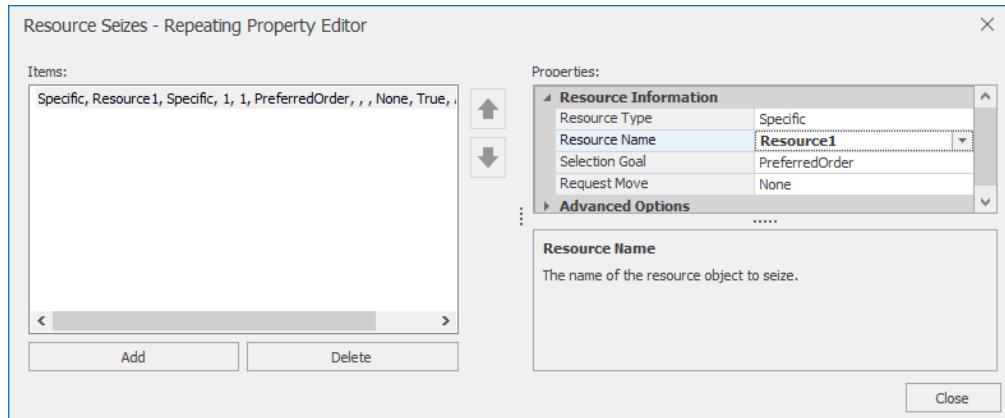


図 4.17 Resource1 が占有される Seize プロパティの設定

- 12.Seize ステップのすぐ後ろに Delay ステップを追加し、そして、Delay Time プロパティに Random.Exponential(1)分と設定する。これは、エンティティの時間進行が平均 1 分（元のサービス率の 60/時間と同等）の指数分布に従うことを示している。
- 13.Delay ステップの直後に Release ステップを追加する。そして、Releases プロパティを Resource1 に設定する。
- 14.もう一つ Assign ステップを Release ステップの次に加える。そして、State Variable Name プロパティへ WIP、New Value プロパティを WIP-1 と設定する。これは、エンティティがリソースを解放するとき、システムを離脱した事実を反映するために、状態変数の値を減少させたいことを示している。
- 15.Tally ステップを加えて、TallyStatisticName プロパティへ TimeInSystem(Tally Statistic はすでに作成したので、プルダウンで利用可能である)、Value プロパティには TimeNow-Token.TimeCreated と設定する。これは、記録される値が、現在のシミュレーション時刻から、現在のトークンが作成された時刻を減算した値であることを示している。この時間間隔は現在のエンティティがシステムに費やされた時間を表す。Tally ステップは、図 4.13 の「記録」機能を実装したものである。特にここでは、図 4.13 における到着時刻の「記憶」について、トークン状態変数の Token.TimeCreated を使っていることに留意してほしい。

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（たとえば、Ending Type を 1,000 時間の固定実行期間に設定する）。

States(状態変数)、Properties(プロパティ)、Tokens(トークン)、および他の Simio Framework の細部については 5 章で議論する。

モデルをテストするために、Project Home リボンの New Experiment アイコンをクリックし、Experiment を作成する。図 4.18 は、各反復で 500 時間のウォームアップ期間を用いたモデルを 10 回反復実行した Pivot Grid の結果を示している。レポートが CurrentWIP と TimeInSystem 統計を含む UserSpecified カテゴリを含んでいることに気づくだろう。4.2 節の Standard ライブラリのオブジェクトモデルで自動的に収集している ModelEntity 統計の NumberInSystem や TimeInSystem と異なり、プロセスモデルでこれらの統計を集めるように設定したためである。モデルがより大きくより複雑になるにつれ、おそらくデフォルトの統計量以上のものを求めるようになるので、ユーザによって指定された統計を理解することは重要である。CurrentWIP 統計量は時間従属の統計量の例である。ここで、Simio の状態変数（ステップ 4）を定義して、必要なときに（増加させるステップ 10 と減少させるステップ 14）状態変数の値をアップデートするのにプロセスロジックを用いた。そして、シミュレーション時間が進行するとき、値の変化を追跡して、サマリ統計量 (\hat{I} 、ステップ 4) を報告するように設定している。TimeInSystem 統計量は観測またはタリー統計量の例である。この場合、それぞれの到着エンティティは、ただ 1 つの観測（エンティティがシステムに費やされる時間）に寄与しており、Simio はこれらの値（この場合、 \hat{W} ）のサマリ統計量を追跡し、報告している。ステップ 7 でこの統計値を設定し、そしてステップ 15 では各観測を記録している。

Drop Filter Fields Here						
					Scenario ▲	
					Scenario1	
Object Type ▲	Object Name ▲	Data Source ▲	Category ▲	Data Item ▲	Statistic ▲	
Model	Model	CurrentWIP	UserSpecified	StateValue	Average	3.9636 3.4841 4.2981 0.1913
				FinalValue	Average	0.9000 0.0000 3.0000 0.7872
				Maximum	Average	32.8000 24.0000 44.0000 3.9439
				TallyValue	Average (Hour)	0.0825 0.0730 0.0893 0.0037
Resource	Resource1	[Resource]	Capacity	ScheduledUtilization	Average	0.6255 0.4759 0.7466 0.0780
				UnitsAllocated	Average	1.0000 1.0000 1.0000 0.0000
				UnitsScheduled	Average	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 (Hour)	0.0828 0.0777 0.0884 0.0023
					Occurrences	4,828.0000 77.0000 167.0000 107.3130
					Percent	79.8263 78.7424 80.9127 0.4707
					Total (Hours)	399.1316 93.7118 104.5634 2.3535
				TimeIdle	Average (Hour)	0.0209 0.0207 0.0211 0.0001
					Occurrences	4,827.9000 76.0000 167.0000 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

プロセスモデルに関して注意する別の問題は、対応する Standard ライブラリオブジェクトのモデルよりも速く実行できるということである。その速度差は、Standard ライブラリオブジェクトには、提供される追加的な機能に関連するすべて（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）を含めているためである。

上述したように、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難しい。しかしながら、プロセスは Simio の基本的なものであり、それらがどのように働いているかを理解することは重要である。さらに詳細に 5.1.4 項でこのことについて再び取り上げる。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。

4.4 Model 4-3 : 現金自動振込機(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任意の待ち行列システムに焦点を合わせた。本節と 4.8 節の焦点は、シミュレーション分析に用いられる「本当の」システムのモデルをより細々と表現するように、いくつかの状況をモデルに追加することである。以降の章では、シミュレーションモデリングに関する一般的な概念について説明し、Simio の特徴について研究しながら、モデルをさらに充実させていく。Model 4-1 と 4-2において、エンティティ到着時間間隔とサービス時間は、指数分布からの観測値を用いた。シミュレーションにおけるランダム性の基本を説明するために、 $M/M/1$ 待ち行列モデルの結果における数学的な「よさ」を利用できるようにした。しかしながら、多くのモデリングの状況で、エンティティ到着時間間隔とサービス時間はあまり指数分布に従わない。Simio と他のシミュレーションパッケージでは、一般的なモデリングを支援するためにさまざまな分布から標本抽出ができる。Model 4-3 と 4-4 では、サービス時間において三角分布の使用を示す。そして、5 章のモデルでは、他のいくつかの標準的な分布の使用を示す。6.1 節では、シミュレーションモデルがモデリングの現実性を正しく表現できることを議論する。

Model 4-3 は図 4.19 に示すような現金自動振込機 (ATM) をモデル化する。顧客は、入口のドアを通って入り、ATM に歩いて行き、ATM を利用して、出口のドアまで歩き、離れる。このモデルにおいて、ATM が置かれた部屋は、ATM の利用を待つために何人の顧客が入れるくらいじゅうぶんに大きいと仮定する (これによって、モデルは少し簡単になる。ただし、後の章で収容数に限界のある待ち行列の利用について再考する)。この仮定によって、基本的には、図 4.1 に示されていたものと同様の単一サーバ待ち行列モデルとなる。

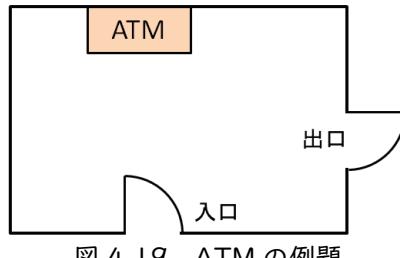


図 4.19 ATM の例題

そして、Model 4-1 から開始して、ATM モデルとなるようにモデルを変更する (Model 4-1 のファイルを上書きしないように、Save Project As オプションを必ず用いて Model 4-3 を保存する)。完成した ATM モデル (Model 4-3) を図 4.20 に示す。

必要な変更は以下の通りである：

1. オブジェクト名を書き換えて、新しいモデルの背景を反映する (エンティティは ATMCustomer、Source オブジェクトは Entrance、Server オブジェクトは ATM1、および Sink オブジェクトは Exit)。
2. 図 4.20 のように、モデルを再配置する。
3. モデルに顧客移動時間を含めるために、Connector とエンティティオブジェクトを変更する。
4. パラメータ(0.25, 1.00, 1.75)分 (すなわち、1.00 分が最頻値であり、0.25 分～1.75 分の間) の三角分布に従うような ATM 取引時間として、ATM 処理時間分布を変更する。

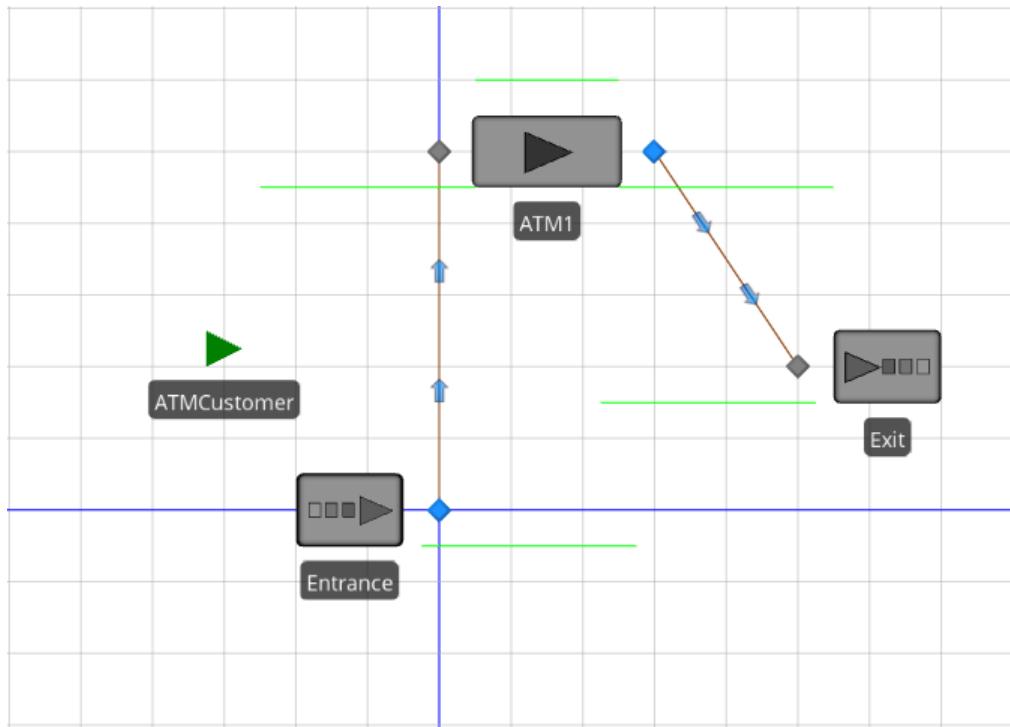


図 4.20 Model 4-3 : ATM の例題

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と、モデルの可読性を（特に大きくて複雑なモデルでは）大いに改良できる。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とよい。オブジェクトを選択し、F2 キーを押して新しい名前をタイプするか、またはオブジェクトの Name プロパティを編集するかによって、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、非常に簡単である。Simio では、モデルの周辺で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カノードも動かすことができることに注意する。

初期の待ち行列モデル (Model 4-1) では、エンティティは到着のときにサーバに単に「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、ATM から出口まで歩くので、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)**：ゼロシミュレーション時間で、オブジェクト間のエンティティを移動させる（すなわち、即座に無限速度で移動する）。
- **Path (パス)**：移動時間を決定するのにオブジェクト間の距離とエンティティ速度を利用して、オブジェクト間でエンティティを移動させる。
- **TimePath (タイムパス)**：ユーザによって指定された移動時間の式を用いて、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)**：物理的なコンベアをモデル化する。

次章からのいくつかの章で、これらの手法がそれぞれ用いられる。ここでは、ATM モデルに Path を用いる (Simio レファレンスガイド (F1 キーか Simio ウィンドウ右上にある?アイコンをクリック) では、これらのオブジェクトのすべての詳説を提供している)。このモデルは、Model 4-1 を変更しているので、オブジェクトは既に Connectors を用いて接続されている。Connector を Path

に変えるもっとも簡単な方法は、Connector を右クリックして、Convert to Type サブメニューから Path オプションを選ぶことである。接続の種類を変えるにはこれだけでよい。あるいは、Connector オブジェクトを削除して、Standard ライブラリで Path をクリックし、開始と終結ノードを選択し、手動で Path オブジェクトを加えることもできる。

パスの長さとエンティティ速度に従って、Path オブジェクトに沿ったエンティティの移動時間が決定される。Simio モデルは、デフォルトで比例するように描かれるので、2つのノード間のパスをえたとき、パスの長さは 2つのノード間の距離として設定される（長さやその他の単位を持つプロパティを入力する場合、プロパティフィールドに表示されている+を押してフィールドを拡張し、入力単位を指定することができる。また、Run リボンの Unit Settings ボタンにより、Facility ウィンドウのラベルや Pivot Grid の数値、および出力の追跡など、出力として表示される単位を変更することができる）。General セクションの Physical Characteristics/Size グループに属する Length プロパティが、Path オブジェクトの現在の長さを与えている。また、グリッドを用いて経路の長さを推定することができる。また、実寸図示でパスを描くことが不都合であれば、手動で論理パスの長さを設定できる。手動で論理的な長さを設定するためには、Drawn To Scale プロパティに False を設定する。そして、希望の長さを Logical Length プロパティに設定する。エンティティ速度は、エンティティのプロパティの Travel Logic セクションの Initial Desired Speed プロパティで設定される。Model 4-3 では、入口から ATM までのパスの長さは 10 meters であり、ATM から出口までのパスの長さは 7 meters である。そして、エンティティ速度は 1 meter/second である。これらの値により、エンティティは入口から ATM まで移動するために 10 秒、ATM から出口まで移動するために 7 秒のシミュレーション時間を必要とする。モデル化されるシステムによって書き換えられるように、容易にパスの長さとエンティティ速度を変更できる。

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。指数分布の特性から、おそらく ATM の取引つまり処理時間をモデル化するには不適当である。特に、指数分布は多くの比較的小さい値と少しの非常に大きい値で特徴づけられ、密度関数の最頻値がゼロである。すべての顧客が銀行キャッシュカードを挿入し、正しく個人識別番号（暗証番号）を入力して、取引を選択しなければならないが、ATM 取引の種類数は通常制限されていることを考えると、境界がある分布のほうがよりよい選択となるだろう。そこで、パラメータは三角分布 (0.25, 1.00, 1.75) 分を用いることにする。適切な分布の利用を決定することは、**入力分析**の一部であり、6.1 節で詳しく解説することにする。当面は、与えられた分布が適切であるとする。処理時間の分布を変更するために、Processing Time プロパティを Random.Triangular(0.25, 1, 1.75) に変更し、Units プロパティは Minutes のままにする。Random キーワードを用いることによって、（この本で取り上げている）約 19 の一般的な分布から統計的に独立な観測値を抽出できる。この分布のなかには、標準的な確率分布に適切に適合しない場合に用いられる、連続および離散型の経験分布も含まれる。これらの分布、必須のパラメータ、および密度関数または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 (数式エディタ、関数、および分布)」の「Distributions (分布)」セクションで詳細に議論している。6.3 節および 6.4 節で、Simio の乱数発生と確率変数について議論することにする。

Model 4-3 が完成したので、4.2.5 項で議論したように、モデルについて検証しなければならない。上で述べたように、検証の過程は、一連の期待値を得ること、モデルを構築して実行すること、そして、モデルの結果が期待値と合っているかを確認することが含まれる。期待値とモデル結果の間にミスマッチがあるとき、モデル、期待値、または両方に関する問題を見つけて、修正しなければならない。Model 4-1 に関しては、期待値を得る過程はかなり簡単であった。パフォーマンス測定基準の厳密値を計算できるように、M/M/1待ち行列システムをモデル化していた。Model 4-3 の過程はそれほど簡単でない。処理時間は指数関数ではなく、また到着とサービスの間とサービスと出発の間でのエンティティ移動時間を加えている。そのうえ、これらの 2つの変更が、待ち行列

測定基準において互いを打ち消す傾向がある。特に、処理時間の変動を減少させたので、システム内時間と同様にシステム内の待ち行列におけるエンティティ数が（Model 4-1 に比べて）減少することが予想される。しかし、エンティティ移動時間を加えたので、システム内のエンティティ数とエンティティがシステムで費やす時間が増加することも予測される。そして、検証可能な一連の期待値がない。より複雑なモデルを構築すると、これはかなりよくあることであるだろう。けれども、モデル検証においては必要なものである。

1つの戦略としては、容易に一連の期待値を得られる修正モデルを構築し、検証にはその修正モデルを用いることである。Model 4-3 では、このアプローチをとる。モデルを修正するための 2 つのもっともな選択がある：Entity Transfer Times を 0 に設定し、 $M/G/1$ の待ち行列近似を用いるか（2 章で説明）、または処理時間を指數分布に変更する。ここでは、後者のオプションを選択し、ATM の Processing Time プロパティを Random.Exponential(1) に変えた。各エンティティに、移動時間の 17 秒を加えただけなので、 ρ 、 L_q および W_q が $M/M/1$ の値（表 4.2）と合致し、対応する $M/M/1$ の値より W が 17 秒大きくなることが予想される。表 4.6 は、500 回の反復実行で、30 時間の反復時間および 20 時間のウォームアップ（4.2.5 項で用いた同じ条件）で実行した結果を示す。

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に合っているように見える（もし平均値と予測の間の小さい偏りに関心があるなら、より長い間各反復を実行するか、または追加反復を実行する）。したがって、修正したモデルが適切に検証されたとすれば、Model 4-3 も同様に検証されたとはいえない場合は、Processing Time プロパティを間違って入力しているか、あるいは Simio の実行時に Triangular 乱数ジェネレータが妥当な値を生成していないかである。

ここで、正しくプロパティに入力されていることを確認し、そして、Simio の乱数および確率変量のジェネレータが正しく働くと仮定しよう。表 4.7 は Model 4-3（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）の実験の結果である。予想されたように、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ので、想定内である）。4.2.5 項で指摘したポイントを繰り返すことには価値がある：モデルが検証されることが、（一般的な）実証ではない。代わりに、確信するまで（おそらくプロセスにおける誤りを見つけて、修正することで）証拠を集めることができるだけである。

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からの結果はランダムであるため、適切な統計的手法で分析される必要があると、すでに何か所かで強調してきた。今までのところ、平均値に焦点を合わせる傾向があった。未知の母平均（または、確率変数の期待値と興味がある出力応答の分布）を推定するためにシミュレーションからの平均を用いている。そのために、おそらくもっとも役に立つ方法は信頼区間を利用することであり、Simio では Experiment Pivot Grid と Reports でそれらをどのように提供しているかを示してきた。（シミュレーション出力データだけではなく、何でも）平均値は重要であるが、それらが定義上、関心のある確率変数における無限数の反復実行の平均であるので、全体像（さらにいえば、両裾）を伝えてはいない。したがって、平均システム内時間、最大の待ち行列の長さ、またはリソース利用のようなシミュレーション出力応答について、データの広がりや、またはありそうな値は何か、ありそうもない値は何か、何も伝えられないようなものである。このことは、Sam Savage の著書 *The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty* の中で、指摘されたポイントである (Savage 2009)。3.2.3 項のたった 1 期間の静的な在庫シミュレーションにおいて、特に図 3.3 において、結果のヒストグラムについて議論した。さらに、平均に加えて、利益に関して上限と下限のリスクがどれだけあるのか、特に利益よりむしろ損失があるかもしれないというリスクを確認した。たとえば、5,000 個の帽子を注文するなら、**平均利益が上向き** (95%信頼区間で 8,734.20 ドル±3,442.36 ドル) であるように見えたが、損をするというリスク（損失）も 30% あった。しかし、平均ではこれらのどちらも記述することはできなかった。

そこで、Experiment Pivot Grid と Report に加えて、Simio は出力統計を報告するための新しいタイプの図が含まれている。それでは、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、1977 年に最初に John Tukey によって説明され、改良された箱のプロットの組み合わせであり、ヒストグラム、および個々の反復サマリ応答の簡単な点プロットが含まれている (Tukey 1977)。SMORE プロットは Barry Nelson (2008) によって開発された、**Measure of Risk and Error (MORE) プロット**に基づいており、図 4.21 はそれらのいくつかの要素を定義している図を示している。SMORE プロットは、出力パフォーマンス尺度（応答）サマリのための実行結果のグラフ表示である。ここで応答とは、複数の反復実行後の、平均システム内時間、待ち行列の最大数、またはリソース稼働率などである。デフォルト設定における箱のプロットに加えて、最小と最大の観測値、標本平均、標本中央、そして「下限」と「上限」のパーセント値を表示している。ここで「標本」は反復からの観測値ではなく、反復を通したサマリ測定値で構成される。したがって、これは複数回反復実行される終結シミュレーション、または適切なウォームアップ期間が指定され、事実上、モデルがウォームアップを持って反復される定常状態シミュレーションを、主に意図している。SMORE プロットは任意に平均値の信頼区間、下限と上側限両方のパーセンタイル値、観測値のヒストグラム、およびそれらの個々の反復からの応答を表示できる。

SMORE プロットは実験の Responses に基づいて自動的に生成される。おそらく、Model 4-3 に関しては、反復において顧客のシステムに費やす平均時間（顧客が ATM に到着し、その顧客が ATM を離れるまでの時間間隔）に关心があるだろう。Simio は自動的にこの統計値を追跡する。そして、式 `ATMCustomer.Population.TimeInSystem.Average` として反復中の平均値にアクセスできる。Response としてこれを加えるために、(Experiment ウィンドウの) Design リボンの Add Response アイコンをクリックし、出力のラベルに Name (`AvgTimeInSystem`) を指定し、Expression プロパティを指定する (`ATMCustomer.Population.TimeInSystem.Average`) (図 4.22 を参照)。ここでよくある質問は「Expression に `ATMCustomer.Population.TimeInSystem.Average` とタイプすべきであることをどのように知るのか？」ということである。よい質問であるが、Simio は多くの場面で、コンテキストに応じたヘルプを提供している。

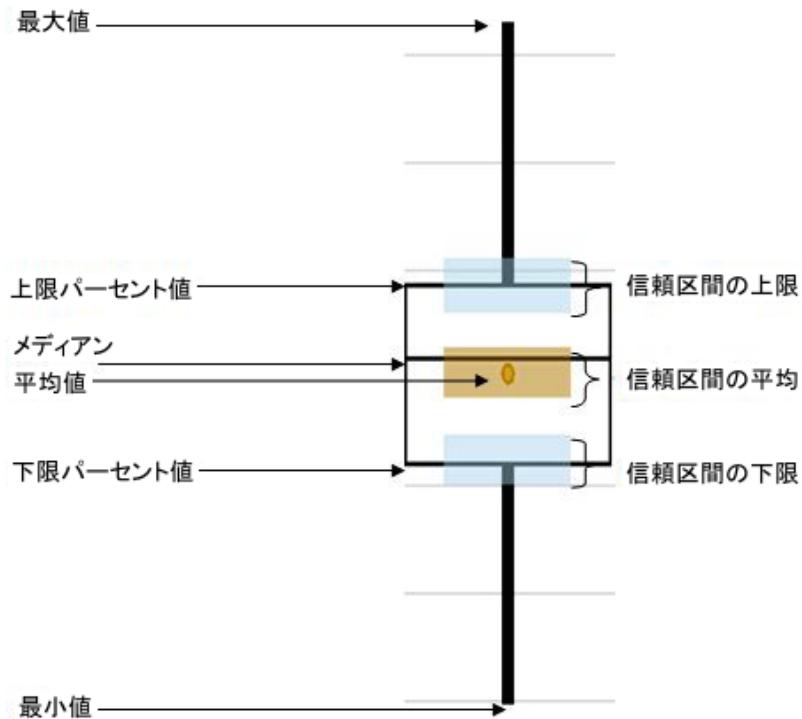


図 4.21 SMORE プロットの構成 (Simio レファレンスガイドより引用)

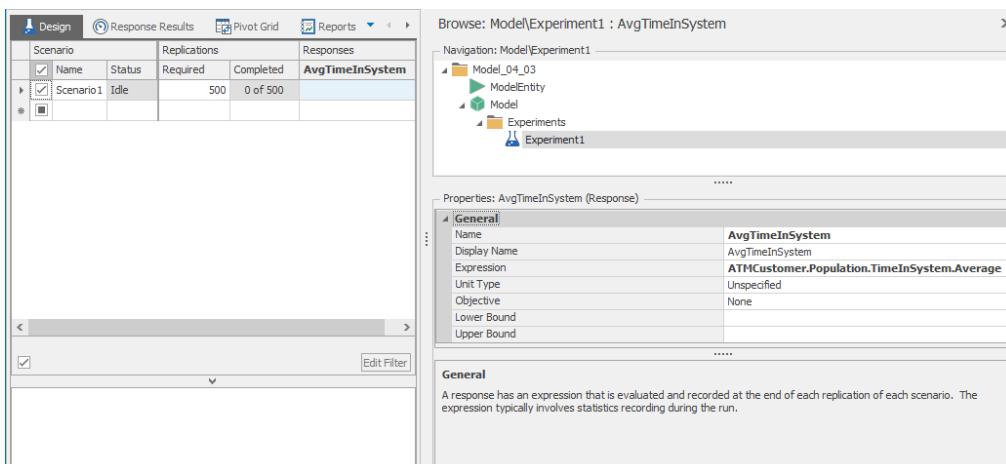


図 4.22 Model 4-3 の平均システム内時間のための実験 Response 定義

Expression フィールドをクリックすると、右に下向きの矢が現れる。それをクリックすると、右側に赤色の×と緑色のチェックマークがある別のフィールドが現れる。これは、Simio の式ビルダであり、5.1.7 項でより詳しく説明する (図 4.23 参照)。

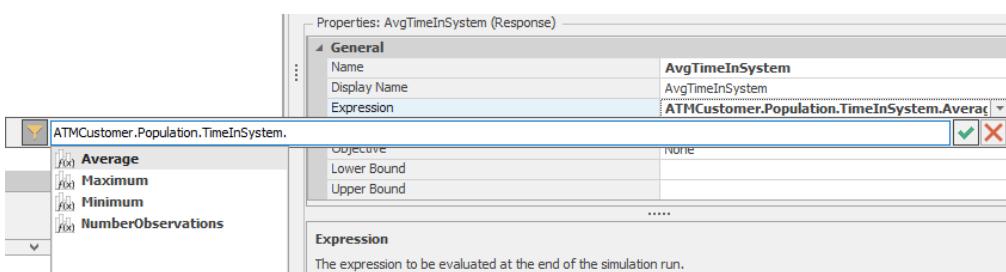


図 4.23 SMORE プロットにおける平均システム内時間 Response に対する Simio 式ビルダの利用

ここで式ビルダを試してみると、空の Expression フィールドをクリックし、次に式の左端でキーボードの下矢印キーを押す。下に現れるリストで、ATMCustomer（エンティティの名前であるので、このエンティティに関する情報、システム内時間がほしい）を見つけ、ダブルクリックする。その名称が Expression フィールドにコピーされることに留意する。次に、Expression フィールドで ATMCustomer のすぐ右にピリオドを入力すると、その名称に続く可能性があるリストがドロップダウンされる。ここでは、特定のエンティティだけではなく、ATMCustomer エンティティ全体の母集団の統計値を探しているので、Population をダブルクリックする。再度、選択のリストが提供される。（ATMCustomer エンティティに関して知りたい事柄として）リストの下部にある TimeInSystem をダブルクリックする。ドロップダウンリストを見失ったら、もう一度下矢印キーを押せばよい。これまでと同様、そのフィールドで徐々に構築している式の右にピリオドをタイプする。そして、リストに現れる Average でダブルクリックする（最大か最小ではなく、ATMCustomer エンティティの平均システム内時間が知りたい。もし平均ではなく、最大や最小に関して知りたいなら、最大や最小を選択する）。これで終わりであるが、別のピリオドを入れ確かめると何も起こらないので、式として確立するために緑色のチェックマークをクリックする。

このように、繰り返し Add Response アイコンをクリックして、上述のように Properties に入力し、より多くの Responses を加えることができる。そして、SMORE プロットを表示する際に、ドロップダウンから選択した Names に交換できる。さらに、もう 2 つの Response を加える：1 つ目は、Name = MaxQueue Length、および Expression = ATM1.AllocationQueue.MaximumNumberWaiting である。2 つ目は Name = ResourceUtilization、および Expression = ATM1.Resource State.PercentTime(1) である。特に最後の(1)を見つけるために、式ビルダを開くことを勧める（式ビルダのドロップダウンにおいて、選択肢にマウスオーバーすると、図 4.23 のように有用な注意がポップアップされる。それは、(1)も含めて、エントリが何であるかを説明している）。対応する実験プロパティを用いて、信頼区間の Lower、Upper Percentiles、Confidence Level の割合を設定する（図 4.24 を参照）。デフォルトでは、下限と上限のパーセンタイルは伝統的な箱のプロットのように 25% と 75%（すなわち、下限と上限の四分位数）に設定される。表示された「箱」は、デフォルトの伝統的な設定においては、反復の真ん中の半分だけからの結果を含んでいるため、それらをさらに広げたいと思うかもしれない。おそらく、半分以上を表すものにしたいかもしれない（たとえば、10% と 90% を設定すると、反復の結果サマリの 80% を含む箱が表示されるだろう）。

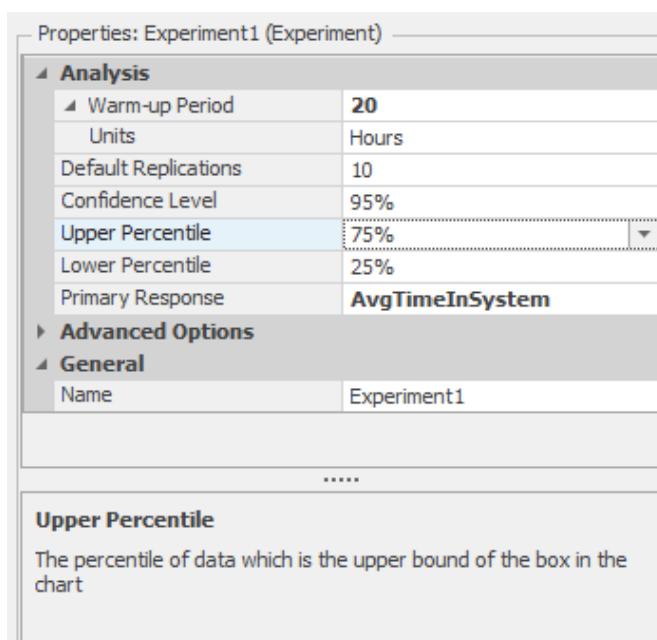


図 4.24 パーセンタイルと信頼区間の水準の設定

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、上述した Model 4-3 の 500 回の反復実行（各反復が 30 時間の実行期間、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 プロットで示している。これらは、Confidence Intervals と Histogram 表示をしており、個々の反復ごとの観測値ではない。デフォルトのままで上限 75% と下限 25% のパーセンタイルである。Rotate Plot ボタンで、垂直方向でなく水平方向のプロットを表示することもできる。SMORE プロットを生成している数値は、SMORE プロットウィンドウの下部の Raw Data タブをクリックすることによって利用可能である。したがって、グラフから読み取らなくても、実際の数値を見ることができる。図 4.25 からシステムの予想された平均時間がちょうど 0.058 時間（3.5 分）未満であることがわかる。そして、中央値は少し低く、右に歪曲している形のヒストグラム形になっている。さらに、箱のプロット（75 番目のパーセンタイル）の上限の端が約 0.064 時間（3.8 分）であるので、反復実行における平均システム内時間がこれ以上である確率は 25% である。そして、500 回の反復が合理的に正確な結論を形成するためにじゅうぶんあることを示しており、信頼区間は合理的によいことがわかる。

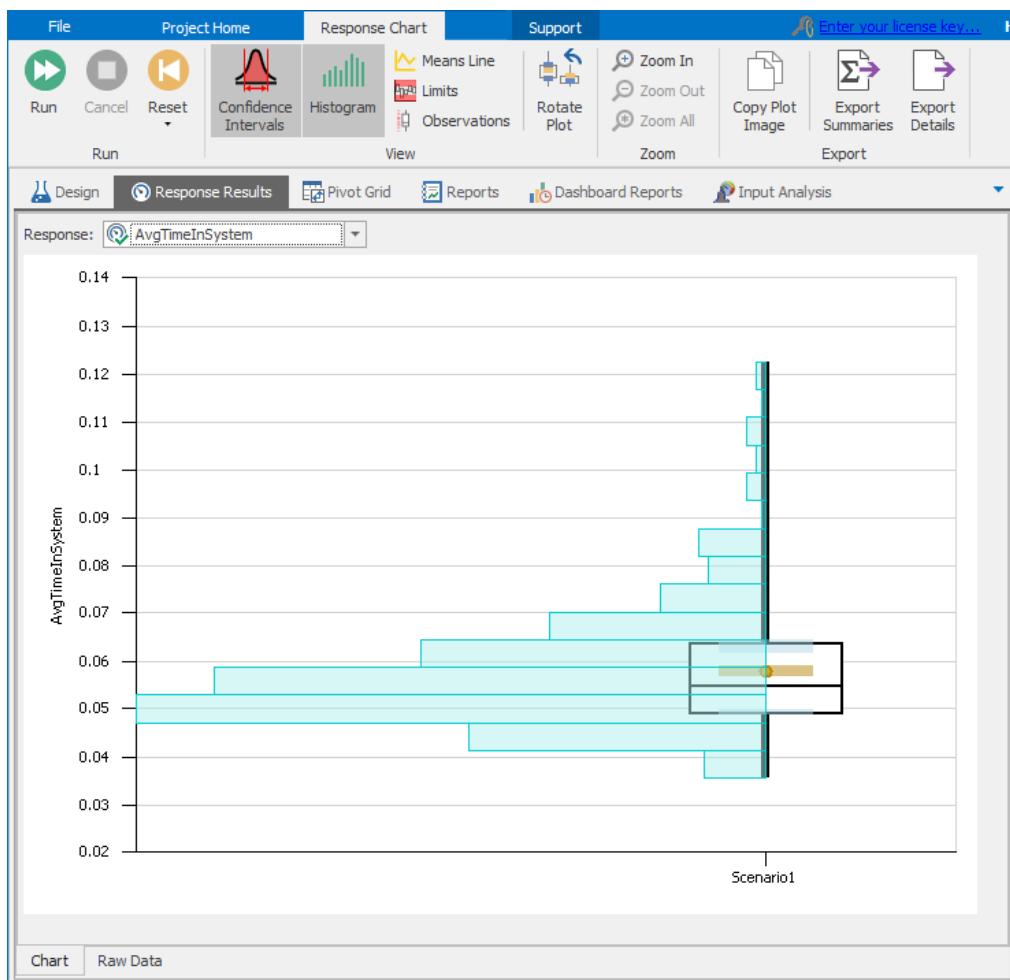


図 4.25 Model 4-3 における平均システム内時間の SMORE プロット

図 4.26 は最大の待ち行列の長さを、SMORE プロット（同じ要素を示している）で示している。これは、Upper Percentile をデフォルトの 75% から、一時的に 90% にしている。各反復における最大（平均ではない）の待ち行列の長さであるので、これは、全反復を通して、どのくらいのスペースがこの待ち行列を保持するのに必要であるかを示している。そして、ATM のロビーでは、14 人か 15 人のスペースを提供すれば、（最大の待ち行列の長さから）反復の約 90% における待ち行列を保持する余地があることがあることがわかる。Experiment Design ウィンドウで設定される

Upper Percentile と Lower Percentile を少し変化させて実行してみてほしい。もちろん、これらのパーセンタイルは、0%と100%の極端な端へ移動し、箱の端も外側へ動く。しかし、興味深いことは、信頼区間がより広くなり、すなわちより正確でなくなるということである。これは、より極端なパーセント点は本質的に変わりやすいためであり、推定することがより難しい。そのために、より広い信頼区間となる。

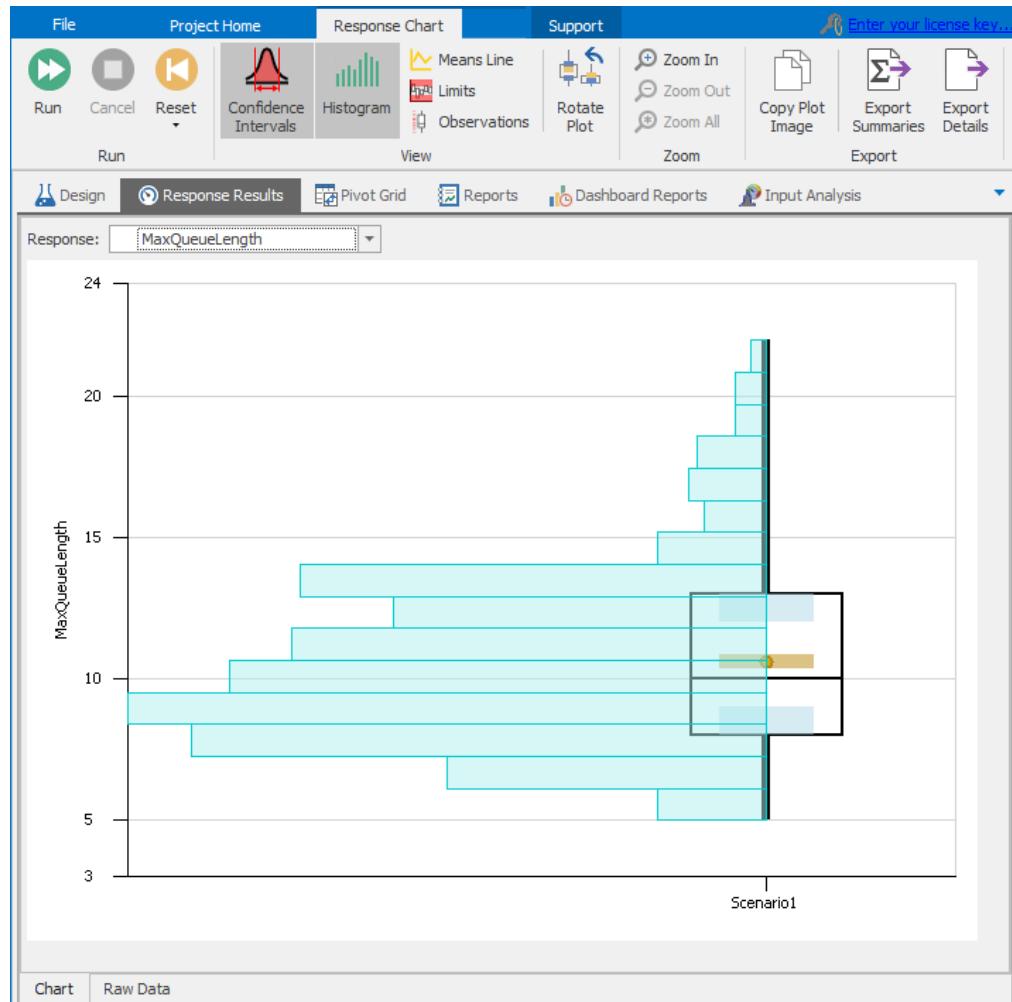


図 4.26 Model 4-3 におけるシステムの最大待ち行列の長さの SMORE プロット

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。それは、時間のおよそ半分の 77% と 83% の間であり、待ち行列理論で予想される稼働率と一致する (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)。

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

しかしながら、ヒストグラムがかなり広がっているので、稼働率が 90%くらい大きいかもしれないという可能性がある(500 回の反復実行では、最大稼働率は 90.87% であった。下部の Raw Data タブで確認できる)。

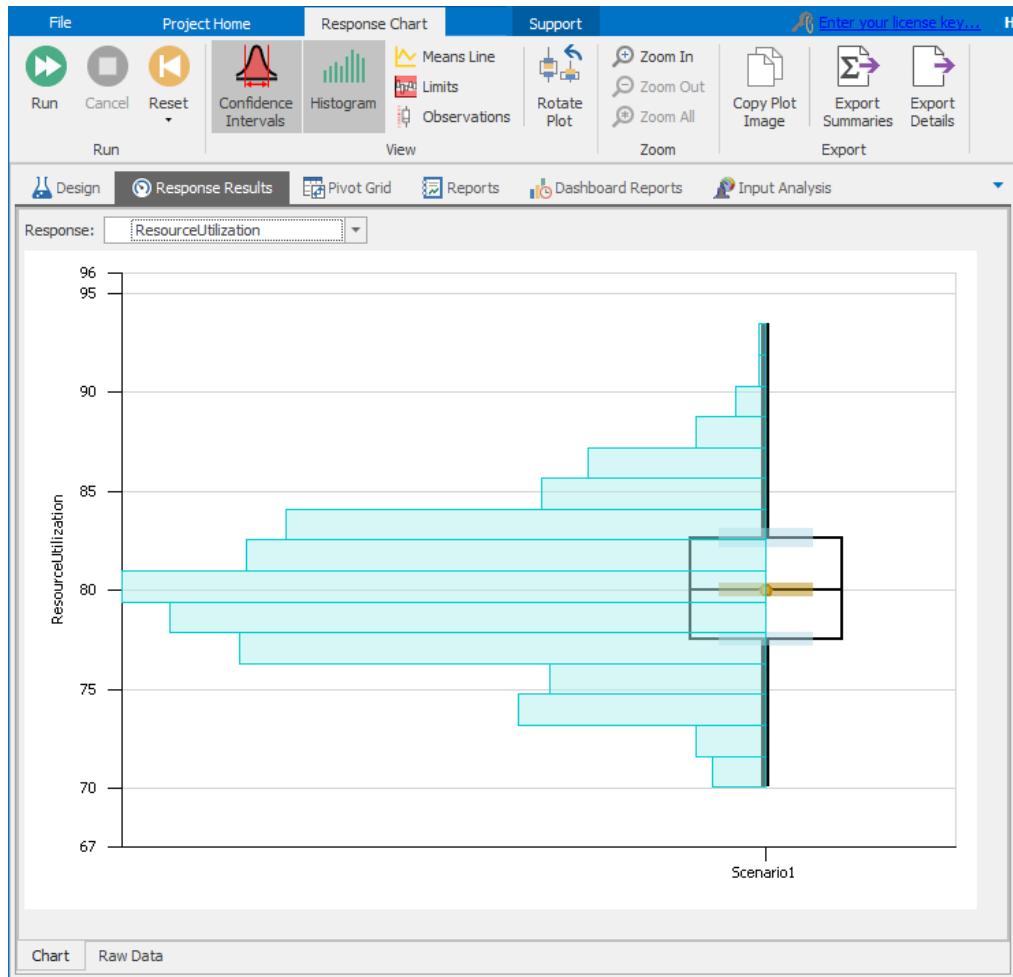


図 4.27 Model 4-3 におけるサーバ稼働率の SMORE プロット

Nelson (2008) でオリジナルに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関連している、解釈しやすいシステムのリスクとサンプリング誤差のグラフ表示を提供している。単なる反復実行の平均や、その平均の信頼区間よりも、はるかに多くの情報を提供している。SMORE プロットにおける信頼区間は、百分順位と平均値の推定におけるサンプリング誤差を表現している。反復回数を増加させることによって、信頼区間の幅（つまり、サンプリング誤差）を減少させることができる。目視により、SMORE プロットの信頼区間の幅が気に入らないほど広いなら、より多くの反復実行をする必要がある。信頼区間がじゅうぶん狭くなる（すなわち、サンプリング誤差に満足である）と、応答に関連している可変性を感じ取るために、上下の百分順位値を用いることができる。また、応答の分布状態に関して発想を得るのにヒストグラムを用いることができる（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、右側または高く歪曲されている分布であったが、稼働率の分布はかなり左右対称であることが、明らかである）。5 章で確認するが、SMORE プロットの出力パフォーマンス尺度は、それらの平均値だけではなく相対的な広がりと分布においても、複数の代替シナリオでの違いが何であるかを確認するためにかなり役に立つ。

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。これまで見てきた信頼区間と SMORE プロットのように、いくつかの統計的なツールを提供しているが（今後の章でもいくつかの機能を紹介する）、Simio はシミュレーションの結果を CSV ファイルに簡単にエクスポートすることができるため、シミュレーション実行後に output データを後処理するために、SAS、JMP、SPSS、Stata、S-PLUS、または R のようなさまざまな専用統計分析パッケージから容易

に読み込むことができる。比較的簡単な統計分析においては、Simio では CSV ファイルを直接 Excel にエクスポートするように設定でき、読み込むことができる。そして、(AVERAGE や STDEV などのような) Excel の組込み関数やデータ分析ツール、または Palisade 社の StatTools のように、おそらくより強力なサードパーティ製の統計的分析アドインを用いることができる。このように便利な形式でエクスポートされた各反復からの応答は、仮説検定、分散分析、または回帰分析などのようなあらゆる分析を自由にできる好みの統計パッケージを利用することができる。反復内からの個々のエンティティ結果ではなく、各反復から 1 つのサマリ値（たとえば、反復実行の平均システム内時間、最大の待ち行列の長さ、またはサーバ稼働率）が得られ、そして、そ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、標準の統計的手法はそれを適用しているということを忘れないでほしい。すなわち、統計用語としての「サンプルサイズ」は、実行した反復回数である。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における Write ステップ、あるいは Standard ライブラリオブジェクトの簡単なカスタマイズによって、手動で個々のエンティティの観測値を集めることができる。また、タリーおよび状態変数統計量の収集にログが導入され、Dashboard Report の表示ができる。

4.2.3 項で、Pivot Grid ウィンドウの Export Details アイコンを経由して、Experiment ウィンドウから CSV ファイルへとエクスポートする方法をすでに説明した。モデルのサイズ、および反復回数によっては、エクスポートした CSV ファイルがかなり大きいことがある。何回か再配列したり、または抽出したりすることによって、結果を得る必要があるかもしれない。しかし、いったんこの便利な形式で求めた数が保存されると、どんな種類の統計分析もできる。おそらく、出力データ自身からパターンと関係を発見しようとするデータマイニングも含まれているだろう。

たとえば、4.5 節の SMORE プロットを作成するのに用いた 500 回の反復からデータをエクスポートして、Excel のスプレッドシートの 1 列目へ 500 個の平均システム内時間を、そして 2 列目に 500 個の稼働率を抽出する。それぞれの 500 行の 1 列目の値は、その反復の平均システム内時間である。そして、2 列目の値はその同じ反復でのサーバ稼働率である。図 4.28 の散布図と相関関係を作成するのに StatTools 統計分析アドインを用いた。

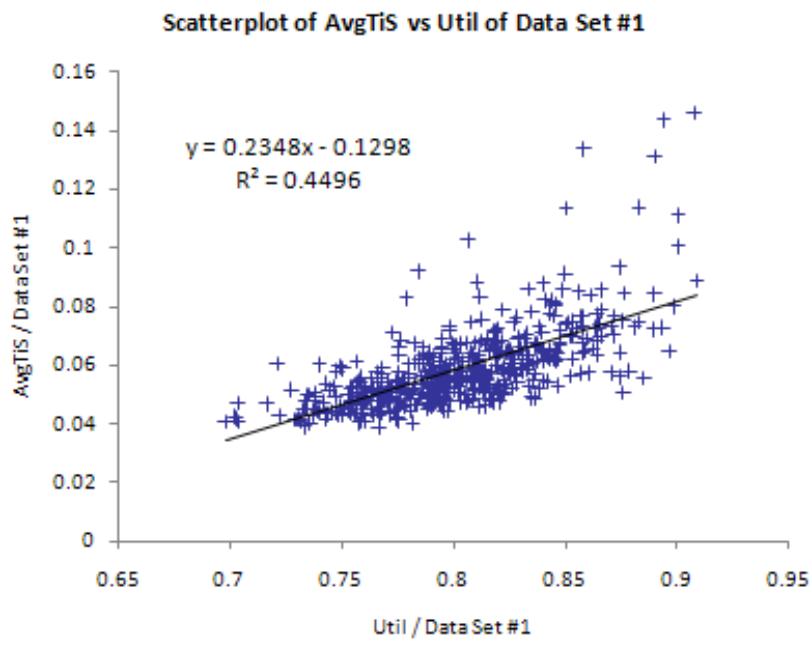


図 4.28 Model 4-3 の平均システム内時間対サーバ稼働率の StatTools 散布図

サーバ稼働率がより高くなると、平均システム内時間がより長くなるという何らかの傾向がある

のがわかる。しかし、これらの結果には多くのランダムな「ノイズ」があるので、多くの例外がある。また、線形回帰直線を重ねて、その方程式と R^2 値を示すのに、内蔵されている Excel グラフツールを用いた。平均システム内時間とサーバ稼働率との正の関係を確認し、平均システム内時間の約 45% の変動は、サーバ稼働率の変動によって説明される。これは、Simio の結果を CSV ファイルへエクスポートすることでできる、1 つの簡単な例である。これは、適切にデータを抽出し（おそらく、カスタムマクロかスクリプト言語を用いる）、それからモデルの挙動と関係性について知見を得るために、強力な統計的パッケージでシミュレーション出力データを読み取ることによって実現できる。

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、種々なデータを記録して、ダッシュボードレポートを作成する機能を提供する。いずれのライセンスタイプであっても、一定のログ機能を提供する。さらに、Academic RPS ライセンスでは、それ以上の機能を提供する。Simio のログとダッシュボード機能は非常に強大であるが、ここでは、シンプルな例を 1 つだけ紹介することにする。さらなる機能の詳細な説明は、Simio Help と SimBits から利用できる。

この例では、Model 4-3 を活用し、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ことにする。まず、ATM リソースの稼働率のログを記録するために、Simio に指示を出す。手順は下記の通りである。

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に、ATM オブジェクトのインスタンスを選択し、Advanced Options リソースグループから Log Resource Usage プロパティの設定を True にする。
3. 早送りモードでモデルをリセットし実行させる。
4. Results タブの Logs セクションをクリックし、Resource Usage Log をクリックする。図 4.29 は、最新の Resource Usage Log の一部を示す。リソース稼働率ログは、特定されたリソースのあらゆる「使われている」状態を記録し、特に、すべてのエンティティ、開始時間、終了時間、およびその期間中の稼働率も特定できる。

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	0.0202772	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 12:20:44 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 12:23:31 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が記録される。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ことにする。手順は、下記の通りである。

- Results タブから、Dashboard Reports セクションを選択し、Dashboards リボンから Create をクリックする。また、Add Dashboard ダイアログボックスに、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を示す。Data Source ドロップダウンボックスから Resource Usage Log をクリックする。

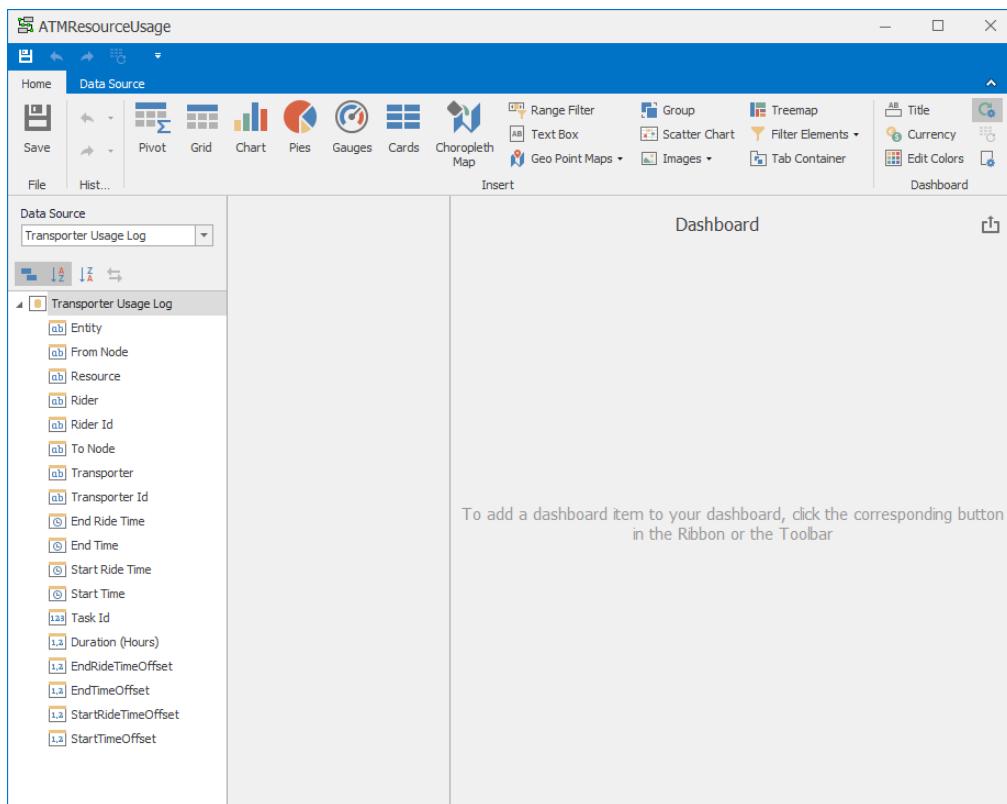


図 4.30 ダッシュボードレポートのダイアログボックス初期画面

- Chart Tools / Home リボンから、Chart オプションを選択し、新しいチャートを作成する。図 4.31 は、新しいダッシュボードレポートで作成できたチャートを示す。ここから、Resource Usage Log から必要な項目をドラッグし、DATA ITEMS の構成要素としてドロップすることによって、チャートをカスタマイズする。
- Duration (Hours) という項目をドラッグし、Values (Pane 1) データ項目としてドロップする。
- データ項目の右側の棒グラフの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。
- Start Time という項目をドラッグし、Arguments データ項目としてドロップする。
- ドロップした Start Time のドロップダウンボックスを開き、リストから Date-Hour-Minute オプションを選択する。作成されたリソース稼働率チャートを図 4.32 に示す。チャートは、リソースの利用開始時間別のリソース稼働率が表示できる。
- 新しく作成されたダッシュボードを保存するには、リボンにある Save をクリックする。

前述のように、Simio のログとダッシュボード機能は非常に強大であるため、これらの機能をシミュレーション実験で体験することをお勧めする。また、関連の Simio ヘルプや SimBits についても参照してほしい。

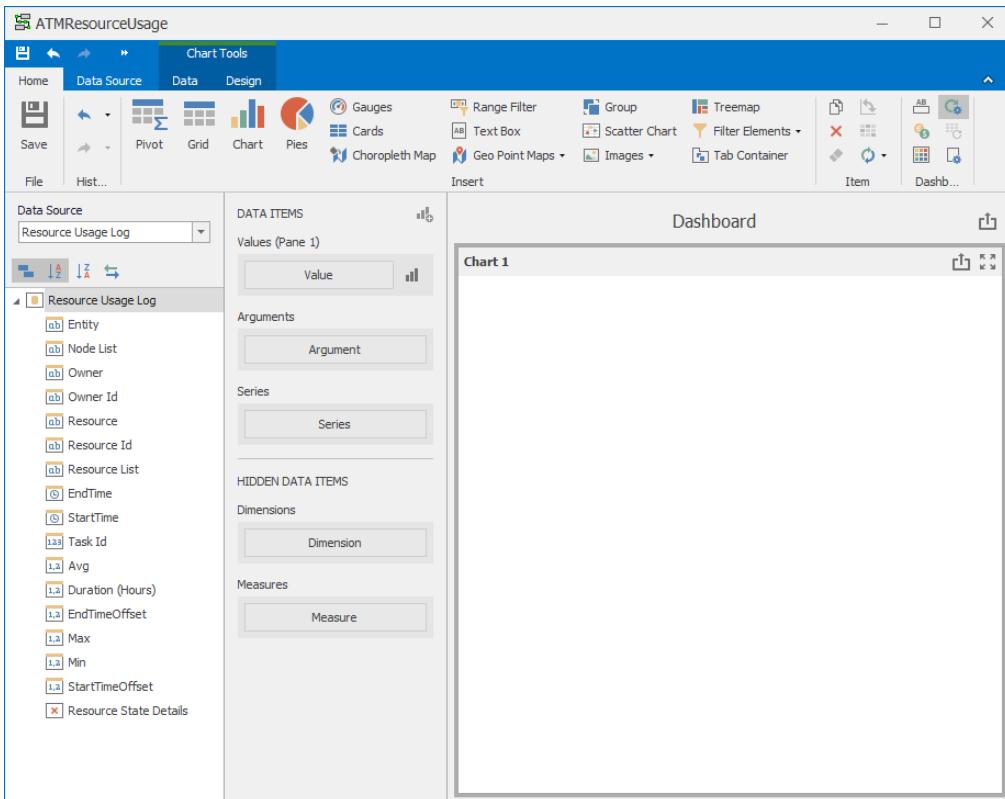


図 4.31 ダッシュボードで作成できた新しいチャート

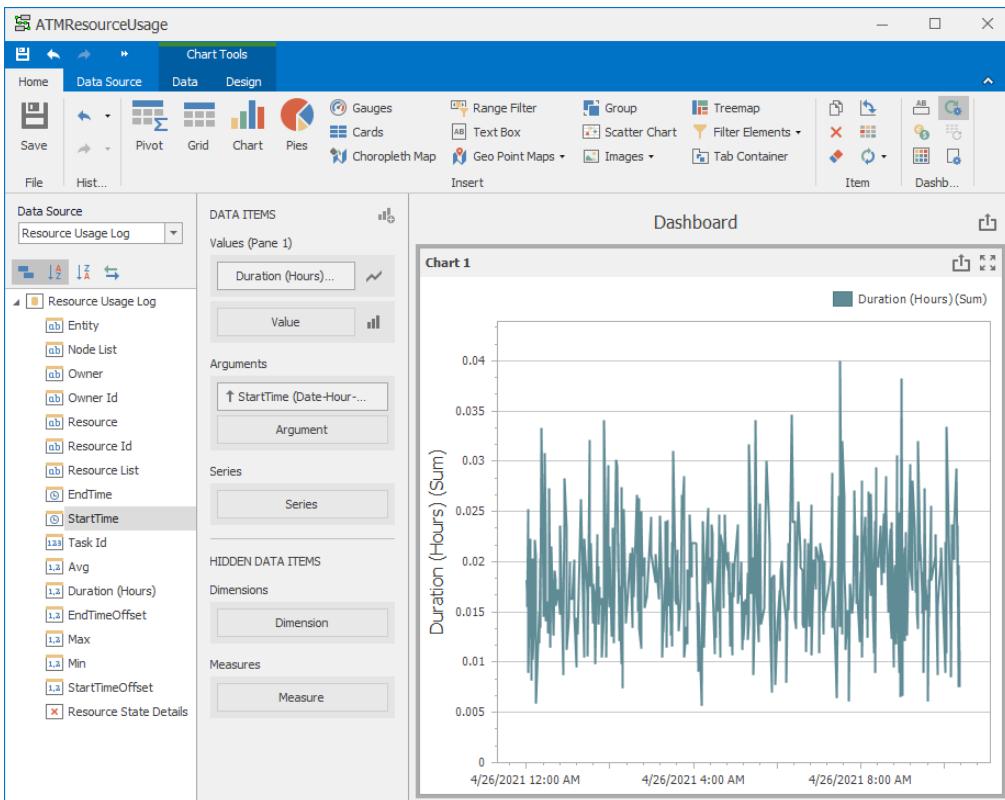


図 4.32 ダッシュボードで作成されたリソースの稼働率チャート

4.8 モデルアニメーションの基礎

ここまでほとんどアニメーションを参照しなかったが、すでに役に立つアニメーションを作成している。本節では、アニメーションに関するいくつかの主要な内容を紹介する。8章で、詳細にアニメーションについて説明する。通常、アニメーションは Facility ビューで行われる。Facility ヴ

インドウをクリックして、Hキーを押すと、アニメーション周辺操作に関するキーボードとマウスの利用についてのヒント表示の有無を切り替えることができる。インターフェースに慣れるまでは、表示したままにしたほうがよいだろう。

Simio の強みの 1 つは、Standard ライブラリを用いてモデルを構築すると、モデルを構築する同時に、アニメーションも構築しているということである。図 4.8 および図 4.20 では、2 次元（2D）のアニメーションモデルが表示されていた。さらに Simio の強みは、モデル構築中、通常は 2D ビューが利用されているが、同様に 3D が自動的に作成されているということである。2D と 3D のビューモードを切り換えるには、キーボードの 2 あるいは 3 のキーを押すか、または View リボンを選択して 2D または 3D のオプションをクリックするだけである。図 4.33 は 3D モードの Model 4-3 を示している。3D モードでは、3D ビューの移動、ズーム、回転にマウスボタンを用いることができる。図 4.33 に示したモデルは、サーバに 1 つの顧客エンティティ（ATM オブジェクトの Processing.Contents 待ち行列）が表示され、5 つの顧客エンティティが ATM の待ち行列（ATM サーバオブジェクトの InputBuffer.Contents 待ち行列）で待っており、そして入口から ATM 待ち行列までの経路に 2 つの顧客エンティティがいる。

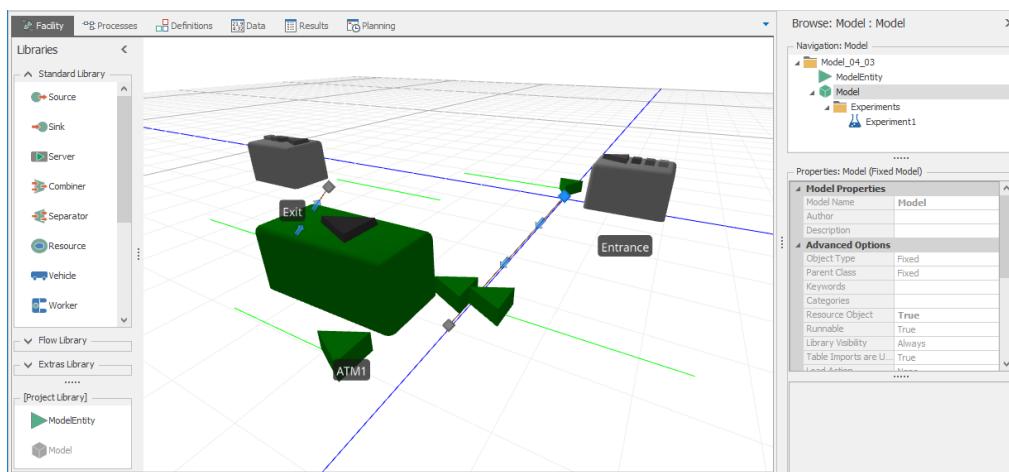


図 4.33 Model 4-3 の 3D ビュー

Model 4-4 では、Model 4-3 を変更することによって、アニメーションを充実させる。もちろん、Model_04_03.spfx に新しい名前をつけてファイルに保存することから開始すべきである。Model_04_04.spfx として、ダウンロードした同じ名前のファイルに上書きしないように、システムの上の異なったディレクトリへ保存するとよい。リボン上の Project Home タブ左の黄色いブルーダウンタブに、Simio の Save As がある。

何かシンボルやオブジェクトをクリックすると、Symbols リボンが前面に出る。このリボンは、シンボルに適用された色やテクスチャを変えたり、追加シンボルを加えたりするオプションや、デフォルトを置き換えるためにシンボルを選択するためのいくつかの方法を提供している。Simio の内蔵のシンボルライブラリから新しいシンボルを選択するという、もっとも簡単なタスクから始める。特に、エンティティのピクチャをデフォルトの緑色の三角形から現実的に見える人間のピクチャに変えよう。

ATM Customer と命名した Entities オブジェクトをクリックすることから開始する。Symbols リボンにある Apply Symbols ボタンをクリックすれば、図 4.34 で示されているように、既存のライブラリを一番上の箇所にあることが確認できる。全ライブラリは、27 のカテゴリに分類された 300 以上のシンボルによって構成される。Domain、Type、および Action という 3 つのフィルタを利用してことで、シンボルを簡単に見つけることができる。フィルタを組み合わせて利用することによって、探しているものの選択肢を抽出しやすくすることができる。これらのフィルタのどんな組合せでも使うことができる。たとえば、Type フィルタを利用し、People オプションだけを

選択すると、ライブラリにあるすべての People を一覧できる。また、一番上には、Library People カテゴリを確認できる。いずれも（クリックはしないで）シンボルの上へマウスを重ねると、選択を支援するための拡大図が見られる。ハイライトされたオブジェクトのエンティティに新しいシンボルとして適用するために、「Female」というシンボルから1つをクリックする。モデルのエンティティは、現在、図 4.35 ようになっているはずである。People フォルダの中には Animated というサブフォルダがある。このフォルダには、Walking、Running、Talking のような行動を表す、よりリアルな人間のシンボルが用意されている。しかし、今回的小規模な ATM モデルには必要ないだろう。

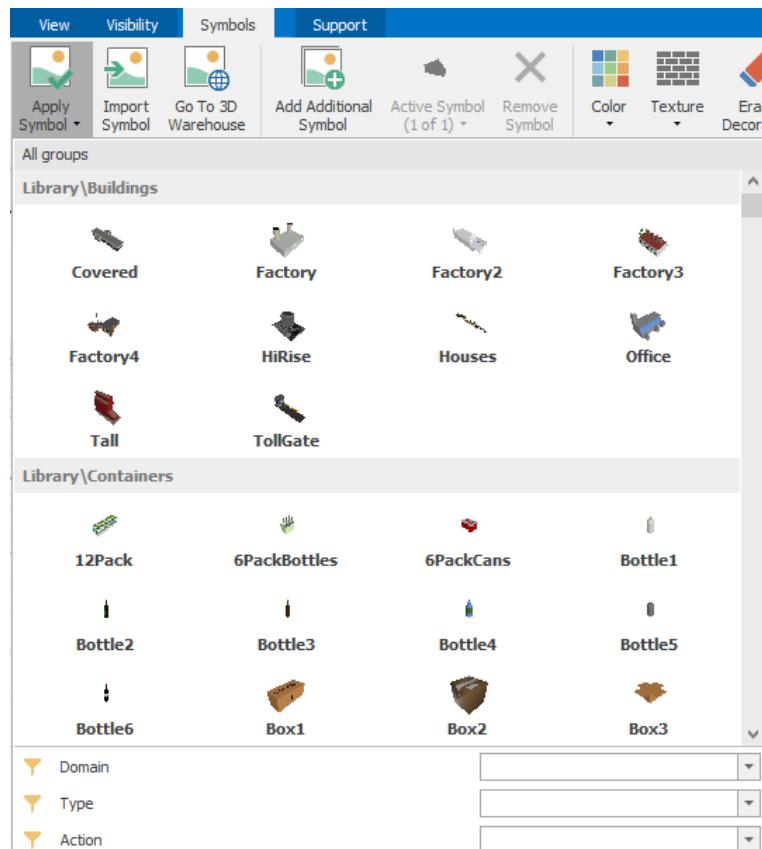


図 4.34 Simio シンボルライブラリのナビゲート

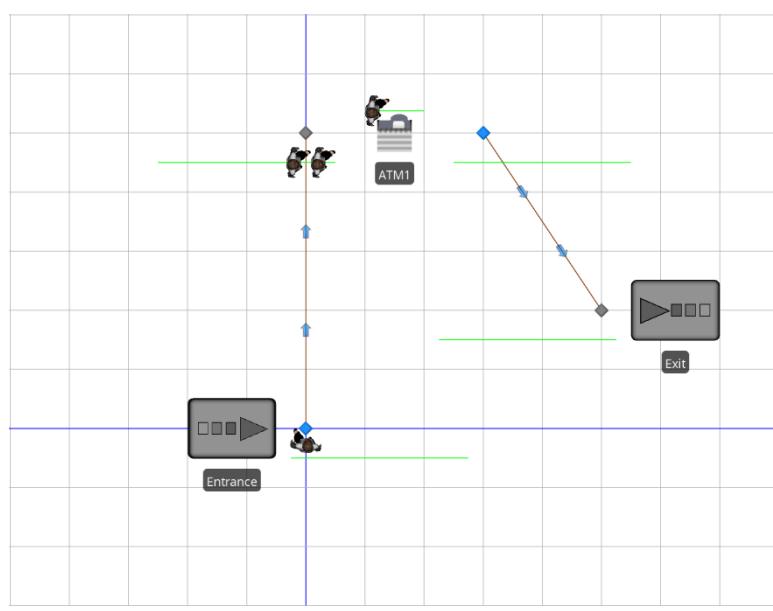


図 4.35 Model4-4 の 2D ビューにおける女性と ATM シンボル

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かれるだろう。残念ながら、これはライブラリのシンボルの 1 つではない。容易に利用可能な ATM シンボルがたまたまあるなら、それをインポートすることができる。または、あなたがひときわ芸術的であるなら、描くことができるかもしれない。しかし、Simio では簡単な解決法を提供している。それは、Trimble 3D Warehouse からダウンロードすることである。これは無料で利用可能なシンボルの巨大倉庫である。Simio はそこへの直接リンクを提供している。では、サーバのピクチャを ATM 機のものに変えてみる。

すでに ATM1 と命名した Server オブジェクトをクリックする。ここで、Symbol リボンを選択し、Go To 3D Warehouse アイコンをクリックする。この操作で、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。検索ボックスの中に ATM と入力し、Search ボタン（虫眼鏡のアイコン）をクリックして、Models タブを選ぶ。名前もしくは内容説明に ATM が含まれている数百個のシンボルのうち、図 4.36 のようなシンボルを見ることができる。ただし、ライブラリは頻繁に更新されるため、検索結果は図とは異なる場合もある。同様に、Web ページ自体の見た目や操作方法も、本章での説明とは異なっているかもしれない。

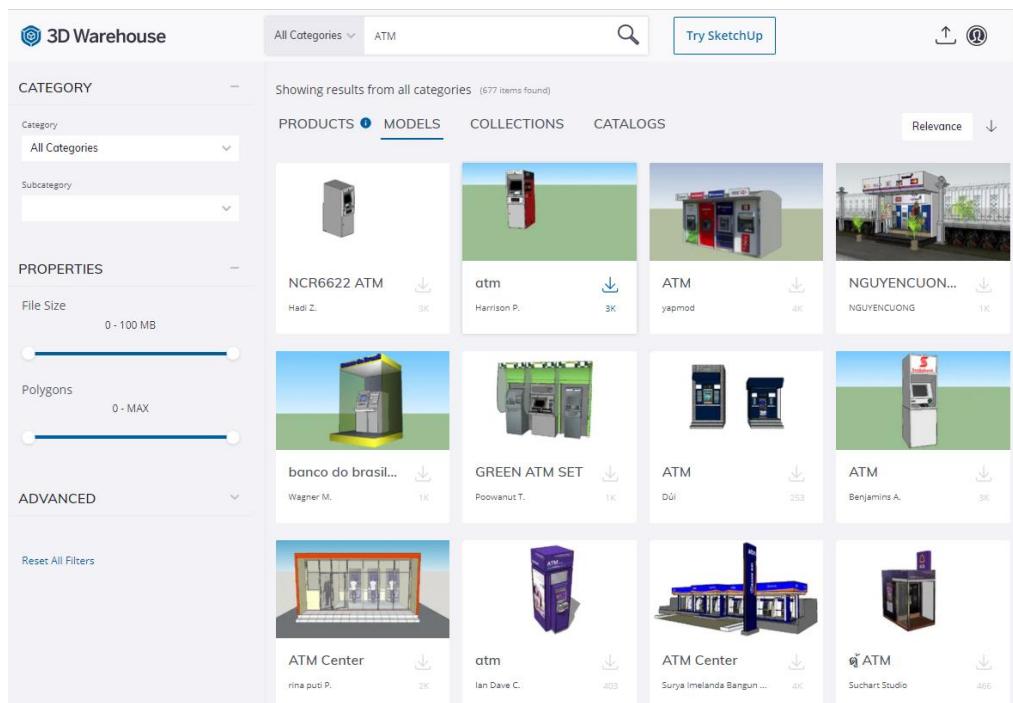


図 4.36 Trimble Warehouse における ATM の検索結果

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 つは見つけられるだろう。関心のあるオブジェクト（ここでは Mesin ATM を選んだ）をクリックすると、ファイルサイズなどの基本的な情報を見ることができる。See more details をクリックすると、ブラウザから 3D モデルを表示したり、回転させたりすることができる。問題がなければ、Download を選択し、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ために、3D warehouse のアカウントを持っていなければ、まずアカウントを設定する必要がある）。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 アイコンを使って、選択した Server にシンボルをインポートし適用することができる。これにより、3D モデルが Simio モデルにインポートされ、シンボルの名前やサイズ、向きを変更して、オブジェクトインスタンスに適用することができる。一度インポートしたシンボルは、（Apply Symbol アイコンを用いて）再インポートす

ることなく、他のオブジェクトインスタンスに適用することができる。

インポートのプロセスで重要なことの1つに、(メートル単位の) サイズが適切であるかどうかを確認することがある。寸法の比率を変えることはできないが、サイズが異なる場合は、どんな値にも変更できる。この場合は、ATMはおよそ幅0.7メートル、高さ2メートルであるため、そのままでもよいだろう。OKをクリックすると、ATMサーバに新しいシンボルの適用が完了している。

ここで、2Dビューで実行しているとき、新しいModel 4-4は図4.35のようになっているはずである。3Dビュー(3キー)に変更すると、図4.37のように選択した新しいシンボルが3Dでよく見えるのがわかる。

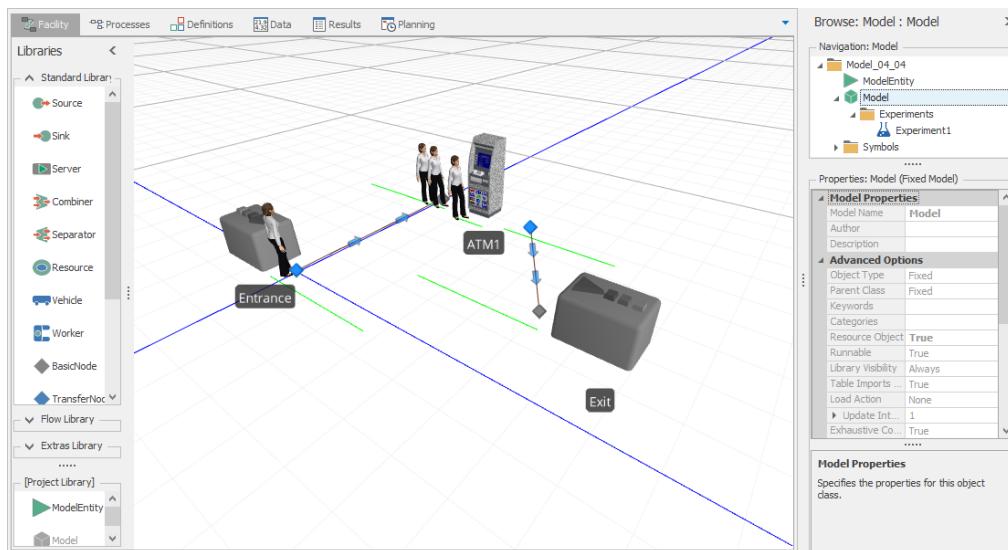


図4.37 Model 4-4の3Dビューにおける女性とATMシンボル

もちろん、ここではアニメーションの可能性について、ほんの少し紹介したに過ぎない。(Drawingリボンで)壁を描いたり、または扉や植物のような外観を加えたりすることができます。より現実的にモデルを表現するために概略図、または他の背景をインポートすることもできる。8章で、これらについて議論する。

4.9 モデルのデバッグ

信じがたいのであるが、人間が誰でもミスを犯すものである。一般的にソフトウェアに存在する誤りはバグと呼ばれる。バグを引き起こす原因はさまざまである。たとえば、入力ミス(3を入力するつもりで、2と入力してしまう)、システムの仕組みに関する誤解、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、あるいはソフトウェアにおける問題などがある。経験豊富なシミュレーション専門家であっても、バグに遭遇するものである。実際、モデリングにおいて、もっとも労力がかかることはバグの解決である。複雑なシステムをモデル化する場合、複雑なソフトウェアを活用するのは当然である。そして、初めて作成した現実のモデルにいくつかのバグがあるのは当たり前であろう。そのときに、バグをいかに効果的に認識し取り除くかが、モデル作成者にとっての効率性に影響する。本節では、デバッグ能力を向上させるために、いくつかの方法を解説することにする。

もっとも優れた方法は、バグの影響を最小限にさせる反復型開発手法である(1.5.3項を参照)。モデルが正しく実行できるかどうかを検証せずに、ひたすらにモデルの構築に何時間も費やすと、複雑かつ見つけにくいバグに間違いなく遭遇するだるう。代わりにモデリングを頻繁に止め、モデルを検証することを勧めたい。問題を見つけたときに、問題を引き起こす原因について考え抜くことで、バグを迅速に見つけ修正する能力を身につけられる。

バグに対するもっとも一般的な初期反応は、ソフトウェア自体のバグと考えることである。いく

らよく書かれたプログラムであって、じゅうぶんに検証されたものであっても、複雑なソフトウェアであれば、たいていのソフトウェアにはバグは存在するのが、大多数の問題はユーザ側の誤りである。つまり、モデル作成者自身に起因する問題である。他の原因があると証明できない限り、誤りは自分にあるとして、バグ修正にすぐ取り掛かるのが最善の道である。

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。多くの問題は見えやすいものである。Run をクリックすれば、何も起こらないか、あるいは劇的な何かが起こる。しかし、最悪の問題は見つけにくいものであり、発見できるまで探し続ける必要がある。

- 4.2.2 項では、モデルを実行する前に、結果の予測をしておくことの重要性を述べた。モデルの結果と予測を比較することが、問題を見つけるもっとも優れた方法である。4.2.5 節では、モデル検証に関する基本プロセスを解説した。下記の手順は、より詳細な検証方法である。
- アニメーションを観察する。動くべきものは動いているか。動いていないのであれば、なぜ動いていないのか。
- アニメーションに詳細な情報を加える。エンティティや他のオブジェクトに診断情報を加えるため、浮動ラベルやフロアラベルを用いる。
- 出力統計結果を入念に確認する。出力結果や結果間の関係は妥当であるか。たとえば、利用率の低いリソースの前に長い行列があることは妥当だろうか。必要であれば、より多くの情報を得るために独自の統計量を追加する。
- 最後に、まだ何らかの問題があるかどうかを確認するために、後述するデバッグツールを活用することもできる。

4.9.2 デバッグプロセス

バグがあることを確信したら、ユーザ自身が責任を持って、自らが行った誤りに起因するバグを見つけ出す必要がある。問題によって、下記に示すようなさまざまなアクションを試すことになる。

- すべてのオブジェクトをひと通り調べる。特に直近に追加したり、頻繁に変更したりするオブジェクトを重点的に調べる。
 - デフォルト値から変更したすべてのプロパティを確認する (Simio では、変更したすべての値は太字で表示され、そのカテゴリは展開されている)。そして、これらの値が、確かに変更を意図した箇所であり、正確に変更されたことを確認する。
 - デフォルトから変更されていないプロパティも確認する。つまり、デフォルト値であることが意味を持つことを確かめる。しばしばそうではないケースがある。
- エンティティの流れを最小限にする。モデルのエンティティを 1 つに限定し、問題が再現されるか確認する。再現されなければ、2 つ目のエンティティを追加する。驚くべきことに、1 つか 2 つのエンティティだけでも、多くの問題が再現され、特定できる。最小限のエンティティを用いることは、他のデバッグプロセスやデバッグツールでも役に立つ。Simio では、ソースごとの Maximum Arrivals を、0、1、あるいは 2 に設定することにより、簡単にエンティティ数を制限することができる。
- モデルを最小化する。モデルを保存し、そして問題に対して影響がないと考えるモデルの要素を削除する。削除しすぎた場合、元の状態に戻して、他の部分を削除する。モデルが小さければ小さいほど、問題を発見しやすく、解決しやすくなる。
- 警告やエラーメッセージが表示される場合、何度も入念に確認する。メッセージはわかりにくい場合もあるが、通常、価値のある情報が盛り込まれている。メッセージを解読してほし

い。

- ・ エンティティの流れを段階的に確認する。エンティティがなぜそのような動きをしているのか、何をしているのかを正確に理解する必要がある。従うべき進路に進まない場合、間違った指示をエンティティに与えた可能性がある。あるいはエンティティに何も指示を与えていない可能性もある。出力結果から手掛かりを探ることも可能である。
- ・ 観点を変える。全く異なる方向から問題を再度検討する。プロパティに注目している場合、先頭から確認するのではなく、最後からチェックしてみる。オブジェクトに注目している場合、通常最後に確認するオブジェクトからスタートする。この方法で、考えやビジョンの新しい道すじが開かれ、初めは見えなかった何かに気づく可能性がある。銀行業では、しばしば、1人の行員が数値の桁を左から右に読み、別の行員が右から左に読むことで、ダブルチェックを行っている。これは、人間が真実よりも、自らの期待に基づいて認識してしまう傾向にあるというパターン認識のサイクルを打破するチェック方法である。
- ・ 友人に協力してもらう。同じ分野のモデリングに精通する仲間に恵まれたのであれば、その仲間に問題を解決するための助言をもらうのもよいだろう。しかし、シミュレーションやモデル化対象の領域についての専門知識のない方に対して、詳細にプロセスを説明するのもよい方法である。実際には、この方法は1人だけのときでも使える。つまり、自宅の熱帯魚やぬいぐるみに対して説明しても効果がある。愚かなように感じられるだろうが、確かに効く方法の1つである。声に出して問題を説明することは、違う方向から問題を検討することになり、問題の発見と解決に役立つ。
- ・ RTFM (Read The Friendly Manual。ユーザに優しい（真実かどうかは疑問だが）マニュアルを読む)。マニュアルを読みたい人は少ないであろう。しかし、他の方法で解決できなかった場合、テキストやマニュアル、あるいは対話型のヘルプ機能を利用すべきときが、とうとうやってきたのである。

上記のステップは、順序どおり従わなくてもよい。最後のステップから行ったり、ところどころをスキップすれば、よりよい結果が得られるかもしれない。次節では、デバッグツールを利用し、デバッグプロセスをより容易にする方法を議論することにする。

4.9.3 デバッグツール

アニメーションおよび数値出力結果はデバッグのきっかけを提供するが、高機能のシミュレーションソフトウェアは、モデル作成者がモデルで起こっていることを理解することを支援するツールを提供している。ここでは、次は、Trace (追跡)、Break (中断)、Watch (監視)、Step (ステップ実行)、Profiler (プロファイル) と Search Window (検索ウィンドウ) という順で基本ツールを解説する。

Trace (追跡) は、モデル実行中に何か起きているかについて詳細な説明をユーザに提供する。エンティティの流れやイベントと、それらが引き起こす結果が一般的に記述される。Simio の Trace は、Process Step レベルである。プロセスにおける各ステップが、1つ以上の追跡命令を生成する。プロセスを学んで理解できるまで、Simio の Trace は、読み取りにくいと感じられる可能性があるが、Step を理解できれば、提供される有効かつ詳細な情報に関心を示し始めるだろう。Simio Trace では、使いやすさのためにフィルタ機能が実装されており、モデル実行後の分析のために外部ファイルへのエクスポートもサポートしている。

Break (中断) は、決められた時点で、シミュレーションを一時停止する機能を提供する。特定の時点でシミュレーションを止めることができ、メインの機能である。また、エンティティが特定の場所に到着した時点で（たとえば、サーバへの到着）、シミュレーションを一時停止する機能は、非常に役に立つ。より高度な機能として、「3つ目のエンティティが A という場所に到着する」、あるいは「時間 125 以降に到着する最初のエンティティ」というような条件つきの一時停止機能もある。

る。Simio の基本的な Break 機能は、オブジェクトあるいはステップの右クリックメニューにある。高度な Break 機能については、Simio の Break ウィンドウからアクセス可能である。

Watch（監視）は、モデルのシステム状態をチェックする機能を提供する。通常は、シミュレーションを一時停止した際に、モデルの状態やオブジェクトレベルの状態を見て、モデル内の決定やアクションがなぜ、どのように起きているのか、その影響はどのようなものか、という事柄への理解を向上させる。Simio の Watch 機能は、あらゆるオブジェクトの右クリックメニューからアクセスできる。Simio の Watch 機能は、プロパティ、状態変数、関数や、各オブジェクトの他の側面にアクセスし、オブジェクトの階層構造に従って、より深く検討する能力を持っている。

Step（ステップ実行）は、ステップと呼ばれる小さな単位の活動に従って時間を進行させ、モデルの実行をコントロールする機能である。この機能を利用すれば、モデルのアクションとそれに起因する結果を入念に検証できる。Simio は 2 つのステップモードを提供している。Facility ビューであれば、Step ボタンにより、アクティブエンティティを次に活動するシステム時間に動かすことができる。Process ウィンドウであれば、Step ボタンでエンティティ（トークン）を次のプロセスステップへと移動させることができる。

Profiler（プロファイル）は、実行スピードに関連する問題に役立つ。この機能は、実行スピードを浪費する原因を探るために内部分析を提供する。プロセッサを集中的に利用する特定のステップを識別することにより、モデルの問題点が明らかになり、別のモデリングアプローチを利用することで実行スピードを改善する機会が得られる。

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、アクティブな方法でプロジェクトにあるすべての箇所を調べるアクティブな機能を提供する。たとえば、「Teller」というオブジェクト名を利用したすべての箇所を検索するのに、Search Window は有効なツールとなるだろう。または、状態変数として利用した「Counter」の数箇所を別の表現に変更したい場合においても、利用できるだろう。

Trace、Break、Watch、および Step は、非常に強力なデバッグのツールセットとして同時に利用できる。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て活用すれば、モデルをよりよく理解したり、ベストな結果を導くための優れたメカニズムになる。

Trace、Errors、Breakpoints、Watch、Search、および Profile のウィンドウは、Project Home リボンからアクセス可能である。これらのウィンドウは、一般的にユーザのアクションにより自動的に開くことができる。たとえば、Run リボンの Model Trace を操作すれば、Trace ウィンドウは開くことになるだろう。また、表現入力の際、構文エラーがあれば、Errors ウィンドウは自動的に開くことになるだろう。しかし、時によって、これらのボタンを利用し、ウィンドウを再開せたり、閉じたり（たとえば、Errors ウィンドウ）、或は新規ウィンドウで、Breakpoints ウィンドウを開けたりすることがあるだろう。

図 4.38 に、これらの機能を活用する典型的な画面を示す。黒丸は、Trace ウィンドウを表示するボタンであり、モデル追跡の生成に利用される。Server2 の進入ノード（赤丸）に設定された Break ポイントに達したとき、実行が自動的に停止（一時停止）するまで、モデルの実行を追跡することができる。その時点で、Step ボタン（青丸）をクリックすれば、エンティティが次の時間進行に至るまで先に進んだ結果として（黄色の背景）、さらに 11 行の追跡が生成される。また、画面右側にある Watch ウィンドウは、Server2 を監視しており、入力バッファとバッファ内の各エンティティを観察できる。

デフォルトで配置されたデバッグウィンドウは、同じ画面にある複数のタブとして表示される。それぞれのウィンドウをドラッグすることで、図 4.38 に示したように、各ウィンドウを再配置できる。また、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を行うことも可能である。これらのウィンドウが他のスクリーンの上に重なって配置することも可能であるため、時には、ウィンドウの居場所がわからなくなることがあるだろう。この場合、Project Home リボンにある Reset ボタンをクリックすることにより、すべてのウィンドウの位置をリセットされる。

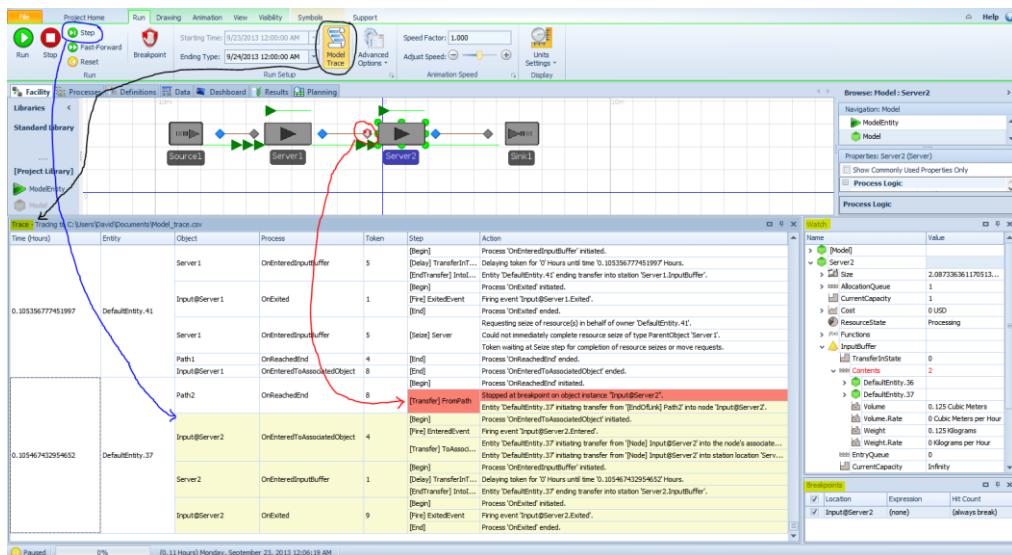


図 4.38 カスタムレイアウトによる Trace、Watch、Break ウィンドウの利用

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、いくつかの簡単な Simio モデルを構築した。途中で、シミュレーションの出力における統計分析について述べた。それは、実際のシミュレーションプロジェクトでモデル化するのと同じくらい重要なことであり、反復、実行期間、ウォームアップ、モデル検証、および強力な SMORE プロットによって可能となる分析能力に関するものである。待ち行列の理論モデルから始めたが、後で現実的な待ち行列システムをモデル化するためのいくつかの関心のある背景を加えた。その過程で、エンティティ移動をモデル化するための Simio の Path の利用や、アニメーションの基本についても議論した。Simio や Simio に関連するトピックは、次章以降もより興味深いモデルを用いて述べることにする。

4.11 問題

- 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 を用いる。100 時間モデルを実行しなさい。そして、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のシステムに費やされた平均時間を報告しなさい。
- 問題 1 から Simio モデルの待ち行列モデルを展開させ、100 時間にエンティティがシステムで費やす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。
- 問題 1 のモデルを用いて、100 回の反復の実験をしなさい。実験を実行し、エンティティがシステムで費やす時間を SMORE プロットで観測しなさい。様々な SMORE プロット設定をして実験すること。上限と下限の百分順位値を変え、プロットを回転させて、ヒストグラムを見ること。
- 問題 3 の実験を 5 回（または何回か）実行すると、到着時間間隔とサービス時間はランダムであるが、いつも全く同じ結果を得る。なぜこのようになるのか？
- あるシステムのモデルを構築する。検証の一部として、得るべきである結果についていくつかの期待値も作成する。しかしながら、モデルを実行すると、結果は期待値と合っていない。このミスマッチとして考えられる 3 つの解釈は何であるか？
- シミュレーションモデリングという状況において、反復とは何か。そして、通常、特定のモデルにおいて何回の反復を実行するかをどのように決定しているか？
- 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は何であるか？**
- 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とは何であるか？**

9. Simio プロセスによって問題 1 のモデルを複製しなさい（すなわち、Standard ライブラリからオブジェクトを用いない）。100 時間の長さで 50 回の反復をし、問題 1 のモデルとこのモデルの実行時間を比較しなさい。
10. ATM モデル (Model 4-3) で、240 時間 (10 日間) の長さで 10 回の反復実行をしなさい。システムの顧客の最大数とシステムの顧客の最大の平均数はいくつであるか (このモデルは、ATM における物理的な空間を考えていないことを思い出してほしい)。仮定は合理的であったか (すなわち、物理的な空間を考える必要はなかったのか) ?
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概観をどのように示すかを説明しなさい。
12. 問題 1 のモデルにアニメーションを追加しなさい。ファストフード店でのレジ係をモデル化し、エンティティが顧客を表し、サーバがレジのレジ係を表すと仮定する。アニメーションに、Simio の標準のシンボルを用いること。
13. 問題 1 からモデルを変更する。そのモデルは、鋼板の掘削孔にかかる製造プロセスをモデル化していると仮定する。ボール盤には、一度に 3 つの部品 (待ち行列用語では、 $c = 3$) まで容量がある。到着率は毎時 120 個の部品とし、そして、プロセス率は毎時 50 個の部品とする。エンティティ (鋼板) とサーバ (ボール盤または他の穴を作る装置) の適切なシンボルを見つけるために Trimble 3D Warehouse を用いなさい。モデルを実行するときに、いくつの部品が処理されているかを示すラベルをアニメーションに追加しなさい。
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結果を確認し、クロスチェックするために、Simio モデルを構築しなさい。それらの正確な定常状態の出力パフォーマンス測定基準は、2.3 節で与えられている。Simio モデルは空かつ遊休の状態に初期化されており、統計的変動を被りやすい結果を生むことを覚えておいてほしい。そして、これらの両方の問題に対処するために Simio Experiments を設計して、実行しなさい。試行錯誤の後、実行期間や反復回数、そしてウォームアップ期間などを決定する。その都度、最初に 2.3 節における結果から、待ち行列理論の定常状態出力パフォーマンス測定基準 ρ 、 L 、 L_q 、 W 、 W_q を計算する。そして、それらとシミュレーション推定値と信頼区間について比較する。また、すべての時間単位は分であり、Simio モデルも同様に分を利用すること。
 - a) $M/M/1$ 待ち行列で、到着率は毎分 $\lambda = 1$ でサービス率が毎分 $\mu = 1/0.9$ である。
 - b) $M/M/4$ の待ち行列で 1 分あたりの到着率 $\lambda = 2.4$ あり、それぞれの 4 つの個々のサーバは 1 分あたりのサービス率が $\mu = 0.7$ である (図 2.2 の mmc.exe コマンドラインプログラムで用いられた同じパラメータ)。
 - c) $M/G/1$ の待ち行列で、1 分あたりの到着率 $\lambda = 1$ あり、そしてサービス時間分布がガンマ分布 ($2.00, 0.45$) に従う (それぞれ形状と尺度パラメータ)。ガンマ分布のプロパティに関して、できれば 6.1.3 項におけるいくつかのウェブリンクを通して、何らかの調査をする必要があるかもしれない。
 - d) $G/M/1$ 待ち行列で、到着時間間隔分布は 1 と 5 の間の連続分布であり、サービス率は 1 分あたり $\mu = 0.4$ である (図 2.3 に示されていた同じ状況)。
15. 2 章における問題 9 の $M/D/1$ 待ち行列の解決策から、定常状態の待ち行列の理論結果について、Simio モデルを構築し、確認とクロスチェックをしなさい。Simio モデルは空かつ遊休の状態で初期化されており、それらは統計的変動を被りやすい結果を生むことを覚えておいてほしい。そして、これら両方の問題に対処するために Simio Experiments を設計して、実行しなさい。試行錯誤の後、実行期間や反復回数、そしてウォームアップ期間などを決定する。それぞれの 5 つの定常状態の待ち行列測定基準のために、最初に、2 章の問題 9 の解答から、待ち行列理論の定常状態出力パフォーマンス測定基準 ρ 、 L 、 L_q 、 W 、 W_q の数値を計算しなさい。そして、それらとシミュレーションの推定値と信頼区間で比較しなさい。また、すべての時間単位は分であり、Simio モデルも同様に分を利用すること。到着率は 1 分あたり $\lambda = 1$ とし、

112 第4章 初めての Simio モデル

サービス率は1分あたり $\mu = 1/0.9$ とする。

16. 2章の問題10のD/D/1待ち行列モデルを利用して、問題15を繰り返しなさい。
 17. 4.3節で構築したプロセスによるモデルでは、システム内時間を決定するために、標準のToken.TimeCreated トークン状態変数を用いた。到着時刻を手作業で記憶するように変更して、同様のモデルを構築しなさい（図4.13に示したが、到着時刻はシステム内時間を記録するために用いられる）。
- ヒント：上記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、そのトークンが生成された時点のシミュレーション時刻を保管するため Assign ステップを用いる必要がある。

第5章 Simio の中級モデリング

本章では、より現実的なシステムのモデルの開発・実験が可能となるよう、前章の基本的な Simio におけるモデリングと分析の概念を基に発展させる。まず、Simio の仕組みと全体的なフレームワークについて、少し詳しく議論することから始める。それから、前章で構築した単一窓口待ち行列モデルに、複数のプロセスや条件による分岐・合流などの機能を付け加える。これらのモデルを開発する際に、新しい Simio の機能を随時紹介する。また、共通の目標に関して複数の代替シナリオを比較するという点に着目し、統計的なシミュレーション実験の計画・分析方法を再考する。この章を終えると、Simio を用いて、ある程度複雑なシステムをモデル化し、分析する方法についてじゅうぶんに理解したことになる。

5.1 Simio のフレームワーク

これまでに、かなりの数の Simio 固有の概念に言及してきた。本節では、残りの節で知っておかねばならない部分を補完するため、これらの概念についてより詳細に解説する。(特にプログラミング経験のない方にとって) 本節の一部は詳細すぎると思われるかもしれないが、本章や本書の後の部分で全体が明らかになるだろう。

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、多くの専門家は、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デファクトスタンダードである。C++や Java、C#などの一般的な言語においても、オブジェクト指向がサポートされている。離散型イベントシミュレーション(DES)も同じ道をたどっている。多くの DES 製品は OOP によって開発されているが、シミュレーション担当者にとって重要なことは、シミュレーションモデルの構築プロセスに対して、本当の意味でオブジェクト指向の利点をもたらす製品は少ないことである。Simio、AnyLogic、FlexSim、Plant Simulation の 4 つの DES 製品は、モデル開発者に真の OOP ツールキットを提供するものである。

なぜこのようなことを考える必要があるのか、不思議に思われるかもしれない。OOP がソフトウェア開発業界に革命を起こしているのとほぼ同じ理由で、シミュレーション業界にも革命を起こしているからである。オブジェクトを利用することにより、大規模な問題を小さくし、より管理しやすい問題へと変えることができる。オブジェクトは、モデルの信頼性、堅牢性、再利用性、拡張性、保守性の向上に寄与する。結果として、全体的なモデリングの柔軟性と効果が劇的に改善され、必要なモデリングの専門知識の水準が低くなるケースもある。

本節の残りの部分では、Simio のオブジェクトにはどのようなものがあるか、そして重要な用語について紹介する。それから、Simio オブジェクトの特徴について、より詳細に議論する。オブジェクトという概念に初めて触れる方には、オブジェクト指向を理解し、効果的に利用することがいかに重要であるか、本節で理解されるであろう。

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、モデルは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構築される。オブジェクトは自己完結型のモデリング要素であり、その要素の特性、データ、ふるまい、ユーザインターフェース、アニメーションを定義する。オブジェクトは、モデル構築に使用されるもっとも一般的な構成要素である。Standard Library (Simio に標準で備わっている汎用オブジェクトのセット) を用いてモデルを作成した際に、すでにオブジェクトを利用している。Model 5-1、5-3 および 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保有し、それはオブジェクトの内部モデルによって定義される。たとえば、生産ラインのモデルは、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設置することにより構築される。他方、病院のモデルは、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を用いてモデル化される。Standard Object Library のオブジェクトを用いてモデルを構築することに加え、特定の応用分野向けにカスタマイズした独自のオブジェクトライブラリを作ることもできる。Standard Library にあるオブジェクトのふるまいは、プロセスロジックを用いることにより、変更したり、拡張したりできる。独自のオブジェクトを構築するための方法は、11章で議論する。

5.1.1.2 オブジェクトに関する用語

これまで、何気なく「オブジェクト」という言葉を用いてきた。ここで、オブジェクトに関する用語をより詳細に見ていく。まず、Simio オブジェクト階層構造における 3 つの層から始めよう：

- **オブジェクト定義 (Object Definition) :**

オブジェクト定義は、オブジェクトの見た目やふるまい、そして他オブジェクトとの相互作用を定義するものである。それは、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、プロジェクトあるいはライブラリの一部であるかもしれない。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、まず、Navigation ウィンドウでそのオブジェクトを選択する。その操作により、他のモデルウィンドウ（たとえば、Processes や Definitions ウィンドウ）のすべてが、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) :**

オブジェクトインスタンスは、モデルにオブジェクトをドラッグ（あるいはインスタンス化）する際に作成される。オブジェクトのインスタンスには、プロパティ値が含まれており、1 つ以上のシンボルが定義される場合もあるが、オブジェクト定義の他のすべての要素については、対応するオブジェクト定義から参照される。オブジェクトのインスタンスは、モデル内にのみ存在するが、単一のオブジェクト定義に対応する複数のインスタンスを保持することができます。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 ビューでこのオブジェクトのインスタンスを選択する。それにより、そのプロパティの値とアニメーションを変更することができる。

- **オブジェクト実行空間 (Object Runspace) :**

オブジェクト実行空間（あるいはオブジェクトの実現とも呼ばれる）は、オブジェクトの状態変数について現在値を保持するオブジェクトの第 3 層である。それぞれが固有の ID を持っており、可変のシンボルを参照することもできる。オブジェクト実行空間は、実行を開始する際に作成され、実行中に動的に作成あるいは破棄されることもある。実行が終了すると、すべてのオブジェクト実行空間は破棄される。動的オブジェクト（たとえば、エンティティ）の実行空間の状態変数は、`InstanceName[InstanceId].StateName` の様式で参照できる。たとえば、`LiftTruck[2].Priority` とすれば、2 番目の LiftTruck に関する優先順位の状態変数を参照できる。

Entity オブジェクトの 3 つの層を、図 5.1 に図示する。1 つのエンティティ定義、図の場合では人、があるとする。エンティティのインスタンスがモデルに配置される。図の場合は、大人(Adult) と子供(Child) である。モデルが実行されると、多くの大人と子供の実行空間が作成され、各々は動的なエンティティの状態変数を保持する。

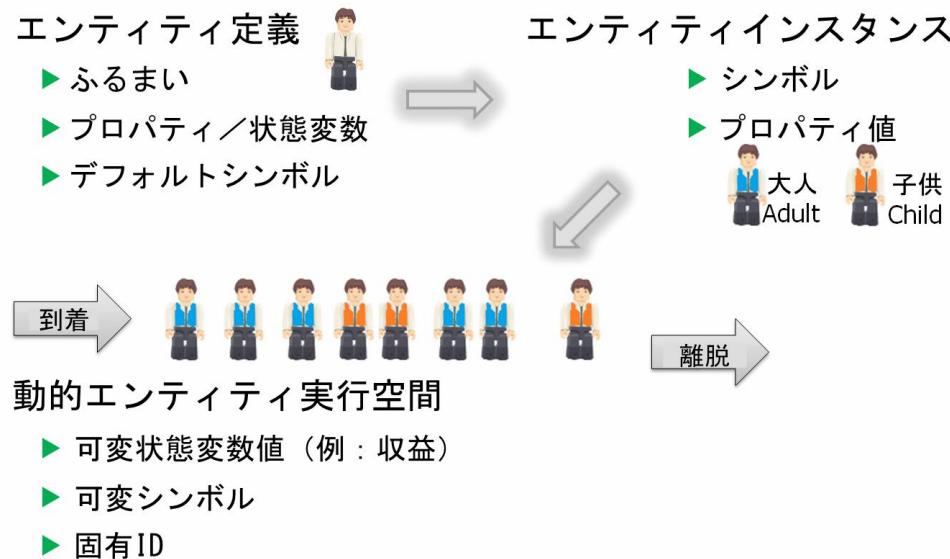


図 5.1 エンティティオブジェクトの 3 つの層

ここで、オブジェクトを議論する際に、頻繁に用いられる 3 つの用語に触れておきたい：

・ **モデル (Model)** :

モデルは、オブジェクト定義の別名である。システムあるいはサブシステムのモデルを構築しているときは、オブジェクトを構築していることになる。逆の場合もそうである。ある構成要素を記述する際にオブジェクトという用語を用いる傾向にあり（たとえば Server オブジェクト）、最上位のオブジェクトに言及するときにはモデルという用語を用いることが多い（たとえば前章の Model 4-3 における Model_04_03 という名称）。しかし、最上位のオブジェクトが他のモデルで構成要素として利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（あるいはモデル）のコレクションである。多くの場合、プロジェクト内のオブジェクトは、連携したり、階層的に動作するように設計されることにより、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の略である。プロジェクトを「モデルを構築すること」として捉えることが一般的であるため、技術的には「プロジェクト」と表現すべき事柄を「モデル」と呼ぶことが多い。本書の例題も同じような流儀で名づけられているが（例：Model 4-1）、ご理解いただけると思う。

・ **ライブラリ (Library)** :

ライブラリは、オブジェクト定義のコレクションであり、実際にはプロジェクトの別名といえる。すべてのプロジェクトファイルは、Project Home リボンの Load Library ボタンでライブラリとして読み込むことができる。

これらは、既存の Simio オブジェクトを活用するためだけでなく、独自のカスタムオブジェクトを構築するためにも、理解しておくべき Simio のキーコンセプトである。次のいくつかの章では、モデルを構築するために既存の Simio オブジェクトを使い、11 章ではカスタム Simio オブジェクトを構築する方法を扱う。技術的には、モデルを構築する際に、すでにカスタム Simio オブジェクト定義を開発しているのであるが、これらのオブジェクトは単にモデルと呼ぶことにする。オブジェクト指向のアプローチ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事実は奇妙に思われるだろうが、Simio で経験を積まれるうちに、うなずけるようになるだろう。

5.1.1.3 オブジェクトの特徴

上述のとおり、オブジェクトは、そのプロパティ、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。ここでは、各々をより詳細に説明する。

プロパティ (Property) は、モデル内にオブジェクトを設置（あるいはインスタンス化）する際に指定することのできる入力値である。たとえば、窓口を表すオブジェクトは、サービス時間を指定するプロパティを持つだろう。モデルに窓口オブジェクトを設置するときに、このプロパティ値も指定することになる。Model 5-1 では、Source オブジェクトの Interarrival Time プロパティを用いて、到着時間間隔の分布を指定し、Server の Processing Time プロパティを利用して、 $M/M/1$ 待ち行列システムの形式で窓口の処理時間の分布を指定した。

状態変数 (State) は、モデルの実行に伴って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および遊休状態は、顧客に対するサービスを開始あるいは終了するたびにオブジェクトによって変更される、Status という名称の状態変数により維持される。Model 4-2 では、システム内のエンティティ数を追跡するために、WIP というモデル状態変数を定義し、活用した。エンティティがシステムに到着したときに WIP 状態変数を増加し、エンティティが離脱する際に WIP 状態変数を減少する。また、シミュレーション時刻の任意の時点で、特定の状態変数が、全体的なオブジェクト状態変数の 1 つの構成要素となりうる。たとえば、先の单一窓口待ち行列システムでは、システム状態変数は、システム内エンティティ数 (WIP モデル状態変数) およびシステム内の各エンティティの到着時刻 (Entity.TimeCreated エンティティ状態変数) から構成される。

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。以前に、3.3.1 節の動的シミュレーションのイベントにおいて概念を紹介した。例題の単一窓口待ち行列モデルでは、エンティティの到着およびサービス完了のイベントを定義した。Simio においては、イベントはオブジェクトに関連づけられる。たとえば、窓口オブジェクトでは、窓口が顧客への対応を完了するごとにイベントが発火するだろうし、またオイルタンクのオブジェクトでは、タンクが満タンあるいは空になった際にイベントが発火するだろう。イベントは、重要な何かが起こったことを他のオブジェクトに伝えたり、カスタムロジックを組み込む場合に有用である。

外観 (External View) は、オブジェクトの 2D あるいは 3D のグラフィカルな表現である。つまり外観は、Facility ビューで目にするグラフィックスのことである。たとえば、Standard Library からオブジェクトを用いるとき、モデルに追加するとオブジェクトの外観を見られる。

ロジック (Logic) は、特定のイベントが発火した際に、オブジェクトが応答する方法を定義する。たとえば、窓口オブジェクトでは、顧客が窓口に到着したときに行うべき活動を指定することができる。この場合のロジックは、そのオブジェクトに固有のふるまいとなる。

Simio の強力な機能の 1 つとして、システムのモデルを構築するときはいつでも、いくつかの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がある。つまりモデルは、上位モデルのサブモデルとして再配置することができる。したがって、Simio では階層的なモデリングを極めて自然に行える。この点については、11 章でより詳しく議論する。

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、前項で簡単に説明したが、これらは頻繁に利用されるため、さらに議論を深めたい。

5.1.2.1 プロパティ

プロパティは、オブジェクトのふるまいをカスタマイズして、ユーザから情報を収集するため、オブジェクト内で定義される（図 5.2 参照）。ここでいうユーザは、そのオブジェクトを配置（インスタンス化）し、プロパティ値を決定する人物を指す。Standard ライブラリのオブジェクトの場合は、Simio LLC が作成したオブジェクトを用いるため、**モデル作成者自身** がユーザとなる。11 章で解説する方法により独自のオブジェクトを作成した場合には、そのオブジェクトを利用する、

オブジェクト開発者以外の人物がユーザとみなされる。



図 5.2 Definitions ウィンドウの Properties ビューにおける Properties リボン

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。このオブジェクト固有のプロパティが（もしあれば）ウィンドウの最上部に表示される。他のオブジェクトから継承されるプロパティは、他のプロパティの下の展開可能な領域に表示される。

プロパティには、次のような種類がある：

- Standard : Integer、Real、Expression、Boolean、DateTime、String…（整数、実数、式、真理値、日付、文字列）
- Element : Station、Network、Material、TallyStatistic…
- Object : Entity、Transporter、あるいは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、そのオブジェクトの Properties ウィンドウに表示される。新しいプロパティを見つけられない場合、そのプロパティを追加したオブジェクトの Properties ウィンドウが表示されているか確認してほしい（例：ModelEntity オブジェクトにプロパティを追加した際には、Model オブジェクトや Server オブジェクトには表示されない）。また、プロパティにカテゴリを設定しない場合には、デフォルトとして、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することはできない。たとえば、ProcessTime プロパティに値 5.3 を定義した場合、そのプロパティは常に 5.3 を返し、実行を止めない限り変更することはできない。しかし、いくつかのプロパティの定義には、定義自体は不变であっても異なる値を返すような、確率変数や動的状態変数を含めることができる。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) を用了。この場合、プロパティ値自体はモデル実行を通して変わらない（常にパラメータが 0.25、1、1.75 の三角分布）が、そのオブジェクトを訪れる各エンティティには、指定された確率分布からの個々のサンプル値が返される。同様に、モデル実行中に変化するような状態変数の名称をプロパティ定義として指定することにより、そのプロパティが参照される度に、状態変数の現在値が返されることになる。

また、プロパティ値は、各オブジェクトのインスタンス（つまり Facility ウィンドウに設置されたオブジェクトのコピー）に固有となる。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、モデル内に ModelEntity の 3 つのインスタンス (PartA、PartB、PartC と呼ぶことにする) を設置するとしよう。これらの各々は、Initial Desired Speed というプロパティを持つが、そのプロパティ値は別々の値とすることができます（たとえば、PartA の Initial Desired Speed と PartB のそれは必ずしも等しい必要はない）。しかし、プロパティは実行中に値を変更することができないため、モデル実行中に生成される動的エンティティ（実行空間）の多寡によらず、そのプロパティ値はインスタンスとして一度保存されたものが用いられる。

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 参照)、モデルの実行中に変化する値を保持する。状態変数の値は変化するため、「状態変数の変数」と呼ばれることがあるが、本書では「状態変数(state)」と「状態変数の変数(state variables)」は同義語として考えてほしい。状態変数は、離散と連続の2種類に分けられる。離散型状態変数(Discrete State;あるいは離散変化型状態変数)は、離散的な時点にのみ値が変化する(例:時刻 1.2457 に変化し、次は時刻 100.2 に変化する)。連続型状態変数(Continuous State;あるいは連続変化型状態変数)は、変化率もしくは加速度が 0 でない場合に、連続的かつ自動的に変化する。

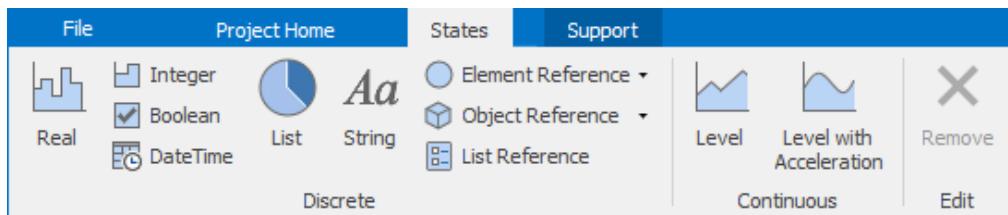


図 5.3 Definitions ウィンドウの States ビューにおける States リボン

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あり、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を指す)
- **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のいずれかを参照する)
- **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。また、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、経時的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。変化率と加速度の双方は、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ができる。

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 項目を選択することができる。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。したがって、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することができる。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することができる。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だろう。

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する（クリックする）と、選択され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。「+/-」をクリックすると、カテゴリを展開あるい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様々な種類を取りうる。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 がドロップダウンリストから選択される一方で、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した場合、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 と同じく、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がある。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 Property 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だろう（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だろう。

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 に示す通り、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。

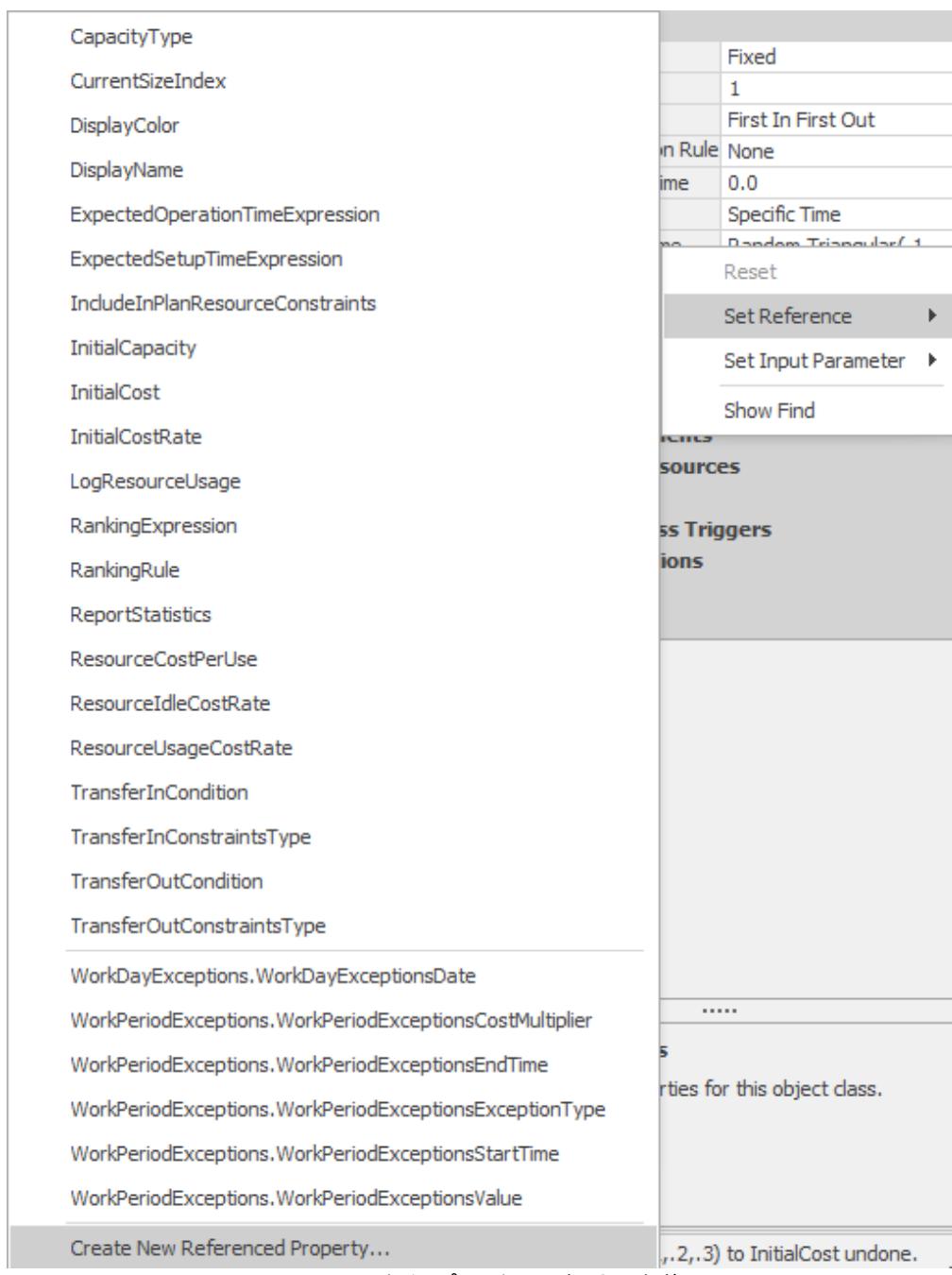


図 5.4 新規プロパティ参照の定義

プロパティ参照を用いる理由として、次の 3 つがある：

- ・ モデル作成者以外の人物がモデルを利用する際に、(Controls として) 主要なパラメータを簡単に検索・変更可能にできる。すべてのプロパティ参照は、モデルの Properties ウィンドウで Controls カテゴリとして表示される (Navigation ウィンドウで Model を右クリックし、Properties を選ぶ)。また、Definitions の Properties ウィンドウで Default Value、Display Name、Category Name やその他の事柄を変更することにより、表示の方法をカスタマイズすることもできる。
- ・ プロパティ参照は、プロパティを共有する簡単な方法である（例：プロパティで値を指定す

れば、複数の場所でそれを利用する（参照する）ことができる）。これにより、異なるプロパティ値を用いた実験が容易になる。

- Experiment ウィンドウでモデル実験を行う場合、プロパティ参照は実験の決定変数（Controls）として自動的に表示される（Visible の値を False にしていない限り）。これらの決定変数は、窓口能力の変更など、複数の代替シナリオで変化するように定義する項目である（5.5 節参照）。

プロパティ参照は、プロパティ値として利用されたとき、緑色の矢印が表示される（図 5.5 を参照）。モデルをインタラクティブに実行しているとき、コントロールの値はモデルの Properties ウィンドウで設定できる。実験を実行する際には、モデルの Properties ウィンドウでコントロールに指定された値は無視され、その代わりに、Experiment Design ウィンドウで各シナリオに対応する値が使用される。プロパティ参照の例は、本章あるいは後続の各章で扱われるモデルで見ることになる。

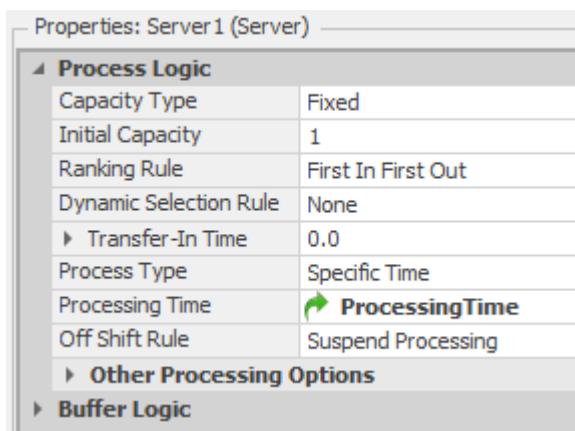


図 5.5 プロパティ参照を利用した例

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。慣れないうちはプロパティと状態変数に関して混同しがちであり、一方を他方よりもよく利用してしまうことがある点に注意してほしい。しかし、多くの機能や構成要素を用いてモデルを構築し、そこでのプロパティや状態変数の用法を議論することで、混乱は徐々に収まるだろう。

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、エンティティの概念が存在する。一般に、エンティテ

イはシステム内を動き回る部品や人間のような物理的な「モノ」を指す。しかし、エンティティが関与しない（システム制御ロジックのような）低水準のロジックや、エンティティが一度に複数のアクションをする（処理時間の進行中にイベントを待つような）場合、別の概念が必要とされる。Simio は、このような柔軟性に対してトークンを提供している。

5.1.3.1 トークン

トークン（token）は、ある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、同じプロセスの終了時に破棄される。トークンがプロセスを移動する際に、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、並行して移動する複数のトークンを保有することがある。また、単一のオブジェクトは、それを表す複数のトークンを持つことができる。

トークンは、独自のプロパティと状態変数を運ぶことがある。たとえば、トークンがロジックの特定のポイントを通過した回数を追跡するような状態変数が考えられる。ほとんどのケースでは、Simio のデフォルトトークンを利用すればよいが、独自のプロパティや状態変数を持つトークンが必要な場合は、Definitions ウィンドウの Token パネルで作成することになる。

トークンは、親オブジェクトおよび関連オブジェクトの双方への参照を持つ（図 5.6 参照）。親オブジェクトは、その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で定義されたプロセスは、その親として窓口（おそらく Server1）を示す。関連オブジェクトは、そのプロセスの実行の引き金となる（親オブジェクトとは別の）関連するオブジェクトである。たとえば、エンティティがオブジェクトに到着すること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みなされる。処理ロジックは、親オブジェクトおよび関連オブジェクトの双方のプロパティ、状態変数、関数を参照することができる。したがって、トークンは Delay ステップの時間進行を指定するために、その到着エンティティを参照することができる。関連オブジェクトを参照するには、オブジェクトのクラス名をプロパティもしくは状態変数の前に付ける必要がある。たとえば、ModelEntity.TimeCreated とすることにより、ModelEntity 型の関連オブジェクトの TimeCreated 関数の値が返されることになる。

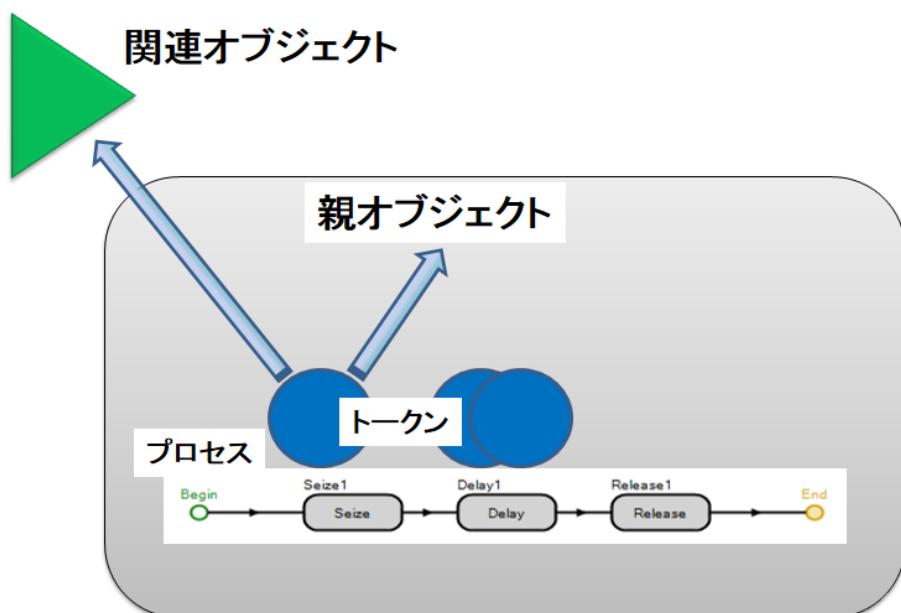


図 5.6 トークン、オブジェクトおよびプロセスの相互関係

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であり、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る。また、それらは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、動的に生成および破棄され、リンクやノードのネットワークを移動し、3D空間を移動し、固定されたオブジェクトの中へあるいは外へと移動できる。

エンティティオブジェクトの例としては、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、プロセス内を移動しない。その代わり、そのプロセスを実行させるために、エンティティの代理としてトークンが生成される。オブジェクト内外へのエンティティの移動は、イベントの引き金となり、対応するプロセスを実行されることになる。プロセスが実行されると、トークンが生成され、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っており、Free Space、Station、Link、Node のいずれかとして存在する(8章でより詳しく議論する)。エンティティの位置の種類については、エンティティの最新の位置を参照することとなる。Simio内のステーションに関する例を示す：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)。

5.1.4 プロセス

4.3節では、Processだけを用いて、全体のモデルを構築した。それは、プロセスを用いた極端なケースであった。事実、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、すべてのモデルはプロセスを用いているといえる。このことは、さらに11章でより詳しく議論する。本項は、別のプロセスの一般的な用法であるアドオンプロセスを扱う。

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あり、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、1つ以上の要素の状態変数を変更することができる。再び図5.6を参照し、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)に注目する。最初の青いトークンから出る矢は、親オブジェクト(たとえば、サーバ)とその関連オブジェクト(たとえば、このプロセス実行を開始したエンティティ)への両方の参照があることを示す。図中の他の青いトークンにも同様の参照がある点に注意してほしい。

いくつかの方法でプロセスを発火することができる：

- **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、初期化に際し、各オブジェクトに対して Simio によって実行される。
- **イベント発火プロセス**は、モデル中で発火するイベントによって発生する、ユーザによって定義されたプロセスである。
- **アドオンプロセス**は、オブジェクトのユーザが、オブジェクトの標準的なふるまいに対してカスタムロジックを挿入できるように、オブジェクト定義に組み込まれている。

ここからは、アドオンプロセスについてもう少し議論する。

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、他方、一般にそれらには重大な難点がある。オブジェクトが適用先に完全に一致するように構築されてないなら(一致することはめったにないが)、食い違いを無視するか(妥当性がない、あるいは利用不可能な結果を招く危険がある)、また

はオブジェクトを変えるか、または独自に構築することになる（ソフトウェアによっては非常に難しい場合がある）。その問題を解決するために、Simio はアドオンプロセスの概念を導入している。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することができる。アドオンプロセスのトリガーは、あらゆる Standard Library オブジェクトに含まれている。それらは、高度な柔軟性をモデル構築者に提供する。

アドオンプロセスは、他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、エンティティがその Server に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入力して Add-on Process Triggers を設定する。これらのプロセスは非常にシンプルであるが、極めて大きな柔軟性を提供する。たとえば、Simio の Server はいくつかの故障の種類をサポートしているが、いずれも修理の開始前に外部のリソース（たとえば、電気技術者）の利用可能性を考慮しない。リソースの制約を無視するか（賢明ではない）、（おそらく退屈だが）独自のオブジェクトを作成する、もしくはこの能力を実装するためにそれぞれ 1 つのステップからなる 2 つのプロセスを単純に加えることができる。Server には、修理時間を開始する前に電気技術者を得るために、Seize ステップにプロセスを加える Repairing アドオンプロセストリガーがある。また、Server には、修理が終了した後に、電気技術者を解放するために、Release ステップにプロセスを加える Repaired アドオンプロセストリガーもある。

これらのプロセスを作成することは、実際にはとても簡単である。プロセスが数個の異なるオブジェクトから共有あるいは参照される場合、Processes タブでプロセスを作成し、ロジックを完成できる。しかし、一度だけプロセスを用いるなら、もっと簡単な方法がある。上述の例に続けて、これを説明する。

新しいモデルを作成し、Source、Server (Server1)、および Sink を設置する。それらをバスで接続し、Worker (Worker1) を加える。Server1 の Failure タイプを Calendar Time Based に変更し、Uptime Between Failures に 10 Minutes を設定して、Time to Repair に 1 Minute を設定する。すべてのオブジェクトの他のプロパティは、デフォルトのまます。これで、Standard Library オブジェクトを用いて、故障を表現する簡単なモデルが完成した。それでは、修理中に Worker を活用するために Server のふるまいをカスタマイズしよう。

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、適切に命名して (Server1_Failed)、同種のプロセスに分類 (Server1 Add-on Processes カテゴリ) し、プロセスを編集するための Process ウィンドウが開かれる。
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完成する：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、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 を選択して、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。ここでもまた、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完成する：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、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 を選択して、ダイアログを閉じる。

モデルは図 5.7 のようになる。モデルを実行して、対話的な結果を見ると、サーバが 10% の故障時間を示していることに対応し、Worker の稼働時間も実行時間の 10% となっている。もちろん、修理を行なっていないときに、Worker に他のことをさせることによって、このモデルをより

興味深いものにできるだろう。タスク間で Worker を動かす方法や、処理における Worker の現在位置を考慮する方法は後に解説する。ここでは、アドオンプロセスを利用して多くの柔軟性が得られ、その設定方法がそれほど難しくないことを理解しておけばよい。

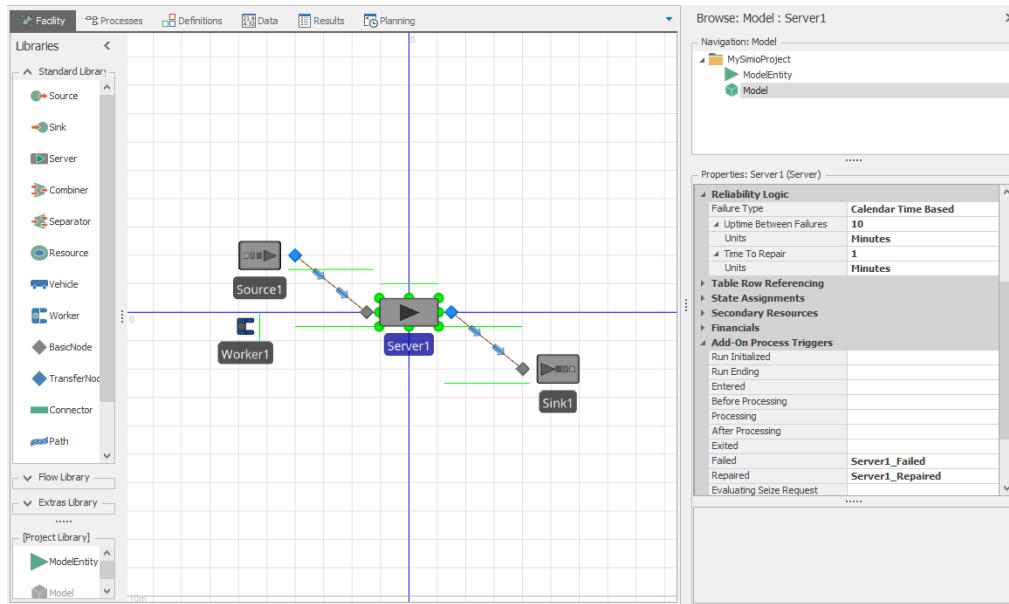


図 5.7 修理に必要なリソースを追加するアドオンプロセスの活用法

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 から特に Resource オブジェクトを用いた。Resource オブジェクトが便宜上ライブラリに提供されているが、その特定のオブジェクトに制限されていない。実際には、リソースとしてどんな Simio オブジェクトも用いることができる（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、またはリソース関連のプロパティの一部あるいはすべてを非表示にすることによって、その機能を無効にすることができることに注意してほしい）。

リソースとして利用可能なのは、以下の能力を持つオブジェクトである：

- ・ 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ための待ち行列がある。
- ・ 計画された、あるいは割り当てられた能力に対する統計量の自動収集が可能である。
- ・ 占有しようとするオブジェクトと相互作用可能である。

より詳細にそれぞれの能力を見ていこう。

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)、またはそれら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 節を参照)。カレンダに基づくスケジュールは、Data タブの Schedules ウィンドウを選択することによって定義できる。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、また休日や計画残業など、スケジュールに例外 (Exceptions) も設定できる。

柔軟性を最大限に發揮するために、リソースの能力は、Process ロジックや Assign ステップを用いて制御される。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 (たとえば、稼働中のリソースがオフシフトに移るときに発生すること) を考慮して、能力を動的に調節することができる。

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ための待ち行列がある。エンティ

ティは直接リソース能力を取らない。むしろエンティティは、リソースが能力をそのエンティティに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っている。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可能でないなら、トークン（および対応するエンティティ）は待たなければならない。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は、静的もしくは動的な順位づけで指定される。静的ランキングルールは、First In First Out、Last In First Out、Smallest Value First、および Largest Value First である。後者の 2 つのルールはよく Entity.Priority のような状態変数による 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であるが、「もっともよい」割当戦略としてリアルタイムの意思決定が必要なときは、動的ランキングのほうが優れた柔軟性を提供する。

動的ランキングルールは、割当が実施されようとするたびに、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ときには、実行速度が遅くなるかもしれない。Smallest Value First や Largest Value First に加え、Critical Ratio や 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、塗装や圧延装置を用いる産業で用いられることが多い。それらの産業では、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に、正確な工程順序が極めて重要である）。上級ユーザに対しては、独自の動的な割当ルールを開発する機能もある（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対する統計量の自動収集が可能である。また、これらは状態変数の統計量で補完することができる（たとえば、Idle、Busy、Blocked、Failed、…）。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や最小、最大の数を追跡する。標準オブジェクトの多くは、状態変数を基に詳細を追跡する List State 機能の特徴を利用している。たとえば、Resource Object は、以下の状態変数を追跡している：

- **Idle**：リソースは任意のタスクに割り当てられていない。
- **Busy**：リソースは 1 つもしくは複数のタスクに割り当てられた能力を保持している。
- **Failed**：リソースは故障し、任意のタスクに割り当てられていない。
- **OffShift**：リソースはシフトを外れており、任意の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 つもしくは複数のタスクに割り当てられており、故障中だが「Busy」状態であるとみなされる。
- **OffShiftBusy**：リソースはシフトを外れているが、1 つあるいは複数のタスクに割り当てられており、オフシフトであるが「Busy」状態であるとみなされる。

また、出力統計に加えて、モデルもしくはコンソールに、これらのデータの円グラフを表示できる。

リソースは、占有しようとするオブジェクトと相互作用可能である。この機能は多大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「賢い」リソースの概念が提供される。Simio では、リソースを占有するプロセスは 2 つのオブジェクトの間の交渉であり、占有が起こる前に双方のオブジェクトが「同意」しなければならない。リソースは、自身が占有される状況およびタイミングを選択できる。たとえば、リソースがシフト終了に近いときには、自身が占有されることを防ぐことができる。あるいは、より高い優先順位の要求がしばらく後に到着することがわかっている場合、リソースは遊休状態のままでいることを選択するかもしれない。また、リソースの現在地の近くに存在する要求のみを考慮することもあるだろう。

上述のように、この機能により、リソースに意義深い知性を組み込むことができる。割り当てに対するインテリジェントな制御は、標準プロセス OnEvaluatingAllocation にロジックを追加することによって達成される。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。この利用方法については、11 章で詳細を学ぶ。

5.1.6 データ範囲

オブジェクトは、定義された時点ではどのように用いられるのか、あるいはどのオブジェクト内に置かれるのかについて知ることができないため、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、包含するオブジェクト内で自身と同等のオブジェクトに関して何もわからない。たとえば、モデル上で定義された DayOfWeek という状態があり、モデル内のオブジェクトから直接 DayOfWeek が参照されていないとしよう。これは、プロパティの項目において、間接的に参照を許可することによって動作する。実際に、それはまさに Standard Library オブジェクトの大部分のプロパティで行わ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だけが、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ているオブジェクトがあり、それが DayOfYear という名前の状態変数を持っており、さらに DayOfYear がパブリックと定義されているなら、モデルと他のオブジェクトはそれを参照することができる。もし DayOfYear がパブリックでないなら、カレンダオブジェクトだけがその値を参照することができる。

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に、アドオンプロセスの Step が包含するモデル（たとえば、メインモデル）の情報を直接参照できることを意味している。11 章で独自オブジェクトの構築を習得するまで、データ範囲の問題は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（たとえば、Processing Time）において、Simio では複雑な式を入力する過程を簡素化するために式ビルダを提供している。4.5 節において簡単な使用方法をいくつかの例を通して紹介したが、その他の詳細についてここで説明する。式フィールドをクリックすると、下向きの矢がある小さいボタンが右に表示される。このボタンをクリックすると、右に赤色のX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「点灯」し、フィルタが自動的に適用されたことを示している。ほとんどのモデルでは、選択肢のリストが少なすぎることはないが、フィルタをオフにするとさらに選択肢が増える。

式ビルダは、マイクロソフト製品群の IntelliSense と非常に似ており、5.1.2.3 で解説したドット表記を用いて入力することで、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、必須あるいは任意のサブ項目がある。サブ項目を持っている項目には、右側の端に>>と表示されており、さらに多くの選択が利用可能であることを示している。太字でない項目は、それがまだ有効な選択でないことを示す。そこで止まると有効な式を表現できないため、サブ項目を選択しなければならない。項目自体が太字である場合、有効な選択であることを示している。それらの項目には、サブ項目がまったくないか、または任意のサブ項目しかない。たとえば、図 5.8 の Color、DateTime、DirectDistanceTo、Elements および Enum は太字でないため、有効な選択ではない。これらを用いるためには、サブ項目を選択しなければならない。他の 3 つの項目（太字のすべて）は有効な選択である。最初の 2 つ（DefaultEntity と Entity）には、矢で示されるように、任意のサブ項目がある。他の太字の項目（False）には、サブ項目がない。

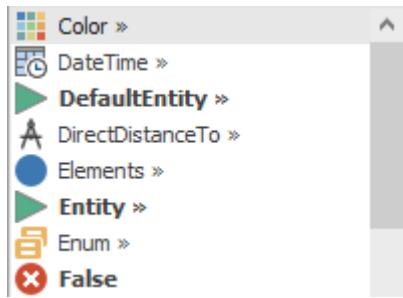


図 5.8 式ビルダ項目の抜粋

$2 * (3 + 4) ^ (4 / 3)$ 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。<、<=、>、>=、!=、==、&&、||、!のような**論理演算子**を用いて、論理式を作成することもできる。論理式は真のときは 1 の数値を返し、偽のときは 0 の数値を返す。たとえば、 $10 * (A > 2)$ は、もし A が 2 より大きければ 10 の値を返す。そうでなければ、0 の値を返す。配列（最大 10 次元）は 1 次元（最初の添字は 0 ではなく 1）であり、角括弧で添字をつける（たとえば、B[2, 3, 1]は B という 3 次元配列である）。モデルのプロパティは、プロパティ名によって参照することができる。たとえば、Time1 と Time2 がモデルに追加されたプロパティであれば、 $(Time1 + Time2) / 2$ のように、式に入力することができる。

式ビルダの一般的な使用は、確率分布を入力することである。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 ないし 4 の間で連続一様なサンプルを返す。式ビルダにこの式を入力するためには、まず「Random」とタイプする。入力すると、ドロップリストでは Random という単語までジャンプする。ピリオドをタイプすると、すべての分布名が表示される。それから、「U」をタイプすると、Uniform(min, max) にジャンプするため、Enter もしくは Tab キーを押し、式に加える。そして、パラメータ名 min および max に数値を入力する。また、リストの分布名（あるいはその他の項目）にマウスを合わせると、その分布に関する説明が自動的に表示される。式ビルダの引数リストには表示されないが、すべての分布は乱数系列を表すパラメータを最後に持っている。たいていはこの既定値のままとするが、特殊な状況では利用する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 を用いて、同様の方法でアクセスできる。まず「Math」をタイプして、利用可能な数学関数のリストを得るために、ピリオドを入力する。リスト内の数学関数をハイライトすると、その関数の説明が自動的に表示される。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することができる。

また、**オブジェクト関数**も関数名によって参照することができる。この関数は、Simio によって、内部的に維持されるか、または計算される値であり、式で用いることはできるが、代入することはできない値である。たとえば、すべてのオブジェクトは ID という関数を持っている。それは、モデル内のすべてのアクティブなオブジェクトに割り当てられているユニークな整数を返す関数である。動的なエンティティの場合は、ID として非常に大きな数を生成しないように、ID 番号が再利用されることに注意してほしい。もう 1 つの例は、リソースとして用いられるすべてのオブジェクトに、現在の状態の数値を返す ResourceState という名前の関数があることである。たとえば、Server1 が Processing の場合、式 Server1.ResourceState は値 1 を返す。ResourceState 関数をハイライト表示すると、その取りうる値が表示される。

オブジェクト関数および数学関数は、Simio ヘルプと Reference Guide (Support リボン) に列挙され、詳細が記述されている。Support リボンの Expression Reference Guide は、現在のモデルにある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 1 つは、期待費用を予測すること、または費用に基づいたシナリオ比較をすることである。いくつかの場合、あるシナリオと別のシナリオで対応する資本投資を比較することによって、これを達成できる。そのような簡単な例では、多くの場合、費用を追跡するいくつかの状態変数を用いてモデル化できるか、もしくはモデルの外部で費用を計算できる。しかし、あるシナリオと別のシナリオで運用経費を測定する必要があるケースも多い。これらの場合は、直接費用と間接費用があり、そして製品ミックスやサブシステムごとのスループット、総スループットの観点からかなり異なる生産であるだろう。これらのより複雑なケースでは、多くの場合、**活動基準原価計算 (ABC ; Activity-Based Costing)** の包括的なアプローチが役に立つ。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る。これは、より多くの費用が間接費ではなく直接費であることを認識するのに役立ち、個々の製品とサービスが様々なシナリオの下でどのように費用に貢献するかに関して、より正確な理解が同様に得られる。

シミュレーションは、ABC を実行するための理想的なツールである。それは、費用が測定されるのと同様の詳細さで、活動がモデル化されるためである。モデルは、どのリソースが各エンティティによって用いられているか、そして、それらが利用される持続時間を通常「知っている」。いくつかのシミュレーション製品では、発生した各費用を追跡するためにロジックとデータ項目を加えて、ABC を実行しなければならない。より包括的なシミュレーション製品には、ABC が備わっている。Simio は後者に分類される。

Simio Standard Library におけるほとんどのオブジェクトには、**Financials** カテゴリがある。そのカテゴリのプロパティは、オブジェクトの複雑さと機能により異なる。時間単位当たりの稼働費用 (Usage Cost Rate) と同じくらいシンプルなこともあるし、すべての費用を正確に捕捉することよりも複雑にもできる。たとえば、ほとんど（すべてではない）の作業者は次の活動を待っている場合、彼らには仕事をしていない間も賃金が支払われている (Idle Cost Rate)。あるリソースには、各エンティティを処理するたびに固定費がかかる (Cost Per Use)。また、何もしていないときにも高額な費用がかかる (Holding Cost Rate) 非常に高価なリソースやエンティティもある（飛行機など）。多くの場合、費用を正確に計るのは簡単ではないが、費用を適切な部門へ配賦する、あるいは配分する必要がある (Parent Cost Center)。

図 5.9 では、Server と Worker の双方にリソース費用を計算するためのよく似たプロパティがあるのに気づくだろう（たとえば、Server あるいは Worker の占有および保持費用、仕事が予定されている間に活動していない費用）。また、Server には、入出力バッファで待っている間に生じる費用を計算するプロパティがある。Worker オブジェクトには、バッファ費用はない。その代わりに、エンティティを運ぶたびに課される費用や、Worker が移動している間にかかる運搬費用がある。あるオブジェクトの 1 つのインスタンスが、これらのすべての費用を持つことはまれである。典型的には、利用ごとの費用が課されるリソースがあれば、稼働および遊休時の原価率を持つリソースもあるだろう。

図 5.9 の左側は自動化された収縮包装機 (Server) を表している。それは、33,500 ユーロの資本費用が必要であり、1 回当たりの使用で 13.852 ユーロの固定費かかる。収縮包装機の費用は、ShippingCenter (発送センター) コストセンターに配賦される。図 5.9 の右側は、稼働しているか否かに関わらず毎時 20US ドルを支払うフォークリフト作業者 (Worker) を表している。作業者が何かを輸送するためにフォークリフトを用いるとき、輸送機械の費用として毎時 31.50US ドルが余分にかかる。フォークリフト作業者の費用は、Transportation (輸送) コストセンターに配賦される。

The screenshot shows two side-by-side property panels in Simio:

- Properties: ShrinkWrapper (Server)**
 - Financials**: Parent Cost Center is set to **ShippingCenter**. Capital Cost is **33500** (Units: EUR).
 - Buffer Costs**: Input Buffer Cost Per Use is **0.0**, Holding Cost Rate is **0.0**.
 - Resource Costs**: Idle Cost Rate is **0.0** (Units: **USD per Hour**), Cost Per Use is **13.852** (Units: EUR), Usage Cost Rate is **0.0** (Units: **USD per Hour**).
 - Add-On Process Triggers**: Run Initialized.
- Properties: ForkLiftOperator (Worker)**
 - Financials**: Parent Cost Center is set to **Transportation**. Capital Cost Per Worker is **0.0** (Units: **USD**).
 - Transport Costs**: Cost Per Rider is **0.0** (Units: **USD**), Transport Cost Rate is **31.50** (Units: **USD per Hour**).
 - Resource Costs**: Idle Cost Rate is **20** (Units: **USD per Hour**), Cost Per Use is **0.0** (Units: **USD**), Usage Cost Rate is **20** (Units: **USD per Hour**).
 - Add-On Process Triggers**: Population.

Below each panel are detailed descriptions of the selected properties:

- Capital Cost**: The initial one-time setup cost to add this object to the system.
- Parent Cost Center**: The parent cost center that costs charged, or accrued, to workers of this type are rolled up into. If a parent cost center is not explicitly specified, then costs accrued to a worker will be automatically rolled up into the...

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

最後に、多国籍の企業やプロジェクトは多くの場合、世界にまたがって仕事が行われ、複数の通貨が関わることを指摘しておこう。Simio はほとんどの一般的な世界の通貨を扱うことができる（ゴートやビットコインは扱っていないが）。Advanced Options の Run タブにある Financials カテゴリで、レポートで用いたいデフォルト通貨を指定することができ、また利用する通貨に対する為替レートも設定できる。

5.2 Model 5-I : PCB 組立

さて、Simio のフレームワークをよりよく理解したので、モデリングと分析に焦点を戻そう。本章のモデルは、プリント基板（PCB）用の簡易組立作業に関するものである。PCB は事実上すべてのエレクトロニクス製品において主要な構成部品である。PCB 組立は、基本的に構成部品間の電子シグナルの伝達を容易にするように設計された特別な基板に各種の電子部品を実装する必要がある。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）。一般に、PCB 組立は複数の連続した加工工程、検査、再加工、包装作業を伴う。本章を通して、これらの工程をモデルに順次追加するが、まずは 4 章の単一窓口待ち行列モデルを拡張することから始める。

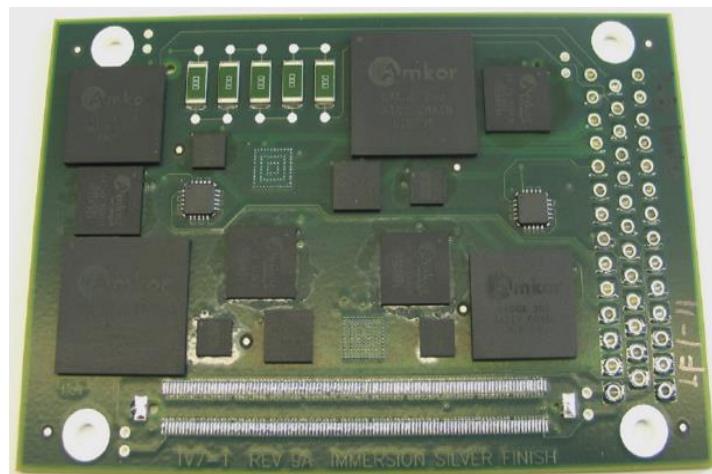


図 5.10 プリント基板の例

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、検査する作業に焦点を当てる。基板は、毎時 10 枚の速度で上流工程から部品配置に到着する（当面、到着過程はポアソンであると仮定する。すなわち、到着時間間隔は指数関数に従う）。部品配置機械の処理時間は、三角分布(3, 4, 5)分である。部品配置の後に、基板は、すべての構成部品が正しく配置されたことを確認するために検査される。検査員が検査を行い、その検査時間は 2 ないし 4 分に一様に分布する。過去の記録では、検査された基板の 92% が「良」、残りの 8% が「不良」であることが判明している。到着時間間隔、部品配置、および検査時間はすべてランダムであるので、いくつかの待ち行列が生じることが予想されるため、部品配置と検査員の待ち行列長と稼働率を推定したい。また、基板がこの小規模なシステムで費やす時間と待ち時間を推定し、さらに、良品および不良品と判定された基板の数を数えたい。

この時点でモデル構築に移ることができるが、(4.2.5 項で述べたとおり) 関心のあるパフォーマンス尺度を推定するために、モデルを用いる前に、モデルを検証しておく必要があるため、ここでシステムに関する期待値を予測しておこう。基本的な検証プロセスは、期待値とモデルの結果との比較を伴うことを想起してほしい。図 5.11 は、最初の 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 ($\lambda = 10$)、各サービス率 ($\mu_p = 15$ および $\mu_i = 20$) を所与とする場合 (すべての単位は 1 時間当たりの基板枚数)、定常状態の正確な窓口稼働率を計算することができる (稼働率は、特定の到着時間間隔および処理時間の分布から独立している)。 $\rho_p = \lambda / \mu_p = 0.667 < 1$ であるため、検査員への到着率はちょうど毎時 10 基板であり (なぜか?)、したがって、 $\rho_i = \lambda / \mu_i = 0.500$ であることが分かっている。サーバ稼働率の双方が厳密に 1 より小さいので、システムは安定している (到着するより早くエンティティを処理できる) ため、システムのエンティティ数は厳密に有限となる。すなわち、システムが永遠に増え続ける基板で「爆発」することはない)。部品配置と検査時間が指数関数に従うと仮定した場合、システム内の基板の期待値 $L = 3.000$ と、基板がシステムで費やす期待時間 $W = 0.3000$ 時間を計算することができる (詳細は 2 章を参照)。もちろんどちらも定常状態である。一方、時間が指数関数的に分布していない場合、4 章で述べたように、検証するために最初からシミュレーションモデルを構築することは簡単である。

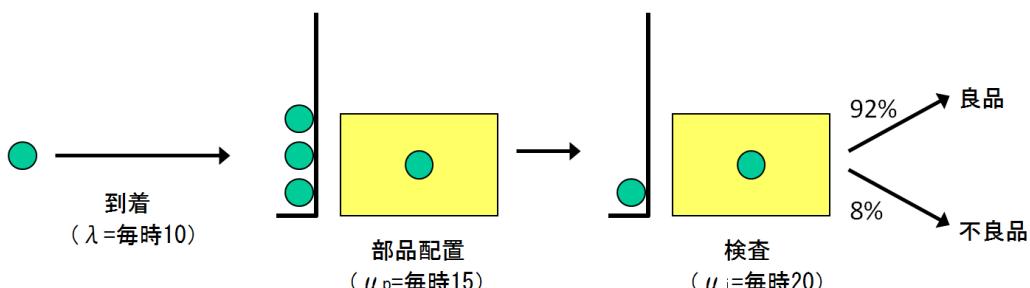


図 5.11 Model 5-1 の待ち行列モデル

期待値が判明したため、これで Simio モデルを構築することができる。4 章のモデルと比べて、Model 5-1 には 2 つの基本的な拡張が含まれている：(部品配置から検査に続く) 連続したプロセスと (検査の後に、基板は「良」もしくは「不良」に分類される) 分岐である。これら 2 つの拡張機能は Simio で広く用いられ、非常に簡単に利用できる。図 5.12 に完成した Simio モデルが示されている。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)、および 2 つの Sink オブジェクト (GoodParts と BadParts) から構成される。

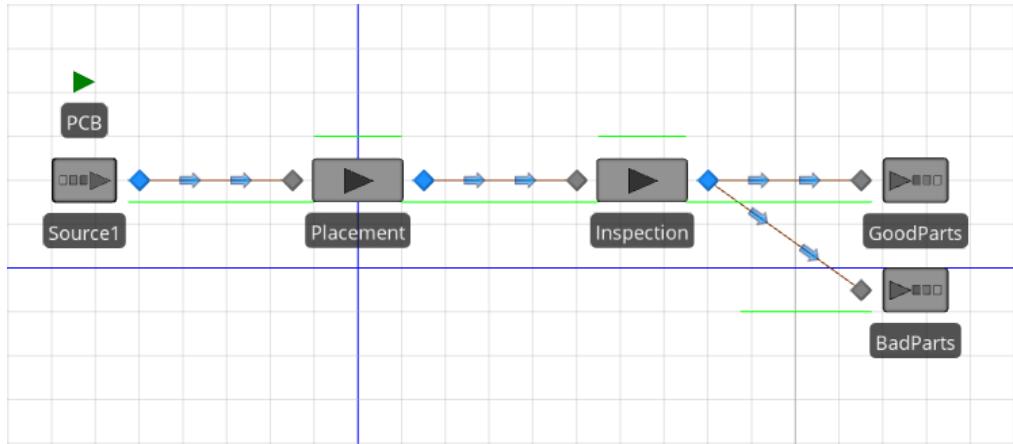


図 5.12 Model 5-1

初期モデルでは、オブジェクトを接続し、エンティティの流れを指定するため、Connector（コネクタ）を用いた。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、ゼロのシミュレーション時間でエンティティを移動させることを想起してほしい。連結したプロセスは、単に 2 つの Server オブジェクトを接続することによって、モデル化される（Placement オブジェクトを離れる基板を Inspection オブジェクトに移動させる）。分岐は、2 つのコネクタを同じ出力ノードから伸ばすことによって、モデル化される（検査を離れる基板をシンク GoodParts かシンク BadParts のどちらかに移動させる）

特定のエンティティがどの分岐を選択するかを決める方法を設定するには、出力ノードの Routing Logic に指定しなければならない（図 5.13 は Model 5-1 の設定である）。ここでは、Inspection オブジェクトの出力ノードが選択され、Routing Logic グループの Outbound Link Rule プロパティが By Link Weight に設定されている。これは、各リンクを選択する相対的な確率を決定するために、コネクタの Selection Weight プロパティを用いて、ランダムに出力リンクの 1 つを選ぶように Simio に指示している（エンティティに目的地セットが設定されている場合、Shortest Path がデフォルトのルールとなる。ルールが Shortest Path に設定されていたとしても、エンティティの目的地が指定されていない場合は By Link Weight が用いられる）。

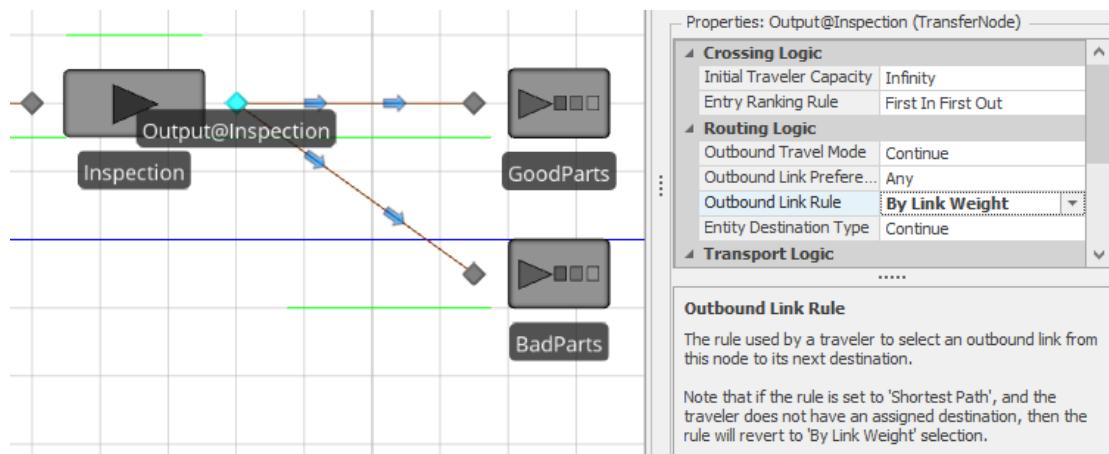


図 5.13 出力ノードの Routing Logic

このモデルの仕様では、検査された基板の 92% が良品であるので、GoodParts オブジェクトのリンクに対して 0.92、BadParts オブジェクトのリンクに対して 0.08 と、Selection Weight プロパティを設定する（図 5.14 参照）。このリンク重みづけのメカニズムは、それぞれの出力リンクに相対的なリンクの重みを割り当てるだけで、n 種類のランダムな分岐を簡単にモデル化することができる。ここで、リンクの重みに対して、92 と 8 または 9.2 と 0.8、あるいはその他の割合（た

とえば、5 単位の小麦粉、2 単位の砂糖、1 単位のふくらし粉などを配合するケーキのレシピ) のいずれも利用することができる点に留意してほしい。Simio は、エンティティが各経路に沿って送られる割合を決定するためだけに、指定されたリンクの重みを用いる。Simio には、基本的な Routing Logic を指定するために、他にもいくつかの手法がある。以降のモデルでこれらを紹介するが、ここではランダムセレクションのみを用いる。

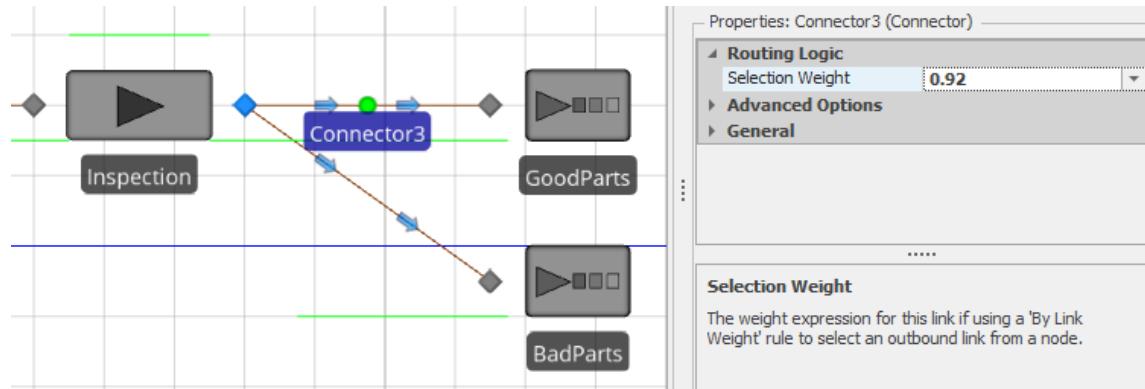


図 5.14 Inspection と GoodParts オブジェクト間のリンクに対する重みづけ

Model 5-1 を完成するための最後のステップは、4 章のモデルでしたように、Interarrival Time と Processing Time プロパティを用いて、到着時間間隔とサービス時間の分布を指定することである。はじめに、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できるよう、処理時間が指数分布に従うように設定する。表 5.2 に、シミュレーション期間 1,200 時間、200 時間のウォームアップ期間における、25 回の反復実行の結果を示す。結果はすべて、4.2 節で説明した Pivot Grid レポートから読み取った。これらの結果は、期待値の信頼区間にあるため、モデルが期待通りに動作していることを示す強固な証拠となる(すなわち、検証されたといえる)。このように、処理時間の分布を指定された値に変更し、モデルを完成することができる。

表 5.2 Model 5-1 の初期結果 (指標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 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・不良品の割合はあまり変化しなかったが、エンティティ数 (L) および滞在時間 (W) についてシミュレーションで生成された推定値が共に小さくなったことに注意してほしい。実際の数値は変化したが、**推定値**は確率的な観点からは変化しなかった。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、指標分布から三角分布あるいは一様分布に変更した際、(どちらの分布も、指標分布とは異なり、右側に有界であるので) 双方のプロセスにおいて分散が減少するため、 L および W の減少も予測できることに留意してほしい。

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加えるために、PCB 組立モデルを拡張する。具体的な追加機能は、以下のとおりである：

- **再加工 (Rework)**：検査済みの基板は、良品もしくは不良品のどちらかで分類されている。ここでは、検査済み基板のうち 26%は、専用の再加工ステーションで再加工が必要であると仮定する。そのステーションでは、作業員が配置された部品を基板から取り外す。これらの再加工が必要な基板は、前節では良品と見なしていたものから分類される。したがって、本節のモデルにおける検査後には、66%が良品、8%が不良品、そして 26%が再加工となる。再加工の処理時間は、三角分布(2, 4, 6)分に従うと仮定する。再加工後、基板は再処理するために部品配置に送り返される。再加工された基板は、部品配置に新たに到着した基板と同様に扱われ、続く検査においても同じ確率で良・不良・再加工の判断がなされる（つまり、基板が再加工と判断される回数に制限はなく、部品配置と検査に何度も戻される可能性がある）。
- **作業者のスケジュール (Worker Schedules)**：検査と再加工の処理は、どちらも人間の作業員が担当するため、作業員が働く就業時間には「食事休憩」も含めて考慮したい。また、検査は操業時間中ずっと行われているが、再加工は「2 番目のシフト」の間のみ行われる。部品配置は自動化されていて、作業員を全く必要としないものとする。
- **機械故障 (Machine Failures)**：部品配置は自動化された工程であり、修理が必要な故障がランダムに生じる。

このモデルに作業者スケジュールの概念を導入したので、ここで明確に**操業時間**を定義しければならない。Model 5-2 に関しては、24 時間操業であるとし、600 日連続してモデルを実行する。

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を追加するために、新しい Server オブジェクト Rework を追加する。Processing Time と Name プロパティを設定して、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、検査を離れる 3 種類の分岐を作る（リンクの重みも更新する必要がある）。再加工された基板は、再処理のために部品配置に送り返されるので、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。ここでは、Placement オブジェクトに入る別々のエンティティの「流れ」が 2 つある。ひとつは Source1 オブジェクト（新たに到着する基板）からで、もうひとつは Rework オブジェクト（一度処理されており、検査で不合格となり再加工された基板）から入るものである。Source1 オブジェクトあるいは Rework オブジェクトから到着したかどうかに関わらず、Placement オブジェクト（そして、その入力ノード）では同じように到着エンティティを扱う。後で、到着エンティティと状態変数に応じて、異なる扱いをする方法も述べることにする。

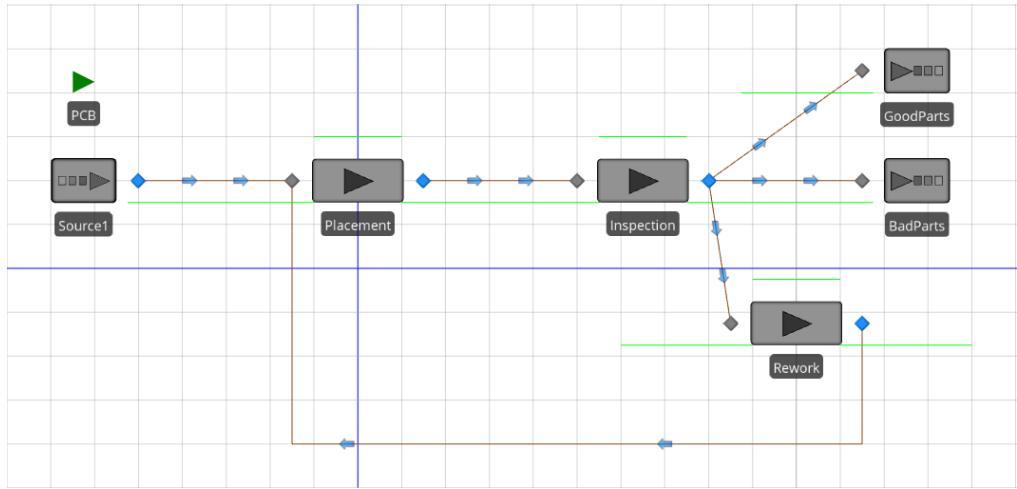


図 5.15 Model 5-2

システム（そして、モデル）の構造からわかる通り、部品配置または検査工程では新しいものと再加工された基板を区別しないので、基板は複数回再加工される可能性がある。その上で、各基板が部品配置で処理された回数を追跡し、その値の統計量を知りたい（基板が再加工された回数を追跡する、より簡単な方法を選ぶこともできた。この場合、最小値は 1 ではなく 0 となる）。これは、関心のある値に対するユーザ定義統計量の例であるが、Simio では自動的に提供されない。この機能をモデルに追加するには、以下の手順が必要である：

1. エンティティが部品配置で処理された回数を追跡するために、エンティティの状態変数を定義する。
2. エンティティの部品配置での処理が終わったときに、その状態変数值を増加するロジックを加える。
3. 実行における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（良品もしくは不良品として）システムを離れるときに、タリー値を記録するロジックを加える。

これは 4.3 節でモデル化した *TimeInSystem* 統計値と同様であることに留意してほしい。これらは、観測統計量（または離散時間過程）の例であり、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、部品配置工程を通過したエンティティの回数であり、Model 5-2 では、エンティティのシステム内時間である。そして、タリー統計量は、すべての観測値（どちらの場合もエンティティ）の平均値や最大値のような要約統計量を報告する（ここには標準偏差は適さない。たとえ、その計算が簡単にできそうであっても偏っており、正しい結果をもたらさない。理由は 3.3.1 項を参照のこと）。

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に伴って変化する値を蓄積する（5.1.2.2）。これは、エンティティが部品配置工程を通過した回数を追跡するのに必要なものである。エンティティが工程を通過（離脱）するたびに、状態変数に蓄積された値を増加させる。この場合、エンティティに関連する状態変数を定義する必要があり、Navigation ウィンドウで *ModelEntity* を選択して、*Definitions* タブを選択し、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 参照）。

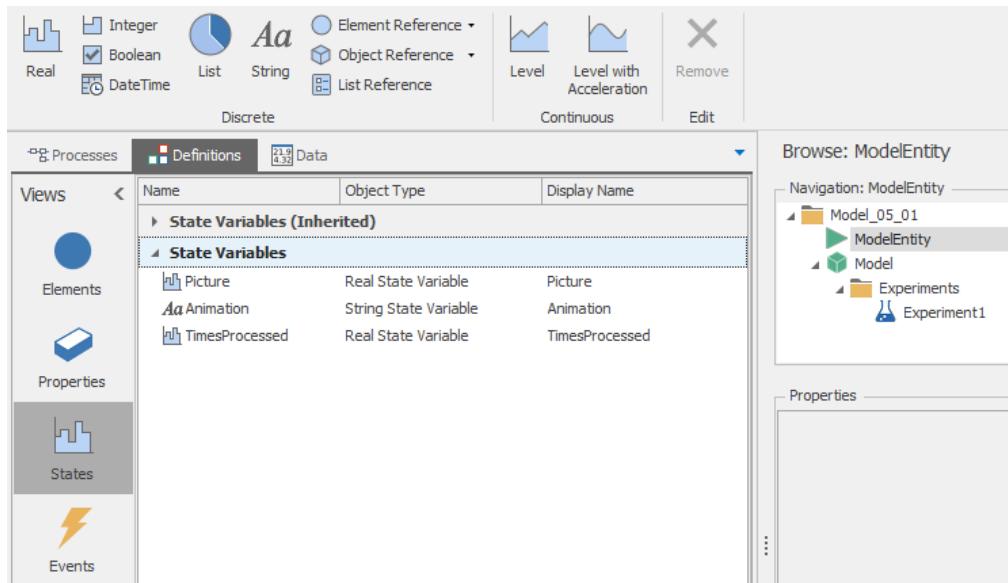


図 5.16 エンティティ状態変数 TimesProcessed の作成

このモデルでは、状態変数に TimesProcessed と命名した。タイプが ModelEntity のあらゆるエンティティオブジェクトは、TimesProcessed という状態変数を持つことになり、シミュレーション実行に伴い、この状態変数の値が更新される。このモデルの場合、部品配置工程の終わりで値を増加する（現在の値に 1 を加える）必要がある。Simio でこのロジックを実装するには、いくつかの方法がある。このモデルでは、Server オブジェクトに関連付けられた *State Assignments* プロパティを用いる。このプロパティで、エンティティがサーバに進入あるいは退出するときに、1 以上上の状態変数を割り当てることができる（図 5.17 参照）。

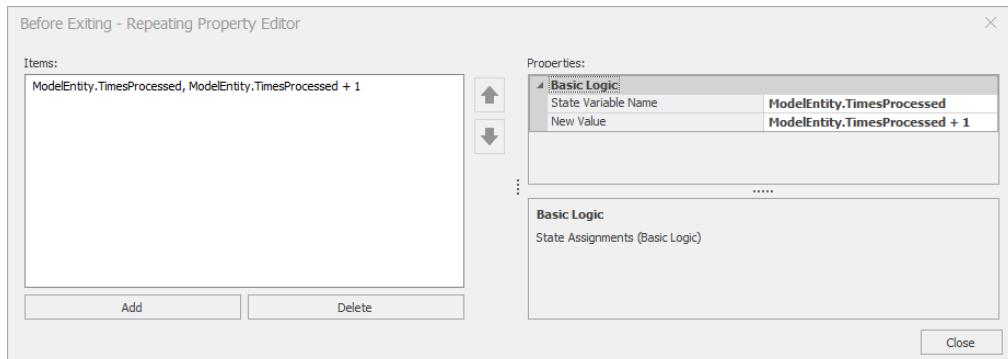


図 5.17 Placement オブジェクトの State Assignments プロパティの設定

ここでは、Placement オブジェクトを選択して、State Assignments セクションの Before Exiting リピートグループを開き、状態変数 TimesProcessed の値を増加するように状態変数の割り当てを追加した。これは、エンティティがオブジェクトを離れる直前に、指定された状態変数のセット（この例では、単一の状態変数）を割り当てるように Simio に設定している。このように、任意のシミュレーション時刻において、エンティティの状態変数 TimesProcessed は、エンティティが Placement オブジェクトによって処理を完了した回数を示すこととなる。

次に、エンティティがシステムを離れる直前に、観測（タリー）統計量として状態変数 TimesProcessed の値を追跡したいことを Simio に指示する。これは 2 段階の手順で行う：

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 に蓄積されている）値を記録する。

タリー統計量を定義するには、まず、Navigation ウィンドウで Model を選択し、Definitions タブをクリックする。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 参照)。ここでは、Name プロパティを入力して、タリー統計量を NumTimesProcessed と命名した。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用いて、状態変数 TimesProcessed の値をタリー統計量に記録する方法と、Sink オブジェクトの入力ノードの Tally Statistics プロパティを用いて記録する方法である(通常はモデル内でどちらかの方法を使用するが、ここでは両方の方法を説明する)。

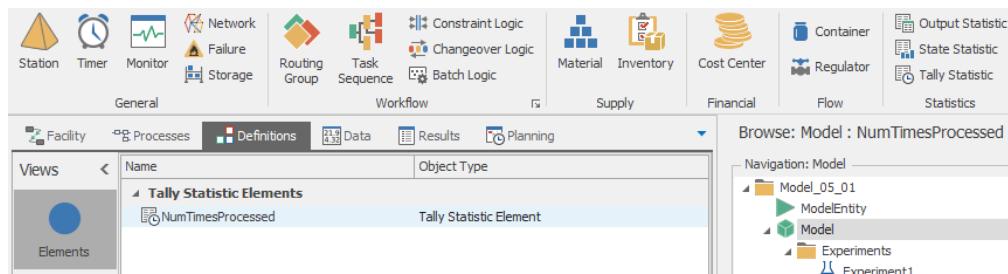


図 5.18 タリー統計量 NumTimesProcessed の作成

最初に、Sink オブジェクト GoodParts へ向かうエンティティでアドオンプロセスの方法を用いる。そのプロセスはエンティティが Sink オブジェクトへ入るときに、実行される。アドオンプロセスを作成するために、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。そこでは、Run Initialized、Run Ending、Entered、Destroying Entity の 4 つのトリガーが定義されている。ここでは、それぞれのエンティティの値を記録したいので、Entered トリガーを選択する。アドオンプロセスには好みの名前を入力できるが、アドオンプロセスを定義するための簡易な方法がある。トリガーのプロパティ名(この場合、Entered)をダブルクリックするだけで、Simio が適切な名前を定義し、新たに作成されたプロセスを選択した状態でプロセスパネルが開くのである。この場合のアドオンプロセスは、適切なタリー統計値にタリー値を記録するだけであり、单一ステップの簡単なものとなる。これには、Tally ステップを用いる(図 5.19 参照)。

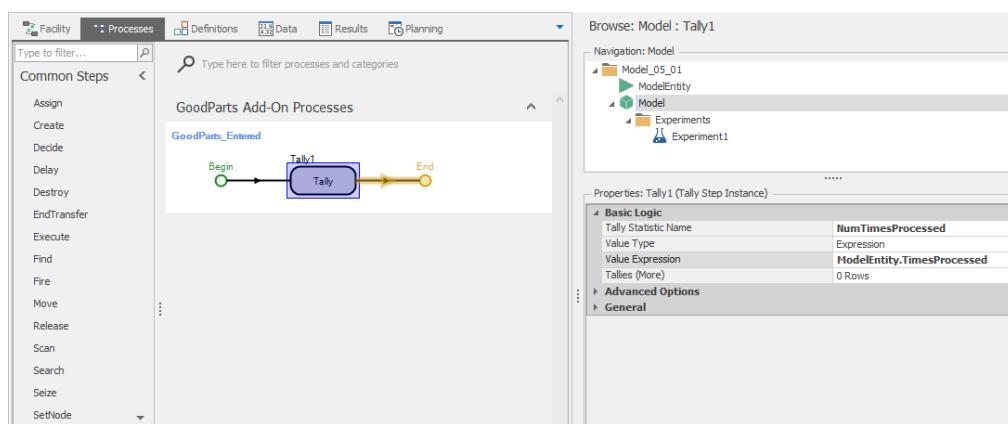


図 5.19 Sink オブジェクト GoodParts のアドオンプロセス

ステップを加えるためには、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 ノードの間に設置するだけである。Tally ステップにおける重要なプロパティは、TallyStatistic Name である。そこでは、以前に定義された Tally Statistic (NumTimesProcessed) と Value を指定し、記録される値(この場合、エンティティ状態変数 TimesProcessed)を設定する。また、評価あるいは記録するために式を用いることを示すために、

Value Type プロパティに Expression を設定する。

二つ目の方法は、Standard Object ライブラリから Basic Node の Tally Statistics プロパティを用いるもので、この方法を Sink オブジェクト BadParts で利用する。図 5.20 はこの方法を示している。

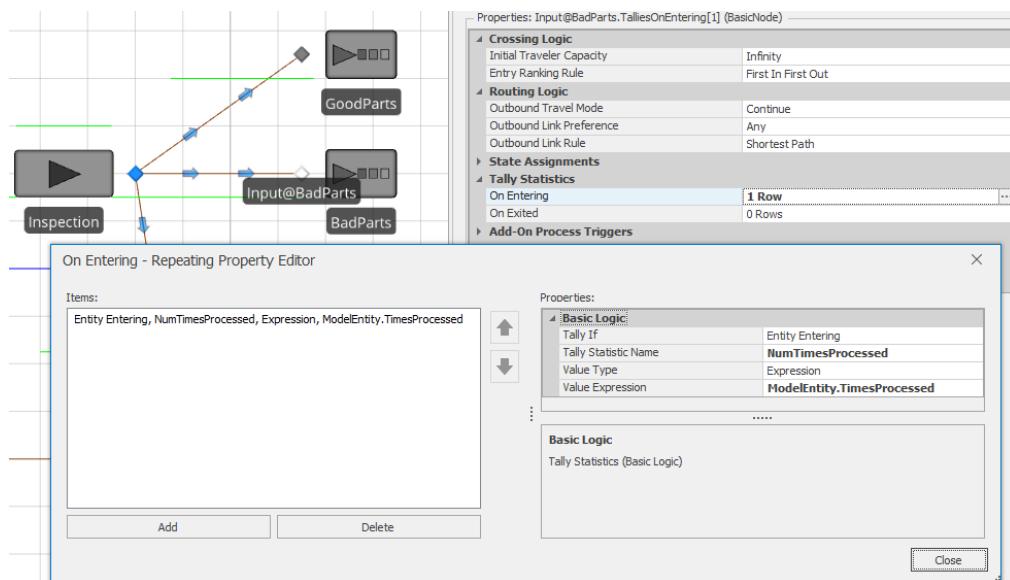


図 5.20 Sink オブジェクト BadParts における Tally Statistics プロパティ

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（図の上部に示されている）。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、図の下の部分に示すようにタリー統計量を追加する。すべての部品に対して処理された回数を追跡する必要があるので、両方の部品タイプで同じタリー統計量を使用することに注意してほしい。部品タイプ（Good と Bad）で統計量を別々に追跡したいならば、単純に 2 つの異なるタリー統計量を使用する。モデルを実行すると、標準の Simio Pivot Grid と Reports でタリー統計量が表示され、プロセスロジックで値が利用可能になる。

ユーザ定義の観測統計量を作成するためにこれまで説明した内容は、頻繁に用いられる一般的なプロセスの具体例である。ユーザ定義の観測（タリー）統計量を作成するための一般的なプロセスをまとめると、以下の通りである：

1. 観測する式を特定する。しばしば、これには（前例でしたように）観測される式を計算して格納することのできる、新しいモデルやエンティティ状態変数の定義を伴う。
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（すでにモデルの一部として構築されていない場合）。
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 ステップを用いて式の値を記録すること、またはノードオブジェクトインスタンスの Tally Statistics プロパティを用いることを設定する。

ユーザ定義の観測統計量を組み込むには、すべてこれと同じ一般的なプロセスをたどる。

5.3.2 リンクの重みに対する式の利用

Model 5-2 を続ける前に、コネクタのリンクの重みに対して式を利用する方法を説明したい。これは、リンク選択に関するロジックを簡単にできるかなり強力な機能であるが、多くの場合見落

とされている。上の例では、3つの可能性がある検査結果（良品：66%、再加工：26%、不良品：8%）に定数の確率を指定して、それに従ってリンクに重みを設定した。ここで、基板を処理する回数を制限したいものとしよう。たとえば、次のようなルールがあるものとする：

- ・ 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。

そして、モデルにこのロジックを実装する。この場合、リンク選択は完全に確率的であるというわけではない。基板が不合格（3回処理されたもの）かどうかを決定する条件はあるが、その条件が満たされない場合には、残りの2つの選択肢を選ぶために確率が用いられる。図5.21は、リンクの重みに対して式を用いて、このロジックを実行する簡単な方法を示している。括弧内の要素でリンクの重みが0あるいは1と評価されるため、基板の加工回数が3回未満の場合のみ確率が適用されることに注意してほしい。3回目の加工の後に、基板は不合格となる。

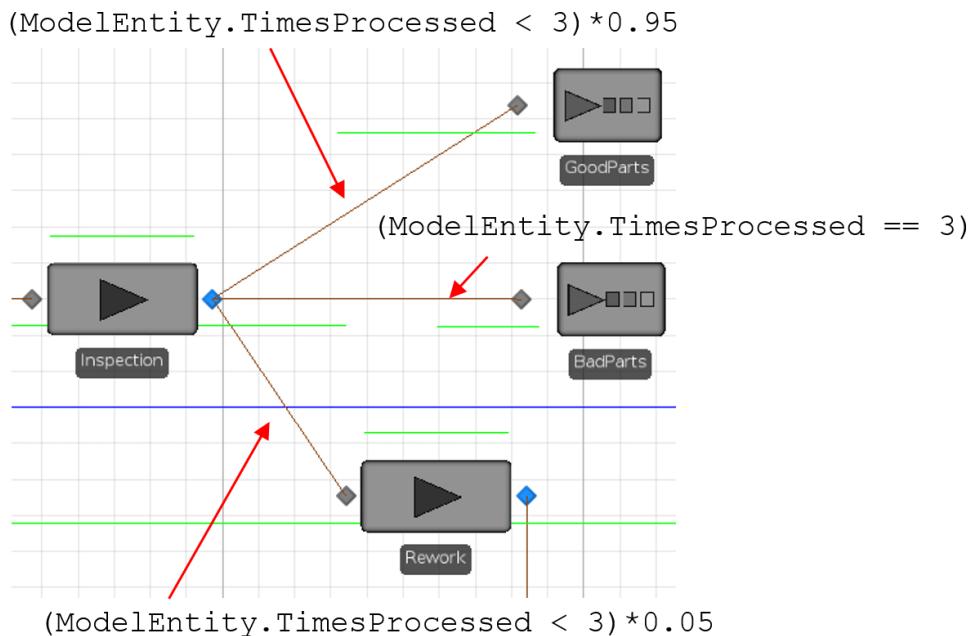


図5.21 リンクの重みに対して式を用いた例

この例で、リンクウェイトの選択ルールで式を用いることのよさを説明する。初心者は多くの場合、この種のロジックを実行するために、不必要に、より複雑なアドオンプロセスを用いようとするだろう。ここで用いたような式は、モデル開発をかなり容易にするため、この例をしっかりと検討されることを推奨する。もちろん、実際にこのシステムを実行するなら、基板が3回処理される前に不合格にしたいだろう。すなわち、2回目に不合格となった検査の直後で、再加工される前に判断したい。この問題は、読者の演習問題とする（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によって行われる（部品配置工程は自動化されている）。作業員は、一般にシフトで働き、操業時間中に休憩を取る。Simioでの作業者スケジュールの実装方法を説明する前に、Simioがシミュレーション時刻に従って1日の時間を追跡する方法について、初めに説明しなければならない。Simioの実行は、デフォルトでは、各「日」が（深夜）12時に始まる24時間の時計に基づいている。これまでのモデルでは、反復実

行期間とウォームアップ期間に焦点を合わせたが、実行がどのように1日24時間に分割されるかについて議論していない。

PCBの例では、施設は1日につき3回の8時間勤務シフトで操業しており、各シフトには1時間の食事休憩が含まれるものとする（したがって、1日の勤務時間は、実際には食事休憩を含む9時間である）。シフトは以下の通りである。

- ・ 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、複数のシフトの工程（この例では検査工程）では、作業員の最初の1時間は、事務処理を行い、シフトの準備に費やされると仮定する。したがって、作業員のシフトが重複する期間において、実際に検査作業を実施する作業員は1人である。

上述の作業スケジュールを考えると、検査工程は1時間の食事休憩を3回（正午～午後1時、午後8時～午後9時、午前4時～午前5時）含む1日24時間で操業されている。このスケジュールは、それぞれが1日当たり8時間ずつ働く3人の検査作業員がいることに対応している。Simioは午前12時に始まる24時間の時計に基づいているため、このスケジュールは表5.4に示すような検査工程のリソース能力スケジュールに変換される。

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、検査員がまったくいなくなる3回の各「食事休憩期間」に、検査ステーションの前に基板の待ち行列が生まれることが予想される。いったんモデルを完成すれば、アニメーションでこのような待ち行列の挙動を間違いなく目にするだろう。この種のリソース能力スケジュールをSimioに実装するには複数の方法がある。Model 5-2では、Simioに内蔵されたScheduleテーブルを利用する。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、それからInspectionオブジェクトに対して、当該オブジェクトのリソース能力を決定するために定義したスケジュールを用いるように設定する。

作業スケジュールを作成するために、NavigationペインでModelを選択し、さらにDataタブを選択して、DataパネルのSchedulesアイコンを押す。そこには2つのタブがある。Work Schedulesタブの下には、StandardWeekと命名されたサンプルスケジュールがすでに作成されており、Day Patternsタブの下には、StandardDayのサンプルが作成されている。

Day Patternsタブで、1日の能力の値をSimioに設定して、毎日の作業サイクルを定義することから始める。サンプルパターンを修正するか、置き換えるかを選ぶことができる。ここではStandardDayを選択し、Removeをクリックして、デフォルトのパターンを削除する。次に、リボ

ンの Create カテゴリで Day Pattern をクリックして、新規のパターンを作成する。新しいパターンに DayPattern1 の Name をつけ、名前の左側の「+」をクリックして Work Period (作業期間) の定義を展開する。最初の作業期間の行では、Start Time に 12:00:00 AM と入力あるいはリストを利用して設定し、Duration (持続期間) に 4 と入力する。End Time が自動的に計算され、Value (リソースの能力) のデフォルトが 1 となっていることを確認する。この手順を続けて、表 5.4 に示された残り 3 つの実作業期間を追加する。完成した 1 日のパターンは、図 5.22 のようになる。

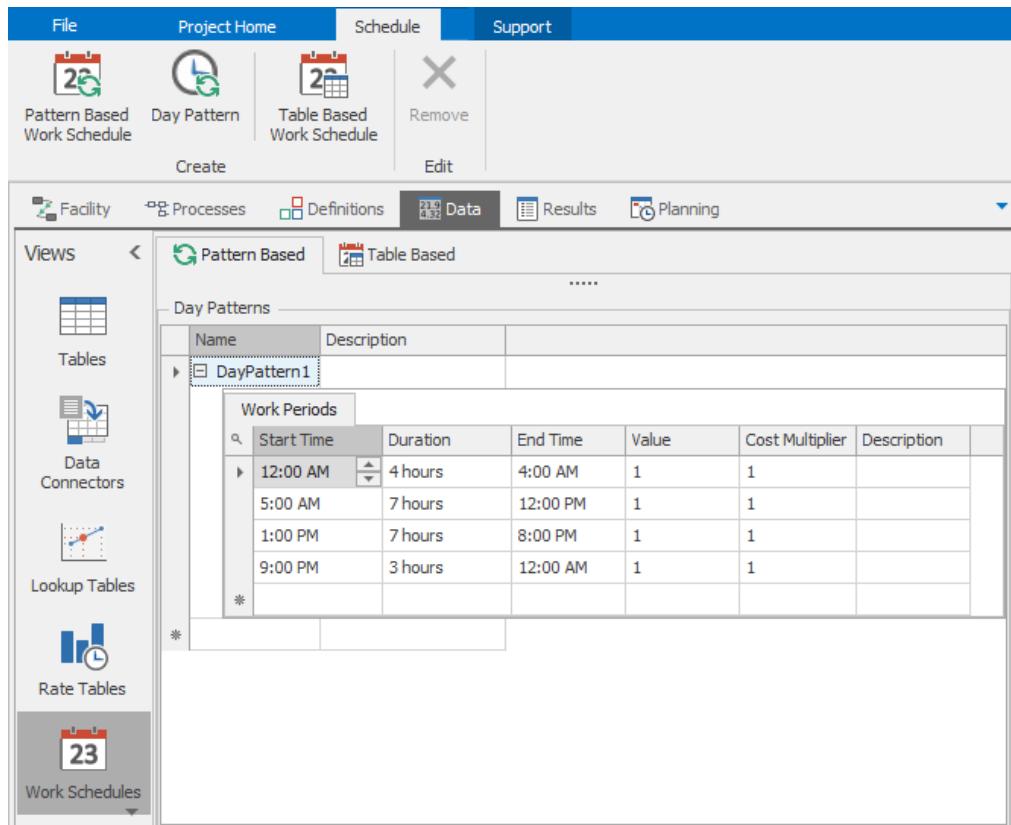


図 5.22 検査工程の 1 日のパターンの定義

ここで、Work Schedules タブに戻る。再びサンプルスケジュールを削除して、新しいスケジュールを作成することもできるが、今回は必要に応じて修正を加えることにする。Work Schedule Name を InspectionSchedule に変更し、Description を Daily schedule for inspectors に変える。このモデルでは作業パターンが毎日同じため、同じパターンが毎日（1 日）繰り返されることを示すために、1 を Days プロパティ値に設定する。最後に、元々ここで参照されていた StandardDay パターンを削除したので、Day 1 列はエラーを示す。エラーを正すために、プルダウンリストから新しい DayPattern1 を選択する。完成した作業スケジュールは、図 5.23 のようになる。

ここで、定義されたスケジュールができたので、リソース能力の指定にスケジュールを用いるように Inspection オブジェクトに設定する。Inspection オブジェクトを選択して、Capacity Type プロパティを WorkSchedule に設定する。そして、Work Schedule プロパティに InspectionSchedule を設定する（図 5.24 参照）。

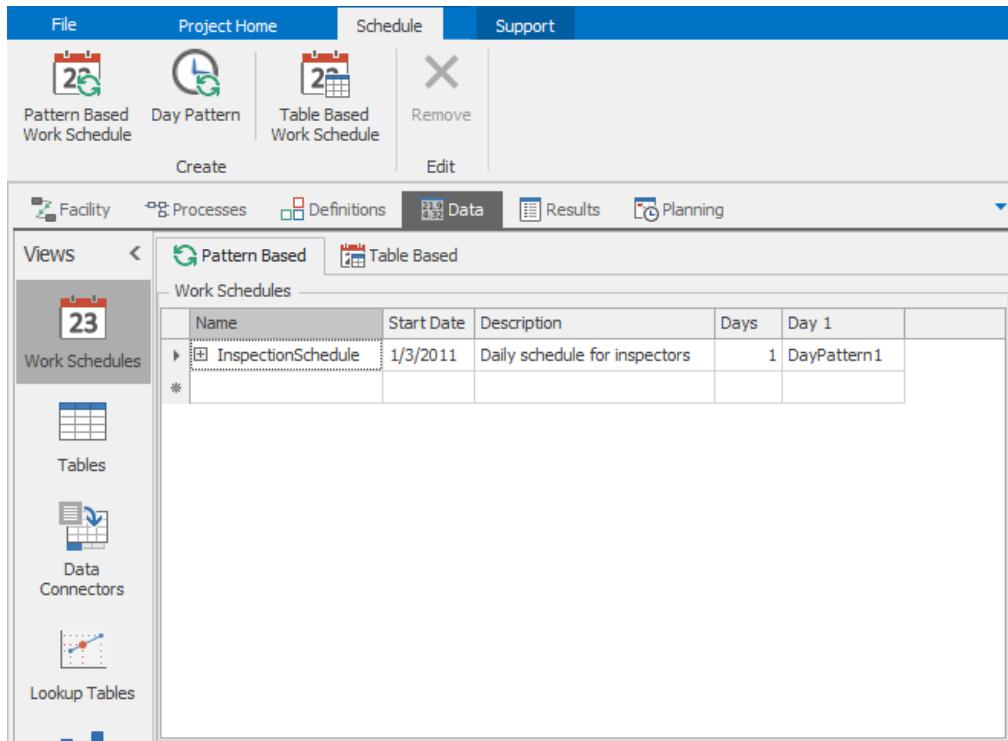


図 5.23 検査工程のスケジュール

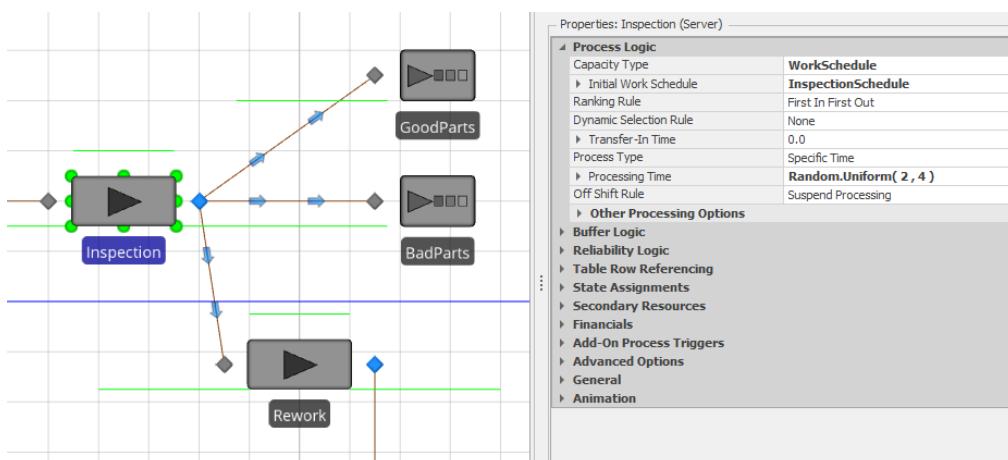


図 5.24 Inspection オブジェクトに WorkSchedule を指定

次に、再加工オペレータについて同様に能力スケジュールを指定する必要がある。上で指定されたように、再加工工程は、2番目のシフトの間だけ操業する。それは、午後4時から午後8時、午後8時から午後9時（食事休憩）と、午後9時から午前1時である。前出と同様、新しい日のパターン DayPattern2 を生成し、新しい仕事スケジュール ReworkSchedule でその日のパターンを参照し、次に、スケジュールを用いるように Rework オブジェクトに設定する（図 5.25 参照）。スケジュールについて、さらに詳細については 7.2 節で議論する。

部品配置機械の故障をモデルに組み入れる前に、リソース能力スケジュールを組み込むことによって、サーバ稼働率の統計量にどのような影響があるかを考えることは、有益である。前出のモデルでは、サーバが稼働していた実行中の割合（または、割合に 100 を掛けたパーセンテージ）として利用率を定義した。サーバリソースの能力は実行中変化しなかったので、これは機能していた。ここでは、モデル実行中のリソース能力が変化するサーバであり、事実上、利用率計算はもう少し難しく、利用率自体の解釈は簡単ではない。どのようにサーバリソースの「オフシフト」時間を評価するかは、特に、明確ではない。たとえば、再加工のオペレータがある日、6時間28分、稼働していたと仮定する。利用率を計算する1つの方法は、1日の長さ（24時間）で稼働時間を割る

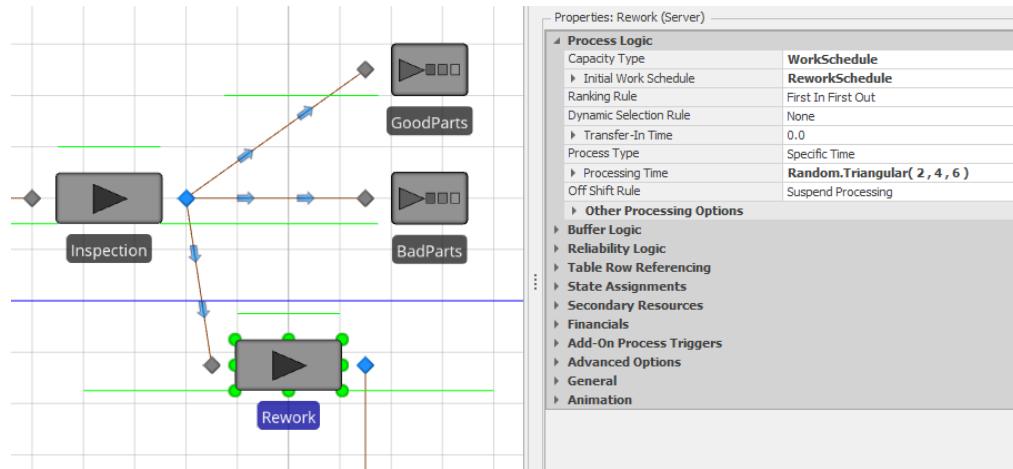


図 5.25 Rework オブジェクトに WorkSchedule を指定

ことによって、26.17%の利用率とする方法である。しかしながら、再加工のオペレータは、「ちょっとまってください。私はそれよりずっと忙しかった！」と、おそらくいうだろう。（オペレーターの立場で見えてくる）問題は、利用率計算が、再加工のオペレーターは 24 時間のうち 8 時間だけ（すなわち、2 番目のシフトだけ）が利用可能であることを考慮していないということである。2 番目の計算方法としては、オペレーターが実際に利用可能である 8 時間で稼働時間を割ることで、78.50%の利用率とする方法である。値について、おそらく再加工のオペレーターは同意するだろう。ここで留意したいのは、これらの解釈のどちらも技術的に「間違っていない」ことである。それらは単にリソース利用率の概念の 2 つの異なった解釈であり、そして、Simio はデフォルト出力として両方を提供している。

図 5.26 は、25 日のウォームアップ期間で 125 日の実行期間として、Model 5-2（部品配置機械に故障なし）における 25 回の反復実行による標準 Pivot Grid レポートの一部を示している。そこでは、Inspection オブジェクトの Server Resource 統計量を示している。稼働率に関する議論において、特に関心を持っている値を表 5.5 に抽出している。

[Resource]	Capacity	ScheduledUtilization	Percent	77.3062	76.3032	78.7940	0.2442
	Capacity	UnitsAllocated	Total	32,462.8800	53.0000	136.0000	104.2211
	ResourceState	UnitsScheduled	Average	0.8750	0.8750	0.8750	0.0000
		Maximum		1.0000	1.0000	1.0000	0.0000
	Capacity	UnitsUtilized	Average	0.6764	0.6677	0.6894	0.0021
	ResourceState	Maximum		1.0000	1.0000	1.0000	0.0000
	Capacity	TimeOffShift	Average (Minutes)	60.0000	60.0000	60.0000	0.0000
	ResourceState	Occurrences		300.0000	00.0000	300.0000	0.0000
	Capacity	TimeProcessing	Average (Minutes)	5.5445	5.3928	5.6699	0.0291
	ResourceState	Occurrences		17,569.4000	81.0000	342.0000	58.6694
	Capacity	TimeStarved	Average (Minutes)	67.6429	66.7653	68.9447	0.2137
	ResourceState	Percent		97,405.8470	42.0028	280.3872	307.6782
	Capacity	Total (Minutes)		28,594.1530	19.6128	357.9972	307.6782
	ResourceState	Average (Minutes)		1.6466	1.5457	1.7258	0.0166
	Capacity	Occurrences		17,365.3600	71.0000	747.0000	60.2163
	ResourceState	Percent		19.8571	18.5553	20.7347	0.2137
	Capacity	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

統計量は以下の通り定義される：

- ScheduledUtilization(Percent)：実行中、オブジェクトのスケジュールされた能力のパーセント利用率であり、式 $100 * (\text{Capacity.Utilized.Average} / \text{Capacity.Average})$ の値を返すことによって計算される。
- UnitsScheduled(Average)：実行中、このオブジェクトのスケジュールされた能力の平均。
- UnitsUtilized(Average)：実行中に利用されているこのオブジェクトの能力単位の平均。
- Offshift Time(Percent)：リソースが Offshift 状態にある実行中のパーセンテージ。
- Processing Time(Percent)：リソースが Processing 状態にある実行中のパーセンテージ。
- Starved Time(Percent)：リソースが Starved 状態にある実行中のパーセンテージ。ここで Starved は、リソースが利用可能であるが、処理するために利用可能なエンティティがないことを意味していることに注意してほしい。

この例題における再加工のオペレータに係る統計量では、26.17%の測定値は UnitsUtilized * 100 と同値であり、そして、78.50%は ScheduledUtilization に同値である。同様に、表 5.5 の値は、検査オペレータがおよそ勤務中の 77.3%稼働しており、そして、検査オペレータが稼働中である割合はおよそ全日中の 67.7%であることを示している。0.8750 の UnitsScheduled 値は、3 回の 1 時間の食事休憩が存在し、検査オペレータが毎日 24 時間のうちの 21 時間（つまり、1 日の 0.8750）がスケジュールされていることを意味する。

Capacity 値と Resource State 値にいつも同じ関係があるというわけではないことに注意してほしい。特に、スケジュールのリソース能力値が 0 か 1 であるため、以下の関係式が成立する。

$$\begin{aligned} \text{UnitsScheduled(Average)} &= \frac{\text{Processing Time}(\%) + \text{Starved Time}(\%)}{100} \\ \text{UnitsUtilized(Average)} &= \frac{\text{Processing Time}(\%)}{100} \end{aligned}$$

あるスケジュールにおいて、リソース能力が 1 以上（たとえば、2、3 人の検査作業員がシフトで働いている）であれば、Resource State 値は稼働リソースのユニット数から独立しているが、Capacity 値はリソース能力のユニット数を考慮する。たとえば、検査工程の処理率を半分にして（Processing Time プロパティを Random.Uniform(4, 8) にする）、勤務サイクルの間、1 人ではなく 2 人の検査作業員がいるようにスケジュールを変更すると、表 5.6 に示した結果が得られた。

このモデルでは、ScheduledUtilization と Offshift Time のパーセンテージは変化しなかったが、他の値は変化した。リソース能力のユニットのどちらかまたは両方が稼働している時間の割合を測定するので、ProcessingTime パーセンテージは増加した。同様に、検査リソースが Starved 状態になるとき、能力の両ユニットが Starved になるため、Starved Time パーセンテージは減少した。

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 への最後の発展は、部品配置の機械ヘランダムな機械の故障を組み込むことである（また、5.1.4 項で機械故障の例を示したことに留意する）。大部分が複雑な機械装置であるため、部品配置の機械は故障しがちであり、基板を処理し続ける前に、それを修理しなければならない。Standard Library の Server オブジェクトは 3 つの異なったタイプの故障をサポートしている：

- ・ **カレンダ時刻基準 (Calendar Time Based)**：稼働時間はカレンダ時刻に基づいており、故障間の稼働時間を指定する。
- ・ **イベント数基準 (Event Count Based)**：稼働時間はイベントが起こる回数に基づいており、イベントの数を指定する。
- ・ **処理数基準 (Processing Count Based)**：稼働時間はサーバがエンティティを処理する回数に基づいており、故障間にサーバが処理する回数を指定する。

部品配置の機械故障に、ここでは Calendar Time Based の故障を用いる。過去の記録では、部品配置の機械は 6 時間の稼働時間後に故障し、そして機械の故障後の修理に必要な時間はおよそ 30 分であることが示されている。稼働時間と修理時間の両方はかなり変動するため、両方が指数分布に従うと仮定する。この故障モデルを Model 5-2 に組み込むために、FailureType プロパティへ Calendar Time Based を、Uptime Between Failures プロパティへ Random.Exponential(6) 時間を、そして Time to Repair プロパティへ Random.Exponential(30) 分として、設定する必要がある（図 5.27 参照）。

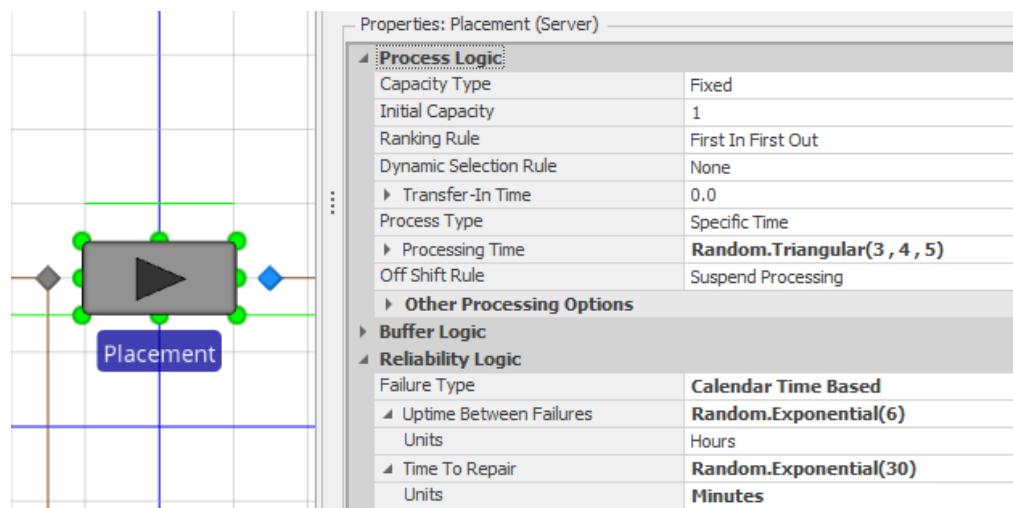


図 5.27 Placement オブジェクトの故障の定義

モデルを実行すると（前項と同様のパラメータを用いて実行する）、Placement オブジェクトの ResourceState カテゴリに Failed Time のセクションが含まれるようになり（図 5.28 参照）、機

械が故障状態である時間の割合の推定値が約 7.83% であると考えることができる。

ResourceState	TimeFailed	Average (Minutes)	30.0046	26.7572	33.0989	0.5971
		Occurrences	376.0400	41.0000	409.0000	7.2392
		Percent	7.8318	6.7293	8.5965	0.1916
		Total (Minutes)	11,277.8352	90.1316	378.9810	275.9076
	TimeProcessing	Average (Minutes)	147.8082	02.2921	226.4377	11.4048
		Occurrences	903.8800	84.0000	253.0000	63.7593
		Percent	90.0323	89.0083	91.8331	0.2523
		Total (Minutes)	129,646.5525	72.0173	239.6147	363.3631
	TimeStarved	Average (Minutes)	5.7175	5.3442	6.0685	0.0898
		Occurrences	537.9200	23.0000	371.0000	65.9470
		Percent	2.1358	0.8935	3.6706	0.2662
		Total (Minutes)	3,075.6123	86.5948	285.6462	383.3170

図 5.28 故障を加えた後の Placement オブジェクトの ResourceState の結果

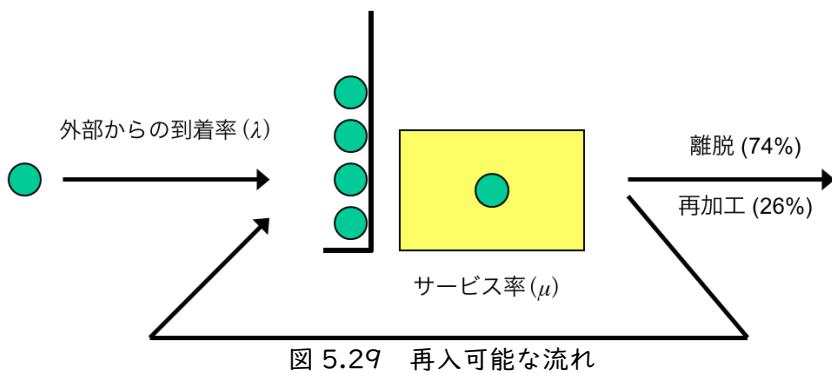
Calendar Time Based の故障は、故障間の時間が時間の関数であるようなランダムな機械の故障をモデル化するのに一般的によく用いられる。Processing Count Based の故障は、故障間の時間が処理回数の関数であるような故障をモデル化するのに一般的によく用いられる。原材料の補給という状況は、このタイプの故障プロセスが必要となる例である。その状況では、機械には部品を加工するために消費される原材料の在庫がある。原材料の在庫が空のときは、機械は原材料の在庫が補給されるまで処理を続けることができない。したがって、故障間の時間は処理された部品数(より明確にいうと Processing Count) の関数になる。Event Count Based の故障は、イベントの発生回数に基づいている。そのイベントは、一般的な Simio 定義イベントもしくはユーザ定義イベントである。

5.3.5 Model 5-2 の検証

システム分析にモデルを用いる前に、モデルが「正しい」という何らかの確からしさが必要である。この文脈では、モデルが予想されたように挙動する（すなわち、正しく Simio モデルに仕様を翻訳している）ことを検証することが含まれる。Model 5-1 と 4 章のモデルについて検証したとき、基本的な方針は、モデル出力を比較するためのいくつかの予測を発展することであった。しかしながら、それらのモデルで予測を立てるために用いた待ち行列解析は、非常に簡単であった。Model 5-2においては、Model 5-2 もまだ非常に簡単なモデルであるが、標準の待ち行列モデルに組み込むには難しいモデルの構成要素が加わっている。もちろん、待ち行列（または、他の分析手法）を用いることでシステムを完全に分析できるなら、シミュレーションの必要はまったくない。さて、Model 5-2 の待ち行列解析を複雑にする 2 つの要素は、以下の通りである：

1. 再入可能な流れ：再加工ステーションから離脱する基板は、再処理のために返送されるので、Placement ステーションへの到着率は外部からの到着率である毎時 10 基板よりも多くなる。
2. 実行中に能力が変化するリソース：リソース能力スケジュールと機械故障を取り入れたことにより、Placement、Inspection および Rework リソースの能力は時間経過とともに変化する。

1 つずつ、これらの問題を考える。Placement ステーションへの到着率が Model 5-1 で設定した毎時 10 基板よりも多くなるので、Placement リソース利用率もより高くなると予想される。新しい利用率を見積るために、新しい到着率を測定する（見積もる）必要がある。図 5.29 に再入可能な流れを図示する。



ここで、 λ' を有効な到着率（プロセスが実際に目にする到着率）とすると：

$$\lambda' = \lambda + 0.26\lambda'$$

λ' について解くと、以下を得る：

$$\begin{aligned}\lambda' &= \lambda / (1 - 0.26) \\ &= 1.3514 \times \lambda\end{aligned}$$

差し当たり機械の故障を無視すると、部品配置の機械の稼働率は $13.514/15 = 0.9019$ であると予測できる。稼働率が厳密に 1 以下であるので、離脱率は到着率と等しく、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無視すると） $13.514/20 = 0.6757$ である。同様に、検査の利用率も 1 以下であるので、再加工ステーションの期待利用率は（再び、リソース能力スケジュールを無視すると）、 $(13.514 \times 0.26)/15 = 0.2342$ である。リソース能力スケジュールと機械故障を無視した Model 5-2 を実行すると、モデルが予測と合っているのがわかる。

リソース能力スケジュールと機械故障には、リソースの利用可能な能力を減少させるという影響がある。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）合計 21 時間／日の間、働いているので、1 日当たりの処理率は $21 \times 20 = 420$ であり、期待利用率は $(24 \times 13.514)/420 = 0.7722$ である。同様に、再加工オペレータは 2 番目のシフトだけの間働いているので、1 日当たりの処理率は $8 \times 15 = 120$ であり、期待される利用率は $(24 \times 13.514 \times 0.26)/120 = 0.7027$ である。

機械の故障には、少し異なったアプローチを取る。部品配置の機械の期待稼働時間が 6 時間であり、それを修理するための期待修理時間が 30 分であった。そうすると、機械は 390 分中に約 30 分、すなわち 7.692% の割合で故障する。部品配置の機械が遊休であると期待される時間の割合 $(1 - 0.9010) \times 100 = 9.91\%$ は、期待故障時間の割合より長いので、システムが安定した状態を保つと期待できる。

表 5.7 は 100 日間のウォーミングアップで 600 日間の実行における、50 回の反復実行に基づいた結果である。これらの値が予測に合っているように見えるため、モデルが正しいという（確かに）有力な証拠となる。そのため、ここで、システムを分析するためにこのモデルを用いることができる。

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)	0.9010 ± 0.1025
Inspection の利用率 (ρ_i)	0.7723 ± 0.0906
Rework の利用率 (ρ_r)	0.7035 ± 0.1545
故障時間の割合 (Placement)	7.7054 ± 0.0714

5.4 Model 5-3：工程選択のある PCB モデル

PCB モデルに追加する最終的な発展は、現在の部品配置の機械の直後に 2 番目の部品配置の処理を加えることである。2 番目の部品配置処理は、ファインピッチ部品を PCB に追加する処理である。これらの部品には配置位置の精度が必要とされるため、この配置処理には通常の部品配置処理よりかなり長く時間がかかる。そのため、ファインピッチ部品の配置処理は、通常の部品配置、検査、および再加工処理が遅れないように、並行して 3 台の部品配置の機械を用いる。図 5.30 は完成したモデルを示している。これは、異なる速度の工程がある連続生産システムの一般的な構成であることに留意する。

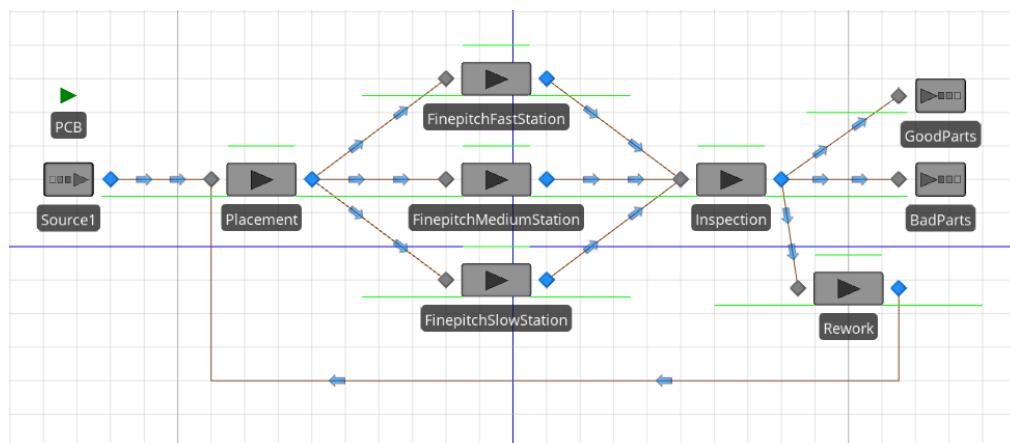


図 5.30 Model 5-3

また、モデルを面白くするために、3 台のファインピッチ部品配置の機械は同じ速度ではないものとする。具体的には、高速の機械、中速の機械、低速の機械があり、それらの処理時間はランダムであり、以下の分布に従うと仮定する：

- ・ 高速 : Triangular(8, 9, 10)
- ・ 中速 : Triangular(10, 12, 14)
- ・ 低速 : Triangular(12, 14, 16)

最後に、高速の機械がランダムに故障することも想定する。その故障では、稼働時間は平均 3 時間の指数分布に従い、そして修理時間は平均 30 分の指数分布に従うと仮定する。このモデルでは、中速および低速の機械は故障しないものとする。

図 5.30 に示されているように、Placement オブジェクトと Inspection オブジェクトの間の Connection 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 および FinepitchSlowStation) を表す 3 個の新しい Server オブジェクトを加えた。各ステーションで、Capacity Type プロパティを Fixed、Initial Capacity プロパティを 1 に設定する。そして、上述した処理時間分布を Processing Time プロパティに設定する。また、FinepitchFastStation オブジェクトに関しては、故障を加えた（図 5.31 を参照）。

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 オブジェクトプロパティ

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 オブジェクトを用いて接続した（分岐）。同様に、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）。

この時点でモデルには 5 つの Server オブジェクトインスタンスがあるので、プロパティスプレッドシートビューの利用法を紹介しよう。このビューは、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、使い慣れたスプレッドシート形式で表示する。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。プロパティスプレッドシートビューを開くには、オブジェクトインスタンスを右クリックし、コンテキストメニューから Open properties spreadsheet view for all objects of this type ... オプションを選択する。また、Project Home リボンの Windows セクションにある Properties ボタンを使用してビューを開くことができる。プロパティスプレッドシートビューには、他のスプレッドシートビューと同様に、標準の並べ替えとフィルタリングオプションが含まれており、同じオブジェクトの複数のインスタンスを持つモデルでは非常に便利である。

5.2 節の最初の分岐の例では、代替のエンティティの目的地を、無作為に選択するリンクウェイトを用いた。ファインピッチ機械の選択において、容易に同じロジックを用いることができるが、ここでは別の方法を利用する。ランダムに機械を選択するより、「最初に利用可能な」機械を選択したいと考える。このロジックを実行するには Simio のノードリストを用いる。ノードリストを用いるには、まずリストを定義して、次に、Placement オブジェクトの出力ノードに PCB オブジェクトの目的地決定にリストを用いることを設定する必要がある。リストを定義するには、Navigation パネルで Model を選択して、Definitions タブを選択し、Definitions パネルから Lists アイコンを押す。Simio は、String、Object、Node および Transporter リストをサポートしている。リボンの Node アイコンをクリックすることで、新しいノードリストを作成し、3 個のファインピッチ・オブジェクトの入力ノード (Input@FinepitchFastStation、Input@FinepitchMediumStation および Input@FinepitchSlowStation) を、メインウィンドウの下部枠のノード

リストに追加する（図 5.33 参照）。下部枠の Node リストの各項目は、モデル内のノードのすべてを含んでいるプルダウンリストから選択されることに注意してほしい。

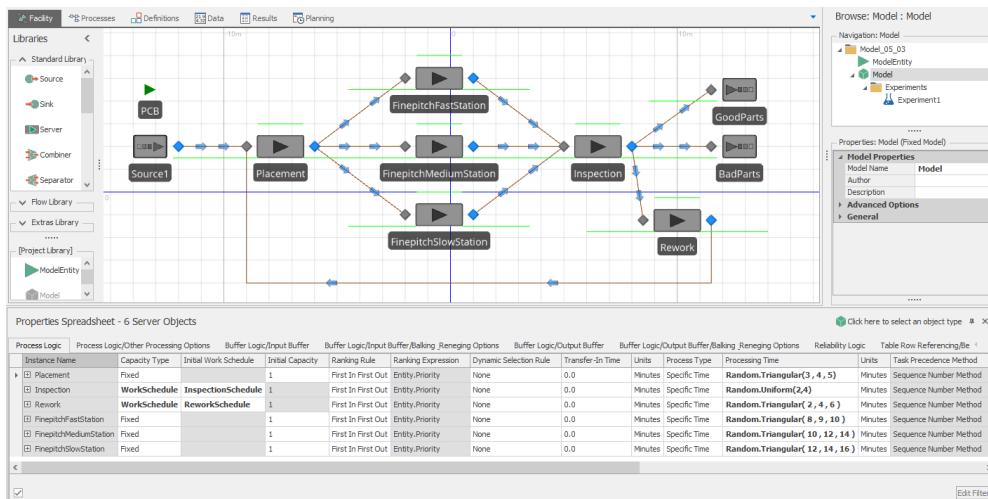


図 5.32 Model 5-3 のプロパティスプレッドシートビュー

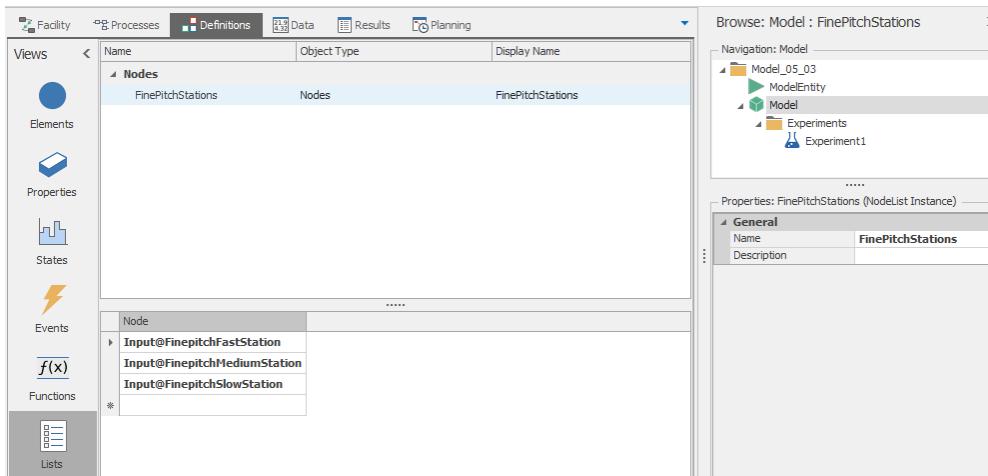


図 5.33 FinePitchStations ノードリスト

さて、リストが定義されたので、リスト要素の 1 つを選択する方法を 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 を送りたいと考える。処理時間がランダムであり、機械が偶発的故障を被りやすいので、どれが最初に利用可能になるかを確認できないため、代理の評価基準を用いる。つまり、待機中あるいは移動中の基板がもっとも少ない機械を選択したい。このロジックを実行するために、Placement オブジェクトの出力ノードを選択し、図 5.34 に示されている値にパラメータを設定する。

Entity Destination Type と Node List Name プロパティは、FinePitchStations リスト（図 5.33）を用いるように Simio に設定し、Selection Goal プロパティは、Selection Expression の最小値とする。この場合、エンティティがノードに到着すると、Simio はリストのそれぞれの候補ノードについて式を評価して、その式の値が最小となるノードへエンティティを送ることになる。

式の値、Candidate.Node.AssociatedStationOverload については、いくつかの説明が必要であろう。Candidate.Node というキーワードは、リストから各候補ノードを考慮することを示す。関数 AssociatedStationOverload は、それぞれのノード（この場合ではファインピッチ機械オブジェクト）に対応する各ステーションの Overload（過負荷）を返す。プロパティ AssociatedStationOverload は Simio Reference Guide で以下のように定義されている：

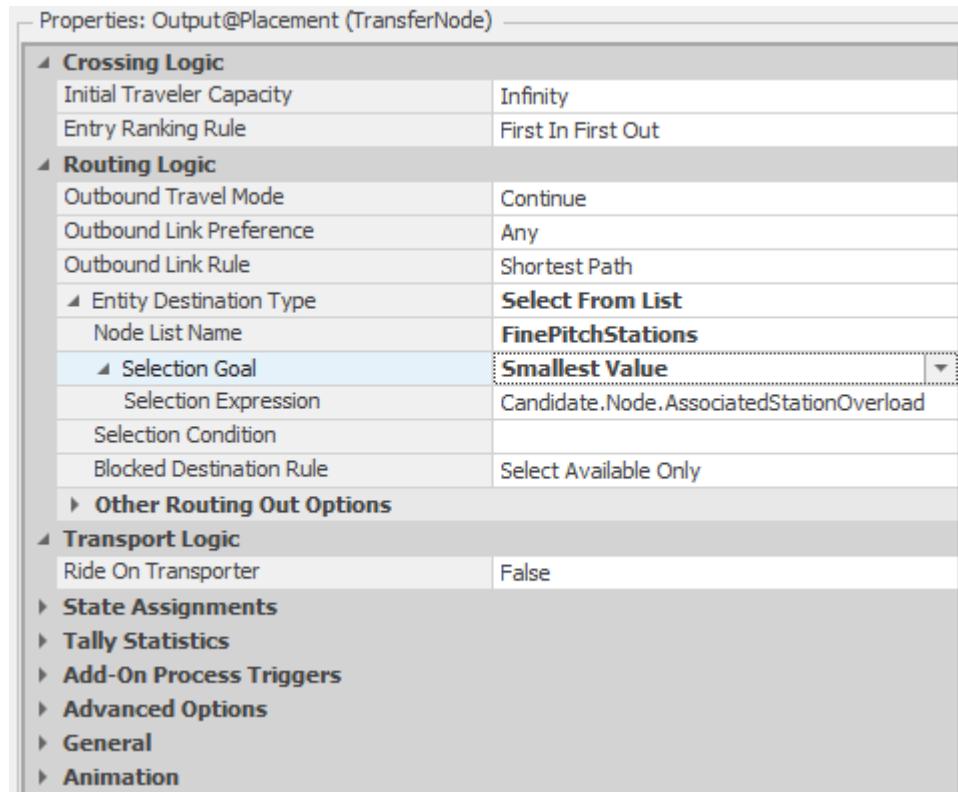


図 5.34 Placement オブジェクトの出力ノードのプロパティ

外部の入力ノードに対して、ノードを利用するために入ろうとしている、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ついて、負荷の値と能力との差の現在値を返す（正の差の場合は「過負荷（Overload）」を意味する）。

対応するステーションの「負荷」は、ステーションに入ろうとノードに向かっているエンティティ数と、ノードに到着したがステーションに入るのを待っているエンティティ数、そしてステーションを占有している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 ($Overload = Load - Capacity$)。

したがって、エンティティが Placement オブジェクトの出力ノードに達するとき、Simio は最小の過負荷（ここでは、機械を待つ作業量の尺度）を持つファインピッチ機械の入力ノードを選択し、そのノードにエンティティを送る。同じ過負荷値の複数のノードがあれば（たとえば、最初の基板が到着したときは、3 台すべての機械は空であって使用されていない）、Simio はリストの順序に基づいてノード（すなわち、最小値を持つ最初のノード）を選択する。

重要な注意：Input Buffer Capacity が Infinity（無限）の場合は、AssociatedStationOverload や AssociatedStationLoad のような関数はうまく働かない。基本的な数学からわかるように、Infinity に何を加算しても Infinity のままであることを思い出してほしい。したがって、無限の能力を持つ待ち行列がリストに含まれると、同順位となる。

この例題では、先頭に高速の機械を置き、タイブレイクの優先順位を持たせる。AssociatedStationOverload 以外にも、キーワード AssociatedStation はステーションに関連する多くの値と組み合わせて用いられる。よくある活用例を以下に示す（式ビルダで値を開き、矢印を用いて表示されることにより、選択肢を確認できる）：

- AssociatedStation.Capacity: 外部入力ノードに対して、ステーションの現在の能力を返す。
- AssociatedStation.Capacity Remaining: 外部入力ノードに対して、ステーションの現在の利用可能な（未使用の）能力を返す。

- AssociatedStation.Contents：外部入力ノードに対して、ステーションで待っているエンティティの現在の数を返す。

ファインピッチ部品配置処理のロジックをモデルに組み込んだので、高速、中速、低速の機械がそれぞれ選択された回数の割合を追跡できるように、ユーザ定義統計量を加える。この統計量を加える理由は、以下の2つである：1つ目は、モデル検証のためである。高速、中速、低速の機械の割合は、それぞれ処理の約38%、33%および29%となることが予測される（章末の問題8参照）。2つ目は、ユーザ定義統計量の設定方法を説明するためである。ユーザ定義統計量を作成するための手順は、以下の通りである：

- 各機械が基板を処理するために利用された回数を記録するモデル状態変数を定義する。
- ファインピッチ機械の1つが処理を完了するときに、対応するモデル状態変数を増加する。
- 各割合をOutput Statisticで報告するようにSimioに設定する。

最初のステップでは、Navigation ウィンドウで Model を選択して、Definitions タブに進み、Definitions パネルから States アイコンを選択する。そして、3つのDiscrete Statesを加える（NumFast、NumMedium および NumSlow。図 5.35）。

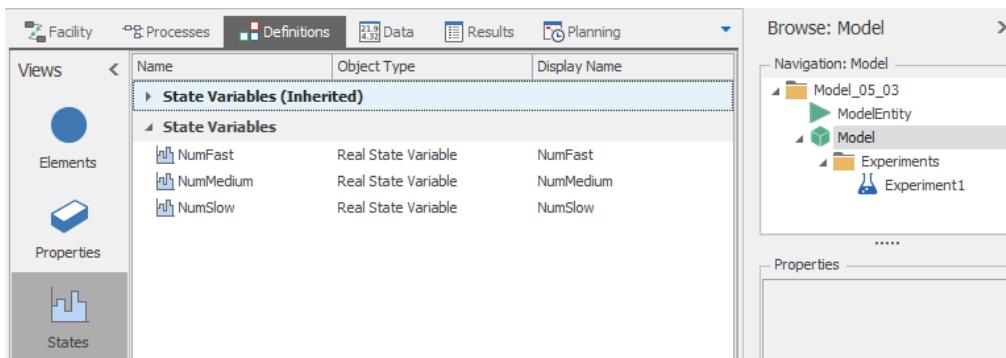


図 5.35 ファインピッチ機械の処理数を追跡するモデル状態変数の定義

部品配置の機械によって処理された基板の数を追跡するためにユーザ統計量を定義したとき（5.3.1節）、モデル状態変数ではなくエンティティ状態変数を用いた。これは、エンティティの特性を追跡していたからであって、今回の場合はモデルの特性を追跡している。ユーザ定義統計量を作成する際には、このことは非常に重要な区別である。2つ目のステップでは、5.3.1項で用いたのと同じ手法を用いる。Serverオブジェクトに対応するState Assignmentsプロパティを用いて、それぞれのリピートグループ BeforeExiting で NumFast、NumMedium および NumSlow を増加する。最後に、モデルの Definitions パネルから Elements アイコンを選択する。そして、各機械で完了した処理の割合を計算するために、3つの Output Statistics を加える。Output Statistic は反復実行の終わりに式の値を報告することを想起してほしい。高速の機械が利用された回数の割合の式は以下の通りである：

$$\text{NumFast} / (\text{NumFast} + \text{NumMedium} + \text{NumSlow})$$

これにより、反復終了時に割合を計算し、反復間の統計量を追跡し報告するようにSimioに設定している。中速および低速の機械に対する出力統計量も、それぞれ分子のNumFastをNumMediumとNumSlowに変更すること以外は、同様に定義される。

また、Simioによって提供された処理数のカウントを用いて出力統計量を定義することで、割合の統計量を追跡することもできた。特にSimioプロパティの、

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：

```
FinepitchFastStation.Processing.NumberExited /
(FinepitchFastStation.Processing.NumberExited +
FinepitchMediumStation.Processing.NumberExited +
FinepitchSlowStation.Processing.NumberExited)
```

これらの 2 つの手法を比較すると、独自の状態変数を用いる利点は、値の追跡方法を正確に知ることができる、そして、自由にそれらをカスタマイズできることである。しかし、Simio によって提供された状態変数を用いるほうがいくらか簡単である（状態変数を定義する必要はないし、また増加する必要がないため）。いずれにせよ、自分で状態変数を定義する方法と Simio 内蔵の状態変数を用いる方法の両方を理解しておくことは、同様に重要である。

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。たとえば、ファインピッチ部品配置の機械を PCB 組立システムに追加購入することを考えていて、高速、中速、または低速の機械のどれかを 1 台購入するかどうか検討したいと仮定する（おそらく、これらの価格は異なるだろう）。本節では、それぞれの代替案の下でシステムの予測パフォーマンスを比較できるように Model 5-3 を変更して、Model 5-4 を作成する。ファインピッチ機械ステーションの能力を 1 つ増加し、モデルを実行して結果を保存し、さらに 2 回繰り返す（その都度、異なるファインピッチ機械ステーションの能力に変える）。このようにして、手動で保存した結果を用いて代替案を比較するように、容易にモデルを変更できるが、ここでは Simio の実験機能の威力を示すために別のアプローチをとってみよう。最初のステップは、実験においてそれぞれのサーバ能力を容易に操作することができるよう、3 個のファインピッチの Server オブジェクトに対する Initial Capacity プロパティのプロパティ参照（5.1.2.4 参照）を定義することである。

図 5.36 は、プロパティ参照 InitialCapacityFast を Initial Capacity に設定した FinepitchFastStation サーバのプロパティを示している（緑色の矢印は、そのプロパティ値がプロパティ参照であることを表す）。プロパティ参照の定義と、そのプロパティ参照に Initial Capacity サーバプロパティの設定を行るために、Properties ウィンドウの Initial Capacity プロパティのラベル上（プロパティ値を入力するためのフィールド上ではなく、ラベル上）で右クリックし、ポップアップメニューから Set Referenced Property を選択して、メニューの下部から Create New Referenced Property を選択する。これにより、Reference Property 名を入力するダイアログボックスが開く（ここでは、InitialCapacityFast と命名した）。次に、同じ手順に従って、中速および低速のファインピッチ機械に対してプロパティ参照を定義する。

3 つのプロパティ参照を定義したら、新たに定義されたプロパティ参照を表示するために、Navigation ウィンドウから Model を選択し、Definitions タブの Properties を選択する（図 5.37 参照）。

ここでは、Default Value プロパティを設定する（Model 5-4 では 1 に設定した）。この時点で、新たなプロパティ参照が定義され、3 つのファインピッチ機械に Initial Capacity プロパティを定義するために、そのプロパティ参照を用いるように Simio に設定できた。インターラクティブな実行に対するプロパティ参照値は、Model Properties で設定できる（Navigation ウィンドウで Model を右クリックし、そのメニューから Properties を選択する）。

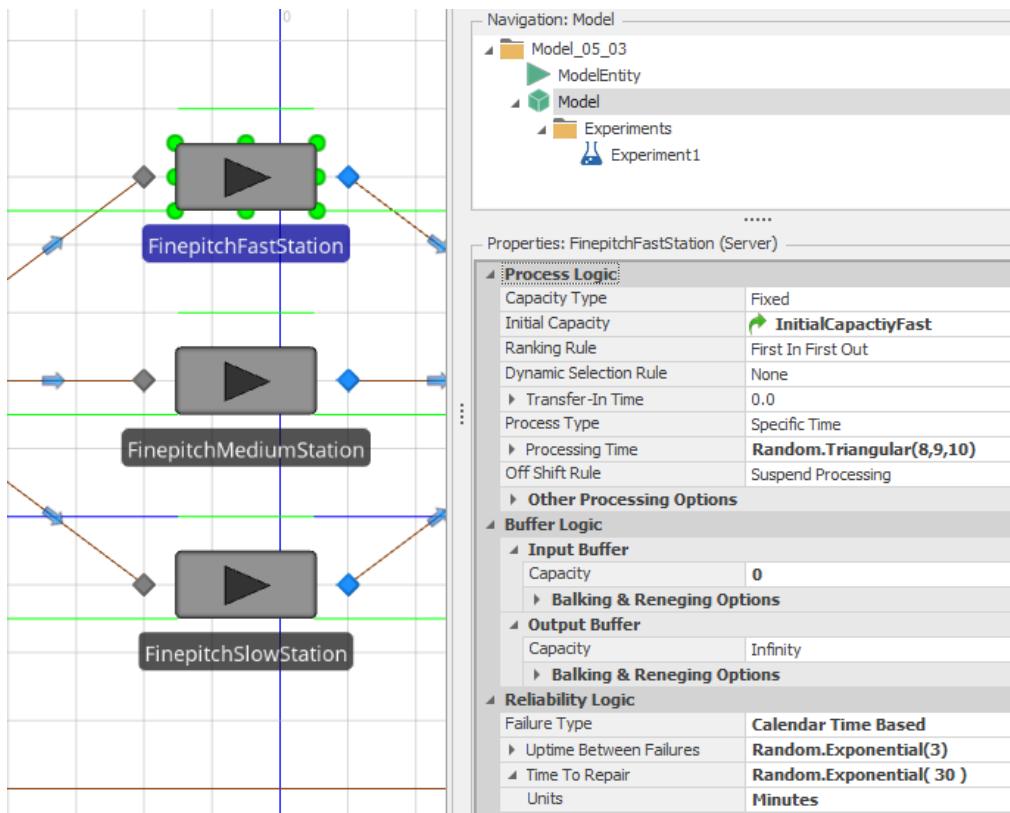


図 5.36 高速のファインピッチ機械ステーションにおけるプロパティ参照の設定

Facility

Views

- Elements
- Properties
- States
- Events
- f(x)
- Functions
- Lists

Processes

Data

Results

Planning

Properties

Name	Object Type	Display Name
InitialCapacityFast	Expression Property	InitialCapacityFast
InitialCapacityMedium	Expression Property	InitialCapacityMedium
InitialCapacitySlow	Expression Property	InitialCapacitySlow

Browse: Model : InitialCapacitySlow

Navigation: Model

- Model_05_03
 - ModelEntity
 - Model
 - Experiments
 - Experiment1

Properties: InitialCapacitySlow (Expression Property)

Value	
Default Value	1
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Unspecified
Appearance	
Display Name	InitialCapacitySlow
Category Name	General
Expanded	False
Parent Property Name	
General	
Name	InitialCapacitySlow
Description	
Required Value	True
Visible	True

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に実験を定義することである。実験を定義する前に、(デモンストレーションの目的のために) パフォーマンスの差がより明確になるように、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、高速は Triangular(8, 9, 10)、中速は Triangular(18, 20, 22)、そして低速は 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。

Scenario	Replications	Controls	Responses					
			InitialCapacityFast	InitialCapacityMedium	InitialCapacitySlow	TIS	WIP	NumTimes
Add Fast	300	300 of 300	2	1	1	9.07304	90.8868	1.35123
Add Medium	300	300 of 300	1	2	1	9.73842	97.436	1.3516
Add Slow	300	300 of 300	1	1	2	12.8645	128.872	1.3506

図 5.38 プロパティ参照が表示された 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、議論が必要であろう。4.5 節で実験にレスポンスを導入し、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自動的にこの統計量 (PCB オブジェクトの `PCB.Population.TimeIn System.Average`) を追跡する。
- 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自動的にこの統計量 (PCB オブジェクトの `PCB.Population.NumberIn System.Average`) を追跡する。
- Num Times : エンティティが部品配置の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たが、その平均値を得るために式 `NumTimesProcessed.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のを見ることができる。Controls 列には、3 つのファインピッチ部品配置の機械オブジェクトの初期能力を指定するために作成し利用される、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の値を設定でき、実行する反復回数や 3 つのプロパティ参照の値を指定することで、シナリオを定義できる。図 5.38 に示されているように、シナリオ構成を定義するためにプロパティ参照を用いて、3 つのシナリオを定義した（高速の機械を 1 つ増やすシナリオ、中速の機械を 1 つ増やすシナリオ、そして低速の機械を 1 つ増やすシナリオの 3 つ）。複数のシナリオが存在する場合、実験テーブルの左列にあるボックスにチェックをつけたり外したりすることにより、実行するシナリオを明確に制御できる。利用しているコンピュータの CPU の種類により、Simio は複数のシナリオを同時に実行できる。

図 5.39 は、TIS レスポンスが選択された 3 シナリオの実験における Response Chart タブ (4.5 節で紹介した SMORE プロット) を示している。直接比較できるように、3 つの SMORE プロットが同じスケールで表示されていることに注意してほしい。これらの SMORE プロットでは、Upper Percentile が 90%、Lower Percentile が 10% に指定されている。したがって、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% が含まれている。また、Response Chart リボンの Rotate Plot ボタンで、値の軸が水平で表示されるようにした。予想したとおり、Add Fast シナリオが他の 2 つのシナリオより良好である（平均システム内時間がより小さい）ように見える。また、Add Medium は Add Slow よりよく見える。図の左上の近くのプルダウンリストから選択されている WIP レスポンスを選んでも、類似したパターンが現れるだろう（図は省略する）。

図 5.40 は、部品配置の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) を SMORE プロットに示している。予想したとおり、基板が処理される回数は、高速、中速、低速のファインピッチ部品配置機械のいずれを購入するかに左右されないように見える。

TIS や WIP に対する SMORE プロットにおいて、あるシナリオが他のものよりもよく「見える」かもしれないが、正式な結論を導き出す前に、統計的に妥当な比較を活用したほうがよいだろう。1 つの方法は、各反復からそれぞれのレスポンスが含まれた CSV ファイルを作成する Simio の Export Details 機能を用いることである (4.6 節で議論した)。そして、統計的比較手法を提供する専用の統計分析パッケージにデータをインポートする。もし平均値を比較するのであれば、手法の 1 つとして、古典的な対応のある t 検定のアプローチがある。そのアプローチでは、2 つのシナリオの平均値の差の信頼区間を構築する、または 2 つの平均値の差の仮説検定をすることになる。なお、ここでは、すべてのシナリオの実行に用いられる乱数が同一であり、それゆえに標本が独立ではないため、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 検定では非独

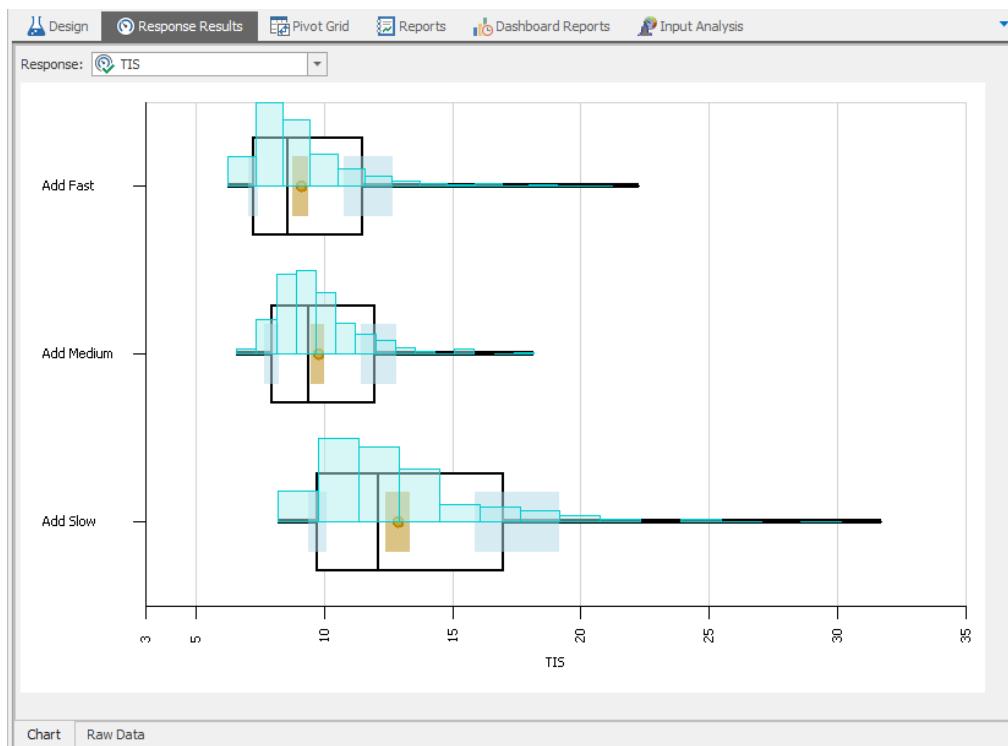


図 5.39 Model 5-4 のシステム内平均時間の SMORE プロット

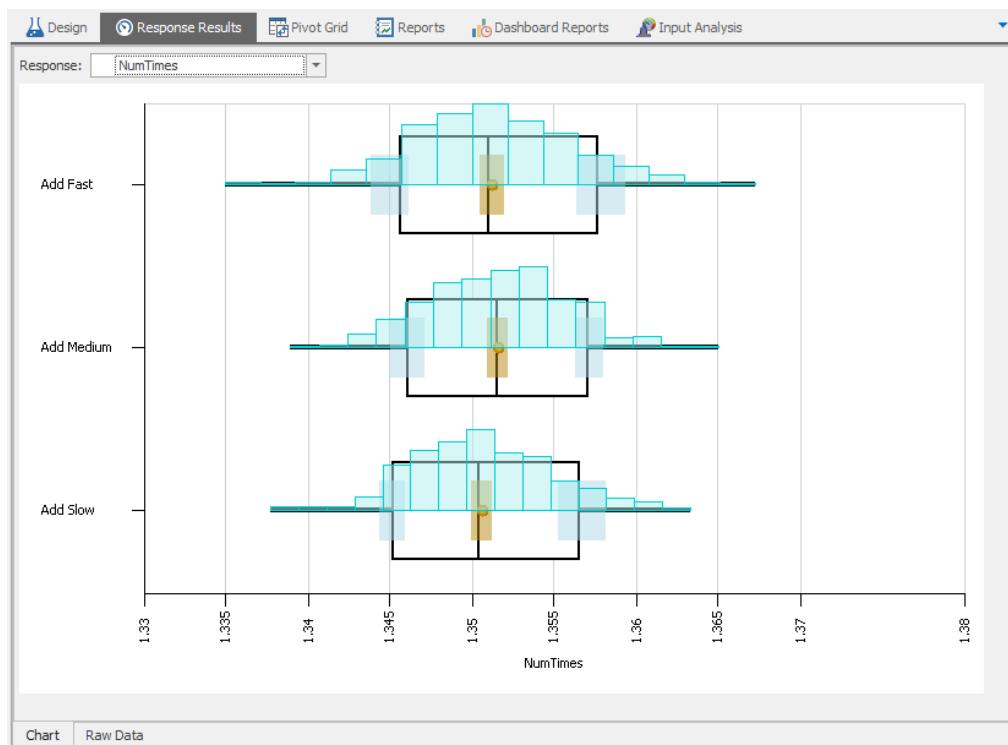


図 5.40 Model 5-4 の NumTimes レスポンスの SMORE プロット

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あり、すべてのシナリオにわたって別々の乱数が必要とされる。この方法では、同時に 2 つのシナリオの比較だけが許容され、ここで行いたいような 3 つ以上のシナリオが存在する場合には、すべての対(高速と中速、高速と低速、中速と低速)について比較することになるだろう。しかし、このようなやり方で複数の統計的な比較を行う場合、全体の信頼水準を低下させてしまうことを念頭に置いておく必要がある。したがって、同時にすべての平均値を比較するには、分散分析 (ANOVA) のような手法を用いて、一般的な多重比較手法 (Tukey, Scheffé, Bonferroni など) を利用する。詳細に関しては、標準的な統

計学のテキストを参照してほしい。

複数の代替シナリオを比較することは、シミュレーションの適用においては一般的であるため、そのために特別な統計的手法が開発されている。図 5.38 の TIS レスポンス列を慎重に見返すと、Add Medium および Add Slow シナリオに対する Response 値は薄茶色で表示されているが、Add Fast シナリオは濃茶色で表示されている。これは、Simio Experiments で利用可能な Subset Selection Analysis の結果であり、応答の期待平均値でシナリオを 2 つの部分集合に分割するために Boesel et al. (2003) の統計的手法を用いている。2 つの部分集合は、各シナリオについて 300 回の反復に基づいて判断された「おそらく最善」なシナリオ（濃茶色）と「棄却」シナリオ（薄茶色）である。この手法に対する Nelson (2013) における重要な解釈は、標本平均に関して「（おそらく最善の）部分集合に分類されるすべては、標本の最適値から統計的に差がない」ということである。この Subset Selection を実行するためには、Experiment 実行を終了した後、まず Experiment Design ウィンドウで TIS レスポンスを選択する。それから、平均システム内時間は小さいほうがよいため、結果の Properties グリッドの Objective に（右側のプルダウンリストから）Minimize を選択する。そして、Experiment Design リボンの Analysis から Subset Selection アイコンをクリックする。そうすると、そのアルゴリズムが、どのシナリオが「おそらく最善」（この場合、濃茶色の AddFast）であり、そしてどれが「棄却」（この場合、薄茶色の AddMedium と AddSlow）であるかを決定する。なお、Experiment Properties の Confidence Level において、この結論を導くための統計的信頼水準を、90%、95%、98%、または 99% から選んでおく必要がある。WIP レスポンスにも同様に繰り返すことができ、こちらも小さい値がよいため、再び Objective プロパティに Minimize を選択する。実際にやってみると、TIS レスポンスで分析をした場合と、たまたま同じ「最善」および「棄却」の部分集合を得る。しかし、すべてのレスポンスでこのように一致することは、一般には期待できない（ここで、NumTimes レスポンスについては、その値が大きいほうがよいのか、反対に小さいほうがよいのか、明確でなかったので、Subset Selection Analysis を行わなかつた）。

部分集合の選定では、この例のように、最終的にいつも解決するというわけではない。なぜなら、「おそらく最善」シナリオの部分集合が、複数のシナリオを含む可能性があるためである。このような場合、選択肢を狭めるため、Experiment リボンの Select Add-In アイコンに、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いた手法である。このアドインは、Kim and Nelson (2001) で開発されたアルゴリズムを用いて Experiment の実行を支配し、選択したレスポンス（一度に 1 つのレスポンスを対象とする）を基準として、「最善」に非常に近い唯一のシナリオを特定するために、シナリオの反復実行回数を増加させる。このアドインでは、この選定に対する信頼水準と検査続行域（選定した平均を真の最善シナリオの平均と比べたとき、この値以下でも容認するような、じゅうぶんに小さな値）や、モデルの初期実験に基づいて設定する反復実行回数の上限を設定しなければならない。Simio におけるこの手順の動作方法についての詳しい情報は、Simio Reference Guide の「Select Best Scenario Using KN Add-In」にある。もちろん、この統計的手法に関しては、Kim and Nelson (2001) に詳しい。また、9.1.2 項では、KN アドインを利用して、大規模な例で最善なシナリオを選択する。

5.6 まとめ

本章では、いくつかの新しい Simio の概念を用いた中級モデリングに話を進めた。これにより、Simio のオブジェクト指向がどのように定義されているか、しっかりと基礎を固められたことだろう。また、複数の代替シナリオを比較する際に、有効な統計分析をする方法について議論した。本章のすべては Simio のシミュレーションをさらに深く進めるために、強力な基盤となるものである。以降の章では、モデルデータ、アニメーション、および高度なモデリングのテクニックについて説明する。

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は何か？
2. Server オブジェクトに対応するプロセスについて考える。トークンの親オブジェクトと関連オブジェクトの違いは何か？
3. 表 5.2 (Model 5-1 で両ステーションの処理時間が指数分布) の値に類似した定常状態の値を与える待ち行列モデルを開発しなさい。
4. 運転免許証の取得のために顧客が訪れるオフィスを考える。到着した顧客に対して、受付／申請、視力検査、筆記試験の 3 ステップのプロセスがある。顧客到着は毎時 6 人のポアソン過程であると仮定する（すなわち、到着時間間隔は平均 10 分の指数分布に従う）。表 5.8 に、3 つのプロセスの処理時間分布を示す。オフィスには、受付／申請の責任者 1 人と視力検査を管理する責任者 1 人がいる。筆記試験はコンピュータで実施され、顧客は 3 台のコンピュータステーションで試験を受けることができる。このシステムの Simio モデルを開発しなさい。オフィスは午前 9 時に開き、午後 5 時に閉まるとして仮定する。関心のあるパフォーマンス指標は、顧客がシステム内で費やす時間、責任者およびコンピュータステーションの利用率、そして受付／申請待ち行列、視力検査待ち行列、および筆記試験待ち行列の顧客の平均と最大の待ち人数である。信頼のある結果を得るために何回反復実行をすべきか？答えを説明しなさい。

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。まだ行っていないなら、合理的なオフィスレイアウトを設定し、オフィス内の顧客移動には Path オブジェクトを用いる。ステーション間の距離がオフィスとして妥当であること、およびエンティティ速度が人間の歩行速度に適切となることに留意すること。
6. Model 5-2 では、検査作業員のシフトが重ならないように仮定されていた（すなわち、2 人の検査員が同時に働くことはなかった）。あるシフトの最初の 1 時間と、次のシフトの最後の 1 時間が重なって検査作業員が作業するように、モデルを変更する。2 つのモデルの結果を比較しなさい。検査作業員が重複することは、システム運営の助けになるだろうか？
7. Model 5-2 の説明では、基板の検査において、再加工の必要性が発見される回数に制限はなかった。ここで、基板が検査に 2 回よりも多く不合格になった場合、不良品として処理されるように、Model 5-2 を変更する。検査に 3 回以上不合格となり不良品とされる基板の数を数えなさい。
8. Model 5-3 の説明では、モデル検証の一部として、高速、中速、低速のファインピッチ部品配置の機械に進む部品の割合（それぞれ 38%、33%、29%）について示した。これらの数値を推定するために、待ち行列モデルを開発しなさい。
9. 顧客が処方薬を受け取る調剤薬局を考える。顧客は医師に処方箋を先にファックスで送ってもらい、後から薬局を訪れて処方薬を受け取ることができる。あるいは、処方箋を持参して、その場で調剤されるのを待つこともできる。ファックスで送られた処方箋は、直接薬剤師によって扱われる。薬剤師は、処方箋の薬を調合し、カウンタの後ろの容器に置いていく。

過去の記録では、約 59% の顧客が先に処方箋をファックスで送っており、薬剤師が調剤する時間は三角分布(2, 5, 8)分であったことが示されている。到着した顧客が既に処方箋をファックスで送っていた場合、レジ係は、容器から処方箋の調剤を取り出して、顧客の支払いを処理する。過去の記録では、レジ係が処方箋を探して、支払い処理を完了するために必要な時間

は、三角分布(2, 4, 6)分であることが示されている。到着した顧客が先に処方箋をファックスで送っていない場合、レジ係は支払い処理して、処方箋の薬を調剤する薬剤師に処方箋を送る。レジ係の時間と薬剤師の時間分布は、先にファックスで送った顧客のときと同じである（それぞれ Triangular(2, 4, 6)、Triangular(2, 5, 8)）。ファックスによる処方箋および顧客の到着率、薬局の人員配置は、1日にわたって変動する。表 5.9 に、到着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 ：ファックスによる処方箋の到着率、 λ_2 ：顧客の到着率）。この薬局の Simio モデルを開発しなさい。関心があるパフォーマンス指標は、ファックスの処方箋が調剤される平均時間、顧客がシステム内で費やす平均時間、そしてレジ係と薬剤師のスケジュール利用率である。薬局は午前 8 時に開き、午後 10 時に閉まる。そして、終業時刻にシステムに存在するファックスと顧客は無視できると仮定する。分析に 500 回の反復を利用し、関心があるパフォーマンス指標の SMORE プロットを生成しなさい。

表 5.9 問題 9 の到着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

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。時間当たり 120 個の割合で部品がシステムに到着し、到着はポアソン過程に従うと仮定する。各機械の特性を表 5.10 に示す。モデルには、次の特性を含める必要がある：
- 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。
 - 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。
 - 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。ここで、「Rework」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）部品である。
 - 次の情報を表示するステータスプロットを作成する – (1) その時点のシステム内のエンティティ数、(2) システム内のエンティティ数の平均、(3) その時点のシステム内の再加工数。なお、プロットの時間範囲は 25 時間とする。
 - モデルの魅力的なアニメーションを作成する。
 - 以下の条件で実験する：
 - ・ 実行時間 – 200 時間
 - ・ ウォームアップ – 50 時間
 - ・ 20 回の反復実行
 - 次のそれについて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）

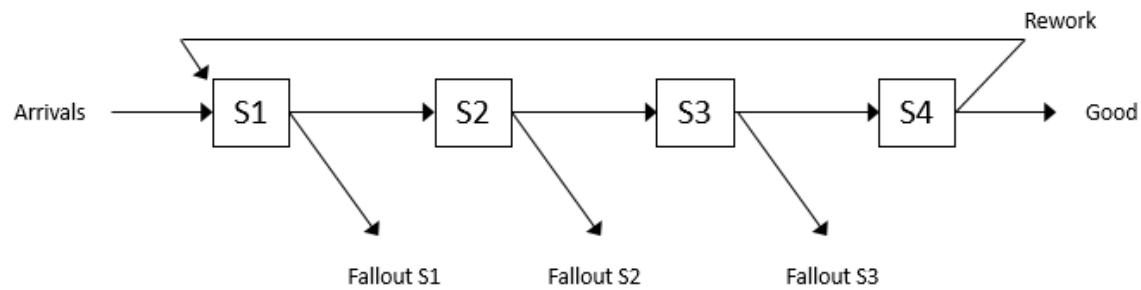


図 5.41 問題 10 の直列生産システム

表 5.10 問題 10 の機械の特性

	S1	S2	S3	S4
サービス時間 (秒)	UNIF(10, 20)	TRIA(20, 30, 40)	UNIF(60, 84)	EXPO(40)
窓口 (容量)	1	1	2	1
不良／再加工の確率	0.15	0.30	0.33	0.08

第6章 入力解析

3章ないし5章のスプレッドシートモデルでは、モデリングプロセスの一環として、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。Model 3-3 では、帽子の需要は、整数 $\{1000, 1001, \dots, 5000\}$ の離散一様確率変数であると述べた。また、Model 3-5、4-1、4-2 では、待ち行列システムの到着時間間隔は、平均 1.25 分の指数分布に従い、サービス時間は、平均 1 分の指数分布に従うと述べた。Model 4-3 および 4-4 では、サービス時間の確率分布を最小 0.25、最頻 1.00、最大 1.75 の三角分布に変更した。これはサービス時間の確率分布として指数分布（最頻値が 0）よりも現実的な形であろう。5章のモデルでは、到着時間間隔や加工時間、検査時間および再加工時間を表現するために、指数分布、三角分布および一様分布の入力確率分布を用いた。そして、本章のモデルでは、定数ではなく、いくつかの不確実性を含めてモデル化した方がよいと思われる種々の入力データのために、多くの確率分布の特定が必要な箇所がある。

これについて、2つの疑問を持たれることだろう：

1. 実際には、どのようにそのような入力分布を特定するか？
2. 入力分布を特定した後、シミュレーションが実行できるように、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（と本書のすべて）では、読者は既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ている。

- 2章の冒頭で箇条書きされたすべての確率のトピックス。
- 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。
- 平均や他の母数の信頼区間、およびそれらの解釈。
- 平均や他の母数の仮説検定（本章では仮説検定の 1つである適合度検定について簡単に説明する）。仮説検定の p 値の概念（観測された有意水準）の解釈。

これらの概念のいずれも手に負えないのであれば、ここを読み進む前に、かつて学習した確率・統計の本で、復習されることを強く勧める。多くの式を暗記したり、正規または t 分布表を覚えていい必要はないが、これらの観念を理解していることが必要である。マルコフや他の種類の確率過程、回帰分析、因果分析やパス解析、データマイニングなど、上記のリストよりもかなり多くの確率・統計に関するトピックが存在する。しかし、本章では、このリストにあるものが、本当に必要なすべてである。

6.1 節では、**单一変量**の入力確率分布の特定方法について説明する。すなわち、モデル内に同時に 1 つだけスカラー変量があり、独立している場合である。6.2 節では、より一般的なシミュレーションモデルへの入力データの種類について概説する。それらには、相関、多変量、および入力過程（時間によって変化する割合で到着を表現する**非定常ポアソン過程**が主要な例）を含む。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）を生成する方法に移る。それらは、多くの

人々が考えるよりはるかに厄介であることが判明しており、Simio に組み込まれている（最高の）乱数ジェネレータを説明する。6.4 節では、乱数をモデルの入力データとして使用を決定する手順と、確率分布から観測値（あるいは実現値またはサンプル）へ変換する方法について説明する。最後に、6.5 節では、Simio の Input Parameters 機能と、サポートされる入力解析の活用法について解説する。

6.1 単一変量の入力確率分布の特定

この節では、シミュレーションへの入力のための単一変量確率変数の分布を特定する方法について、一般的な手順を概説する。6.1.1 項では全体的なテーマを説明し、6.1.2 項では、観測された実世界のデータを使用するための選択肢を概説する。6.1.3 項と 6.1.4 項では、1 つまたは複数の候補分布の選択について検討し、それから、それらの分布をデータに当てはめる。6.1.5 項では、適合された分布が本当に当てはまり具合であるといえるのかどうかを評価する方法について、より詳しく説明する。6.1.6 項では、分布の特定における一般的な問題について概説する。

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、ランダムな数値入力を表す確率分布の特定が必要となる箇所がいくつかある。たとえば、帽子の需要や到着時間間隔、サービス時間、機械の稼働／修理時間、移動時間やバッチサイズなど、枚挙にいとまがない。そこで、それぞれの入力のために、**単一変量確率分布**（つまり、多変量またはベクトル値の RV ではなく、1 次元の RV）を自分自身で特定する必要がある。また、一般的にこれらの入力分布は、シミュレーションモデル全体で互いに**独立**であることを前提としている（6.2.2 項で、シミュレーションモデルに、相関またはベクトル値のランダムな入力を利用することへの可能性と重要性を簡単に説明する）。

たとえば、多くの待ち行列型のシミュレーションで、サービス時間のための分布を特定する必要がある。それらは、製造システムにおける機械の部品加工や救命救急での患者の検診などである。一般的に、すでに使用可能であるか、またはシミュレーションプロジェクトの一環として収集された、これらの時間についての実世界のデータを所持している。図 6.1 は、Excel スプレッドシートファイルの Data_06_01.xls の一部を示している（付録 C で説明するように本書ウェブサイトの students からダウンロード可能）。A 列の行ごとに 1 つずつ、各数分の 47 のサービス時間のサンプルが含まれている。これらのデータは、実際のサービス時間からの IID 観測であり、関心のある期間の定常的で、代表的な部分から収集されたことを前提としている。以下に示す統計的適合度仮説検定に「合格する」という意味で、観測データを適切に表している確率分布を求める。その後シミュレーションを実行するとき、シミュレーションを駆動するために、この適合した分布から、**確率変量**（サービスタイム RV を元とする観測値（あるいは、サンプル、実現値）を「生成」する。

	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 47 の観測されたサービス時間（分）の Data_06_01.xls の抜粋

6.1.2 観測された実データを使用する方法

シミュレーションを実施するために、観測された実際のサービス時間を読み込んでシミュレーシ

ョンへ直接利用するという自然なアプローチを取らず、確率分布を当てはめ、そこから確率変数を生成するという間接的なアプローチをとることを、不思議に思われるかもしれない。それには、いくつかの理由がある：

- ・ 後の章で説明するように、一般的に非常に長い時間のシミュレーションを実行したい。そして、出力データから統計的に有効かつ正確な結論を引き出すことができるようになるため、莫大な回数の IID 反復実行を繰り返すことがある。そのため、すぐに実世界の入力データが不足する。
- ・ 図 6.1 と Data_06_01.xls の観測データは、データが取られたその時点で、何が起こったのかのみを表している。またあるときは、別のデータを観測し、おそらく単なる代表として、とりわけ異なる範囲（最小値と最大値）のデータを得ることがある。もしシミュレーションを駆動するために直接観測データを使用した場合、偶然得た値で行き詰るだろうし、観測範囲外の値を生成することはないだろう。特にサービス時間をモデル化するとき、たとえまれであるとしても、大きな値は典型的な待ち行列の混雑尺度（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ので、シミュレーションに用いるサービス時間の右側を「丸める」ことによって、シミュレーション結果に反映させる。
- ・ 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、現実的には起こり得る値ではあるが、観測ではたまたま取得できなかったというような範囲のギャップが残る場合がある。したがって、そのような値は、シミュレーションの間、決して起こることはない。

したがって、上述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生成することが、しばしば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。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、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ため、モデルの妥当性の確認を行うケースが考えられる。しかし、この場合には、シミュレーション出力に対応した、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。データの収集活動が入力分布を特定すること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ろう。しかしながら、確率分布の適合に際して、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、6.5 節で簡単に解説する。しなければならないことは、どの確率分布が観測データを適切に表しているか見つけ出すことである。

6.1.3 確率分布の選択

もちろん、モデルへのランダムな入力をモデル化するために、選択可能な多くの確率分布があり、おそらくいくつかには精通しているだろう。一般的な連続分布は、正規、指数、連続一様、三角であり、そして離散的な分布には離散一様、二項分布、ポアソン、幾何、そして負の二項分布がある。他のどこかで容易に入手できるので、ここでは、これらすべての分布について詳細には言及しない。Simio Reference Guide (Simio 内で、F1 キーを押すかまたは右上隅の?のアイコンをクリックする) では、およそ 20 のサポートされた分布について、基本的な情報を提供している (すなわち、Simio モデルでそ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、次に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ており、右側にガンマ分布のエントリを示している。そこには、PDF のプロット (この例では連続分布であるので、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、そして Simio での構文とパラメータなど基本的な情報が上部に示されている。Selia et al. (2003) や Law (2015) のような文献では、シミュレーション入力モデリングのための有用な

確率分布が述べられた発展的議論や章もある。さらに、複数巻におよぶ書籍（たとえば、Johnson et al. 1994, Johnson et al. 1995, Johnson et al. 2005, Evans et al. 2000）では、かなり深く、多くの確率分布について記述されている。もちろん、多数のウェブサイトで、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返される。en.wikipedia.org/wiki/List_of_probability_distributions では、約100種の单变量分布についてWebページへのリンクがある。そこでは、離散分布と連続分布の両方のために範囲（あるいはsupport（台））に基づいてカテゴリに分けられている。en.wikipedia.org/wiki/Gamma_distribution は、図6.2と同じガンマ分布である。分布は、いつも同じ様式でパラメータが指定されるとは限らないことに注意が必要である。そこで、Simioでは何が対応しているのか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 $\beta > 0$ によってパラメータ化されている。しかし、平均速度 λ で発生するイベントに関連づけられているポアソン過程の割合は $\lambda = 1/\beta$ と表現されることになるが、Simioにおいては、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。

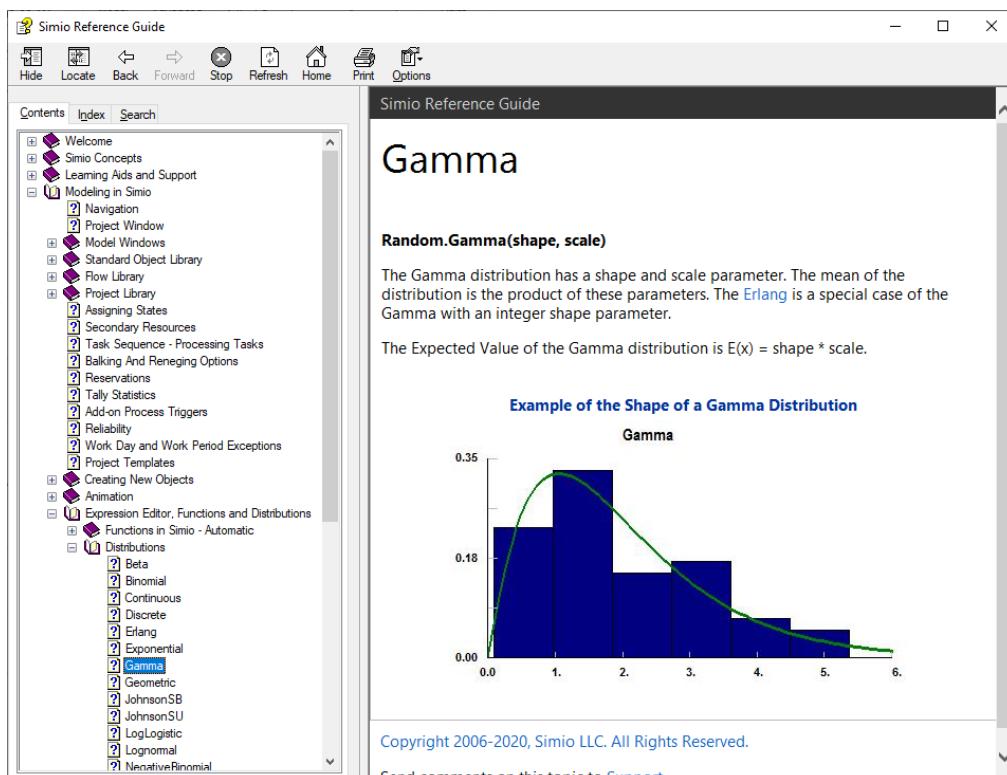


図6.2 Simio Reference Guide のガンマ分布エントリ

何百もの途方もなく多くの種類の確率分布から、どのように1つを選択するのか？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または連続的のどちらである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るべきではなく（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。
- 確率変数がとることができる値の範囲に注意する。具体的には、それが右と左に有限または無限かどうか、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または無限であるかどうかは、待ち行列またはシステム内の時間（または数）のような待ち行列モデルの結果を得るのに非常に大きな影響を与える。
- 上述の点に関連して、無限の裾を両方に、特に左に持っている分布の使用に注意しなければ

ならない。もちろんこれには、（それを知っていて利用するのではあるが）正規分布が含まれる。その PDF は、常に無限の左（および右）裾を有しているので、ゼロの左側にまで伸びており、そのため負の値を生成する可能性がある。これは、サービス時間やその他の持続時間の入力データでは意味をなさない。ただし、平均がゼロ以上の 3 または 4 の標準偏差である場合、負の値が実現する確率が「小さい」ことは事実である。基本的な統計の本では、それを無視することができると書かれている。負の値を取得する実際の確率は、平均がゼロを超えた 3 あるいは 4 の標準偏差である場合、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においては、入力確率分布から簡単に数十万もの、さらに何百万もの確率変量を生成することができる。したがって、最終的には負の値の生成が起こり得る可能性がある。そのような場合、負の値をシミュレーションでどのように扱うかに依存しており、望ましくない、あるいは誤った結果を作り出すかもしれない（Simio は、これが発生した場合、実際に実行を止め、エラーメッセージを生成し、正しい実行内容であるかを尋ねる）。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、シミュレーションで可能性のある領域を外れているとはいえない。もしあなた自身が、その形状で正規分布を使用したいと考えるとき（そして、実データによく適合しているとき）、他の分布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、そして範囲が適切であるかどうかを判断することが、最初のステップであり、適切な分布を絞り込めるだろう。

しかし、まだ、選択可能なかなりの数の分布があるので、今度はその形状（離散の PMF または連続の PDF）が、観測された実データのヒストグラムの形状に少なくともおおよそ、似ているものを探す必要がある。この理由は、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であるという点にある。図 6.3 は、図 6.1 と Data_06_01.xls の 47 の観測されたサービス時間のヒストグラムである（詳細は以下で述べるが、分布適合ソフトウェアの Stat::Fit®で作成した）。これはサービス時間であるので、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。

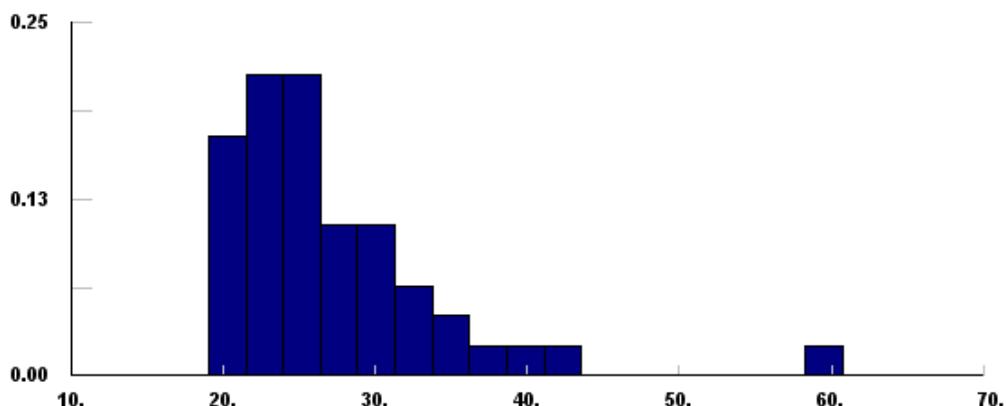


図 6.3 Data06_01.xls の 47 の観測サービス時間（分）のヒストグラム

そして、どれだけ長いサービス時間の可能性があるのか、上限を設ける情報がない場合には、おそらく右側は無限となる。（上述の Simio Reference Guide や、Simio 内で F1 キーを押すかまたは右上隅の?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みると、可能性があるものは、アーラン、ガンマ（これは、アーランの一般化である）、対数正規分布、ピアソン VI、またはワイブルだろう。しかし、これらにはそれぞれ、推定するために必要なパラメータ（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、許容で

きる適合、すなわち、データをじゅうぶんに表したものを見ているかどうか、その分布を検定する必要がある。この推定と適合度検定は、観測データに分布を当てはめることを意味する。

6.1.4 観測された実データへの分布適合

このような分布の適合とそれに続く適合度検定には、いくつかのパッケージがあるため、Simioではこの機能を提供していない。そのようなパッケージの1つは、Geer Mountain Software Corporation (www.geerms.com) のStat::Fitである。本書ウェブサイトのstudentsからダウンロードできる無料の教科書バージョンがあるので、一部を抜粋して説明する。Stat::Fitを選んだもう1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／貼り付けて、エクスポートできるということにある。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ので、Stat::Fitの機能についてすべての説明をしないが、その基本的な部分を説明する。他のパッケージでは、@RISK®(Palisade Corporation, www.palisade.com)、および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の教科書バージョンの入手およびインストール方法は、付録Cで述べる。

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、より詳しい説明と、問題を解決してStat::Fitのヘルプシステムへのアクセスを許可するWinHlp32.exeのオペレーティングシステム固有のバージョン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することができるStat::Fitと同じZIPファイルに、pdfマニュアルが含まれている。

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を表している。Excelから直接、観測された実データをコピー／貼り付けることができ、47の値のうち最初の19が表示されているが、47のすべてデータがそこには存在し、Stat::Fitで使用される。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)で作成する。また、Input→Options、またはツールバーでOPSとラベルされた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、観測データのいくつかの基本記述統計量であり、Statistics→Descriptiveで作成する。

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の中から選択する。無料の教科書バージョンのStat::Fitでは7つの分布(離散ため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が、商用のフルバージョンは、33の分布がサポートされており、観察データのポイントは8,000まで増やせる。

サービス時間をモデル化するための連続的な分布が必要で、さらには負の値を生成するわずかな機会も許したくない場合、このリストから質的に賢明な選択肢となるのは、指数、対数正規、三角、一様である(これは無料の教科書バージョンからの選択であり、もちろんフルバージョンでは他にも多くの分布に可能性がある)。ヒストグラムの形状は明らかに一様分布を除外するように見えるが、もしあまりよく適合していない分布を適合させたらどうなるかを示すためにリストに含めている。適合度検定を行うために、これら4つの分布を選択するには、メニューのFit→Setup、またはツールバーのSetup Calculationsボタン SETUPのラベル)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。右側のリストから削除したい場合は、その分布をクリックすればよい。

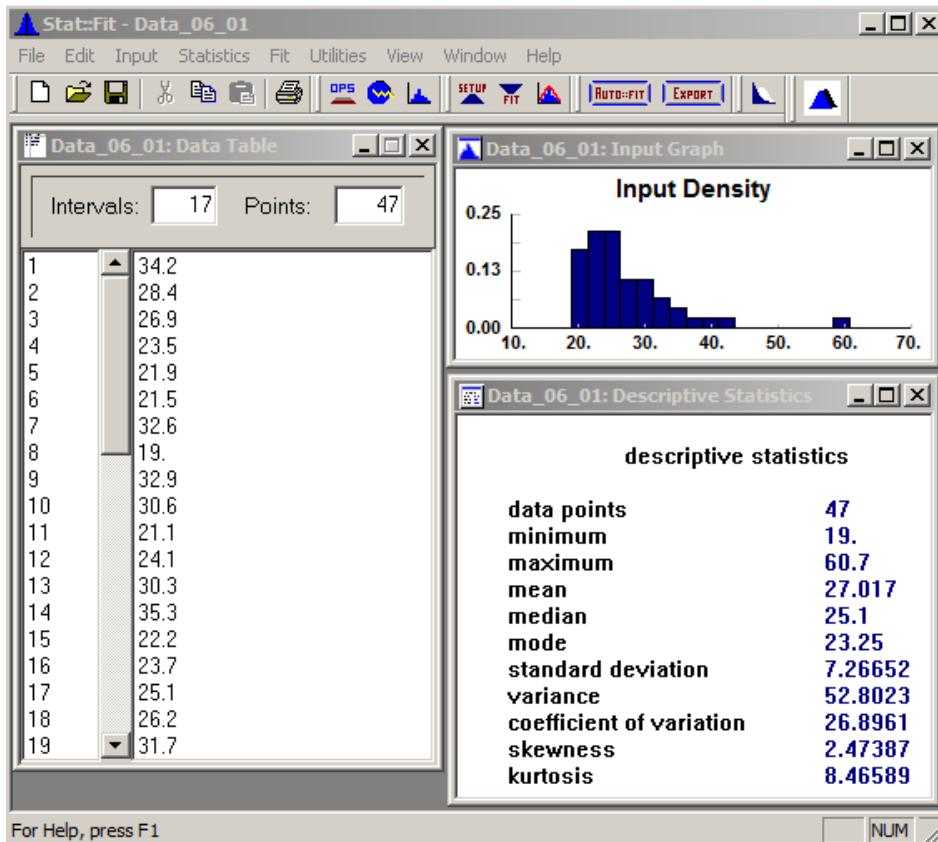


図 6.4 Stat::Fit のデータファイル、ヒストグラム、記述統計量（単位は分）

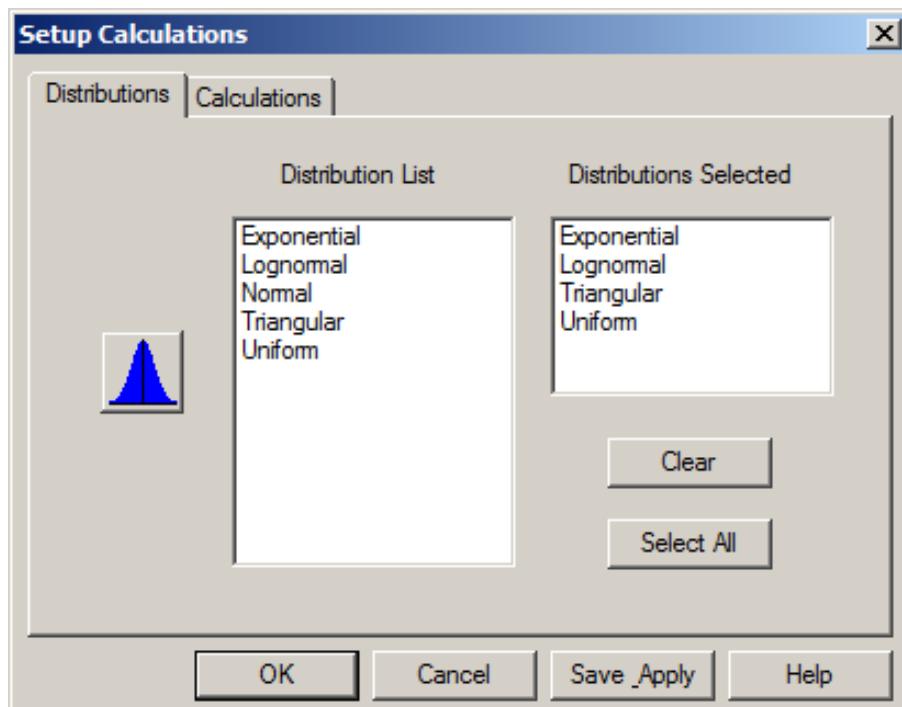


図 6.5 Stat::Fit の Setup Calculations ウィンドウの Distributions タブ

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 (カイニ乗検定、コルモゴロフ-スマルノフ検定、アンダーソン-ダーリング検定) を選択できる。さらに、

カイニ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。さらに詳しくは Banks et al. (2005) または Law (2015) を参照のこと。

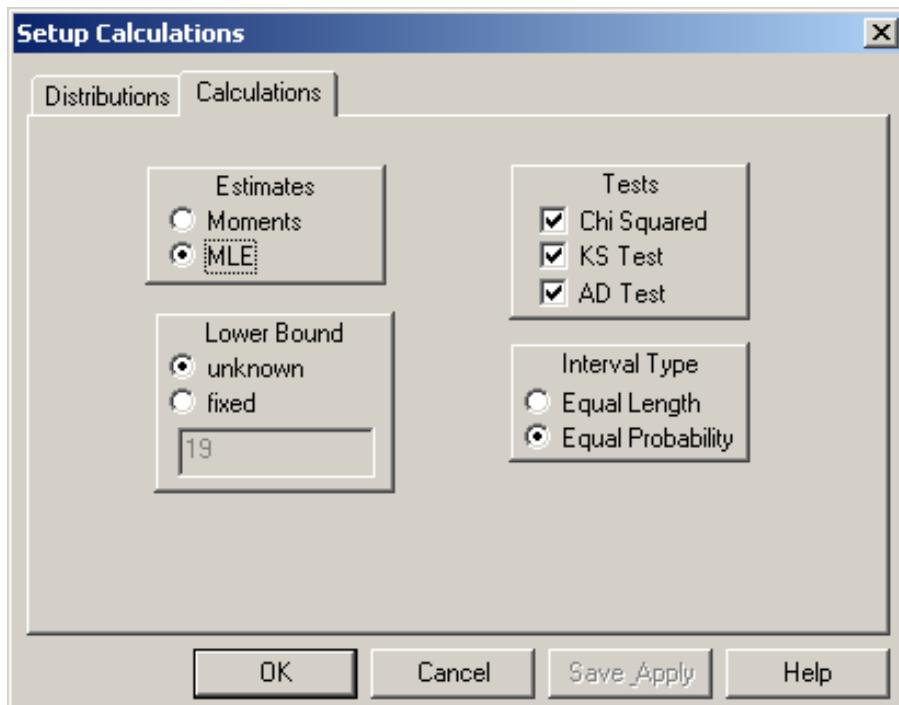


図 6.6 Stat::Fit の Setup Calculations ウィンドウの Calculations タブ

Lower Bound (下限) の仕様はもう少し説明が必要であろう。リストに挙げた指數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。

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。

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 (カイニ乗検定) 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、指數、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、指數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。指數の適合においては、カイニ乗検定は 0.769、コルモゴロフ-スマルノフ検定は 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。

goodness of fit			
	data points estimates accuracy of fit level of significance	47 maximum likelihood estimates 3.e-004 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

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 種の誤りの (誤って真の帰無仮説を棄却する) 確率として選択する。そして、 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おり、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、もちろん、本当は丸め誤差によ

り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、さらに三角や一様の適合の詳細が示されている下にまでスクロールすれば、*p*値が極めて小さく、三角や一様はデータへの適合がよくないことを示している（既に、一様の場合にはヒストограмから分かっていたが、しかし三角分布についてはあまりよく分かっていなかった）。

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ればよい。Auto::Fit ダイアログが最初に表示されるので、（無限の左裾はいらないので）サービス時間に対しては、continuous distributions（連続分布）と lower bound（下限）ボタンを選択する。それから、OK を押すと、図 6.8 の Automatic Fitting の結果ウィンドウを得る。

The screenshot shows a Windows application window titled "Data_06_01: Automatic Fitting". Inside, a table titled "Auto::Fit of Distributions" lists four distribution fits with their ranks and acceptance status:

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ograms 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、取り除くことができる。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、一様分布がどれくらい馬鹿げているか、そして三角分布もあまりよい適合ではないことを確認できる）。

最後のステップは、コピーして直接 Simio の expression フィールドに貼り付けるために、適切な構文に結果を翻訳することである。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。そして、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 $17.4 + \text{Random.Lognormal}(2.04, 0.672)$ が得られる。この式は、今、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り、確率分布を式のフィールドへ貼り付ける（CTRL-V）と、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とられる 0 の下端ではなく、17.4 だけ対数正規を上に（右に）移すことを推奨している。シフトに用いる 17.4 という値は、データセットの最小値である 19.0 より少しだけ小さい値である。そのため、適合した対数正規の確率密度関数は、最小データ値の少しだけ左が 0 となる（したがって、変数を生成するとき、もっとも小さな可能性がある値は 17.4 となる）。

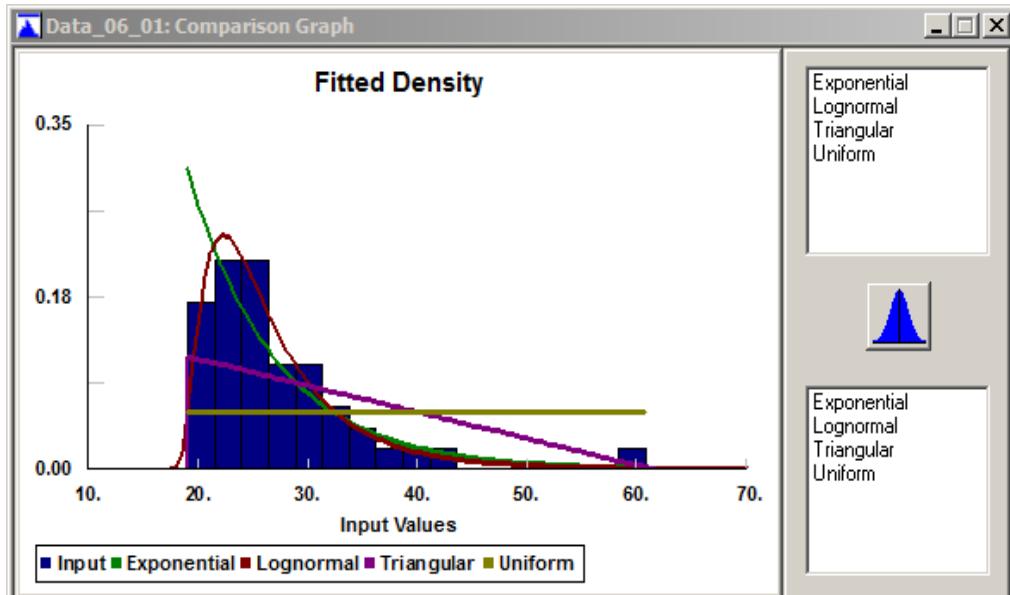


図 6.9 Stat::Fit のデータヒストограм上に重ね合わせた適合した分布の密度関数

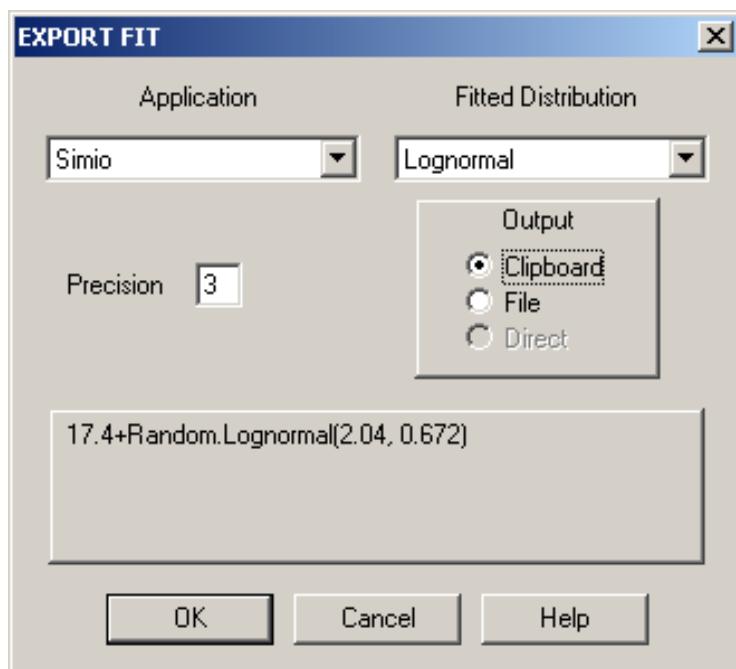


図 6.10 Stat::Fit の EXPORT FIT ダイアログ

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 とするか、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 としている。

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 カイニ乗適合度検定

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ニ乗検定やその他の多くのケースで、その前提是実際に必要ない。さて、区間 k の左端を $x_0, x_1, x_2, \dots, x_{k-1}$ とし、 k 番目の区間の右端を x_k とすると、 $j = 1, 2, \dots, k$ 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、 $\hat{f}(x)$ を適合を検討中の分布の PDF としよう。これがデータを表現するまさに正しい PDF とすれば、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（おおよそ）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。すなわち、各区間の観測度数および期待度数（ \hat{f} がデータに対してまさに正しい密度関数の場合）は、相互に「接近した」ものとなる。このため、多くの区間において実質的に合致しないなら、不十分な適合として疑うことになる。これは、実際には以前に「見た目から」検定したときに行つたことである。

しかし、本当に良好な適合の場合においても、ランダムサンプリングにおける自然な変動のため、すべての区間 j について（正確に） $O_j = np_j$ であることを要求しない。この概念を形式化するため、次のカイニ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} に従うとする帰無仮説 H_0 の下で、 χ^2_{k-1} は（おおよそ。より正確には漸近する。すなわち、 $n \rightarrow \infty$ ）自由度 $k-1$ のカイニ乗分布となる。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」とする判断基準はカイニ乗分布表に依る。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 α で自由度 $k-1$ のカイニ乗分布に含まれる臨界値である。結論を導くもう 1 つの方法は、 p 値による検定を行うことである。ここで、 p 値は自由度 $k-1$ のカイニ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 よりも小さいならば H_0 を棄却する。

検定統計量の値 χ^2_{k-1} およびおそらく検定の結果も、区間の選び方に依存することに留意してほしい。この区間をどのように選べばよいかについては、相当数の研究が行われている問題であるが、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。すなわち、積分の確率値 p_j が互いに等しく（あるい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（およそ）5

とする（後者の条件は基本的に、サンプルサイズが極めて小さい場合、カイニ乗適合度検定の適用を抑止する条件である）。等確率の間隔の端点を見つける1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 は逆関数を示す（算術逆数ではない）。これは x_j について式 $\hat{F}(x_j) = j/k$ として解くことができるが、 \hat{F} の形状により、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を用いる必要があるだろう。

カイニ乗適合度検定は、（バッチサイズのような）値が離散的であるデータに離散確率分布を適合すること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 値の和と置き換えるだけで、手順は同様である。離散分布においては、一般に、区間の選択に対して正確な等確率性を達成することはできないことに注意してほしい。

6.1.5.2 コルモゴロフ-スミルノフ適合度検定

カイニ乗検定が適合 PDF（離散の場合は PMF）と経験 PDF あるいは PMF（ヒストグラム）を比較するものであるのに対し、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、ここでの目的のためには、 $F_{\text{emp}}(x) = (\text{すべての } x \text{ に対して } \leq x \text{ である観測データの割合})$ 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。前と同じように、 $\hat{F}(x)$ を特定の適合分布の CDF とする。K-S 検定統計量は、 x の取り得る値の全体について、 $F_{\text{emp}}(x)$ と $\hat{F}(x)$ との間の垂直方向の乖離における最大値である。数学的に表現すると、次のようになる：

$$V_n = \sup_x |F_{\text{emp}}(x) - \hat{F}(x)|$$

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 $F_{\text{emp}}(x)$ 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, \dots, X_n が観測サンプルを表すものとし、 $i = 1, 2, \dots, n$ 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。

$$V_n = \max \left\{ \max_{i=1,2,\dots,n} \left[\frac{i}{n} - \hat{F}(X_{(i)}) \right], \max_{i=1,2,\dots,n} \left[\hat{F}(X_{(i)}) - \frac{i-1}{n} \right] \right\}$$

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が決めるために、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、カイニ乗検定とは違い、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。カイニ乗検定に対する 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 に対しても正確なことである（ $n \rightarrow \infty$ の場合だけではない）。

Stat::Fit では、メニューで Fit → Result Graphs → Distribution と選ぶことで、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、様々な適合分布の CDF が下部の凡例に挙げられた色で示されている。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（適合分布のそれぞれについては、経験 CDF と各適合分布間

のもっとも大きな乖離を、縦軸の長さから判断するしかない)、一様および三角 CDF は乖離が大きいため経験 CDF との当てはまりはよくなく、対数正規は良好に適合していることが見て取れる(実際、図 6.11において、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)。

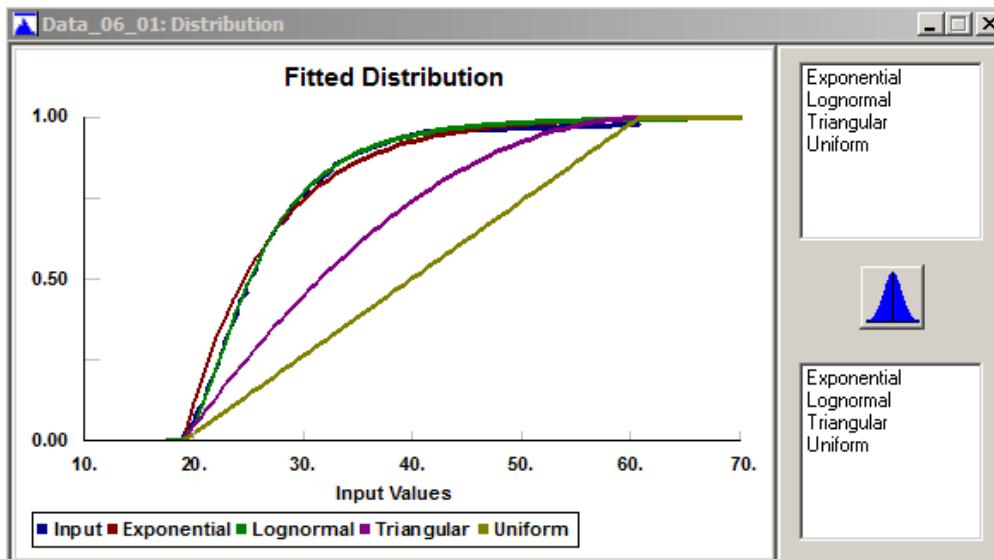


図 6.11 Stat::Fit による経験分布への適合 CDF の重ね合わせ

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 $i = 1, 2, \dots, n$ 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 を出現率から取り除く。それにより、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、 $i = 1, 2, \dots, n$ について点 $((i-1)/n, \hat{F}(X_{(i)}))$ をプロットして、 $i = 1, 2, \dots, n$ について $(i-1)/n \approx \hat{F}(X_{(i)})$ であるかどうかを調べる。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 $(1,1)$ への対角線として示される。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、あるいは 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 Fit→Result Graphs→PP Plot を選べばよい。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、よい当てはまりを示している。また三角および一様分布の P-P プロットは対角線から非常に離れており、よい適合とは到底いえない状態である。指数分布も不合理とまではいえないが、対数正規ほど良好ではない。

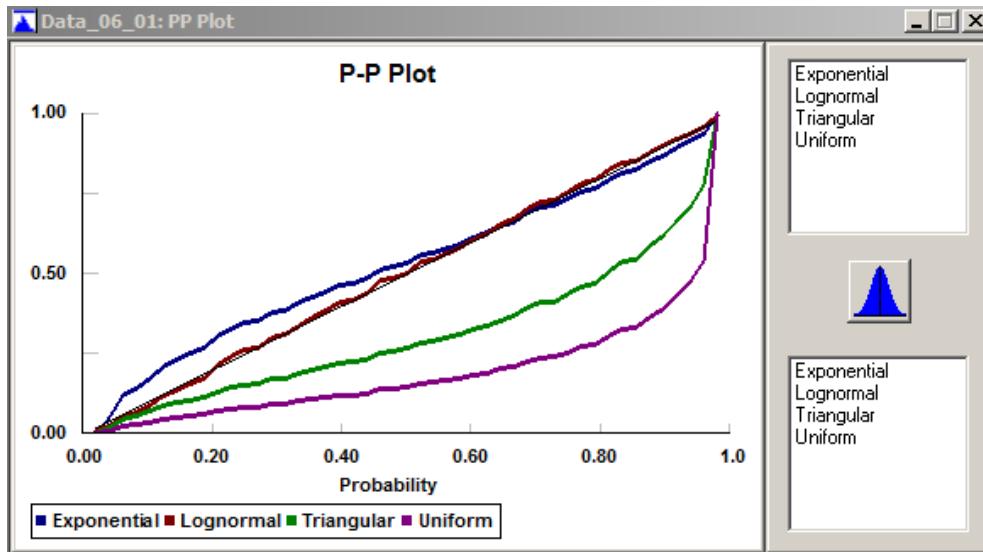


図 6.12 Stat::Fit の P-P プロット

6.1.5.4 Q-Q プロット

P-P プロットは、 $i = 1, 2, \dots, n$ において、 $(i-1)/n \approx \hat{F}(X_{(i)})$ か否かを確かめるものであった。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、 $i = 1, 2, \dots, n$ について $\hat{F}^{-1}((i-1)/n) \approx X_{(i)}$ が得られ、 \hat{F}^{-1} と \hat{F} は互いに対称である。ここで、左辺は適合分布の分位数である（任意の確率もしくは出現率（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式が利用できない場合、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、 $i = 1, 2, \dots, n$ について点 $(\hat{F}^{-1}((i-1)/n), X_{(i)})$ をプロットすると、おおよそ対角線を示すことになる。ただし、今度は $(0,0)$ ではなく、 $(X_{(1)}, X_{(1)})$ と $(X_{(n)}, X_{(n)})$ 間の対角線となる（実際には、Stat::Fit は順序を逆にし、点 $(X_{(i)}, \hat{F}^{-1}((i-1)/n))$ をプロットするが、直線を目にするとという事実は変わらない）。さて、これらの点の x 軸および y 軸が分位数（それぞれ経験および適合）であるため、このプロットは分位数プロット、あるいは 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。対数正規および指数の 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。三角および一様分布の 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 プロットは分布の内部において乖離がより強くなるのに対して、Q-Q プロットは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。

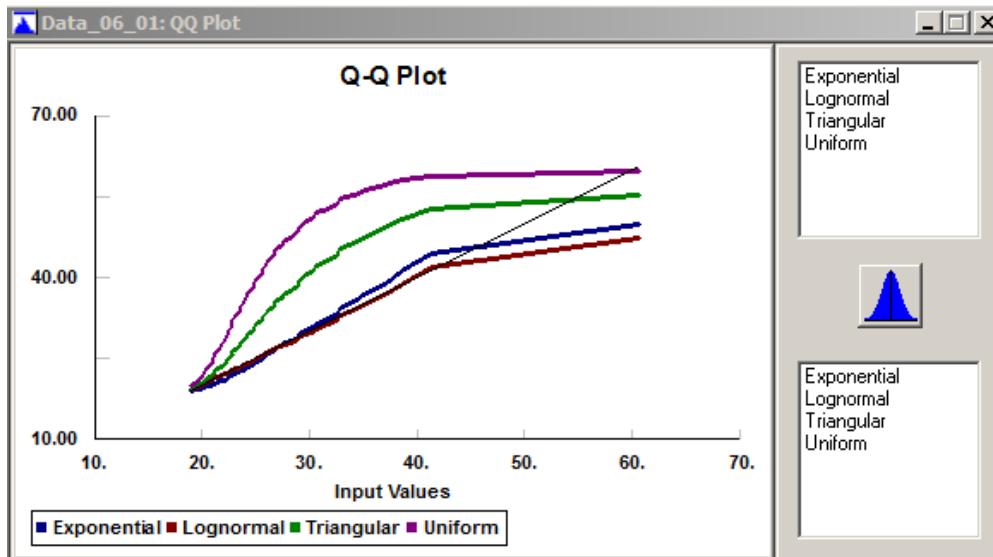


図 6.13 Stat::Fit の Q-Q プロット

6.1.6 分布の適合に関する問題

この節では、観測データに分布を適合しようとしたとき、頻繁に発生するいくつかの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。違う場合はどうするか

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、IID（独立同分布；Independent and Identically Distributed）。論理的観点から言うと、次の 2 つの仮定から構成されており、6.1.3 項ないし 6.1.5 項で扱った手法が妥当であるためには、いずれの仮定も保た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、疑わしくなる。なぜなら、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、もしくは「物理的な」因果関係や明らかに統計的な相関関係を持つからである。
- ・ 同一分布 (ID)：データを生じさせる確率分布あるいは確率過程が、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やり方でデータ収集中に条件が変わる場合や、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, \dots, X_n を観測値とするが、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 項ないし 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。このデータにおいては、後者が成り立ってしまっているように思われる。

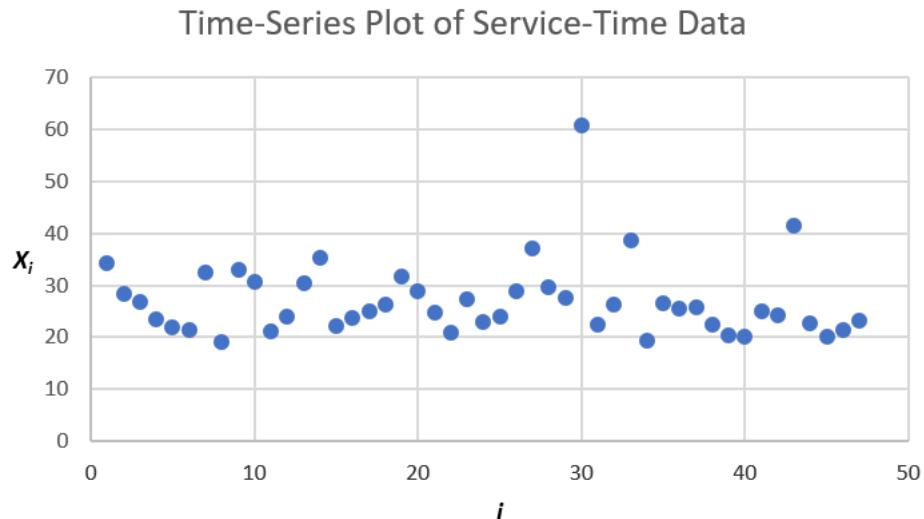


図 6.14 47 のサービス時間の時系列プロット

他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 あるいは lag-1）と呼ぶ）間の非独立性を検出するものとして散布図がある。図 6.15 に 47 のサービス時間について、 $i = 1, \dots, n - 1$ 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、 $i = 1, \dots, n - 1$ 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り）、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。もし、 $k > 1$ のラグにおける非独立性を検討したければ、 $i = 1, \dots, n - k$ のペア (X_i, X_{i+k}) に対する散布図を描けばよいだろう（Stat::Fit では作成できないため、Excel で工夫する必要がある）。

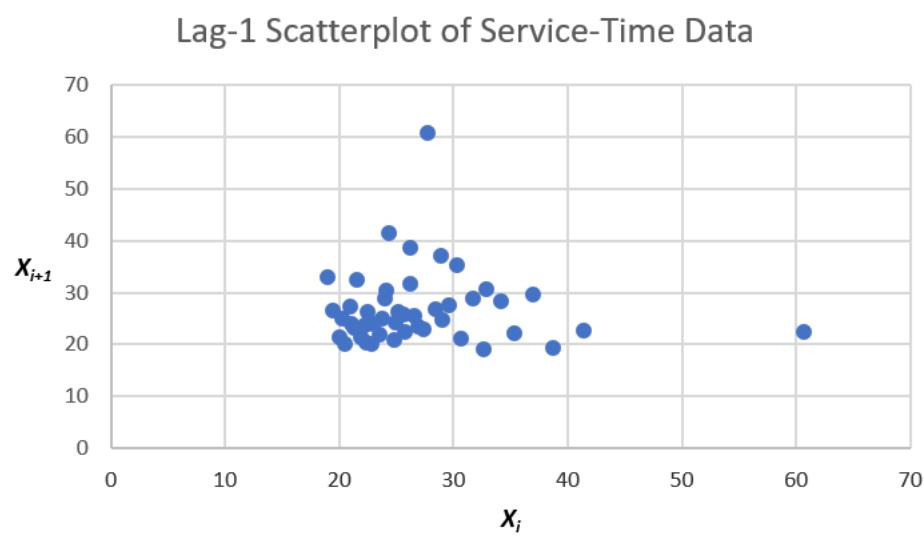


図 6.15 47 のサービス時間の散布図（ラグ 1）

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に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ないし3行だけ(観測時間順にラグを表現するために)シフトしたデータである。

	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 Data_06_02.xlsの自己相関行列に対するデータ整理

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。それから入力範囲として\$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行は除いてある)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(ここでは、[出力先]を\$G\$37とした)。図6.17に結果を示す。

Autoorrelation Matrix Through 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 Data_06_02.xls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数値であり、またすべては極めて小さい(相関は常に-1から+1の間の数値をとる)。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、非常に小さい数値である。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、これらの自己相関に関する

る推定量が、統計的に十分に 0 から離れているか否かについて、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なされる(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。

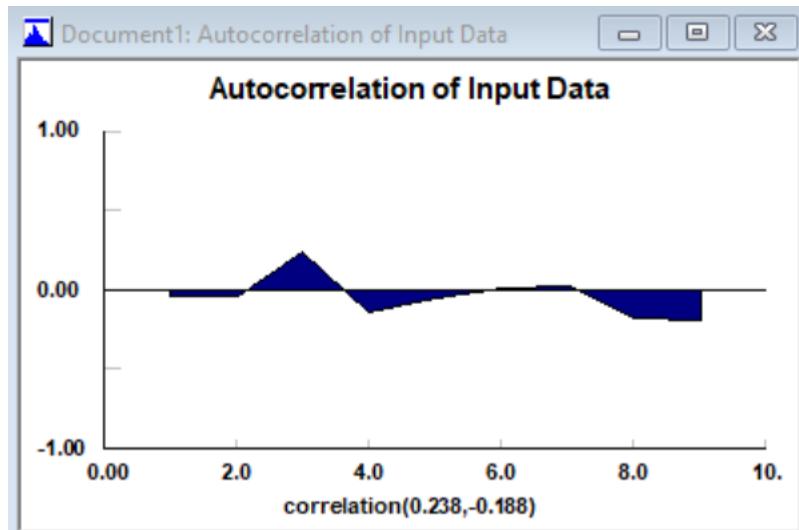


図 6.18 Data_06_02.xls の Stat::Fit によるコレログラム

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もあり、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(上昇連あるいは下降連)について、ある連の長さおよび度数の期待値を計算し、データ内の他の連の長さおよび度数と一致するか否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)。

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、反対する強いエビデンスがないため、少なくとももっともらしいといえる。データに関して独立性を明らかに示すものが場合、モデルリングを始めるか、あるいはさらにデータ収集をするかは、モデル作成者に委ねられる。シミュレーションにおける重要な特殊なケースとして、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 ではうまくモデルングできる。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ルングすることは困難であり、本書の範囲を超える内容である。興味のある読者は、時系列に関する書籍を熟読してほしい(たとえば Box et al. (1994))。そこでは、さまざま自己回帰過程の適合に関する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サポートされていない。

これまで「I」(独立性)の仮定に焦点を合わせてきたので、ここからは「ID」(同一分布)の仮定を考える。この仮定を外れるのは、異なる訓練およびスキル水準にある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で、いくつかの鮮明なモード(ピーク)が検出されることがある。この場合の対処法は、どのオペレータから得られたサービス時間であるかを追跡し、サービス時間について異なる確率分布を指定した、別々のサーバリソースとなるようにシミュレーションモデルを変更することである。そして、訓練あるいはスキルに応じたサービス時間について、別々の適切な確率分布を用いればよい。他の例としては、病院の救急部門のシミュレーションがある。そこでは、患者がそれぞれ異なる症状の緊急度(重症度)で到着し、さまざまなサービス時間は緊急度に依存するため、異なる緊急度には別々のサービス時間の確率分布を必要とする。また、

確率分布が時間によって変化するようなとき、「ID」の仮定から外れることになる。この場合、**変化点**を（システムに対する物理的な知識やデータ自身から）識別し、異なるデータ期間において別個の確率分布を用いるように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 1 つのケースであり、物事はいつもスムーズにいくわけではない。あなた（および Stat::Fit）が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、あなたの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だろうか？いや、それは反対に、分布の「標準的な」リストがデータを収めるエントリを持つことに失敗したことを意味する（結局データは、サービス時間のように、観察される現象についての「真実」である。その責任は、データをそれ自体に合わせるための分布にある）。

このケースでは、Stat::Fit は、経験分布を生成することができる。それらの分布は、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、そこから変数を生成することができます。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。これは、Windows のクリップボードに一連の対応する v_i 以下の変量を生成する v_i と c_i の組をコピーする。 v_i はデータ値の i 番目に小さいものである。そして 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。実際に、Simio にこの情報をコピーした後にどうなるかは、離散的または連続的な分布かどうかによって異なる。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、希望に従って、Discrete (離散) または Continuous (連続) のいずれかを選択する。

- 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だけ一連のペアが続き、そして、累積確率 c_i で各々の v_i の値を生成する（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ことになる）
- 連続の場合、Simio 式は Random.Continuous(v1, c1, v2, c2, ...)となる。そして、生成された CDF は、各々の (v_i, c_i) を通過して、直線で結ばれる。したがって、 v_1 の 0 から最大（最後）の v_i の 1 まで上がっていく、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では、下方（最小データポイント）と、上方（最大データポイント）の双方で有限の分布である。すなわち、どちらの方向にも無限の裾を持っていないことに留意してほしい。

6.1.6.3 データがない場合

明らかにこれはよい状況ではない。人々が通常していることは、そのシステム（または類似システム）に詳しい「専門家」に、たとえば合理的に起こりうる最小値と最大値を尋ね、その後、一様分布を指定する。その一様分布が両極端に多くの重みを与えすぎていると感じた場合は、代わりに三角分布を使うことができる。三角分布は、状況に応じて、範囲の中央付近をとる最頻値 (PDF のピークであり、必ずしも平均ではない) を使う。このような対処法でモデルを動かせるなら、実際には感度分析ツールとしてモデルを利用すべきであり（この後の「データの適切な量」を参照してほしい）、出力に対してどの入力がもっとも重要か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関するいくつかのデータを今後必死に集め、分布への適合を試みるべきである。6.5 節で議論するが、Simio の Input Parameters 機能により、主要な出力パフォーマンス尺度に

対するもっとも適切な入力に関する目安が提供される。

6.1.6.4 データが「多すぎる」場合

もう1つの極端な状況は、観察した現実のデータが何千もの、非常大きなサンプルサイズがあるときである。一般に多くのデータを持つことは喜ばしいことである。非常に大きなサンプルサイズであるとき、適合度仮説検定は、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）。したがって、大きなサンプルサイズは、グラフによりビジュアルで確認するには完全に合理的にみえるかもしれないが、適合度検定においてはすべての分布の適合を棄却する可能性が高い。このような場合には、すべての仮説検定と同時に適合度検定も、決して完璧というわけではないことを忘れてはいけない。そして、サンプルサイズが大きいことで適合度検定が棄却した分布であっても、見た目でよさそうなものであれば、その分布を利用すべきだろう。

6.1.6.5 データの適切な量

サンプルサイズに関して、人々はよく、分布に適合するのに必要な実データはどれくらいであるかという疑問を持つ。当然のことながら、そのような質問に答えられる普遍的な答えはない。しかしながら、それに対処する1つの方法は、いくつかの重要な出力が入力分布の変化にどの程度敏感であるか測定するために、感度分析ツールとしてシミュレーションモデル自体を使用することである。「しかし、入力分布が何であるかについて分からなければ、モデルの構築さえできない」と考えているなら、厳密にいえば、あなたは正しい。しかし、入力の確率分布に適合するために実データを収集する前であっても、最初に架空の（任意で、おかしい分布でない）入力分布を利用すれば、まずモデルを構築することができる。ほとんどの場合には、簡単な一様や三角の入力分布で合理的な推測を行うことができ、そして、誰かはそのシステムに精通しているため、それらの入力分布のパラメータを尋ねればよい。それから、どれが出力にもっと多くの影響を及ぼすかについて確認するために、これらの入力分布を変化させる。それらは重要な入力分布であるので、あまり出力に影響を及ぼさないような他の入力分布ではなく、感度分析のためのデータ収集に集中した方がよい。「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機能により、モデルのさまざまな入力に対して実世界のデータをどの程度収集すればよいのかについて、目安が提供される。

6.1.6.6 正しい答えは何か？

最後のコメントは、分布の指定は、厳密な科学ではないということである。2人の人物が同じデータセットを用いて、異なる分布を出すことがある（両方は完全に合理的である）。合理的とは、すなわち、データへのじゅうぶんな適合は提供するが、異なる分布となるものである。そのような場合、第2の基準（たとえば、分布の平均に変化をもたらすパラメータ操作における容易さ）を考慮するかもしれない。ある分布が他の分布よりも操作が容易であるならば、異なる入力分布の平均を試したい場合（たとえば、平均して20%高速の窓口を設置したらどうなるかなどの場合）に備えて、簡単なものにするかもしれない。

6.2 入力のタイプ

6.1節で单变量の分布の適合を説明したので、少し離れて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。確定的と確率的、およびスカラーと多変量と確率過程の2×3区分での2次元で、これらを分類する。

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に、单

なる定数である。これは、シミュレーションの実行中に変更されることはない。もちろん、キオスクが時間内のランダムな点で故障し、その後、ランダムな時間を継続して修理を受けなければならぬ場合は除く。確定的入力のもう1つの例は、歯科医院における患者の事前予約された到着時間が挙げられる。

しかし、これまでに、歯科医院の予約に遅れて（または早く）行ったことはないだろうか？ 到着時間は、より現実的に（確定的に）スケジュールされた時間としてモデル化され、その上、到着の遅れは正で早い到着は負となる**偏差 RV**としてモデル化されるかもしれない（たとえすべてのケースでなくとも、もし患者が平均してちょうどの時間に来たと仮定したなら、期待値は0になる）。これは RV を伴う確率的入力の例である。実際に、このような状況では、待ち行列シミュレーションをスプレッドシートで行った Model 3-3 のように、到着時間間隔の分布を特定することが一般的である。Simio は、Source オブジェクトで Arrival Type に Arrival Table を用いることによって、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与えられた入力は、ほぼ間違いなく確定的または確率的のどちらかである：

- ・ チェックインキオスクからセキュリティまでの空港における旅客の歩行時間。距離は、誰にとっても同じであるが、歩行速度は明らかに異なる。
- ・（確定的な）注文数とは対照的な出荷における実際の商品数。
- ・ 打ち抜き加工工場の機械で材料をプレスし、カットする時間。もしロールオフシートメタルが一定の割合で引っ張られ、プレス金型の昇降が一定の割合である場合、これは確定的に近いかもしれない。この RV を利用して小さな変動をモデル化しようとするかどうかは、モデル構築の詳細レベルで扱う問題である（ちなみに、詳細なモデルが常によりよいモデルにつながるものではない）。
- ・ 軍用車両の「定期的な」メンテナンスを行うための時間。計画は確定的である一方、多くの問題が発見された場合には、必要な追加（およびランダム）時間をモデル化しようとする。

確定的または確率的のいずれを用いてモデル化するかは、モデル化の意思決定である。実際のシステムと一致する方を採用すべきであることは明らかであるが、シミュレーション出力に対する影響を考慮する必要もあるだろう。12章では、所与の入力について、計画フェーズでは確定的として、リスク分析フェーズでは確率的とする方法について検討する。

6.2.2 スカラーと多変量と確率過程

入力が1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、**单変量**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1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ある。これは、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、我々の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单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单変量の入力は、部品を処理するための時間、そしてその後の検査のための時間を表すだろう。救命救急では、このようなスカラー单変量入力は、患者の連続した到着時間間隔、年齢、性別、保険の状況、診察室でかかった時間、診断コードと処置コード（マイカーで病院の診察室へ行く、または救急車を呼んで病院へ行くなど）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の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、それらを見なければならない。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より現実的な入力とすることができることがある。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、所定の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すことができる。また、関節炎を患っている幼児や妊娠による合併症をもつ初老男性のような、救命救急で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されたなら、その両方が発生する可能性がある。また別の例として、消防署のシミュレーションを考える。そこでは、特定のコールで送られる消防車と救急車の数に正の相関がある（大規模な火災は、それぞれ数台が要求されるが、小規模な火災では、おそらくそれぞれ1台のみが必要になるだろう）。これをどのようにモデル化し、1つのプロジェクトで実行するかは、Mueller (2009) を参照してほしい。そのような状況のいくつかは、実に論理的にモデル化することができるが（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）、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することができる。このような非独立性が、現実世界のシステムで存在する場合、シミュレーションの出力結果に影響を与えることがあるため、それを無視してモデル全体で独立したものとしてすべての入力を生成することは、誤った結果につながる。

入力確率ベクトルを特定する1つの方法は、6.1節のように、最初に座標の各単変量確率変数を一度に1つの分布に適合することである。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にされることがある）。それから、統計書で述べられる通常の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してほしい。ただし、同時分布するものが正規確率分布である場合は除く。

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）を考えるだろう。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（それが特定の到着時間であり、特定のサイズである）。シミュレーションへの入力ストリームとして使うために、非常に一般的な時系列モデルに適合させるロバスト法については、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、たとえば、Kelton (2006) や Kelton (2009) 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、時間の経過と共に著しく変化する。例として思い浮かぶのは、1日にわたるファストフードのレストランや1年にわたる（フルシーズンの）救急室、1日にわたる都市高速道路の交通のようなものである。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発生す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、高速公路の通行を（間違って）モデリングすることを想像してほしい。それは、午前3時あたりも含まれており、またラッシュアワーの混雑をひどく軽視していることとなる(Harrod and Kelton (2006) で、この方法により生成された誤差についての数値例を参照してほしい)。

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、必要に応じて速度関数 $\lambda(t)$ が高くなると、到着の平均数は、時間間隔の間に高くなる（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とに注意してほしい。これは平均=1/ λ で IID の指數関数の RV を到着間隔時間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、たとえば 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 1 つの簡単な方法は、**区分的定数関数**による方法である。ここでは、一定期間（議論を具体化するために高速道路の交通の例で 10 分としておく）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 分の間隔の終わりではおそらく異なるレベルで上下に移動することがある。各 10 分の期間内に一定の割合を仮定するのが妥当であると分かるためには、そのシステムの知識を持つ必要がある。それぞれ 10 分の期間にわたって割合のレベルを指定するには、よい精度でその期間中の到着をカウントし、期間ごとに複数の平日にわたってうまくいけば平均をとる（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であるが、Leemis (1991) に示されているように、それには良質な理論的背景がある。

Simio は、エンティティがその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、病院の救急部門で、区分定数の到着率によって非定常ポアソン到着過程を実装する一連の例を紹介する。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**の単位としなければならないことに注意してほしい。到着率のデータが、それぞれ 10 分の期間中の到着数であるとすれば、**時間当たり**の到着率に変換するには、最初に期間ごとにレートの推定値を 6 倍する必要がある。

6.3 亂数生成法

すべての確率的シミュレーションは、**乱数**を「生成」する方法のルーツから始める必要がある。乱数は、シミュレーションでは特に、一様かつ連続的に 0 ないし 1 の間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。それは理想論であり、文字通りには実現できない。その代わりに、独立しており一様に分布しているように見える 0 と 1 の間に多くの値の列を生成する数値計算アルゴリズムが開発されている。生成された乱数は、特定の証明可能な理論上の条件を満たす（極めて長い間に自身を繰り返さないなど）と同時に、一様性と独立性に対する統計および理論上の総合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。

実際には、よい RNG を構築するための多くの研究がなされている。それらの研究は、ほとんどの人が思っているよりもはるかに困難なことである。（過去のものとなったが）1 つの古典的な方法は、**線形合同法** (Linear Congruential Generator ; LCG) と呼ばれる方法である。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。

$$Z_i = (aZ_{i-1} + c)(\text{mod } m)$$

ここで a 、 c 、 m は慎重に選択する必要がある非負の整数定数 (a と m は 0 より大きい) であり、シード値 $Z_0 \in \{0, 1, 2, \dots, m - 1\}$ を指定することとする。ここで $\text{mod } m$ は、 $(aZ_{i-1} + c)$ を m によって除し、この除算の余りを Z_i とすることに注意してほしい。 m で割った余りなので、各 Z_i は 0 と $m - 1$ の間の整数になる。0 と 1 の間をとる乱数 U_i が必要となるので、 $U_i = Z_i/m$ とする。代わりに、 $m - 1$ で割ることができるが、実際には m は非常に大きいので、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 ('はじめに' で説明したようにダウンロード可能) の一部を示している。これは、最初の 100 個の乱数を生成するために LCG を実装している。Excel の組み込み関数 MOD を用いて、列 F に剩余を返している。

	A	B	C	D	E	F	G	H
1	線形合同法による乱数生成							
2	a:	17	i	Z _i	U _i			
3	c:	8	0	7	n/a			
4	m:	23	1	12	0.5217			
5	Z ₀ :	7	2	5	0.2174			
6			3	1	0.0435			
7			4	2	0.0870			
8			5	19	0.8261			
9			6	9	0.3913			
10			7	0	0.0000			
11			8	8	0.3478			
12			9	6	0.2609			
13			10	18	0.7826			
14			11	15	0.6522			
15			12	10	0.4348			
16			13	17	0.7391			
17			14	21	0.9130			
18			15	20	0.8696			
19			16	3	0.1304			
20			17	13	0.5652			
21			18	22	0.9565			
22			19	14	0.6087			
23			20	16	0.6957			
24			21	4	0.1739			
25			22	7	0.3043			
26			23	12	0.5217			
27			24	5	0.2174			
28			25	1	0.0435			
29								

図 6.19 線形合同法による乱数生成 (Model_06_01.xls)

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、少し深く見ていくと、シードは $Z_0 = 7$ であり、 $Z_{22} = 7$ に再び現れ、その後に生成された数列は、ちょうど始まりからまったく同じ順序で繰り返される。シードがこのように早く繰り返されるのを防ぐために、 Z_0 を他の値に変更してほしい。しかし、すぐに不可能だと気づくことになる。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のである（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に見えるような整数を得るのは難しく、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。

いくつかの比較的良好な LCG が考案され（つまり、 a 、 c および m の許容値が発見され）、1951 年の Lehmer (1951) における開発後、長年にわたり成功裏に利用された。しかし、 m は明らかに大きい値であった（多くの場合、 $m = 2^{31} - 1 = 2,147,483,547$ であり、約 2.1 億または 10^9 のオーダー）。しかしながら、コンピュータの速度は、1951 年以来大きな発展を遂げたため、高品質の RNG に対して LCG はもはや候補として挙げられなくなってしまった。1 つには、 10^9 の周期の LCG でさえ、今日ではすべてのサイクルを一般的なパーソナルコンピュータにおいて、わずか数分で実行可能であるためである。したがって、他の異なる方法が開発してきたものの、多くは未だ内部でモジュロ剰余演算を用いているため、より長い周期がよりよい統計的性質（独立性と一様性）を示すとみら

れている。ここではそれらを解説しないため、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・ツイスター (Mersenne Twister, Matsumoto and Nishimura (1998) または開発者のウェブサイト www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html 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 つの憂慮すべき点がある。それは、乱数の不足および低品質の乱数生成である。

Simio に実装されるとき、メルセンヌ・ツイスターは、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。それは、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) にアクセスできない一方、そうする必要がない。もし望むなら、分布の特性を表す拡張的なパラメータとして、使用する乱数列を指定することができる。たとえば、6.1 節で Stat::Fit でサービス時間データを適合し、シフトした対数正規分布のために、(デフォルトの乱数列 0 ではなく) 乱数列 28 を使用したいとき、`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? 1 つの理由は、複数のシナリオ (たとえば、異なる工場のレイアウト) を比較しているとき、出力の違いはレイアウトの違いによるものであり、乱数による違いではないと、より確信を得たい場合のためである。別々の乱数列をモデル内の各入力分布に専用のものとすると、工場レイアウトのすべてのシナリオをシミュレートするとき、異なるシナリオ間の様々な入力に対して乱数を同期できる。それにより、各レイアウトのシナリオにおいて、同じときに同じジョブが到着することとなり、そのジョブに対する処理要件などもシナリオ間で同一とすることができる。この方法により、代替シナリオ間の異なる結果の説明に関して、少なくとも部分的に「乱数列の違いによる影響」を取り除くことができる。

このように乱数を再利用することは、いくつかある分散減少法の一種であり、Banks et al. (2005) または Law (2015) のような一般的なシミュレーションの文献で説明されている。この分散減少法は、複数のシナリオで同じ目的のために同じ (共通) の乱数を使用することを試みるため、共通乱数法と呼ばれる。直感的に魅力的に見えることに加え、実際に共通乱数には確率的な背景がある。 Y_A および Y_B が A および B のシナリオに対して同じ出力パフォーマンス RV (たとえば、総システム内時間) であるとすると、その後の比較においては、 $Y_A - Y_B$ 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、それぞれの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 となるだろう) よりも、出力における差 $Y_A - Y_B$ 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。様々なシナリオについて複数の反復実行を行う場合、Simio は各シナリオのそれぞれの反復実行において、使用する乱数列の同じところから始めるように調整する。これにより、共通乱数法を活用した同期が、最初の反復実行から保たれる。また、Simio で複数のシナリオを実行する場合でも、各シナリオでは、利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・ツイスターの場合にはシードベクトル) 用いる場合、もちろん正確に同じ「ランダム」な乱数列を得る。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ことにより、正確に同一の数値結果を得られる。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であると直感的に考えてしまうシミュレーション初心者に驚きを与える。しかし、同一の実行において複数回、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。実際には、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事実は、（たとえば、デバックにおいて）極めて有効である。

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、乱数（0 と 1 の間に連続一様に分布する）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と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、少なくとも Simio がサポートするおよそ 20 の分布を生成するために、内部に備えられている。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のために乱数発生法を説明してきた。3.2.3 項では、整数 1000, 1001, …, 5000 上に一様に分布した離散確率変量を生成する必要があった。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必要とした。そして、3.3.2 項では、指数分布から連続確率変量を生成する必要があった。それぞれのケースにおいて、分布固有の方法を考案し、離散一様の場合には、少なくともその方法がなぜ正しいのかについての合理的な洞察を行った（その他のケースは本節で言及する）。

いくつかの一般的な原則があり、乱数から希望する入力分布の確率変量に変換する方法を考え出す際の目安となる。おそらくもっとも重要な方法は、CDF の逆関数法であろう。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）を探す方法である（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想起してほしい。つまり、 $F_X(x) = P(X \leq x)$ のとき、RV X が CDF F_X を持つ。

はじめに、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 について解くことを試みるものである。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、分布に依存するのであるが、式を単純に解ける場合とそうでない場合があるのである。解は（単純であるか否かに問わらず） $X = F_X^{-1}(U)$ で表される。ここで、 F_X^{-1} は F_X の逆関数である。

なぜこのようなことをするのか？ すなわち、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 が連続一様に 0 と 1 の間に分布していることにある。それが少なくとも非常に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。 $[0, 1]$ の任意の区間を取る場合、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 は以下の通りである：

$$f_U(x) = \begin{cases} 1 & 0 \leq x \leq 1 \text{ の場合} \\ 0 & \text{その他の場合} \end{cases}$$

なお、RV がある区間に落ちる確率は、その区間にに対する PDF 下の面積である（図 6.20 参照）。

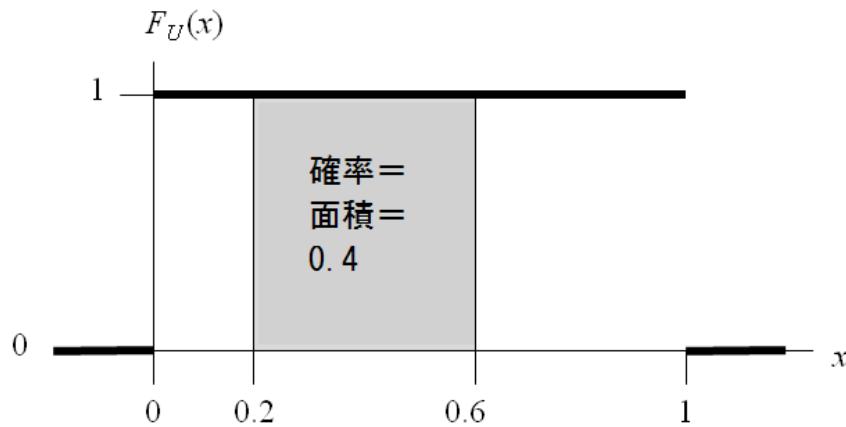


図 6.20 0 と 1 の間の連続一様分布の PDF

特に、0 と 1 の間の任意の値 w について、 $P(U \leq w) = w$ である。そのため、「生成された」変量 $X = F_X^{-1}(U)$ から、 X の範囲で任意の値 x 以下となる確率は、次のようになる：

$$\begin{aligned}
 P(F_X^{-1}(U) \leq x) &= P(F_X(F_X^{-1}(U)) \leq F_X(x)) \\
 &= P(U \leq F_X(x)) \\
 &= F_X(x) \\
 &\quad \text{(両辺に増加関数 } F_X \text{ を適用)} \\
 &\quad \text{(逆関数の定義より)} \\
 &\quad \text{(} U \text{ は } 0 \text{ と } 1 \text{ の間に一様に分布するため、} \\
 &\quad F_X(x) \text{ も } 0 \text{ と } 1 \text{ の間をとる)}
 \end{aligned}$$

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。すなわち、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（この乱数は常に 0 と 1 の間にあるため、任意の CDF（これも 0 と 1 の間をとる）に対するプロットの縦軸内にいつも「適合」する）、対応する生成された変量 X_1 および X_2 が、CDF $F_X(x)$ の範囲内の x 軸にとられる（図の CDF では、ガンマあるいは対数正規分布のように、すべてが正の数の範囲をとる）。したがって、逆 CDF 法は、図を用いて縦軸に乱数をプロットし、それから CDF 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 が常に正であるため、右側に読み取る）、 x 軸に垂線を下ろすことにより、生成された変量を得る。

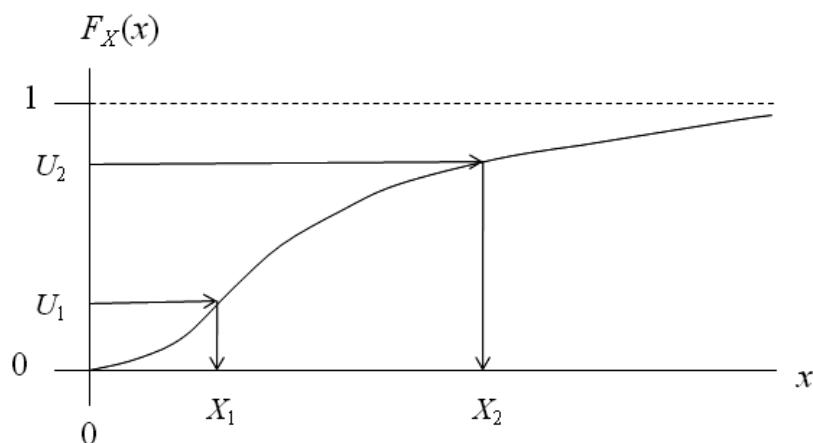


図 6.21 連続型の確率変量生成に対する逆 CDF 法

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じであるため）。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（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はより高密度で、PDFも高い（PDFが密度関数と呼ばれる所以である）。つまり、逆CDF法は、希望するRV X の分布にしたがって、乱数の一様分布を「変形する」方法であるといえる。

例として、3.3.2項と同じ分布を利用し、 X が平均 $\beta > 0$ の指指数分布を持つものと仮定する。多くの確率統計の書籍で述べられるとおり、指数分布のCDFは次の通りである：

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

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として閉じた形の式であったため、すべてがうまく行った。さらに、指数分布のCDFも、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) 簡単に逆関数にすることができた。他の分布では、(たとえば正規分布のように) CDFとして閉じた形の式を持たず、逆関数にすることができないものもある。また、CDFに閉じた形の式を持つ分布であっても、解析的に逆関数を得られないものもある (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、実施するために求根アルゴリズムのような数値解析手法を用いる場合がある。

離散のケースでは、CDFが連続的でないことを除いて、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、これを実施する場合には一般に、段に対する適切な「上昇」を発見するために探索を伴う。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必要がある。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、 $X = x_3$)。実際には、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、離散型の逆CDF法であり、探索というよりも式として、より効率的に実施できるものである。特定の分布に対する上記のような特別な「操作」は一般的であり、場合によっては逆CDF法と等しい方法になることがある。

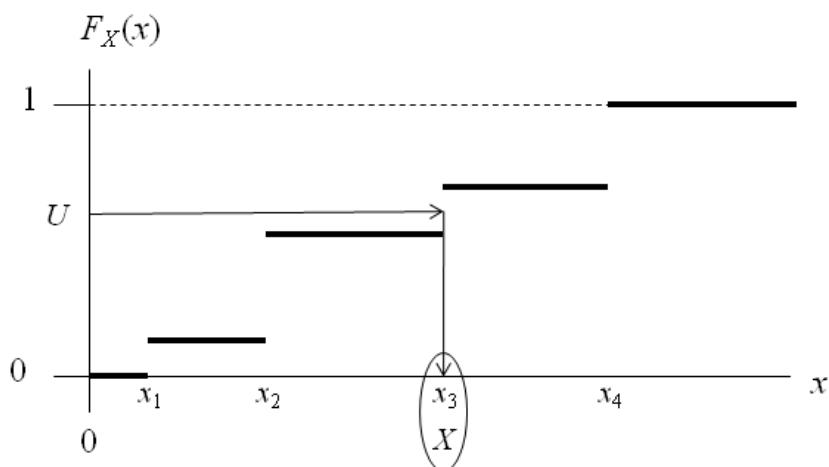


図6.22 離散型の確率変量に対する逆CDF法

逆 CDF 法はある意味で、最高の確率変量生成法だが、唯一の方法ではない。高速、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。この話題に関する概要は、たとえば Banks et al. (2005) あるいは Law (2015) を参照してほしい。また、Devroye (1986) では非常に詳細にわたる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関し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。これらの方針について議論することは、本書の範囲を超えていたため、前掲の参考文献が多くの状況において示唆に富むだろう。6.2.3 項で指摘したとおり、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は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) 分あるいは $1 + \text{Random.Lognormal}(0.0694, 0.5545)$ 分と入力していた。Model 6-2 では代わりに、4 つの入力の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。なお、このモデルのすべてのノードは、縮尺に合わせて描かれた Path で接続されている。したがって、このモデル（そして縮尺に合わせて描かれた Path を用いるその他のモデル）では、長さが変更されれば、結果が異なることがある。

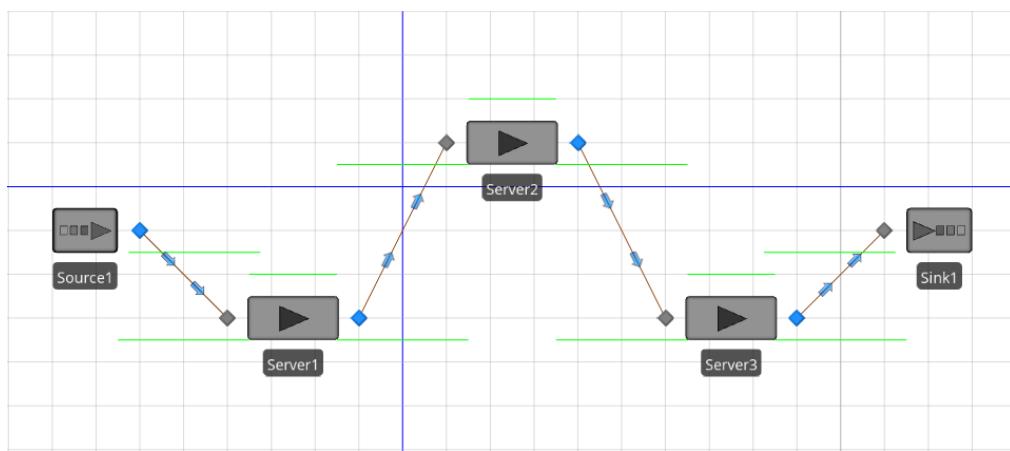


図 6.23 Model 6-2 : Input Parameters を用いる 3 つのサーバ

Model 6-2 における設定済みの Input Parameters の定義を図 6.24 に示す。なお、Input Parameters 定義ページは、Model オブジェクトの Data タブ上にある。メインウィンドウは上下に分かれている。上部には定義された 4 つの 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（Expression タイプのパラメータに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 PTimes2 (Server2 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は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として EntityArrivals が、Server1、Server2、Server3 の処理時間として、それぞれ PTimes1、PTimes2、PTimes3 が用いられている。

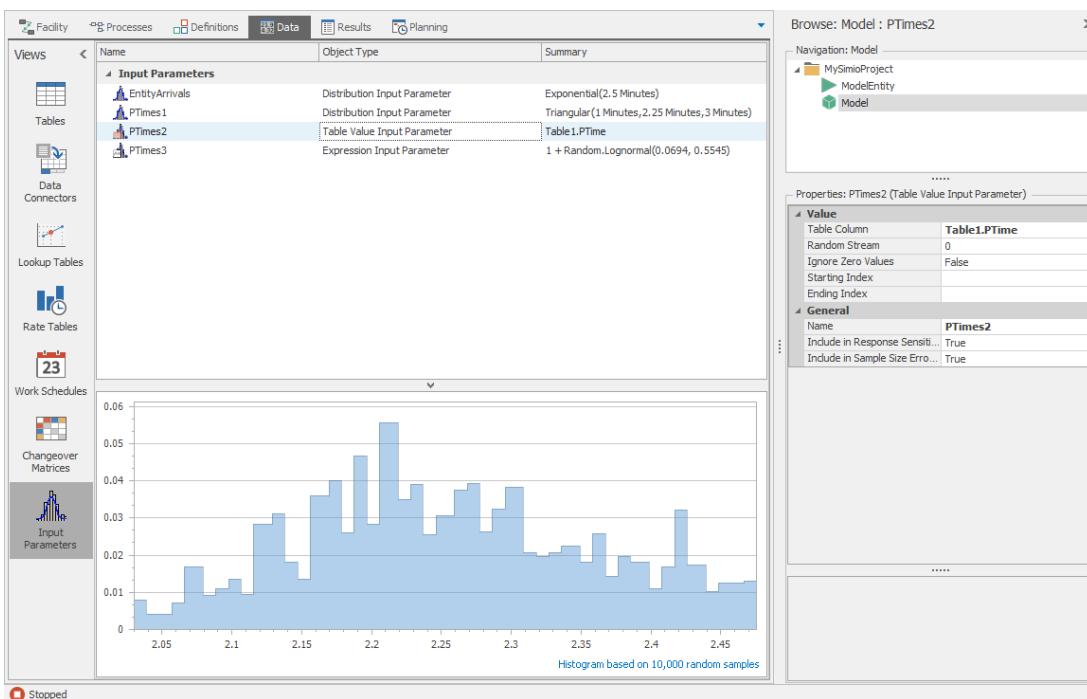


図 6.24 Model 6-2 の Input Parameters 定義

Simio は、Input Parameters として、現時点で 3 種類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 6.25 に、Model 6-2 の PTimes1 という名称の Input Parameter を示す。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 の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際に、これらの値がおおよそ意図した範囲と形状をしているかどうか、簡単に確認する手段をモデル作成者に提供することである。プロパティ Number of Data Samples は 300 と指定されている。これは、この分布に当てはめるために、300 の実世界の観測値を利用したことを意味する。このプロパティは 6.5.2 節で詳説する。

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。シミュレーション中に抽出を行う観点から、確率分布の Input Parameter を用いることは、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、運用上は同義であり、本章すでに議論した。しかし、このようにして Simio Input Parameter として定義することにより、モデル内の複数箇所で再利用する能力を得られる。また、6.5.1 項および 6.5.2 項で述べるとおり、感度分析やサンプルサイズ誤差の推定にも利点がある。

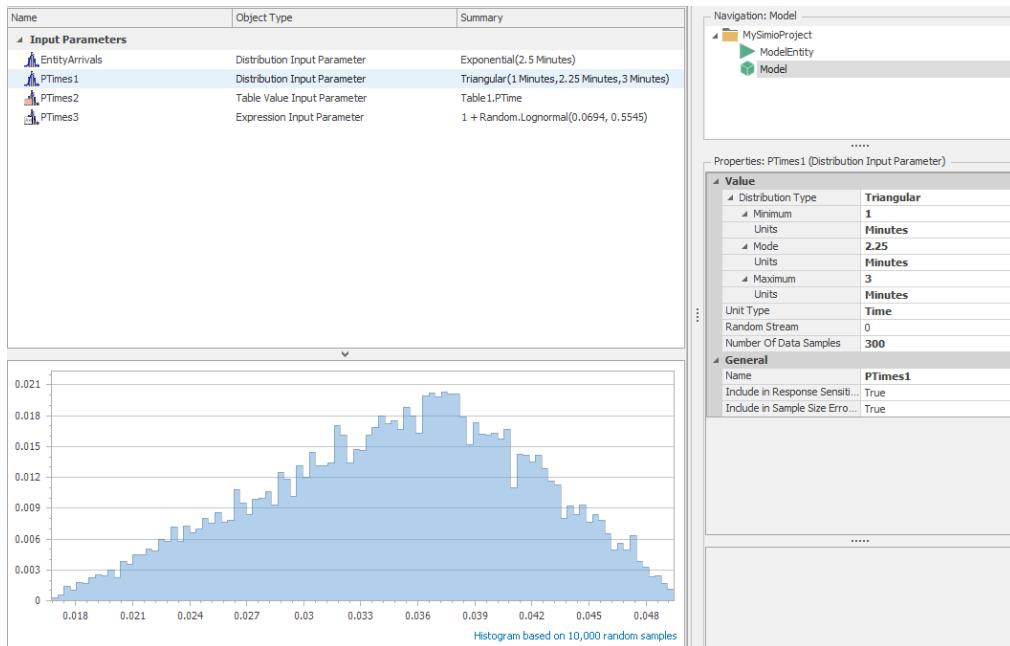


図 6.25 Model 6-2 の PTimes1 Input Parameters 定義

- 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。図 6.24 に、Model 6-2 の PTimes2 という名称の Input Parameter が示されている。プロパティ Table Column には、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）。このパラメータのヒストグラムは、指定された Table 列のデータから生成される。

Input Parameter に Table Value タイプを指定すると、このヒストグラムに対して、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、技術的には、テーブル列にある明らかなデータポイントのすべてが取りうる値となる離散（連続ではない）分布からの観測値のいずれかと常に等しい。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）で割ったものとなる。Model 6-2 の Table1 の列 PTime をみると、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 $17 / 300 = 0.0567$ となることがわかる。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）における最大の値であり、2.48 より大きな値は生成されないため、モデル作成者にとって重要か否かによらず、モデルの文脈に依存して、分布の右裾の確率が重要かどうかを考える必要がある。

つまり、この種の Input Parameter を用いる場合、分布の適合あるいは適合度検定は行われない。ある研究者たちが提案しているように、この方法の可能性（おそらく特に分布の適合が、受け入れられるデータの適合を示す分布を生まなかった場合）は、データへの不十分な「適合」を懸念せず、テーブル列にある実世界の観測値のサンプルサイズが適度に大きいため、生成される正しい範囲の値をカバーすると信じられる（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では右裾が重要となる）、データセットの最大値で生成される値を制限することは、モデルの妥当性に問題を生じさせるだろう。すなわち、（実世界のデータで観測された最大値を超えるような）極めて大きなサービス時間が生成されないため、出力尺度に下方のバイアスがかかることになる。

- 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 $1 + \text{Random. Lognormal}(0.0694, 0.5545)$ 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 の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 6.26 参照）。または、プロパティ名を右クリックして、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た。

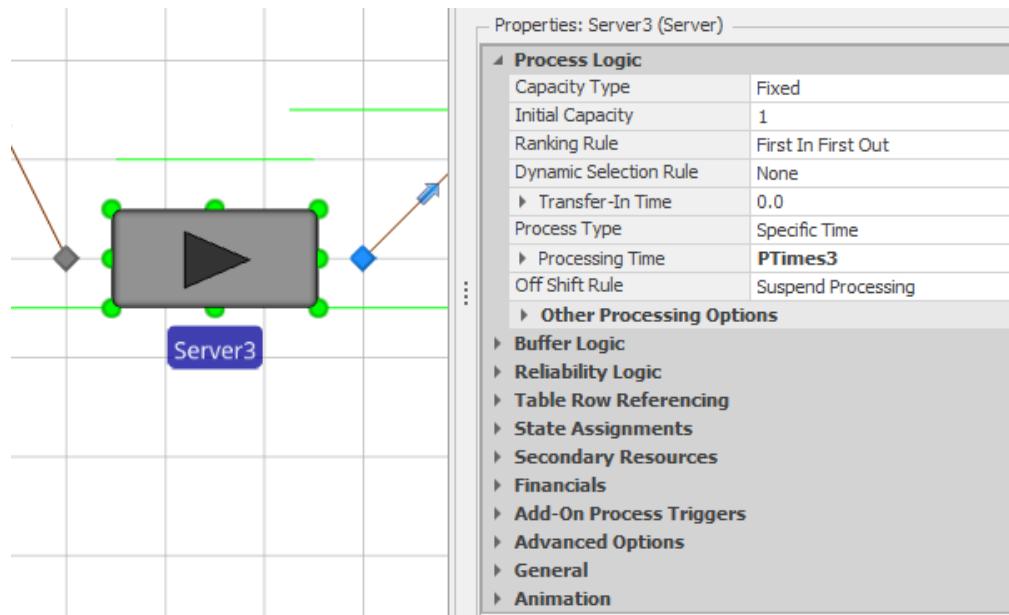


図 6.26 Model 6-2 における Server3 の Processing Time への PTimes3 の設定

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になるだろう（各入力のモデル応答に対する重要度を判断できる）。Simio のサンプルサイズ誤差推定分析は、信頼区間にに基づく方法を利用して、実験応答の不確実性において期待される（すなわち不確実な）Input Parameters の包括的な影響を推定する（6.5.2 項）。

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）関連付ける。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、この 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十分な自由度を確保する必要がある。しかし、この Response Sensitivity 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。Response Sensitivity 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 の応答 TIS に対する Response Sensitivity 分析の Tornado Chart ビューを図 6.27 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、逆に右に伸びていれば、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ティップスが現れ、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るために用いられたデータを見ることができる。

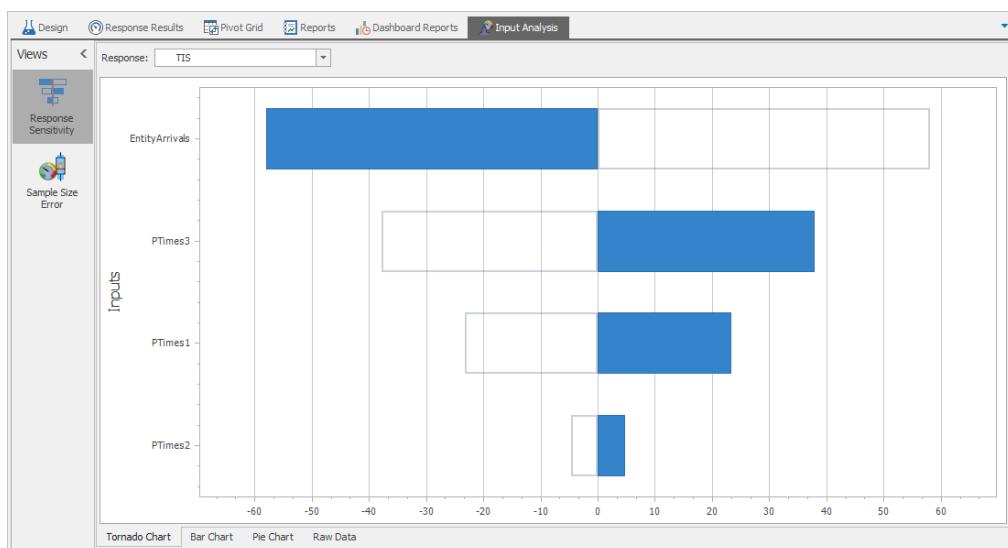


図 6.27 Model 6-2 の TIS に対する Response Sensitivity 分析

Tornado Chart から、応答 TIS は PTimes3 (Server3 の処理時間) に対して正の方向に（他の 3 つの Input Parameters よりも）比較的高い感度を示していることがわかる。つまり、PTimes3 が増加するにつれ、TIS も比較的に著しく増加するといえる。また、TIS は EntityArrivals に対して負の方向に比較的高い感度を示している（すなわち、EntityArrivals が上昇すると、TIS が比較的じゅうぶんに下降する）。さらに、PTimes1 (Server1 の処理時間) に対して正の方向に比較的まずまずの感度を示している。最後に、PTimes2 (Server2 の処理時間) に対して比較的感度はよくない。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、再びより多くの実世界のデータを収集しようとする場合、Tornado Chart により、PTimes3 と Entity Arrivals に対してより多くのデータを集め、よりよい推定量を得るために最大の努力を払うべきである。次いで PTimes1 についてもデータを集める価値はあるが、PTimes2 に対してはさらなるデータを収集する必要はないだろう。なお、これは応答 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んで（あるいは Bar Chart や Pie Chart すべての応答を一覧して）、同様の分析を行う必要がある。

前述のとおり、図 6.27 にあるような水平の青色の感度バーにマウスを合わせると、応答 TIS への回帰として予測される、適合した線形回帰係数 $\hat{\beta}$ の推定量について、数値結果は次のようになる：

- EntityArrivals : -58.036
- PTimes3 : 37.897
- PTimes1 : 23.258
- PTimes2 : 4.709

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に留意することが重要である。このモデルでは、4つすべての入力と応答 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）モデルは、入力 PTimes1 が 1 分増加すると、出力 TIS が 23.258 分増加すると予測する。同様に、EntityArrivals が 1 分増加すると、予測では TIS が 58.036 分減少する（係数が負であるため）。TIS の単位をたとえば時間に変更するには、PTimes1 の 1 分の増加により、TIS が $23.258 / 60 = 0.388$ 時間だけ増加する。同様に、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_1 の推定量の単位）が変化するため、PTimes1 の 1 単位（ここでは 1 時間）の増加により、TIS がおよそ $60 \times 23.258 = 1395.48$ 分だけ増加すると予測される。このように、**数値 23.258** は、選択された応答および Input Parameters の単位内で解釈される必要がある。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 の推定量は、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 の増分（単位を変更しない限り、分）として数値的に解釈される。Input Parameters が表すもの（時間、バッチサイズ、その他）が何であれ、適合した数値的な回帰係数は、対応する Input Parameter の 1 単位増加（入力が何であれ、それ自体の単位）に起因する応答への影響（応答の単位）として解釈する必要がある。もちろん、回帰分析では常に、これはただの**限界効果**であり、偏微分における対象の**変数以外を定数として固定する操作**と同じである（他のすべての Input Parameters を現在値で固定したままにする）。その他の留意点として、線形回帰では非線形の応答（これはシミュレーションではしばしば起こりうる）に対して固有の線形近似を得られることはほとんどないため、対応する Input Parameters について実世界の観測値の範囲を離れて、拡大して推定することはできない点が挙げられる。さらに、これらの応答感度は、シミュレーションモデルから得られたこの応答が、Input Parameters の +1 単位（一度に 1 単位）の変化に対してどの程度反応するかを近似的に予測することはほとんどできない。すなわち、何か異なった結果を得ようとして Input Parameters にそのような変化を加えてシミュレーションを実際に**実行**しても、シミュレーションを**再実行**する（おそらく大規模で複雑な、多くの長い反復実行を行う）必要があるため、Response Sensitivity で推定された予測値はその瞬間のものとなる（つまり、初期のシミュレーション実行で得られた結果からは自由である）。

ところで、図 6.27 の Response Sensitivity の**兆候**は、完全に直感的なものである。応答 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過密に関するいくつかの出力尺度の 1 つである（他には、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、システムに入る連続するエンティティ間の到着時間間隔を意味し（平均到着時間間隔の逆数である到着率ではない）、到着時間間隔を増加させた（この Input Parameter について +1 の変化を与えた）場合、エンティティが現れる時間間隔を広くすることを意味するため、システム内の混雑を当然減少させる（負の変化を与える）。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）を意味するため、サービス時間を増加させれば、システムの混雑を増加させる（正の変化を与える）ことが見込まれる。したがって、Response Sensitivity の兆候（正あるいは負）はモデルの妥当性に対する簡易的なチェックとなるだろう（Simio「コード」の正しさは考慮されない）。しかし、Response Sensitivity の兆候における気づきは、モデルあるいはコードに何らかの誤りが含まれることには必ずしもならず、さらに調査が必要になるだろう（その「気づき」が、モデリングあるいはコーディングのエラーの

帰結でなければ、モデル／システムに関する新しい知見を伝えようとしている)。Response Sensitivity の兆候が直感的なもので、多分に推測が含まれることは事実であるが、兆候の重要さは概して誤っていないため、簡易的に近似的な定量化を行うことには一考の価値があるだろう。

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) に示された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、確率分布あるいはその他の確率過程からの入力実現値をとるため）不確実性をもった結果をもたらす。シミュレーション研究の重要な側面の一つとして、シミュレートされる実世界のシステムに対して、妥当で、信頼でき、正確な結論をもたらすように、出力の不確実性を考慮する（あるいは減少させる）適切な統計的手法を用いて不確実性を評価することがある。本書では概して、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（無限に多数の反復実行を行えないことによるシミュレーションサンプリングの誤差）に焦点を当てる。これには、本章で議論した手法によっ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）方法が含まれる。

たとえば、Model 6-2において、Server2 の処理（サービス）時間として、このサービス時間に関する実世界の 300 の観測値のサンプルから適合した分布に基づき、Triangular(1 Minutes, 2.25 Minutes, 3 Minutes) を指定した（図 6.25 参照）。統計的な出力解析では、出力パフォーマンス尺度の応答（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。そのバラツキを減少させるには、それぞれの長さが 500 シミュレーション時間の 100 回の反復実行よりも多く（あるいは長く）モデルの反復実行を行う必要があるだろう。しかしこれは、すべての入力確率分布（たとえばこのパラメータ（1, 2.25, 3）分の三角分布）が対象の実世界のシステムに関して、数値（1, 2.25, 3）を含むことが本当に正確で正しいことを仮定している。したがって、実際には、すべての入力確率分布およびパラメータが推定値と本当に正確で正しいという条件付きの不確実性の評価を行っている。しかし、もちろん、それらの入力確率分布（およびそのパラメータ）が、より広い意味でいうと、出力尺度と対象の実世界のシステムとの条件付きでない分散と常に本当に正確で正しいことはない。それゆえ、結果を得た際には、そこに含まれるものと対象の実世界のシステムとの間の不確実性は、所与のモデルにおける構造的な不確実性（および入力確率分布）に依存し、また当初より少なくともいくらかは誤りを含んだ入力確率分布（およびそのパラメータ）を用いらざるを得ないことに起因することになる（この例では、無限ではなく 300 のデータ点しか持たない）。Simio の Sample Size Error Estimation はこれを定量化し、包括的な無条件の不確実性を減少させるために、追加的なシミュレーション実行、あるいは追加的な実世界のデータ収集に対して、どの程度の労力を割けばよいのかについてガイドとなるだろう。

図 6.28 に、Model 6-2 の応答 TIS（平均システム内時間）に関する Sample Size Error Estimation 分析を示す。ここで、Response Sensitivity 分析とは異なり、Sample Size Error Estimation 分析には追加的なモデル実行が必要であり、実験に対する実行の一部として自動的に行われないことに留意してほしい。この分析を行うには、Sample Size Error リボンの Run Analysis を用いる。図 6.28 に示された結果は、次の 3 つの要素からなる：

- ・ **半幅拡張係数**（Model 6-2 の応答 TIS では 5.49）：有限の多数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 5.49 倍であることを意味する。
- ・ **棒グラフ**：包括的な無条件の不確実性に対して、各 Input Parameter に起因する相対的な寄与率を表す。したがって、EntityArrivals に関する不確実性が、他の 2 つの入力に比べて、もっとも大きな要因となっている。
- ・ **拡張SMORE プロット（黄色）**：入力パラメータの推定による不確実性に起因する拡張信頼区間に覆われる。

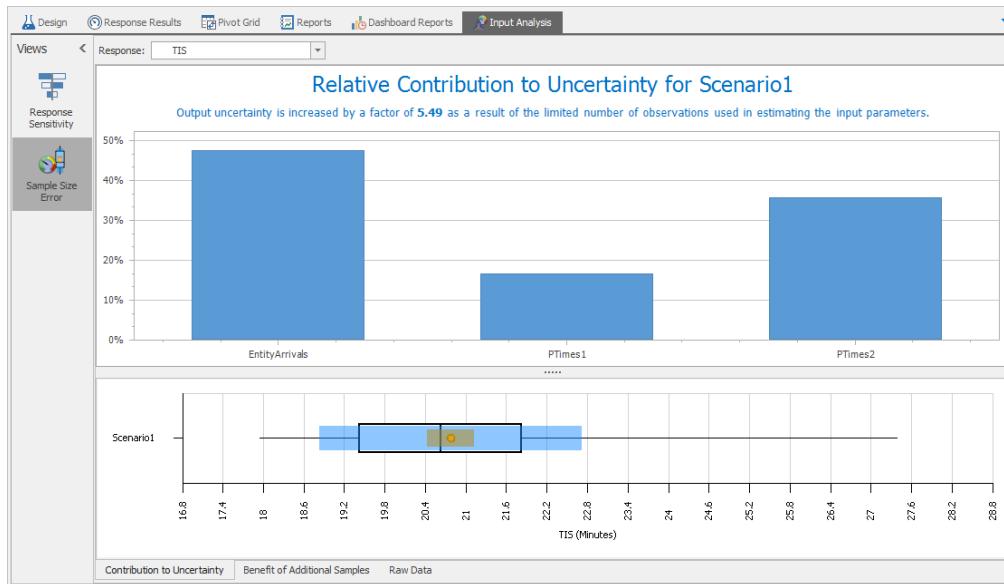


図 6.28 Model 6-2 の応答 TIS に対する Sample Size Error 分析

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 0.3537 と表示され、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として 1.9415 が得られる。拡張係数は、入力に関する不確実性（確実性のあるものとして入力確率分布を推定することに起因する不確実性）が、実験の不確実性（つまり、有限の反復実行回数の実験からのサンプリング誤差）よりも約 $1.9415 / 0.3537 \approx 5.49$ 倍であることを示している。このように、主要な不確実性の問題は、じゅうぶんな反復実行回数（あるい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、むしろ実世界から得た 300 のサンプルサイズが入力確率分布の正確な定義を行うためには不十分であることに起因する。したがって、実世界のパフォーマンス TIS の推定をより正確に行うためには、現在のシミュレーションモデルについてより多く、長い反復実行を行うのではなく、現場に戻って入力分布に関するより多くのデータを収集し、次いで Input Parameter を再定義することが肝要である。

6.6 問題

- Excel ファイル Problem_Dataset_06_01.xls ('はじめに' に記載されているとおり、本書ウェブサイトの students からダウンロード可能) には、コールセンタへの到着時間間隔(分)に対する 42 の観測値が記録されている。Stat::Fit (あるいは他のソフトウェア) を用いて、これらのデータに対して適合度検定と確率分布のプロット作成を行い、1つ以上の確率分布を適合しなさい。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ために、どの分布を推奨するか？すべてのパラメータ化の影響を考慮し、正しい Simio の式を示しなさい。
- Excel ファイル Problem_Dataset_06_02.xls ('はじめに' に記載されているとおり、本書ウェブサイトの students からダウンロード可能) には、問題 1 のコールセンタにおける通話時間(分)に対する 47 の観測値が記録されている。Stat::Fit (あるいは他のソフトウェア) を用いて、これらのデータに対して適合度検定と確率分布のプロット作成を行い、1つ以上の確率分布を適合しなさい。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ために、どの分布を推奨するか？すべてのパラメータ化の影響を考慮し、正しい Simio の式を示しなさい。
- Excel ファイル Problem_Dataset_06_03.xls ('はじめに' に記載されているとおり、本書ウェブサイトの students からダウンロード可能) には、問題 1 および 2 のコールセンタにおける問題を解決するために（最初にコールを受けつけた者以外に）必要なテクニカルサポート

の人数に対する 45 の観測値が記録されている。Stat::Fit (あるいは他のソフトウェア) を用いて、これらのデータに対して適合度検定と確率分布のプロット作成を行い、1 つ以上の確率分布を適合しなさい。シミュレーションモデルで通話に対して必要なテクニカルサポートの人数を生成するために、どの分布を推奨するか？すべてのパラメータ化の影響を考慮し、正しい Simio の式を示しなさい。

4. 実数 a と b ($a < b$) 間の連続一様分布から、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あるいはオンラインで、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たかどうかをチェックするため、マニュアルで Simio 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たかどうかをチェックするため、マニュアルで Simio 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 衡まで）計算しなさい。それから、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、また別に独立した $n = 1,000$ の IID 変量を生成しなさい。その後、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、誤差および精度の双方に関して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ならば、乱数生成法として、組み込みのもの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 関数を活用するとよい。
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。今度は、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選ぶことを意味する)。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て、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、これを達成する Simio の式を開発しなさい。
10. 3.2.3 項で、整数 1000, 1001, ..., 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 アルゴリズムとまったく同じ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。これまでのところ、データを入力する際に、多くの場合は、Standard Library オブジェクトのプロパティに直接に入力する方法でやってきた。たとえば、到着時間間隔の平均値を直接 Source オブジェクトに入力したり、パラメータとして処理時間の分布を Server オブジェクトに直接に入力していた。一部のデータに対してこのようなやり方は問題がないが、多くの場合では別の方法が必要である。とりわけ、一部の特殊な種類のデータ、たとえば、時間の経過と共に到着パターンが変化するなどのようなデータに対しては、特別な表現方法が必要である。ときにはデータ量が多く、より便利な形式でデータを表現する必要があり、外部ソースからデータを取り込むことが必要な場合もあるだろう。また、モデルを使用しているアナリストが必ずしもモデルの作成者でないこともあります。このような場合には、データをモデル内に散在させるのではなく、統合することが必要となってくるだろう。この章では、さまざまな種類のデータについて説明し、これらのデータをもっともよく表現するために利用できる Simio の構文をいくつか探究する。

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のテーブルに類似しており、プロパティを表す列とデータを表す行より構成された矩形のデータマトリックスである。1つの列は1つのプロパティを表し、そのプロパティは Simio に 50 以上もあるデータタイプから選択することができる。プロパティには、標準プロパティ (Standard Properties。たとえば、Integer (整数)、Real (実数)、Expression (式)、Boolean (論理値)、エレメントリファレンス (Element References。たとえば、Tally Statistics、Station や Material)、オブジェクトリファレンス (Object References。たとえば、Entity や Node List) がある。一般に、各行はなんらかの意味を持っており、たとえば、特定のエンティティタイプ、オブジェクト、あるいはデータの組織的側面を表すことができる。

データテーブルは外部ファイルとの間でインポート、エクスポート、さらにバインドすることができる (7.1.7 項を参照)。データテーブルへのアクセスは、順次アクセス、ランダムアクセス、直接アクセス、さらには自動アクセスも可能である。また、テーブル間に関連づけ (リレーション) をしておけば、あるのテーブルのエントリが、別のテーブル内のデータを参照できるようにすることも可能である。基本テーブルのほかに、Simio には Sequence Tables や Arrival Tables のような、基本テーブルから用途に応じて特化したテーブルも存在する。これらはすべてこの節で説明していく。

モデル実行中にファイルの読み書きを行うと、モデル実行のスピードが著しく低下してしまうことがある。これに対して、テーブルは、データをメモリ上に保持するため、非常に高速にデータにアクセスできる。Simioにおいては、テーブルの数に制限を設けていない。そして、それぞれのテーブルが持つ列の数にも制限を設けていない。行も同様であり、基本的にはシステムのメモリに制約されるだけである。テーブルはデータの整理、表現と使用だけでなく、外部のデータと連携する際にも役に立つ。

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 Tables パネルを使って定義する。Table リボンで Tables セクションの Add Data Table をクリックすることで新しいテーブルを追加することができる。テーブルを追加した後に、テーブルのタブをクリックすると、Property ウィンドウでテーブルの名前を変えたり説明文を加えたりするなど、テーブルのプロパティを設定することができる。

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ことになる。また、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ことを想起してほしい。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。

表 7.1 データテーブルの列を表す Simio Standard Properties

プロパティのタイプ	説明
Boolean	真（1 または非ゼロ）あるいは偽（0）
Color	色を設定するためのプロパティ
Changeover Matrix	転換行列への参照
Date Time	特定の日と時刻（例：7:30:00 November 18, 2010）
Day Pattern	スケジュールにおける Day Pattern への参照
Enumeration	事前に定義された一覧表に記載された値のセット
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	文字列のリストに記載された値のセット
Rate Table	レートテーブルへの参照
Real	10 進数（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によるシーケンスの指定
Sequence Table	シーケンステーブルへの参照
State	状態変数の参照
String	文字の情報（例：Red や Blue）
Table	データテーブルあるいはシーケンステーブルへの参照
Task Dependency	タスクの前後関係の定義

テーブルにオブジェクトのインスタンス、あるいは Entity、Node、Transporter または他のモデルオブジェクトのリストを参照してデータを取得させたい場合は、Object Reference を使う。同様に、テーブルで TallyStatistic あるいは Material のような特定のエレメントを参照したい場合は、Element Reference を使用する。各列の名前、位置、データ型、およびその他のプロパティを定義すること自体は、実はデータテーブルの Schema（スキーマ）を定義することになる。Simio ではユーザの必要に応じてスキーマを定義することができる。Simio では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、変換もしくは変換をせずにその外部データをインポートして使用することができる。

7.1.2 Model 7-1：データテーブルを用いた ED モデル

簡単なヘルスケアの事例を通して、前述したテーブルの諸概念を説明しよう。ここでは、ある救急診療部門（ED；Emergency Department）の事例を考えることにしよう。この医療部門では、重症度の異なる患者をどのように処置するかに関して、いくつかの既知のデータを持っている。具

体的には、表 7.2 で示されるように、4 種類の患者、それぞれの優先順位、および標準的な治療時間のデータがある。

表 7.2 Model 7-1 の救急診療病院における患者基本データ

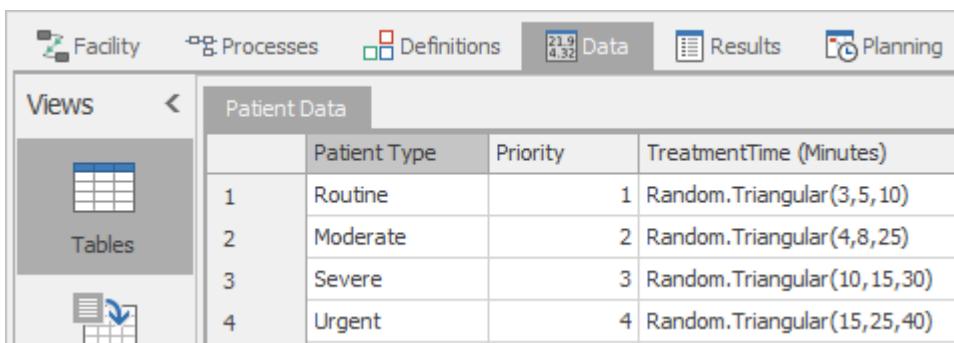
患者タイプ	優先順位	治療時間（分）
Routine	1	Random.Triangular(3, 5, 10)
Moderate	2	Random.Triangular(4, 8, 25)
Severe	3	Random.Triangular(10, 15, 30)
Urgent	4	Random.Triangular(15, 25, 40)

モデル構築の第一歩として、モデルのエンティティとデータテーブルを先に定義しておこう：

- これまで紹介した手順で Simio を立ち上げ、新しいモデルを新規作成する。まず、モデルの Facility ビューで 4 つの ModelEntity のインスタンスをドラッグする。それぞれのインスタンスをクリックして、プロパティから（または F2 キーを押して）インスタンスの名前をそれぞれ Routine、Moderate、Severe、Urgent に変更する。
- それぞれのエンティティをよく区別できるようにするために、4.8 節で記述した手順に従って、エンティティのアニメーションを設定しておこう。エンティティインスタンスをクリックしてから、シンボルライブラリをクリックする。下へスクロールし、People カテゴリから Man6 を選択する。この手順を繰り返して、すべてのエンティティインスタンスに対して Man6 のシンボルを適用する。ここでは、すべてのエンティティは同じシンボルを使用しているが、各人のシンボルのシャツを異なる色に変えることで、彼らを区別させることができるだろう。まずはシンボルがよく見えるようにズームインしておこう。Routine エンティティをクリックすると、シンボルリボンの右側に Color ボタンが表示される。Color ボタンの下半分をクリックするとカラーパレットが表示される。ここで、緑色を選択してからシンボルの男性のシャツをクリックして、色を Routine エンティティに適用させる。同じ手順を繰り返しながら、Severe の患者と Urgent 患者にそれぞれ水色と赤色を適用する。Moderate の患者にはデフォルトの色をそのまま残して使用することにする。
- 次に、データテーブルを作成しよう。リボンの真下にある Data タブを選択する。左側の Tables パネルが選択されていない場合には、それをまず選択しておく。この際、もしアクティブなテーブルがまだない場合には、Table リボンは表示されているものの、大部分のボタンは利用できないようになっている（淡色表示）。Add Data Table ボタンをクリックして空白のテーブルを追加してから、プロパティウィンドウの Name プロパティをクリックして、名前を PatientData と変更する（Simio では名称にはスペースを使用できない）。
- それから、3 つの列を追加しよう。最初の列はエンティティタイプ（オブジェクト）を参照するため、リボンにおいて Object Reference をクリックしてからリストより Entity を選択する。これで Entity Instance を持つ列を作成することができた。プロパティウィンドウの Name プロパティ（Display Name プロパティではない）で PatientType と名前を変更する。2 列目に整数を入れたいので、リボンから Standard Property をクリックし、リストから Integer を選択する。Name プロパティのところで Priority と改名する。最後に、3 列目を追加したいので、リボンで Standard Property をクリックし、リストから Expression を選択する。Name プロパティで TreatmentTime と改名する。この列は時間を表すことから、ここでは特別にいくつかの追加のステップが必要となる。プロパティウィンドウの Logic カテゴリにおける Unit Type を Time と指定してから、Default Units を Minutes と選択し指定する。
- ここまでではテーブルの構造を定義した。次にテーブルに 4 行のデータを追加し、表 7.2 で示

したデータを追加しよう。データを入力する際に、横方向（行ごとに）に入力することもできるし、縦方向（列ごとに）に入力することもできる。ここでは、行ごとに入力することにしよう。左上のセルをクリックすると、4つのエンティティのタイプが含まれたリストが表示される（この際に、リストが表示されない場合は、2つ前のステップへ戻って確認してほしい）。1行目で Patient Type から Routine を選択する。次に、Priority の列に移動し、1を入力してからキーボードの Enter キーを押すと、TreatmentTime の列に移動してくれる。ここでは、表 7.2 で示した式 Random.Triangular(3, 5, 10)を入力する。ヒント：データの値が部分的に隠れる場合、当該データの列の右端をダブルクリックすると、データを完全に表示するために必要な列幅になるまで自動的に拡張してくれる。PatientData テーブルの次の行に移動して、表 7.2 に示した残りのデータについても同様な手順で入力していく。

図 7.1 に完成したテーブルを示す。ここまで、モデルで使われる患者の関連データを定義してきた。次節では、テーブルからデータがどのようにアクセスする方法を説明する。



The screenshot shows the Simio software interface with the 'Data' tab selected. On the left, there's a sidebar with 'Views' and 'Tables'. The main area is titled 'Patient Data' and contains a table with four rows and four columns. The columns are labeled 'Patient Type', 'Priority', and 'TreatmentTime (Minutes)'. The data rows are:

	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 7.1 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テーブル名、行番号、列名または列番号を指定することにより、`TableName[RowIndex].ColumnName` もしくは `TableName[RowIndex, ColumnIndex]` の構文を用いて参照することができる。この構文を使ってデータテーブルを参照し、モデルの構築を続けよう。たとえば、`PatientData[3].TreatmentTime` という構文を用いて、タイプが Severe の患者の治療時間を参照することができる。この構文を用いれば、テーブルのセルを直接参照するのが容易である。また、多くの場合、ある特定のエンティティは、常に特定のデータの行を参照している。たとえば、前出の構文の例では、患者のタイプと対応する行を決める同時に、エンティティが常にこの行からデータを参照することも指定している。これを追跡するためには、自分でプロパティ、あるいは状態変数を追加して追跡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。その機能にアクセスするもっとも簡単な方法は、Source オブジェクトの Table Reference Assignment カテゴリの属性を設定することである。

たとえばこの例では、4種類の患者がタイプごとにそれぞれ異なる流れで到着することを設定するために、このテクニックを使用できるだろう。それでは、患者タイプを作成するためにソースオブジェクトを作成しておこう。図 7.2 は、タイプが Severe の患者の 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ことによって設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してある場合には、次のような短い構文でテーブルのデータを参照することができる：`TableName.ColumnName`。ここでは行の番号はすでに既知である。たとえば、`PatientData.TreatmentTime` を用いて任意の患者タイプの治療時間を参照することができる。

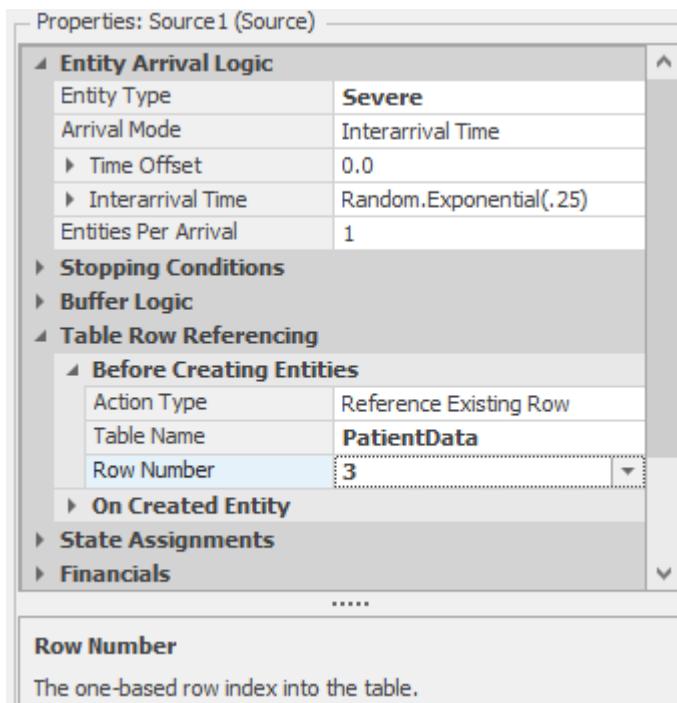


図 7.2 エンティティと特定の行との関連づけ

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、テーブルの他の側面について、もう少し探究しよう。テーブルにおいては、このモデルで行われたことと同様に、1つの行は1つのエンティティタイプに対応していることが一般的であるが、エンティティタイプがランダムに選択されるのもよくあることである。Simioでは、同じ構成でこの2つのことと一緒にこなすことができる。たとえば、1つの行にいくつかの列を追加し、各行（あるいはエンティティタイプ）のウェイトを指定することができる。一方で、`TableName.ColumnName.RandomRow`という式を用いれば、ある列における行をランダムに選択することを指定できる。

では、下記のいくつかのステップに従い、このモデルを完成させよう。

1. この救急診療部門の過去のデータによれば、患者の割合は、Routine (40%)、Moderate (31%)、Severe (24%)、Urgent (5%) である。この新情報をテーブルに追加する必要がある。DataタブのTablesのパネルに戻って、リボンからStandard Propertyをクリックし、Realを選択する。NameプロパティでPatientMixに改名する。そして、新しくできた列に上記の患者のデータを追加していく。完成したテーブルは、図7.3で示されるようになるはずである。なお、5.2節で述べたように、Simioは相対的割合によりPatient Mix列の値を解釈する。ここでは、各タイプの患者の占めるパーセンテージによって値を入力したが、確率(0.40, 0.31, 0.24, 0.05)あるいは上記数値の倍数値（たとえば、4000, 3100, 2400, 500）を入力しても構わない。
2. モデルの構築を続けよう。先の変更により、1つのSourceを用いて特定の分布に従う4種類の患者を生成させられるようになった。1つのSourceをモデルに配置し、Interarrival TimeとUnitsの値をそれぞれRandom.Exponential(4)とMinutesと指定する。図7.2では特定の行番号を用いてEntity Typeプロパティで患者タイプを指定する方法が示された。ここでは、この方法の代わりに、Simioに行番号を選ばせてから、その行のPatientTypeの値に応じてEntity Typeを決定する方法に変更する。エンティティを生成するまでにエンティティタイプを決めないといけないことから、エンティティを生成する前に、まずはテーブルの行を選択しておく必要がある。そのため、Table Reference AssignmentのBefore

Creating Entities カテゴリにおいて、Table Name を PatientData に、Row Number を PatientData.PatientMix.RandomRow と指定する。行が選択された後に、Source はどのタイプのエンティティを生成すべきかを決定するために Entity Type プロパティを参照する。ここでは、プルダウンリストから PatientData.PatientType を選択する。図 7.4 にはこれらの操作の結果を示している。

	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 7.3 Model 7-1 の救急患者データの拡張

Properties: Source1 (Source)

Entity Arrival Logic

Entity Type	PatientData.PatientType
Arrival Mode	Interarrival Time
Time Offset	0.0
Interarrival Time	Random.Exponential(4)
Entities Per Arrival	1

Stopping Conditions

Buffer Logic

Table Row Referencing

Action Type	Reference Existing Row
Table Name	PatientData
Row Number	PatientData.PatientMix.RandomRow

Before Creating Entities

Action Type	Reference Existing Row
Table Name	PatientData
Row Number	PatientData.PatientMix.RandomRow

On Created Entity

State Assignments

Row Number

The one-based row index into the table.

図 7.4 テーブルからエンティティタイプの選択

- ここまでできたら、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 7.5 に完成後のモデルのイメージを示す。

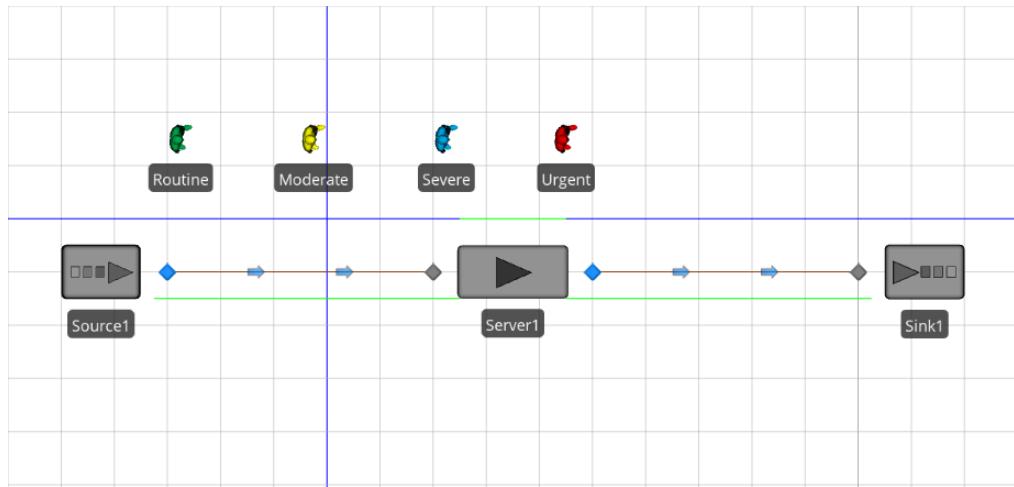


図 7.5 完成した ED モデル Model 7-1

モデルを拡張する前に、データテーブルが正確に実装しているかどうかを確認するために、簡単な検証ステップを踏んでおく必要がある。患者タイプの割合とそれぞれのタイプの患者の予想サービス時間を用いて、全体の予想サービス時間（11.96 分）を計算することができる。患者全体の到着率が 1 時間当たり 15 人であることから、定常状態では 99.64% のサーバ稼働率が期待できる。25 日間のウォームアップ期間設け、Model 7-1 を 200 日間の長さで 25 回繰り返して実行した。結果として、スケジュール稼働率は $99.67\% \pm 0.1043$ (95% 信頼区間の半幅) であった。これはモデルの検証に関する重要なポイントを示唆している：モデルの妥当性の検証は、モデルが「完成」するまで待つよりも、モデルを構築しながら検証する方が、はるかに簡単な場合がよくある。観測された稼働率は当初の予想とかなり一致していることから、患者データのテーブルをモデルに正確に実装したことを確認できた。これで、モデルを拡張するステップに移ることができる。

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の一連の順序を指定するために使用される特殊なタイプのデータテーブルである。ジョブショップ型の生産工場の場合、シーケンスは部品を完成させるために訪れなければならないステーションや機械（たとえば、研削、研磨、組立、配送）を表す。交通ネットワークの場合は、シーケンスはバス路線の一連の停留所（たとえば、メインストリート、5 丁目、9 丁目、山の手）を表すかもしれない。

シーケンステーブルは通常のデータテーブルと同様に、Data ウィンドウの Tables パネルで作成するが、その際に Add Sequence Table ボタンを使用する。このボタンを押すとテーブルが作成され、エンティティのルーティング順序を指定するための列 Sequence が自動的に追加される。通常のテーブルとシーケンステーブルとの主な違いは、この必須の列にある。シーケンステーブルの設定方法は、通常のテーブルとほぼ同様である。また、値を参照する際には、他のテーブルと同じ方法でシーケンステーブルのプロパティ（列）の値を参照することができる（たとえば、`TableName.PropertyName`）。

シーケンステーブルを設定するには、単純シーケンステーブルとリレーションナルデータシーケンステーブルという 2 つの方法がある。単純シーケンステーブルは 1 つのシーケンスを単独で使う場合に適している。たとえば、1 つのシーケンスあるいはエンティティタイプを 1 種類しか使わない場合である。一方、リレーションナルデータテーブルは、複数のエンティティが同じオブジェクトを異なるシーケンスでたどるような、複雑なシーケンスの場合に適しており、対応しやすいという利点を持っている。この 2 種類のテーブルの使い方は同じであるが、テーブル自体の設定方法が異なる。以下では、単純シーケンステーブルの設定と使用方法から説明を始めることにする。

7.1.3.1 単純シーケンステーブル

1つの単純シーケンステーブルは1つの経路計画を定義する。複数の経路計画（たとえば、いくつかのバス路線）がある場合には、それぞれの経路計画に対応するシーケンステーブルを定義する必要がある。シーケンステーブルの各行は、特定の場所に対応している。その行には通常、処理時間、優先度など、特定場所にまつわる固有のプロパティに使用される。

シーケンステーブルを作成したら、エンティティとシーケンステーブル間の関連づけを作成する必要がある（言葉を換えれば、エンティティにどのシーケンスに従うかを指示する必要がある）。これを行うには2つの方法があるが、もっとも簡単な方法は、モデルに置いたエンティティのインスタンスで行う方法である。エンティティのインスタンスの Routing Logic カテゴリには Initial Sequence というプロパティがあり、シーケンステーブルの名前をここで指定できる。エンティティはこのシーケンステーブルの1行目からスタートし、指定されたステーションを順次に訪問していく。この一連の動きは通常自動的に行われるが、必要に応じて現在の行を変更したり、従うシーケンスを変えることもできる。アドオンプロセスの SetRow ステップを使用すれば、この変更はいつでも行うことができる。

この時点では鋭い読者なら「しかし、シーケンスにおいて、エンティティは次のステップに進むタイミングをどのようにして判断するのか」と疑問を持たれるかもしれない。シーケンスにおいて、エンティティに次にどこへ行くべきか指示を与える必要がある。通常、指示を与える場所は Transfer ノードの Routing Logic カテゴリである (Standard Library のすべてのオブジェクトの出力ノードは Transfer ノードであることを想起してほしい)。ここでは、Entity Destination Type を By Sequence と指定する。ここでは下記の3つのことが起きている：

1. エンティティのシーケンステーブルの次の行が、たった今作成される。
2. エンティティの行き先は、その行で指定された移動先になる。
3. 他のテーブルで指定したプロパティ（たとえば、ProcessingTime）は、その新しい行の値を参照する。

エンティティに次のシーケンスに移動するタイミングを明確に指示しているため、必要に応じて、シーケンス間で他のステーションを訪問することもできる。その方法は簡単で、Entity Destination Type を By Sequence 以外の何かで指定すればよい（たとえば、Specific）。これは何回か繰り返し行っても構わない。次回、エンティティがあるオブジェクトを離れる際に、再度シーケンスどおりに移動させたい場合には、By Sequence を使えば、離れる場所からシーケンスを再開することができる。

7.1.3.2 リレーションナルシーケンステーブル

前述した多くの概念は、リレーションナルシーケンステーブルに関しても同様である。リレーションナルテーブルではほぼ同じ方法で使用できるが、構成は少し異なる。相違点の1つは、いくつかの異なったシーケンス（たとえば、特定のエンティティの訪問セット）を1つのシーケンステーブルに組み合わせることができるということである。エンティティのインスタンスでシーケンスを設定する (Initial Sequence プロパティを用いる) 代わりに、別のテーブルでその情報を提供することができる。これらの機能は、キーと外部キー (Foreign Key) として定義した特別な列を介して、関連シーケンステーブルをリンクすることによって実現される。このテクニックは Model 7-2 で紹介する。

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

前述した ED モデル (Model 7-1) をもう少し詳細に説明し、興味深いモデルにしていく。すべての患者はまず受付ステーションを訪問し、Urgent (緊急) タイプの患者を除いて、全員診療受

付エリアに向かう。診療受付を済ませた後に、患者はもっとも早く利用可能な診察室に行く。診察が終わったら、Routine（通常）タイプの患者は去っていくが、他の患者は追加治療を受けるために引き続き残る。一方、Urgent タイプの患者は受付の後、より深刻な病状を治療するために直接に検査室に送られる。Urgent タイプのすべての患者は病状が落ち着くまで検査室に留まり、状態が安定したら治療室に移され、離脱していく。

Model 7-1 の構築を始めた際に、最初に行なったことはエンティティの設置であったことを思い出せるだろうか。一般的には、テーブルで参照を定義したオブジェクトをモデルに配置することから構築を開始する。ここでは、もう一度オブジェクトの配置を行おう。シーケンステーブルは主に場所（より具体的にいうと、オブジェクトの入力ノード）を参照することから、それらの場所を表す Server オブジェクトを置くことから作業を始めよう。その後、新しいテーブルを作って、モデルにプロパティを追加する作業に戻ることにしよう。

1. Model 7-1 を開く。Server と Sink の間のパスは必要ないため、削除しておく。同様に、Server 1 のプロパティに移動し、Processing Time を右クリックして Reset を選択する。
2. Standard Library の Server をダブルクリックしてから、モデルの作業エリアで 4 回クリックして 4 つのサーバを追加する。そして、Esc キーを押すか、右クリックして配置モードを終了する。次に、追加した 5 つのサーバに次のように名前をつける：SignIn、Registration、ExamRooms、TraumaRooms、TreatmentRooms。1 つシンクを加えて、2 つのシンクの名前を各々 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にするために、まず既存の PatientType の列を唯一の 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。この時、TreatmentTime 列を削除しておこう（数分後に、これを新しい ServiceTime 列で置きかえることになる）。削除するには、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する準備が整った。リボンの Add Sequence Table をクリックする。シーケンステーブルのプロパティエリアをクリックして、名前を Treatments と設定する。Sequence という列が自動的に追加されることを確認できるだろう。
5. 治療の種類を識別するために、別の列を追加する必要がある。リボンから Foreign Key をクリックして、特定の治療法を一意に識別するための列を作成する。この事例においては、治療の種類は PatientData テーブル内の PatientType と正確に対応している。新しい列のプロパティに移動し、TreatmentType と命名する。これは外部参照であることから、実際に外部のどこから値を取得することを意味する。Table Key プロパティは、Treatment Type の値がどこから取得されるかを指定する場所である。プルダウンリストから PatientData. PatientType を選択する。
6. 最後に、もう 1 つの列を Treatments テーブルに追加する必要がある。それは、処理時間である。リボンの Standard Property をクリックして、Expression を選択する。すると、1 つのプロパティが作成され、当該場所に滞在する間に使用される数値や式をこのプロパティに入力する。この値は別の場所で多少異なった意味を持っているので、プロパティウィンドウに移動し、Name (Display Name ではない) をより汎用的な名前の ServiceTime に変える。この列は時間を表すので、ServiceTime のプロパティの Logic カテゴリで、Unit Type を Time に選択してから時間単位の Default Units を Minutes に指定する。
7. テーブルにデータを入力する前に、データ入力をより便利にするために、もう 1 つことを変更する。Sequences の下にある最初のセル内のプルダウンリストを見れば、モデル内のすべてのノードのリストが表示されていることがわかるだろう。このリストから任意のノードを目的地として指定することができる。選択可能な目的地として独立したノードを持つ場合

には、この方法を使えるが、このモデルではそのようなノードはまだ持っていない。オプションを利用して、表示されるリストをオブジェクトの入力ノードのみに制限することができる。実際、このオプションはオブジェクト名を表示させることにより、選択をより簡単なものにしてくれる。Sequence 列をクリックし、プロパティウィンドウに移動する。Accepts Any Node を False に変更する。今、プリダウンリストをもう一度確認すると、より単純で短いリストが表示されることがわかるだろう。

8. 次に、治療データを入力しよう。1 行目では、Sequence の下のプルダウンメニューから SignIn を選択する。次に、Treatment Type の下のプルダウンリストから Routine を選択し、Service Time 欄に 2 を入力する。以下同様に、図 7.6 に示すようになるまで行を追加してデータの入力を続けていく。

	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 7.6 Model 7-2 の Treatments を定義するための関係シーケンステーブル

9. データの入力が終わったら、Facility ウィンドウに戻ってモデルを完成させよう。
- 10.これまでの全体の議論を思い出してほしい。エンティティの出発がシーケンスに従う場合は、出力ノードでそれぞれ指定する必要がある。Source の出力側にある青色の TransferNode をクリックする。Routing Logic カテゴリで Entity Destination Type 欄を By Sequence に変更する。各出力ノードに対して、この手順を繰り返して操作することができるが、Simio はショートカットを用意してくれている。この例では、エンティティのすべての動きがシーケンスに従うので、これを一括して変更することができる。任意の青いノードをクリックしてから、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）をクリックする。これで、選択した 6 つのノードの Entity Destination Type を一度にすべて変えることができる。
- 11.使われる可能性のあるすべてのノードをパスでつなげる。この際に、もし不要なパスまで追加したとしても、作業が少々無駄にはなるが、問題を起こすことはない。たとえば、タイプ

が Urgent の患者は Registration を省略することから、Registration から TraumaRooms へのパスは使われることは決してないだろう。

- 12.これまでのところ、患者がサーバに滞在する期間（サーバの Process Time プロパティ）について言及していなかった。すべてのサーバに対する答えは同じで、特定の患者と結びつけられたシーケンスのアクティブなステップの情報を使用する。この式は、TableName.PropertyName の構文となっており、この例の場合、Treatments.ServiceTime で表現される。この式をサーバの Process Time の値として入力する。エンティティ（患者）の情報が現在のテーブルと行に関連づけられていることに留意してほしい。
- 13.一部の患者が他の患者より重要な場合がある。これは政治家とスポーツ選手のことをいいいるわけではなく、患者の重症度を指す。たとえば、重度の心臓病の発作で苦しんでいる患者がいるときに、咳やかぜの患者の治療に時間をかけたくない。前出の患者テーブルで患者の優先順位はすでに定義したが、サーバを選択する際の優先順位をデフォルトの先入れ先出し（FIFO；First In First Out）から変更する必要がある。各サーバに対して、Ranking Rule 属性を Largest Value First に変更する必要がある。テーブルの Ranking Expression 属性を PatientData.Priority で設定する。こうすることによって、Urgent タイプの緊急患者（優先順位 4）が他の患者（優先順位が 1 ないし 3）よりも優先して治療を受けることを保証できる。
- 14.このモデルのサーバは、1 つの単独の窓口ではなく、救急診療部門における 1 つのエリアを表すものである。それぞれのサーバの Initial Capacity を指定する必要がある。それぞれのサーバをクリックして、表 7.3 で示されるように Initial Capacity を変える。それぞれのサーバの上の緑のラインは現在処理中のエンティティを表す。そのキャパシティを満たすような長さになるまで、それらのラインを伸ばすことができる。同様に、サーバの左側の線は行列内で待っている患者を表す。必要であれば、それらのラインを伸ばすこともできる。モデルが実行している間でも、これらの線を適切な長さになるまで容易に調整することができる。

表 7.3 救急診療部門（ED）の各エリアにおけるサーバの数

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

上記で紹介したすべての手順に従っている場合は、モデルはもうすでに完成しているだろう。オブジェクトの配置は異なるかもしれないが、おおむね図 7.7 に示すようになるだろう（Browse ウィンドウは非表示としている）。

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。ここでは、待ち行列ネットワークモデル（図 7.8）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（ウォームアップ期間を 25 日間、実行期間の長さを 200 日間に設定し、シミュレーションを 25 回繰り返して実行したときの結果）を示している。表 7.4 から分かるように、実験結果は明らかに期待に沿ったものである。モデルの妥当性を検証することができたことから、モデルの拡張を引き続いて行うことができる。

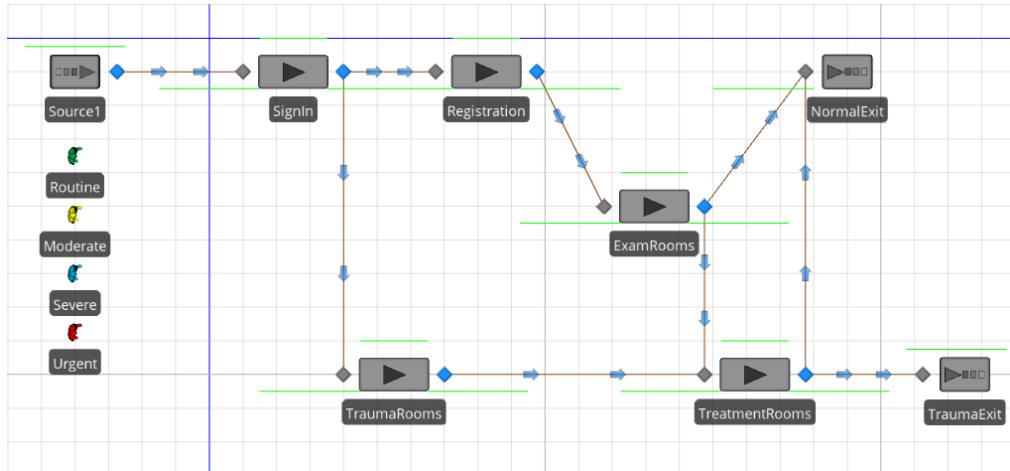


図 7.7 シーケンスを適用した後の ED モデル Model 7-2

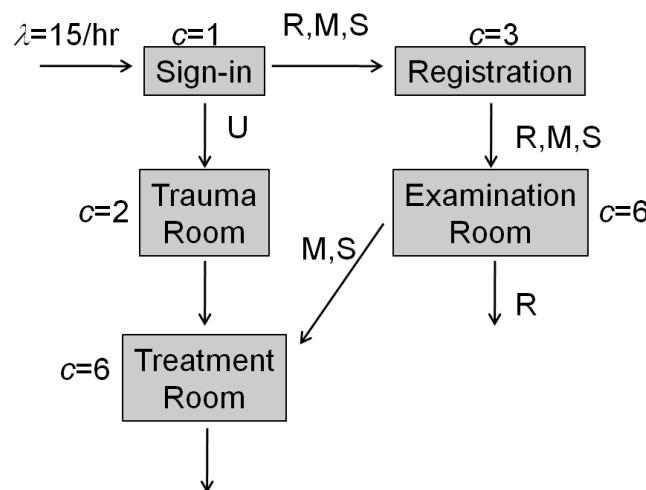


図 7.8 シーケンスを用いた ED モデルの待ち行列ネットワーク

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	0.4215 ± 0.0422
Registration	0.3358	0.3361 ± 0.0380
Trauma Rooms	0.1563	0.1571 ± 0.1005
Examination Rooms	0.5604	0.5607 ± 0.0540
Treatment Rooms	0.4032	0.4036 ± 0.0790

7.1.5 到着テーブルと Model 7-3

これまで、まだ議論していないもう 1 つの種類のテーブルは到着テーブルである。これは、Source オブジェクトの Arrival Mode プロパティと併用することで、テーブルで指定された特定の到着セットを生成する際に用いる。この到着は、確率的なもの、もしくは確定的なもののどちらでも構わない。

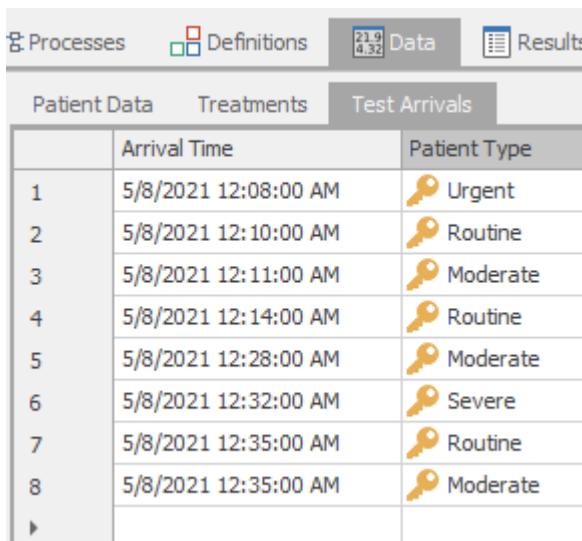
一般的な確率論的シミュレーションモデルでは、確定的到着 (Deterministic Arrivals) はあまり見られないかもしれないが、スケジューリング (Scheduling) と検証 (Validation) という 2 つ重要な用途がある。スケジューリング型の用途では、完了するべき作業が決まっており、システム目標を達成するために、項目を処理する最適な方法を決定するときに用いる。これに対して、検証のテクニックとは、既知の一連のイベントをモデルで実行して、実際の結果とモデルの結果との相違を分析することである。いずれの場合においても、到着テーブルを使えば、入荷作業や資材の配

達などのアクティビティを表現することができる。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している。

Source オブジェクトで 2 つの追加プロパティを利用することにより、同じテーブルを確率的到着 (Stochastic Arrivals) にも活用することができる。Arrival Time Deviation (到着時間差異) を用いると、指定した到着時刻前後の確率差異 (Random Deviation) を指定することができる。Arrival No-Show Probability を使えば、特定の到着が起こらない可能性を指定することができる。これらの属性を使用することによって、患者（後述）、入荷原材料および外回り配達用のトラックなどの到着予定を容易にモデル化することができる。なお、Run タブにある Advanced Options の Disable Randomness オプションを選択することによって、モデルは確定的モードで実行される。この場合、他のモデル側面と同様に、確率の機能が無効になる。

到着時刻のリストが含まれる列であれば、どんなテーブルでも到着テーブルとして使うことができる。到着テーブルの用途は、到着時間の提供に限定されない。テーブルの他の部分には、期待されるルーティングや期待される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を持たせることも可能である。それでは、この ED モデルで、到着テーブルをどのように使用するかを紹介しよう。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 Panel に移動する。Add Data Table をクリックし、TestArrivals と名前をつける。
2. Standard Property をクリックして Date Time を選び、属性名を ArrivalTime と入力する。
3. 次に、Foreign Key をクリックし、エンティティの到着タイプを決めるために、1 つの属性を追加する。このプロパティの名前を PatientType と命名し、Table Key 欄に PatientData.PatientType と入力して、Default Value 欄に Routine と入力する。外部キー (Foreign Key) を使用することにより、この列はすでに入力した他のデータテーブルにリンクされる。ここではこれ以上のことを行う必要はないが、前節で述べた Source オブジェクトの Table Reference Assignments の設定を削除することもできる。
4. さて、いくつかのサンプルデータ（例：昨日の実際の患者到着データ）を入力しよう。図 7.9 に示すデータを入力する。シミュレーションの開始時間が真夜中から始まる方が便利な場合は、そのようにしても構わない。Date Time プロパティを作る代わりに、Real タイプのプロパティを作成し、その Time と時間単位を自由に指定することができる（たとえば、Minutes）。時間のレコードは作成される前に時間順でソートされることになるため、エンティティは厳密な到着時間順になっている必要はない。



The screenshot shows the AnyLogic software interface with the Data panel selected. The table 'Test Arrivals' has three columns: 'Arrival Time' and 'Patient Type'. The data rows are as follows:

	Arrival Time	Patient Type
1	5/8/2021 12:08:00 AM	Urgent
2	5/8/2021 12:10:00 AM	Routine
3	5/8/2021 12:11:00 AM	Moderate
4	5/8/2021 12:14:00 AM	Routine
5	5/8/2021 12:28:00 AM	Moderate
6	5/8/2021 12:32:00 AM	Severe
7	5/8/2021 12:35:00 AM	Routine
8	5/8/2021 12:35:00 AM	Moderate

図 7.9 到着テーブルのサンプル

5. このテーブルを使用するために、Source で設定する必要がある。Arrival Mode プロパティで Arrival Table を選ぶ。Arrival Time Property は、到着を決定するテーブルとプロパティを指定する場所である。ここでは図 7.10 のように、TestArrivals.ArrivalTime を選択する。

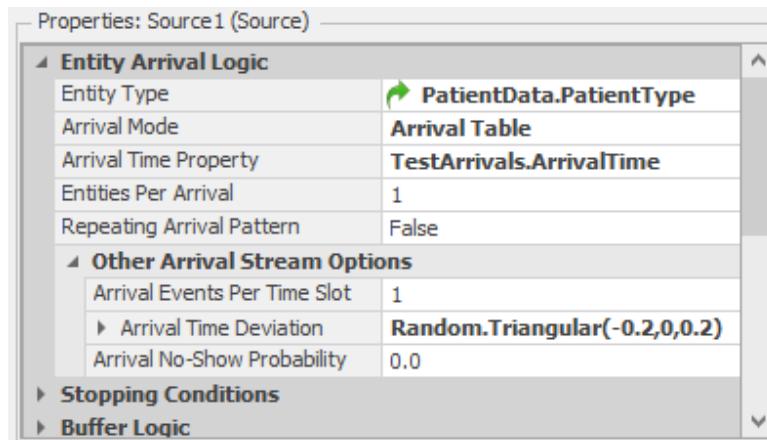


図 7.10 Source オブジェクトにおける到着テーブルの使用

6. また、Sourceにおいては、ランダムな到着時刻を指定するために、Arrival Time Distribution を Random.Triangular(-0.2, 0, 0.2) と設定する。これは、各患者が指定時刻より最大 0.2 時間早く、あるいは最大 0.2 時間遅れて（つまり予定された時間後 12 分以内に）到着する可能性があることを意味する。

7.1.6 リレーションナルテーブル

名前の通り、リレーションナルテーブル (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もつテーブルのことである。これらの関係は、**テーブルキー (Table Key)** と **外部キー (Foreign Key)** の機能を用いて形成される。Set Column As Key ボタンを使用すると、ハイライトされた列が別のデータテーブルから参照される可能性があることを示すことができる。つまり、このボタンはその列をテーブルの主キーとし、テーブル間をリンクさせるものである。そして、この列には、キー値ごとに一意のインスタンスを持つ必要がある。キーの値は重複することができない。ただし、複数の列をまとめて、それぞれが集合として一意の値を取る場合は、一つのテーブルのキーとして複数の列を用いることができる（複合キーと呼ばれる）。

他の任意のテーブル（複数のテーブルかもしれない）は、メインテーブルとそのテーブルキーを指定する外部キーの列を含めることにより、メインテーブルとリンクすることができる。リンクが作成されれば、リンクを明示的にトラバースしなくても、リンクしたテーブルの任意の列を使うことができるようになる。たとえば、Model 7-2 では、PatientData テーブルにおいて、4 つの一意の値を含む PatientType をキーとして指定した。その後、Treatments というシーケンステーブルを作成し、PatientType を外部キーの列として設定した。この場合、エンティティを PatientData テーブルの特定の行に関連付けると、そのエンティティは自動的に Treatments テーブルの一連の関連行にも関連づけられるため、Treatments.ServiceTime の値を使用することができる。

リレーションナルテーブルには、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) がある。キー列を持つテーブルでは、詳細ビューを使えば、テーブル内の特定のキー付き行を指す行のコレクションを見ることができる。行の前にある小さいな「+」記号は、詳細ビューが利用可能なことを示す。「+」を押すと、展開されたスペースに関連テーブルの部分が表示

される。Model 7-3 に戻って、これを確認してみよう。テーブル PatientData を確認してみると、PatientType の各エントリー（行）には「+」記号がついていることが分かる。タイプが Moderate の患者の前の「+」記号を押すと、図 7.11 に示された画面が表示される。

図 7.11 リレーションナルテーブルのマスター詳細ビュー

Treatments タブの下には、Treatments テーブルに関連するすべての行が表示されている。この例では、Moderate タイプの患者の治療順序に関連するすべての場所とデータが表示される。TestArrivals タブも存在することに留意してほしい。そのタブでは、TestArrivals テーブルで指定された Moderate タイプの患者の到着に関するすべてのデータが表示される。詳細ビューを閉じるには、「-」をクリックする。これらのキーは独立しており、それぞれ開いたり閉じたりすることができます。

リレーションナルテーブルは、非常にパワフルな機能を備えている。これを使えば、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ようになる。また、この機能は外部データとリンクする際にも非常に役立つ。同じ効率上の理由から、シミュレーションに必要な外部データが、リレーションナルデータベースで利用できる場合がよくある。独自に作成したデータテーブルにおいても、同じ関係を表現することができる。

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して、テーブルの価値と有用性についての理解は深まっただろう。これまで使ってきたような少量のデータであれば、手動でテーブルに入力するのが合理的である。しかし、データの量が増えるにつれて、ファイルのインポートとエクスポートのオプションを活用することを勧める。

Content リボンメニューの Save to CSV をクリックして、テーブルを CSV (コンマ区切り値) ファイルにエクスポートすることができる。この機能は、適切にフォーマットされたファイルを作成したり、既存のテーブルをファイルに書き込んで外部 (Excel など) で編集したり、バックアップを取りたりするのに便利である。図 7.12 は、先に完成した Model 7-3 を例にして、このインターフェースを示したものである。リボンの右端に、Save to CSV ボタンがある。

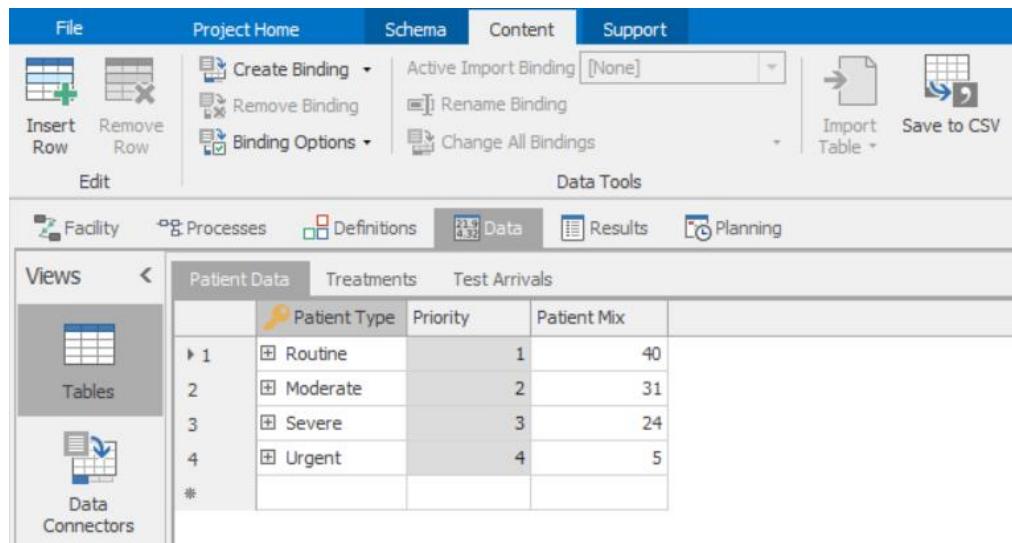


図 7.12 データ接続とバインディングのインターフェース

一方で、データ接続およびバインディング機能を使用すれば、CSV や Excel、あるいは他の一般的なデータベースファイル形式からデータをテーブルにインポート／エクスポートすることができる。データ接続 (Data Connectors) は、ファイルの読み書きに必要なファイル名とその他のパラメータを設定する。バインディング (Binding) (特に Create Binding ボタン) を使用すると、テーブルを 1 つ以上のデータ接続にバインドすることができる。どちらのボタンも図 7.12 に示されている。現在利用可能なインポートバインディングの種類を図 7.13 に示す。

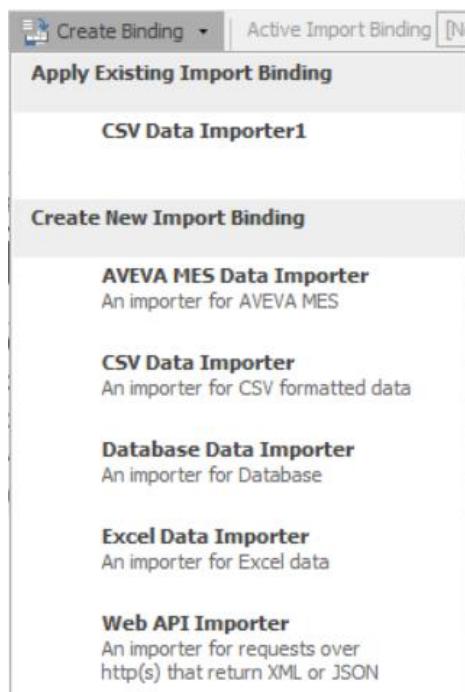


図 7.13 バインディングの種類

インポートまたはエクスポートを行う前に、まず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どのような順序で作成しても構わない。たとえば、最初に各テーブルを選択してバインドを作成し、次にデータ接続に移動して、各データ接続で使用するファイル名とパラメータを指定することができる。図 7.14 に、CSV コネクタのデータ接続のパラメータを示す。

テーブルを特定のファイルにバインドしたら、Binding Options を指定することもできる（図 7.15）。これにより、モデル実行を開始するたびに、データを手動でインポートする（Import ボタンをクリックする場合に限るが）か、もしくは自動的にインポートするかを選択できる。初期状態から現在のシステム状態まで、データ収集が頻繁に変化するものである場合は、後者が望ましいだろう。他にもより高度なオプションも利用可能である。

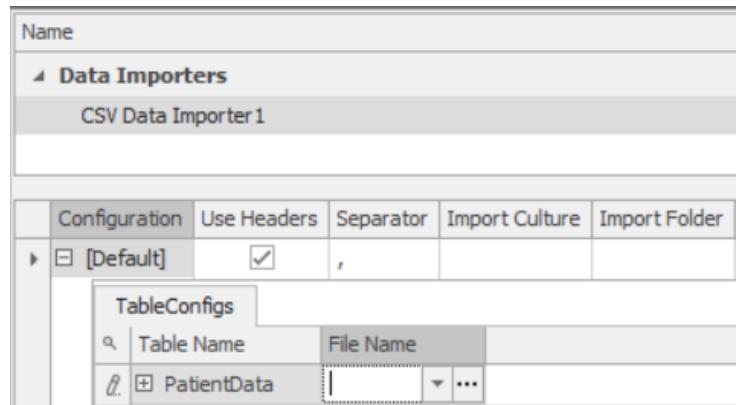


図 7.14 CSV データ接続のパラメータ

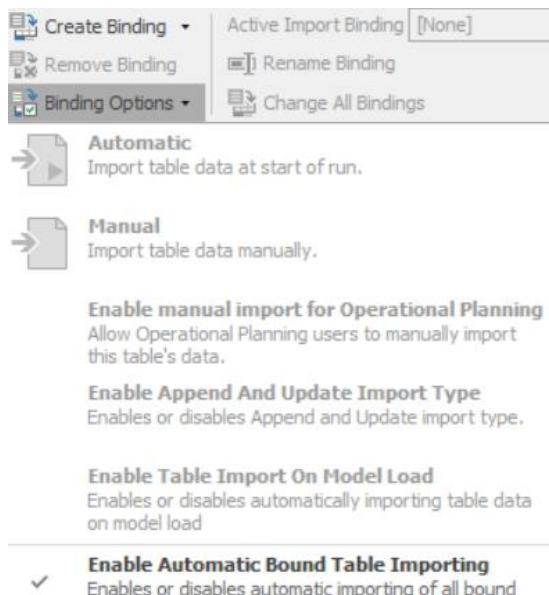


図 7.15 インポートとエクスポートを制御するバインドオプション

同様の手順とオプションは、テーブルをファイルにエクスポートする際にも利用できる。これは、Simio の外部でデータを編集したり（たとえば、エクスポート、編集、再インポート）、データテーブルの永続的なコピーを記録したり、実行結果をエクスポートする場合に、特に便利である。バインディングの設定と構成については、Simio ヘルプの Importing and Binding to Tables を参照するとよいだろう。

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。リソースは、時間の経過と共に変化する能力（たとえば、使用可能なユニット数）をもっている。5.3.3 節で述べたように、オブジェクトの能力が時間と共に変化するモデルの状況をモデリングする方法の一つは、スケジュールの使用である。多くのオブジェクトは、能力が時間の経過と共に自動的に変更できるように作業スケジュールに対応している。数値による能力は、リソースがオンシフト状態

(能力がゼロより大きい)であるか、オフシフト状態(能力がゼロに等しい)であるかを判別するために使用される。一部のオブジェクト(たとえば、Worker)においては、キャパシティは、0(オフシフト)または1(オンシフト)のどちらかである。ほとんどのオブジェクトでは、スケジュールは可変的な能力(たとえば、8時間の5、2時間の4、14時間の0)を表すことができる。

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)とExceptions(例外)の3つの主要構成要素からなる。

Day Patternは1日における作業期間を定義する。作業期間は、必要な数だけもつことができる。指定されていない期間は、非作業期間であると見なされる。SimioにはStandard Dayという名前のサンプルDay Patternが含まれている(図7.16の下部)。これは、1時間の休憩等で区切られた、休憩前後の作業期間が4時間ずつのスケジュールである。

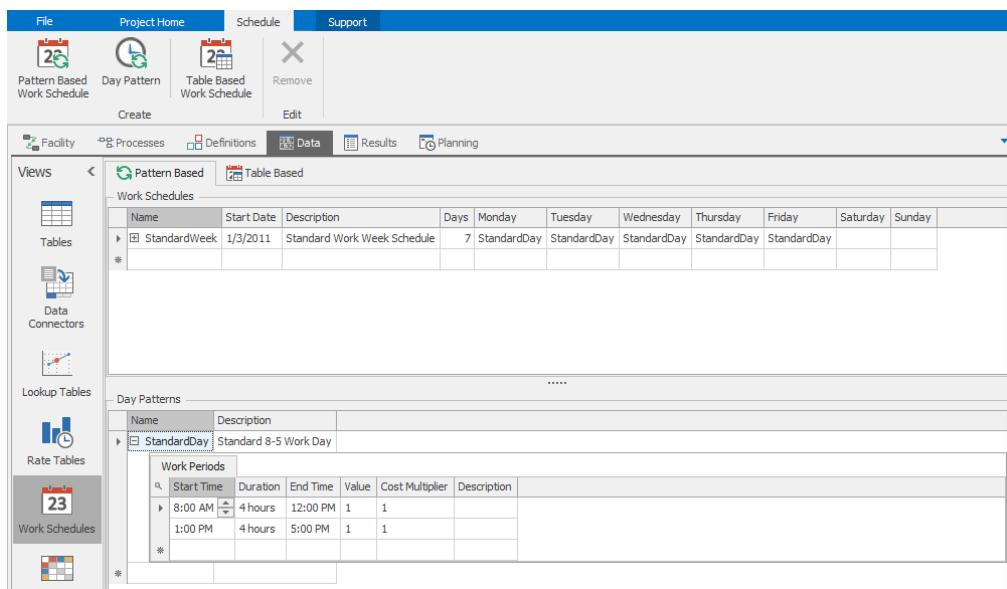


図7.16 Pattern-Based作業スケジュールのサンプル

もちろん、このサンプルを変更したり、独自に置き換えたりすることもできる。Day Patternの各作業期間においては、Start Time(開始時刻)と、Duration(継続期間)あるいはEnd Time(終了時刻)を指定する必要がある。スケジュールされる数(たとえば、リソース能力)が初期値の1ではない場合、列Valueでその値を指定する必要がある。列Cost Multiplierは、1つ以上の期間において、通常のリソース使用レート以外のコストが発生する場合に用いられる。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には、この2時間のコスト乗数を1.5(倍)として設定することができる。

Simioには、図7.16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も含まれている。作業スケジュールは、繰り返しの作業期間を構成するため、デイパターンの組み合わせが含まれている。繰り返しの期間は1日ないし28日の間で設定できる。一般的には週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いつからでも開始することができる。すべての作業スケジュールの長さが7日間で、週の同じ曜日(たとえば、いつも月曜日)からスタートする場合は、図7.16で示されるように、列ラベルは週の実際の日数となる。スケジュールが毎週異なる曜日から始まったり(たとえば、今週は月曜日から始まり、次の週には別の曜日から始まる)、スケジュールの期間が7日間でない場合は、列ラベルは単純にDay 1, Day 2, …, Day nのようにすればよい。この場合、スケジュールされていない日は、データが入力され

ることを防ぐために、灰色で表示される。

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、例外の期間に対して繰り返しの基本スケジュールを上書きする。例外は、残業や計画的なメンテナンス、休暇期間などを定義するために用いることができる。例外は、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ことができる。例外には2つのタイプがある。1つ目のタイプは、ワークデイの例外 (Work Day Exception) である。ワークデイの例外は、ある特定の日に、通常の作業パターンと異なる作業パターンを使用することを表す。たとえば、早めに退勤する時間を定義するために、August Fridays と呼ばれるデイパターンを定義し、8月のすべての金曜日にこの作業パターンを使用するように指定することができる。もう1つの例外のタイプは、作業期間の例外 (Work Period Exception) である。これは、ある特定の日付と時間帯に対して、スケジュールでの指定とは関係なく、特別に設定された値でスケジュールが動作することを指す。これは、多くの場合、長期休暇の期間、もしくは作業時間の延長期間などに用いられる。

スケジュールが大量にある場合は、パターンによるスケジュールを入力するのが面倒な場合があるだろう（特に、データがすでにデータベースに存在している場合）。そのような状況のために、Simio はテーブルベースの作業スケジュール (Table-Based Work Schedules) を用意している。これらのスケジュールは、デイパターンの場合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、およびコスト乗数の情報が、どのテーブルのどの列を参照するかを記述するだけで簡単に指定することができる（図 7.17 参照）。

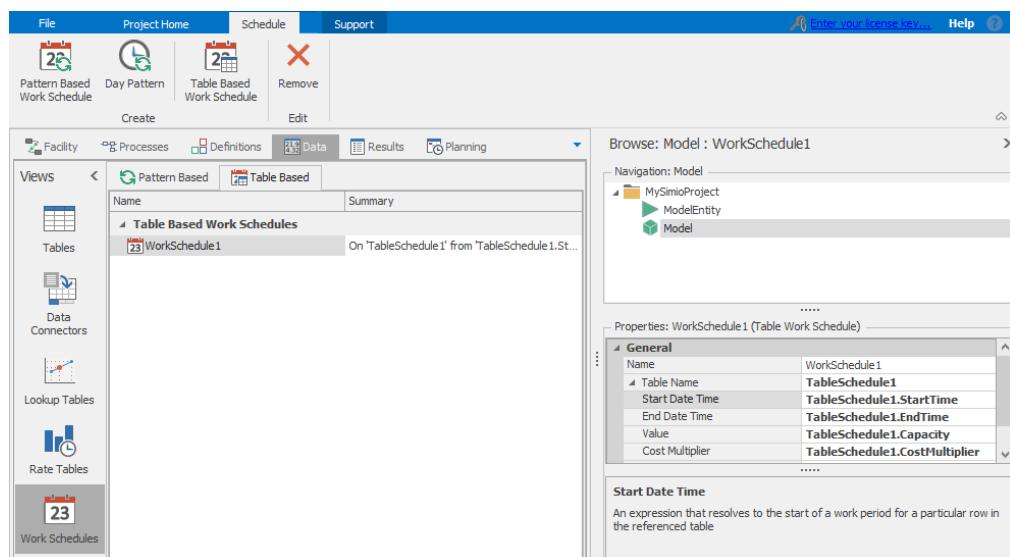


図 7.17 Table-Based 作業スケジュールのサンプル

カレンダ作業スケジュールの設定や使用方法に関する詳細な説明については、Simio ヘルプまたは Simio Reference Guide で「Schedule」と入力して検索することができる。

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であるが、場合によっては追加的な制御、あるいはその逆で単純に手動による制御が必要になるときもある。手動スケジュール (Manual Schedule) はそのため用意されている。もっとも基本的なレベルの手動スケジュールは非常にシンプルである。能力に希望する値を代入し、その値が保たれる時間だけ期間を継続して、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、以下の通りである：

1. Run Initialized Add-on Process Trigger をダブルクリックする。Server オブジェクトが初期化されるときに、このプロセスが自動的に1度だけ実行される。
2. Assign と Delay ステップを1つずつ追加してから、その後にさらに Assign と Delay のステップをもう1つずつ追加する。これらを用いて、それぞれ Server1.CurrentCapacity を1に、Delay を8 Hours に設定し、Server1.CurrentCapacity を0、Delay を16 Hours に設定する。
3. 図 7.18 に示されるように、End コネクタをドラッグして、最初の Assign ステップに戻す。これで4つのステップは無限に繰り返し続ける。

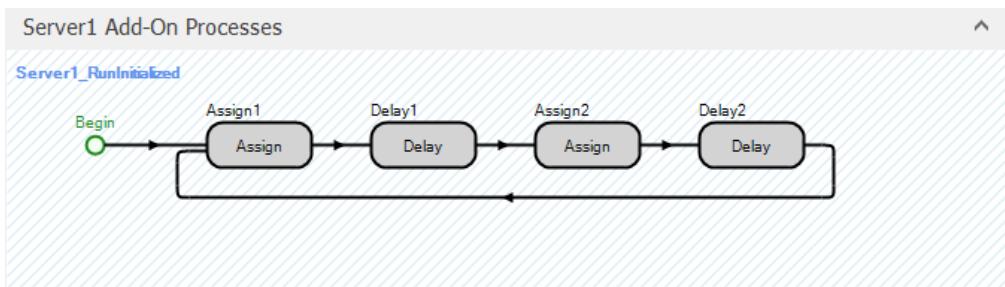


図 7.18 一日の作業時間が8時間の手動スケジュールの作成

手動スケジュールには、かなりの柔軟性を持たせることができる。スケジュールは必要に応じて細かく定義し、推移ロジックを追加することもできる。たとえば、昼食時間になると、あるリソースは直ちに作業を停止するか、進行中の作業を一時停止し、昼食終了時に中断した作業を再開するというロジックを実装することができる。しかし、これは別のロジックで実装することもできる。たとえば、後片付けの時間を確保するために、リソースはシフト終了の30分前になると、新たな仕事の受け入れを停止する。あるいは、シフト終了時にまだ仕事が進行中の場合には、今の作業が完了するまで仕事を継続する、といった一連のロジックである。これはほんの一例に過ぎないが、プロセスをフルに活用すれば、高度なスケジュールの挙動をモデル化することが可能である。このように、モデルの必要に応じてインテリジェントなオブジェクトを作ることができるのである。

7.3 レートテーブルと Model 7-4

レートテーブル (Rate Table) はこれまで議論してきたテーブルとかなり異なっている。レートテーブルには汎用性がなく、列を追加することもできない。むしろ、レートテーブルの目的はただ1つに特化しており、期間ごとのエンティティの時変到着率を指定することである。これまでの例においては、ある実行期間における到着率がずっと一定だった。しかし、現実世界、とりわけサービス業においては、到着率が時間と共に変動することがむしろ一般的である。たとえば、銀行では、他の時間帯と比べて、ある時間帯において顧客が入店する頻度が高い。また、別の例として、たとえば、サポートを求めてコールセンタに電話する人の数は、ある時間帯に対して比較的集中する。

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、それぞれ独立した到着は、到着時間間隔が一定の平均を持つ指数関数的な確率分布に従って、1つずつ発生としていた。時間と共に変化する到着率を実装するために、非定常ポアソン過程 (Nonstationary Poisson Process) が必要である。この種の到着プロセスに関する全般的な議論については 6.2.3 項を参照してほしい。また、モデルの妥当性を検証するためにも、現実世界での到着率の「ピーク」と「谷」をモデル内で反映することが必要である。

もしかすると、到着時間間隔の分布が指数関数的であることに固執して、単純に平均到着時間を状態として指定し、その状態の値を離散的な時点で変化させればよいと考えているかもしれない。しかし、このような考え方は間違っている。そのアプローチでは、1つの期間から次の期間への移

行が適切に考慮されていないため、誤った結果が導かれる。レートテーブルを利用すれば、時間に伴って変動する到着率を正しく提供することができる。

レートテーブルは、Source オブジェクトが時変到着率を有する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用いられ、時間間隔(Interval)の数や各間隔の継続期間(Interval Time Duration)を指定することができる。レートテーブルは、間隔の等しい時間間隔と各間隔における平均到着率との組で構成されている。各時間間隔中の平均到着率は 1 つの間隔内において一定であることを仮定しているが、次の時間間隔の開始時に変更することができる。各時間間隔の到着率の時間単位は時間(Hour)であり、レートテーブルで時間間隔を指定する際の時間単位とは無関係である。たとえば、レートテーブルで時間間隔を 10 分と指定したとしても、この 10 分間隔の到着率を 1 時間あたりの到着率に変換して表す必要がある。レートテーブルでは最後の時間間隔の終わりに達したら、自動的に巻き戻され、最初から繰り返される。このように、到着率の関数を区分的に一定としてモデル化することは、かなり限定的なケースのように見えるかもしれないが、実際にはこれは現実における真の到着率関数と非常によく近似していることが示されている(Leemis (1991) を参照)。

標準の Source オブジェクトにレートテーブルを使うには、Source の Arrival Mode 属性を Time Varying Arrival Rate に設定し、Rate Table プロパティに適切なレートテーブルを選択する。

ED の Model 7-2 の到着をより正確なものに強化する例を通して、この概念を詳しく説明しよう。最初のモデルでは、エンティティの到着を固定の平均 4 分の到着時間間隔で近似していたが、実は到着率は 1 日にわたって変動していることが分かっていた。具体的には、表 7.5 で示されているように、すでにある 1 日における 4 時間ごとの患者の平均到着数のデータを持っている。

表 7.5 Model 7-4 における過去の到着データ

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しよう。

- Model 7-2 を立ち上げて、Data ウィンドウをクリックし、Rate Tables パネルを選択する。
- Rate Table ボタンをクリックし、新しいレートテーブルを追加する。名前を ED_Arrivals に変える。
- 到着率の設定をデフォルトのままにして、1 時間単位で 24 回分の時間間隔を設定することができるが、今回は 4 時間単位で 6 つの時間帯の到着情報しか把握していないことから、テーブルのプロパティの Interval Size を 4 Hours に設定して、Number of Intervals を 6 で設定する。
- 既知の情報は 4 時間のおきの患者の平均到着数であることから、これを 1 時間当たりの患者の到着率に変換する必要がある。そのため、表 7.5 に示した到着率を 4 で割った後にレートテーブルに入力する。図 7.19 は設定完了後の画面を示している。
- もう 1 つの作業がまだ残っている。Facility ウィンドウに戻って、このレートテーブルを使用するように、Source を設定する必要がある。Arrival Mode プロパティに Time Varying Arrival Rate を選択する。そして、Rate Table プロパティには図 7.20 に示すように ED_Arrivals を選択する。

The screenshot shows a 'Rate Table' named 'ED_Arrivals'. The table has six rows with the following data:

	Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 04:00:00		12.25
Day 1, 04:00:00	Day 1, 08:00:00		7.75
Day 1, 08:00:00	Day 1, 12:00:00		9.5
Day 1, 12:00:00	Day 1, 16:00:00		9
Day 1, 16:00:00	Day 1, 20:00:00		15
Day 1, 20:00:00	Day 2, 00:00:00		17.5

The properties panel on the right shows:

- Basic Logic:**
 - Interval Size: 4
 - Units: Hours
 - Number of Intervals: 6
- General:**
 - Name: ED_Arrivals

A note at the bottom states: "A table containing user-specified rates across time intervals."

図 7.19 患者の時変到着率を表すレートテーブル

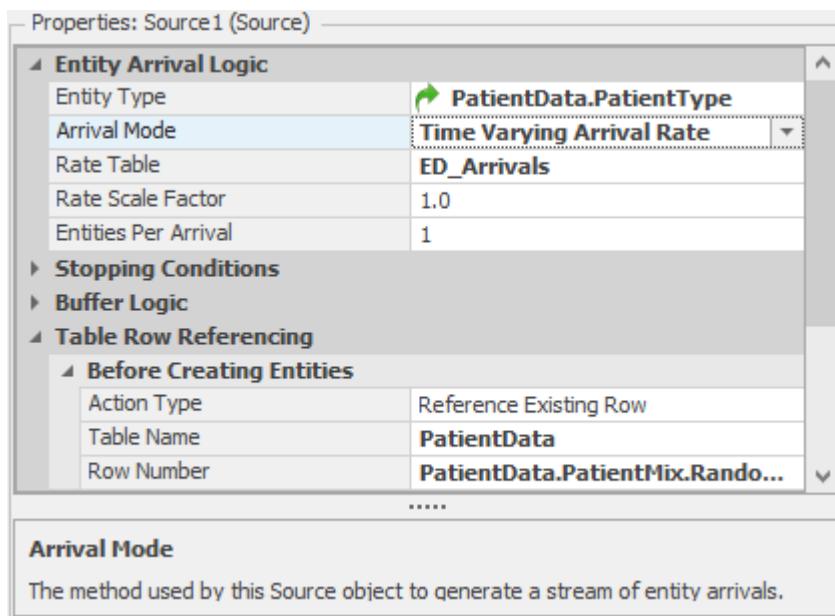


図 7.20 時変到着率の Source への適用

修正後のモデルを実行すると、かなり異なった様子が表示されるだろう。全体的な患者数が減少しているだけでなく、到着患者数の多いピーク時に対処しなければならなくなつたためである。

このモデルでは使用しなかつたが、Source オブジェクトには Rate Scale Factor というプロパティも含まれている。このプロパティを使えば、値を個別に変更する代わりに、係数を変えることで、テーブルに含まれる値をまとめて変えることができる。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 の値を 1.5 に指定すればいい。これにより、サービスの負荷変化に対する実験が簡単にできる（たとえば、平均患者負荷が 50% 増になった場合、システムがどのように反応するかなど）。

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）が必要な場合があるだろう。また、簡単な式で済む場合（たとえば、Server1. CycleCount * 3.5 または Math.Sqrt(Server1. CycleCount)）や、直接テーブル参照を行うこともあるだろう（たとえば、MyTable[Server1.CycleCount]. ProcessTime）。他には、非線形ルックアップテーブルも有用だろう。Lookup テーブルは $f(x)$ の機能を提供し、ここに x は時間、カウント数、もしくはその他の独立変数や数式を表す。Lookup テーブルの一般的な使い方として、仕事を完了する時間が作業者の熟練度に依存するような学習曲線のモデル化がよく挙げられる。また、別の応用例としては、部品

のサイズや重量に基づいて、作業の処理時間やバッテリー放電率などを求める際に用いられることがある。

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 Lookup テーブルを追加する。表には x （独立）の値と $f(x)$ （従属）値の列がある。TableName[X_Expression]の構文で式を指定することにより、式の中で Lookup テーブルを使用することができる。ここに、TableName は Lookup テーブルの名前であり、X_Expression は独立したインデックス値のことを表す。たとえば、ProcessingTime[Server1.CycleCount]では、Server1.CycleCount の現在値に基づいて ProcessingTime という名前の Lookup テーブルから値を返される。Lookup テーブル値は、定義された値または定義された値の間の線形補間を返す。インデックスが定義されている範囲外にある場合、範囲内のもっとも近い端点（最初／最後）の点が返される。

それでは、Model 7-4 を拡張して、Registration Time を調節するための Lookup テーブルを加えてみよう。ここでは、診療受付プロセスの処理にコンピュータシステムを使うことを仮定する。なお、このコンピュータシステムは早朝と夕方には処理が速いが、正午にかけて少し遅くなる（図 7.21）。

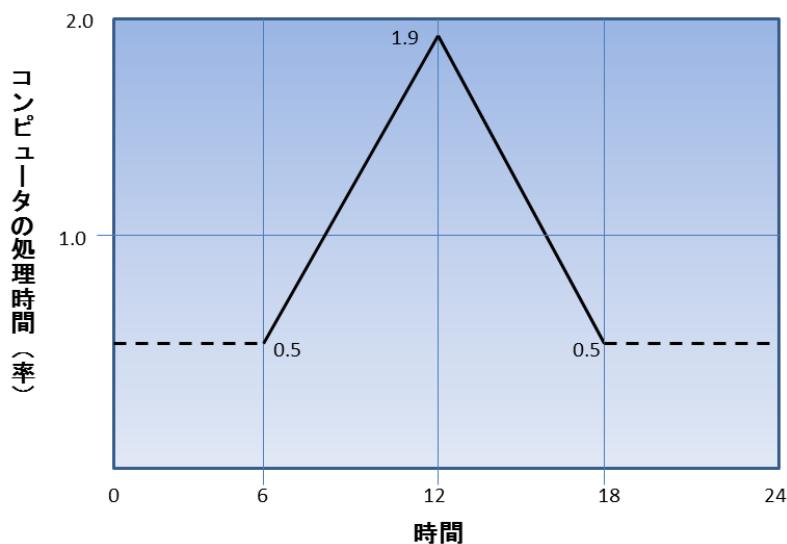


図 7.21 1 日にわたる時間ごとのコンピュータの応答性（速度係数）

このコンピュータの速度を表現するために、処理時間を調整係数として活用する Lookup テーブルを作る。

1. Model 7.4 を開く。Data ウィンドウの Lookup Tables パネルを選択する。
2. 新しい Lookup テーブルを追加するには Lookup Table をクリックする。名前を ComputerAdjustment に変更する。
3. 図 7.21 で示されたデータを追加する。データのポイントは 3 つあり、時刻 6 時の値が 0.5、12 時の値が 1.9、そして 18 時の値が 0.5 である。6 時以前または 18 時以降の時間帯は、最初または最後の値（この例では、両方共に 0.5）が返される。時刻が 12 時ちょうどには 1.9 という正確な値が生成されるが、その他の時間は、指定された値との間で直線的に補間する。たとえば、8 時は 2 つの x 値の間の $1/3$ のところにあることから、この 2 つの $f(x)$ 値に繋がった直線上の $1/3$ のところにあたる値が返される。つまり、 $0.5 + 0.466 = 0.966$ 。
4. 最後に、Facility ウィンドウの Registration サーバに戻って、Processing Time を修正する。Process Time プロパティを右クリックして Reset を選択するところから始めよう。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として新しい Lookup

テーブルから得る値を掛け算する。そのため、その Lookup テーブルに当日の経過時間を表す値を渡す必要がある。Simio に組み込まれた DateTime 関数を利用すれば、その日の時間が返される。以上のことから、Registration の Process Time プロパティの式全体は `Treatments.ServiceTime * ComputerAdjustment[DateTime.Hour(TimeNow)]` となる。

このように、修正後のモデルは、夜間の受付での処理時間を短縮し、昼間にコンピュータの応答が遅いことを反映して、診療受付での処理時間を長くすることが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、あるいは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。リストはまた、転換行列（Changeover Matrix）の転換可能な状態（たとえば、色、サイズなど）を定義したり、選択を行うため（たとえば、リソースの専有、トランスポータの選択）のリストを提供するために使用される。リストは Definitions ウィンドウの Lists パネルから追加することができる。

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っている。リスト値は、`List.ListName.Value` の構文の式を用いて参照することができる。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っている場合、式 `List.Color.Yellow` では 3 の値が返される。BikeColor という名前の ListProperty（すなわち、取り得る値がリストメンバーであるプロパティ）を持っている場合には、`BikeColor == List.Color.Yellow` を用いて条件をテストすることができる。

リストに関する他の種類の応用については 8.2.7.3 で後述するが、ここでは文字列型リスト（String List）だけを検討する。名前から簡単にわかるように、文字列型リストとは、文字列のリストのことをいう。このリストは、数値ではなく、人間が読めるような文字列で項目を識別したい場合に用いる。

文字列型リストを作成するのは簡単である。Definitions ウィンドウの Lists パネルで String ボタンをクリックすれば、新しい文字列型リストを作成することができる。図 7.22 で文字列型リストの一例を示しているように、ウィンドウの下部に各項目を表す文字を入力する。

Name	Object Type	Display Name
▲ Strings		
Colors	Strings	Colors
.....		
0	String	
1	Red	
2	Green	
3	Blue	
	Yellow	

図 7.22 色を表す文字列リストの例

チェンジオーバーとは、タイプの異なるエンティティの間で必要な変換を記述するために使用される一般的な用語である。製造業においては、ある製品について、小から大へのサイズの変更や色の変更などがこれに当たる。チェンジオーバーは、一般的にサービス分野ではありませんが、まれに使われることもある。たとえば患者を入れ替えする際に、タイプが Routine (通常) の 2 人の患者間で生じる清掃時間あるいは遷移時間は、タイプが Serious(危篤) または Urgent (緊急) の患者の場合と比べて大幅に少ないだろう。チェンジオーバーは、Server の Task

Sequences (10.3 節で説明) と関連する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追加することもできる。

チェンジオーバーは 3 種類あり、それぞれ Specific、Change-Dependent と Sequence-Dependent である。Specific はもっとも簡単なチェンジオーバーであり、すべてのエンティティが同じ式を使う。これは、簡単な定数 (たとえば、5.2 分) でもできるし、エンティティの状態変数あるいはルックアップテーブルなどを含む複雑な式 (たとえば Entity.MySetupTime または MyDataTable.ThisSetupTime) でもできる。他のタイプのチェンジオーバーでは、処理された最後のエンティティに関する何らかの情報を追跡し、それを用いてこれから処理しようとするエンティティの情報と比較するようなものがある。Change-Dependent では、正確な値が何であるかよりも、その値が変更されたかどうかに关心がある。たとえば、色が変更された場合にはあるチェンジオーバーの時間が必要されるが、色が変更されなかった場合には別のチェンジオーバー時間が必要となってくる例が考えられる (0.0 かもしれない)。Sequence-Dependent では、値の離散的な集合内の変化を追跡している。その値の組は、一般的な Simio リストで定義される (たとえば、Small、Medium、Large)。セット内における個々の「From」と個々の「To」の間の値の唯一の組み合わせには、それぞれのチェンジオーバーの時間を持つことができる。これには、転換行列と呼ばれる別の Data ウィンドウの機能が必要になる。

転換行列 (Changeover Matrix) はリストに基づく行列である。多くの場合、リストは名前で特性を識別できる文字列のリストである (たとえば、名前が Colors のリストには、Red、Green、Blue、Yellow の項目が含まれている)。エンティティは通常、リストから値を与えられたプロパティまたは状態変数 (たとえば、Color) を持っている。転換行列は、特定のリスト内の値を From/To 行列の形で表示する。各セルには、From の値を有するエンティティから To の値を有するエンティティに変わる際に要求される相応の移行時間を格納する。転換行列が適用されると、行は前のエンティティの行の値に基づいて選択される。そして列は、現在のエンティティの列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することができる。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 プロパティで、先に作成したリストの名前を記入する。この時点で、下のウィンドウが拡張され、リストにあるすべてのメンバーの行列が表示される。ペア間の移行時間を示すために、各交点の値を変更する。図 7.23 では Yellow が 10 分の移行時間を示しているが、他の色への移行時間は 2 分である。色の変更がない場合、移行時間は必要としない。

From \ To →	Red	Green	Blue	Yellow
Red	0	2	2	10
Green	2	0	2	10
Blue	2	2	0	10
Yellow	2	2	2	0

図 7.23 色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、2つの重要な話題である配列と初期化については、後回しにしていた。状態変数配列 (State Arrays) は簡単にいえば、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である。

状態変数は、Dimension プロパティの値を設定することによって、配列とすることができます。たとえば、Dimension プロパティが Vector と設定された（あるいは Dimension プロパティに 1 を入力した）場合、状態変数は1次元であり、Rows プロパティは配列内の行数を決定する。Dimension プロパティが Matrix で設定された（あるいは Dimension プロパティに 2 を入力した）場合、Rows と Columns プロパティは行および列の数を決定する。配列を2次元よりも大きく設定したい場合には、Dimension プロパティに最大 10 までの整数を入力すればよい。Simio では配列を最大 10 次元までサポートしている。

状態変数配列を参照する際には、一般的に、`StateName[row]`、`StateName[row, column]`、または`StateName[row, column, dimension 3, ..., dimension 10]`のような構文で書く。たとえば、`Weight` という名前の状態変数配列の 7 行 5 列を参照する際には、`Weight[7, 5]` を用いる。ここでは、状態変数配列のインデックスが 1 から始まることに留意してほしい。たとえば、5 つの配列要素を持つベクトルを指定する際には、0~4ではなく、1~5を使用してアドレス指定する。

各状態変数は Initial Value というプロパティを持っており、これを使えば、配列の中のすべての項目を同じ値に初期化することができる。配列の中のそれぞれ項目を一意の値で初期化したい場合には、プロセス（おそらくオブジェクトの OnInitialized プロセス）で Assign ステップを使用して行うことができる。また、別の方法として、状態変数あるいは状態変数配列を初期化するために、Read ステップを介して、外部データソースから初期値を読み込むこともできる。

実は、もっと簡単な方法があり、配列の次元の設定と初期化を一括して行うことができる。それは状態変数の Dimension プロパティを [Table] に設定することである。この方法ではまず、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。それから、テーブルの各行が配列の行を生成し、テーブル内の各数値列（たとえば、整数、式、日付時刻など。ただし Element や Object Reference は除外）が、状態変数配列内の列を生成する。次に、初期化の際に、各数値フィールドの値を評価して、その値を適切な状態変数配列の項目にコピーする。この方法を、テーブルを外部ファイルにバインドする機能うまく組み合わせると、外部データソースと完全に一致する配列を簡単に作成・設定することができる。

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もつモデルを作成したいと思うことはよくあるだろう。たとえば、ある複雑なワークステーションを構築し、デバックも済ませていたら、そのワークステーションを同じ施設や他の施設にある同様のワークステーションに再利用したいと考えるかもしれない。オブジェクトの利用はソリューションの重要な部分であるが、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、より柔軟性を持たせることができる。

また、そのコンセプトを少し拡張して、モデル全体を再利用可能にしたいときもあるだろう。たとえば、あるシステムには類似の施設（港、工場、病院など）が多数存在し、各施設が他の施設と似ていても主な構成データや運用データが異なる場合には、このコンセプトは役立つだろう。

幸いなことに、汎用コンポーネントを使用するか、汎用モデルを構築することで上記の2つの目的を達成することができる。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のではなく、「渡された」主要なデータで汎用モデルを作成する概念である。訪れるすべてのエンティティは、自身がどのように処理されるかを決定するために、自らのデータをオブジェクトに提供することができる。オブジェクトは、エンティティあるいは関連データテーブルからの指示に従って、エンティティを処理するように設定される。このコン

セプトにより、カスタマイズはオブジェクトよりも主にエンティティと設定データの責任となるため、オブジェクトを比較的簡素化することができ、その種類を比較的少なくすることできる。このように、データはオブジェクトから抽象化され、多くのオブジェクトに分散していたデータを一ヵ所に集約することができる。これは、いわゆるデータ駆動型モデルと呼ばれる。

Simio は、データ駆動型モデリングをサポートするいくつかの機能を提供しており、すでにその中のいくつかを紹介した。5.1.2 項の後半では、Referenced Properties を Controls として使うことにより、主要パラメータの検索と変更が容易になることを述べた。コントロールを利用するこことにより、データ駆動モデルの作成が容易になる。なぜなら、主要なデータはモデルの属性または実験のコントロール値を介して供給することができるからである。

Model 7-2においては、患者のタイプ、患者構成、治療を受ける場所、治療時間などのデータをテーブルから取得することにより、モデルのオブジェクトにデータを直接埋め込むことなく、モデルを構築することができた。このより包括的な手法により、データ駆動型のモデリングを実現した。つまり、モデルオブジェクトは、特定のアクションの代わりに「テーブルで指定されたタイプの患者を、テーブルで指定された割合で生成し、テーブルで指定された場所に向かわせ、さらにテーブルで指定された所要時間で治療を受けさせる」と本質的に指示されていた。このため、モデリングに関してあまり知識のない人でも、データテーブルの値を少し変えるだけで、モデルを用いて実験することができるようになる。

オブジェクト参照およびエレメント参照の状態変数と属性は、このコンセプトをさらに発展させることができる。オブジェクト参照とは、その名の通り、オブジェクトまたはオブジェクトリストを指定することを想起してほしい。これにより、患者種類ごとに治療や付き添いに必要なスタッフや機器（リソース）のリストを指定し、前のモデルを拡張させることができる。同様に、エレメント参照は、材料や統計量などの要素（エレメント）を指定する。これを使えば、患者に必要な特別な材料（たとえば、手術用パックなど）や、患者のデータを記録するための特別な統計量などを指定することができる。

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についての議論は、Simio のテーブルとリピートグループ（Repeat Group）との間の特殊な関係を抜きにしては語れない。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ことを想起してほしい。その一例が、ほとんどのオブジェクト（例：Source）で利用可能な Assignments である。リピートグループがテーブルと似ていると思うのならば、その通りである。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見なすことができる。実際、Simio ではテーブルとリピートグループを自由に変換できる。

簡単な例で説明しよう。状態変数の割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としよう。

1. 新しいモデルから始めて、Source を 1 つ追加する。
2. Data タブに移動して新しいテーブルを追加するが、Add Table ボタンの下矢印をクリックし、Add Data Table From Schema をクリックする。これにより、モデルに定義されている任意のスキーマ（または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することができる。
3. このモデルでは、下にスクロールして、Source スキーマを選択する。すると、図 7.24 のようなテーブルが表示される。

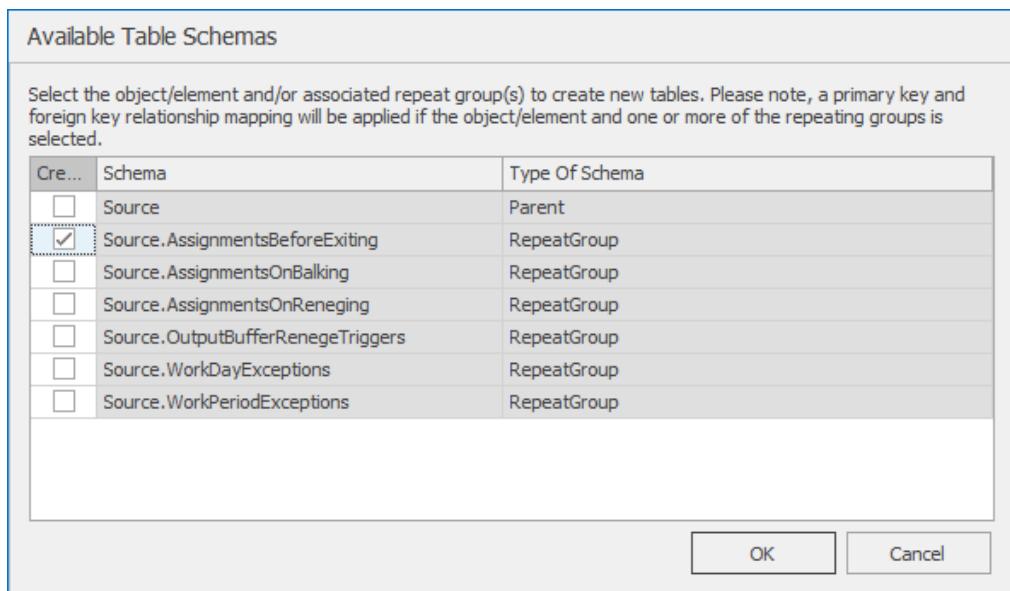


図 7.24 Source オブジェクトによって定義されたデータスキーマ

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、最初の列が State Variable で、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新しいテーブルが作成される。新しいテーブルに StartingAssignments という名前を付ける。
5. 図 7.25 に示すように、テーブルに 3 つの行を追加しよう。最初の行では、モデルの Cost の状態を 100 ずつ増やす。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 とする。

	State Variable Name	New Value 0
1	Cost	Cost + 100
2	ModelEntity.Priority	3
3	ModelEntity.Picture	1

図 7.25 テーブルで定義した割当のサンプル

6. テーブルの作成と入力が完了したので、今まで使用してきたプロパティ参照の機能を使用して、このテーブルを使用することができる。Facility ビューの Source オブジェクトに移動し、State Assignments の下の Before Exiting プロパティの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、リストに表示されている先ほど定義したテーブルがリストに表示され、選択できることがわかる（図 7.26）。それを選択すると、プロパティ参照で見たのと同じ緑色の矢印が表示され、コンテキストも同じで、データを取得するための場所が示される。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する。

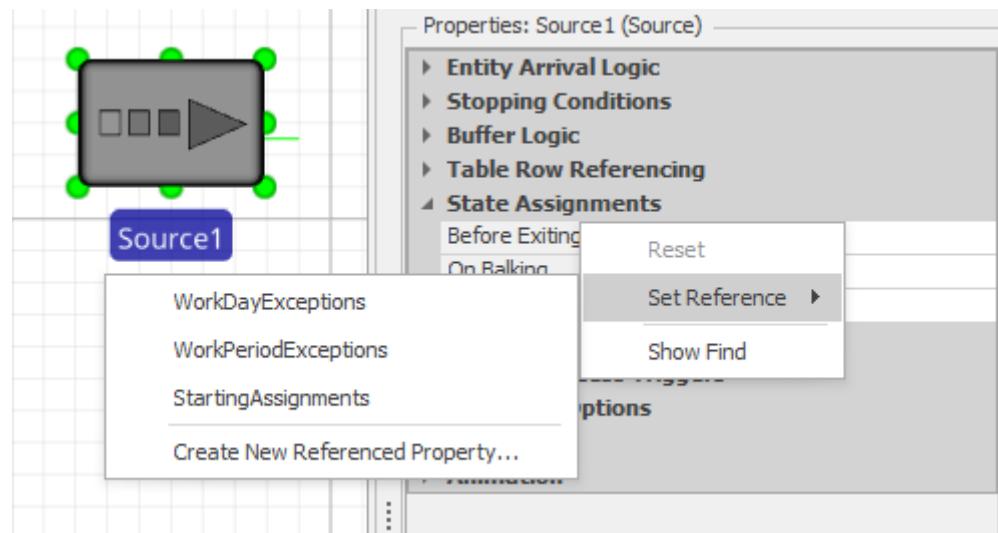


図 7.26 リピートグループのテーブルデータを用いたプロパティ参照の指定

この簡単な例では、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ンナルデータテーブルと組み合わせて使えば、オブジェクトとデータテーブルを結び付けて、より本格なデータ駆動型モデリングへと近づけることが可能になる。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使用して、簡単なデータ駆動型スケジューリングモデルを構築する方法を紹介する。

マクロモデリングの主な目的は、多くの異なるものを扱うことのできる、やや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のようなものである。データ駆動型モデリングの主な目的は、データを抽象化することにより、データを一箇所にまとめて管理できるようにし、モデルの実験とメンテナンスを容易にすることである。マクロモデリングとデータ駆動型モデリングは目標が異なるが、オブジェクトの値をプロパティで直接指定するよりも、データをオブジェクトの外部から取得することで、ある程度の汎用性のあるオブジェクトを作成する手法を利用しているという点で共通している。そして、これらの手法により、モデルの理解、使用と保守はよりシンプルになる。

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)とも呼ばれる)とは、モデル構造自体のデータの多くを外部ファイルからインポートして汎用モデルを自ら生成するという概念である。シミュレーションの価値を大きく変える可能性があるという観点から考えれば、これは本当の「ゲームチェンジャー」といえるだろう。モデリングに関して専門的知識が少なくてモデルをより素早く構築できることは非常に魅力的だろう。なお、この章の内容を超えてはいるが、12.5.1 項で紹介されるデジタルツイン(Digital Twin)は、頻繁に変更される大規模なモデルを取り扱っている。データ生成モデルは、データベースとの直接バイディング、スプレッドシートまたは CSV バイディング、XML 変換などを利用して、あらゆるデータソースにバインドするためのオプションを備えた効果的な解決策を提供してくれる。この方法で作ったモデルは、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の役割として、うまく展開することができる。

既存のデータから直接モデルを構築することが望ましい状況は、さまざまにある。組織では Microsoft Visio のような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構築することがよくある。標準的な Visio ステンシルには通常、完全なシミュレーションモデルを作成するために必要なすべてのデータが含まれていないが、カスタムまたは拡張ステンシルには、それを実行する

のにじゅうぶんな情報が含まれている場合がある。そのモデル（データだけでなくモデル自体も）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り、さらにシミュレーションを追加構築すればフローチャートモデルを拡張することも可能だ。これは、既存のデータからモデルを生成するための、ドメインに依存しない（domain-neutral）簡単な応用である。Simio Shared Items フォルダに、このアプローチを紹介する例がある。

多くの組織がデータを表現するために活用する国際的なデータ標準がいくつかある。B2MML は、Business to Manufacturing Markup Language の略で、一般的なデータ標準の 1 つである。

B2MML は、国際標準では IEC/ISO62264 として知られている ANSI/ISA-95 規格群（ISA-95）の XML の実装である。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、データ構成、インポートおよびエクスポートをモデル化することもできるが、Simio は別の方法を用意している。まず、企業が B2MML のようなデータ規格に従う場合、一般的には単に 1 つのテーブルを越えて、一連のデータおよびモデリング技術に拡張されることを認識してほしい。それをサポートするために、Simio はテンプレートと呼ばれる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（たとえば、Word の履歴書テンプレート）と同様に、標準的な方法でプロジェクトを実装するためのスケルトンで構成されており、データテーブルスキーマ、カスタムオブジェクト、それらのオブジェクトとデータテーブル間の関係などが含まれる。

新しいプロジェクトを最初に開始するときに、テンプレートを適用することができる。File→New From Template を選択すると、図 7.27 に示すようなオプションが表示される。

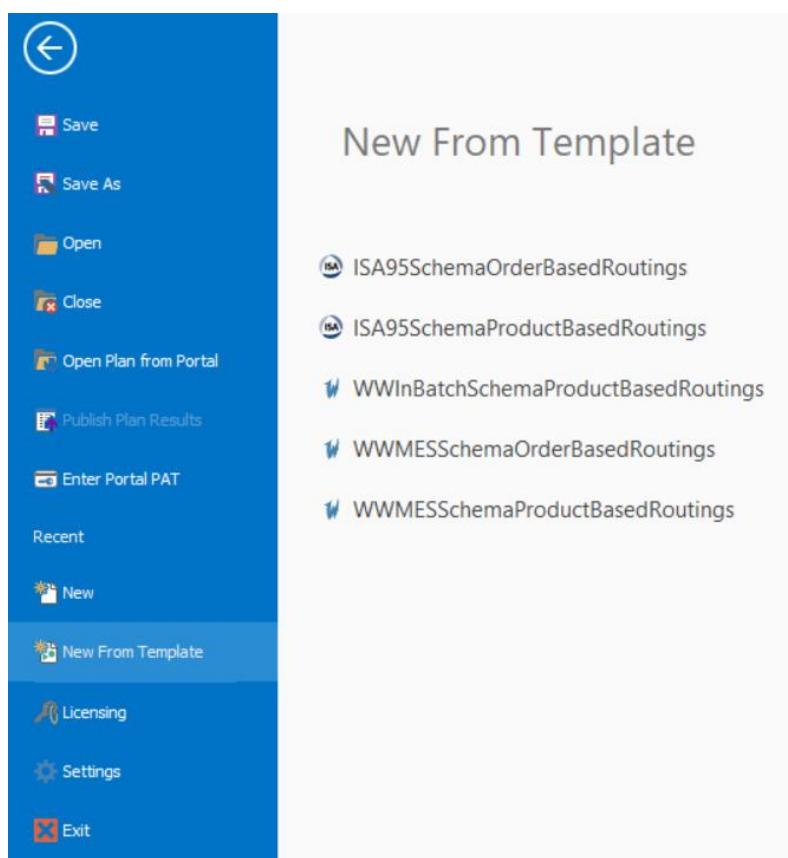


図 7.27 新規プロジェクトを開始する際に利用可能なテンプレート

データが注文基準か品目基準かに応じて、2つのISA95 (B2MML) データスキーマが提供される。これらのスキーマで新しいプロジェクトを開始すると、モデルはリレーショナルデータテーブルスキーマ（ただしデータはない）とカスタムオブジェクトがあらかじめ設定されており、これらのカスタムオブジェクトは、すでにデータテーブルから直接構成されるように接続されていることがわかる。ISA95 規格に準拠した既存のデータは、これらのテーブルにインポート可能である。注：ISA95 はガイドラインであるため、企業の実装は標準と若干異なることが多いが、Simio のデータテーブルをカスタマイズしたり、それらを変換することもできる。MESA (Manufacturing Enterprise Solutions Association) は、B2MML 規格に関する優れた情報源の 1 つである (International 2018)。12.12 節では、B2MML データファイルに基づいてモデルを構築する。

製造実行システム (MES : Manufacturing Execution Systems) は、生産を追跡・制御するためのソフトウェアベースのシステムであり、主に物理的操作に集中している。それが効果的に機能するためには、「モデル」に基づいている必要がある。上記の ISA95 の例のように、Simio は MES から直接データを抽出し、そしてそのデータから Simio モデルを構築および設定することができる。図 7.27 の下 3 つのテンプレートによって、Wonderware (AVEVA 社による人気 MES 製品) のデータをサポートするスキーマを使用してデータテーブルを作成できる。SAP などの製品から直接 ERP データを抽出する際にも、このプロセスを使用できる。実は、自分でテンプレートを作成することもできる。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、その中から選択することが可能となる。

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いて、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。詳しい手順については、Simio のヘルプトピックの Table-Based Elements (Auto-Create) を参照してほしい。実はこの機能は、これまでのテンプレートにすでに組み込まれている。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 ウィンドウに設置されるのがわかる。

ごく少数のケースでは、このデータ生成アプローチを使えば、実行により意味のある結果を生成できる完全なモデルを構築することができるが、残念ながらこの手法は一般的ではない。より一般的なのは、既存のデータにこのようなアプローチを適用することで基本となるモデルが素早く生成され、それを基に段階的な改善を加え、そこから意味のある結果を得られるという点である。もしデータファイルから完全なモデルを生成したいと考える場合には、既存のデータソースを拡張して不足のデータを追加することができる。この方法はいまだに非常に有用である。

7.9 まとめ

データのインポート、保存、そしてアクセスは、ほとんどのモデルにおいて重要な部分である。この章においては、データの種類と、データを Simio に格納するためのいくつかの方法を説明した。テーブルの構造、特にリレーショナルテーブルは、データを処理する際に非常に柔軟な方法を提供してくれる。また、オブジェクトのデータを抽象化することにより、モデルの使用とメンテナンスを容易にしてくれる。モデルは非常に多くのデータを必要とするため、可能な限りデータをインポートしたり、場合によってはモデル全体をインポートするオプションを検討する価値がある。

7.10 問題

- 例題 Model 7-1 の Server の能力を 4 に調整する。利用率 Utilization (Server1.Capacity, ScheduledUtilization) と滞在時間の長さ Length of Stay (Sink1.TimeInSystem.Average) を調べるために実験を行いなさい。モデルに 10 日間のウォームアップ期間を設け、シミュレーションの長さが 100 日間になる実験を 25 回繰り返して実行する。季節が変わり、Model

- 7-1で患者タイプのPatient Mix のUrgent 患者の割合が10%に増加し、Moderate 患者とRoutine 患者の割合がそれぞれ24%と36%に減少する場合に、変更前と変更後のモデルで患者の待ち時間を比較しなさい。
2. 修正後の例題 Model 7-1 の患者カテゴリの変更に加え、ある小さな町の救急病院では「再診(Returns)」と呼ばれる一般カテゴリの患者もいる。彼らは来院したことのある患者であり、包帯の交換、装具の調整、ギプスや抜糸などのために戻ってくる。この再診のカテゴリを Model 7-1 の患者構成に追加すると、Returns、Routine、Moderate、Urgent 患者の割合はそれぞれ8%、30%、26%と10%になる。この Returns 患者の処置時間は3分～30分の間で変動する。この結果を問題1の結果と比較しなさい。
 3. より現実的には、問題2の患者の約3分の2は、日中（午前8:00～午後8:00）に到着する。1日の全体の到着率を変更せずに、この新しい到着パターンに対応するために、問題2のモデルを修正しなさい。変更後のモデルのパフォーマンスを問題2のものと比較しなさい。夜間と日中の滞在時間の長さ（LOS）をそれぞれ収集する。両時間帯の平均LOSを0.5時間以下に維持しながら、病院スタッフの人数を最小限に抑えるためにはどのような変更案を考えられるだろうか。
 4. さらなる分析を行ってみると、問題2の再診患者のうち、わずか10%の人が20分～30分の診療時間を要することが分かった。また、残りの患者は3～20分の治療時間を必要としていることが分かった。この場合、新しいLOSはどうなるだろうか。LOSを目標値の0.5以下に保つつゝ、スタッフの稼働効率を向上させる方法について、経営陣に提案しなさい。
 5. Routine および Return 患者は、治療を待つことに対する忍耐力がやや低く（30～90分）、ときには治療を受けずに帰ってしまうこともある（Leave Without Being Seen; LWBS）。他の「本当に具合が悪い」と分類された患者は、診療を受けるまで待ち続ける。問題2のモデルを修正して、待つこと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。また、待ち時間と、治療を受けずに去った患者の総数に対する割合を報告しなさい。
 6. ある小さな無料のクリニックでは、午前8時から正午までの1人の医師が患者を診察している。この医師は1人の患者を診療するのに6～14（平均10）分かかるため、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と想定される。現在、このクリニックは患者が10分ごとに到着するという前提で、患者の予約スケジュールを組んでいる。しかし、一部の患者は15分も早く到着したり、30分も遅れて到着したりすることもあり、他の患者に迷惑をかけていることが分かった。さらに悪いことに、約10%の患者は来院せず、医師の時間を無駄にしてしまうだけでなく、本来は他の患者が使えるはずの予約時間を浪費してしまう。そこで、このクリニックは、医師の稼働率を最大化するために予約の方針を変えることを検討している。たとえば、20分おきに2～3人ずつの患者を予約に入れる方針に変えて、この代替案の効果を評価したい（可能であれば、1日に24人の患者を診たい）。この代替案では、予定されたすべての患者の診療が終わるまで、医師が勤務し続けることを仮定する。しかし、12:30を過ぎても診療が終わらない場合、医師は非常に不満になる。システムのパフォーマンスを測る際には次の基準を用いる：実際に診た患者の人数、患者の平均待ち時間、予定された患者の診療がすべて終わるまで医師が通常何時まで滞在する必要があるのか。
 7. 救急部門のスタッフにとって大きな関心事となるのは、到着時に重症（Severe）に分類される患者が、治療を受ける前に容態が緊急（Urgent）のレベルまで悪化し、検査室（Trauma Rooms）で直ちに応急措置を行い、容態を安定させなければならない場合である。診察中に10%のSevere患者にこのような容態の悪化がみられると仮定し、Model 7-2を修正しなさい。そして、修正前後のモデルで待ち時間とスループットの変化について比較しなさい。ここでは、「直ちに応急措置を行う」ことを、どのようにモデリングするかがポイントとなる。
 8. 救急部門のスタッフは、各シフトに1時間の食事休憩（できればシフトの半ば頃に）を含む「理想的な」スケジュールに従って、8.5時間のシフトで勤務している。食事休憩時間中に、

本人担当のサービスは他のスタッフに引き継がれる。たとえば、「診療受付 (Registration)」担当のスタッフは、「受付 (Sign In)」の担当者が食事をしている間に彼女の仕事をカバーする。各部門の最小要員は表 7.6 に示されている。次の新しいシフトに変わるために、新旧 2 つのシフトのスタッフが情報の共有と仕事の引継ぎをするために 30 分の「ブリーフィング時間」を設ける。合理的な勤務スケジュールを開発して、Model 7-2 を更新する。スタッフの稼働率を再度見積りなさい。休憩中の「引継ぎ」に対して、どのようなモデリングを行うだろうか。

表 7.6 休憩中のサービスエリアにおける最小要員（問題 8）

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

9. 緊急時には、検査室 (Trauma Rooms) のスタッフは、患者に処置を施している最中に食事休憩の時間が来ても仕事を継続し、休憩時間を見送るしかるべきことは明らかである。彼らは、仕事の合間を見て一口食べるようしている。これを反映するために問題 8 のモデルを更新しなさい。そして「シフトにおける実質勤務時間」を見積りなさい。
10. あなたの仕事は、宝石加工システムの Simio モデルを設計して構築することである。このシステムでは、石は鉱山から届き、「宝石加工」は研磨 (polishing)、鑑定 (grading)、仕上げ (finishing) の工程を経てシステムから離脱していく。石によっては再研磨 (repolishing) と再鑑定 (regrading) が必要とする場合もある (フロー図については図 7.28 を参照)。この施設は 24 時間 × 7 日、3 シフト体制で稼働しており、鉱山から石の到着は非定型である。表 7.7 に、3 つのシフトの開始時間と終了時間ならびに石の到着率を示す (注: このモデルでは休憩時間と食事時間は無視され、シフト変更の間に生産性の損失がないと仮定している)。到着プロセスは非定常ポアソン過程 (NSPP) であると仮定する。作業にかかる特性等は表 7.8 に示す。すべてのプロセスには 1 人の作業者 ($c=1$) がいるが、再研磨と再鑑定の工程の作業者は第 1 シフトの時のみ作業する。非定常な到着とリソースのスケジュールに加えて、モデルには次の機能が必要である:
 - a) 石を表すエンティティは、プロセスに入るときはデフォルトの緑色である。再研磨 (repolishing) が必要な石は赤色に切り替わり、そして最終的に仕上げ (Finishing) プロセスに到着するとまた緑色に戻る。
 - b) ステータスプロットを作成し、研磨 (polishing) と再研磨 (repolishing) プロセスの待ち行列におけるエンティティの数と、システム内の石の平均数を表しなさい (これらの 3 つの項目を 1 つのプロットにまとめて表示すること)。プロットの時間軸の範囲を 24 時間にする。
 - c) ユーザ指定統計量を作成し、再研磨と再鑑定の両プロセス内の石の数を時系列で追跡する。
 - d) ユーザ指定統計量を作成し、各石が再研磨を受けた回数 (再研磨をまったく受けない石も含む) を追跡する。
 - e) ユーザ指定統計量を作成し、再研磨が必要な石 (回数に関係なく) のシステム内の滞在時間を追跡する。
 - f) 長さが 200 日 (それぞれ) の 15 回の反復実験を実行する。実験では次の実験応答を求めたい:
 - i. 再研磨と再鑑定エリア内の石の平均数；
 - ii. 再研磨と再鑑定エリア内の石の最大数；

- iii. 石の再研磨の平均回数；
- iv. 再研磨を受けた石の平均システム内滞在時間；
- v. 研磨プロセスの稼働率

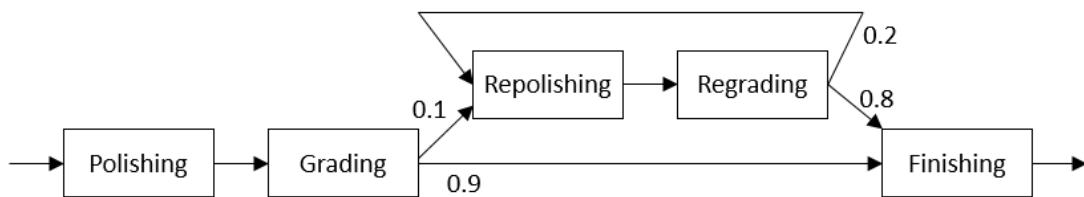


図 7.28 問題 10 の宝石処理システムの図

表 7.7 問題 10 のシフト時間と石の到着率（1 時間あたりの石数）

シフト	時間	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 10 の宝石処理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

本章の目標は、これまでの章で学んだ事柄やさらに概念を追加し、アニメーションとエンティティ移動に関する知識を広げることである。まず、典型的なプロジェクトで、2Dと3Dのアニメーションの有用性について検討することから始める。そして、自由に利用可能であるいくつかのアニメーションツールについて説明する。エンティティ移動の各種タイプについては、フリースペース（オフネットワーク）移動について簡潔に紹介し、そして、エンティティ移動をサポートしているStandard Libraryオブジェクトについてより詳しく述べることにする。

8.1 アニメーション

4.8節では、基本モデルをアニメートするために、いくつかの基本的なアニメーション概念を紹介した。本節では、再度これらのトピックスについて詳述する。まず、アニメーションがなぜ重要であるのか、そしてどの程度でじゅうぶんなのかについて説明することから始める。本節の後半で、アニメーションを作るために必要であり、利用可能ないくつかの追加的な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？

これまでに構築したモデルには、いくつかのアニメーションが含まれていた。アニメーションを持たないテキスト・ベースの商用シミュレーション製品はまだいくつか存在しているが、ほとんどのシミュレーション製品は、この数十年間で一般的な水準のアニメーションが含まれるようになつた。その理由は、アニメーションによって、モデル構築や理解がかなり容易になるためである。

8.1.1.1 システムの理解

シミュレーションモデルは、多くの場合が大規模で複雑である。モデル製作者の立場からすると、この複雑さを管理し、プロジェクトの詳細と「全体像」の両方を理解することは難しい。しかし、たとえ簡単なアニメーションであっても、可視化することは理解に役立つものとなる。理想的なシミュレーションソフトウェアとは、単純なアニメーション構築が非常に簡単にできることである。最近のほとんどの製品はまさしくそのようなものであり、事実、Simioにおいてもデフォルトの2Dアニメーションは簡単にできるため、モデル製作者側にはあまり余計な労力がかからない。

有効なモデル検証をするためには多くの要素があるが、たいていはアニメーションを慎重に見ることが重要である。段階的にモデルの進行を見ることによって、詳細に検討することができる。また、それらのイベント周辺で発生するモデルの反応を観察することによって、即座に一般的な状況かまたはまれな状況かを分析できる。そして、詳細を観察するために早送りしたり、一時停止したりしてアニメーションを見るだけで、多くの場合、問題と好機を発見することができる。

注意:アニメーションは検証において重要ではあるが、それは決して検証のための唯一の方法ではなく、また主要なメカニズムであるべきではない。有効な検証には、多くの異なる技術の利用が必要である。

8.1.1.2 コミュニケーション

一般的なハイレベルなシミュレーションの目標は2つあり、それは、システムを理解することと、利害関係者へ利益を提供することである。アニメーションは、その両方に役立つ。もし利害関係者にモデルロジックの数値やダイヤグラムを1枚の紙で提供したならば、典型的な反応は一目見て無視されるだろう。時にはめまいを起こされるかもしれない。しかし、彼らが見慣れているアニメーションを示すと、即座に元気になる。すぐに、彼ら自身の施設に関する知識を、アニメーシ

ョンにおける構成要素、動き、および状況と比較し始めるだろう。まるで実システムについて窓を通して見ているかのように、彼ら同士で、設計に関する議論を始めるかもしれない。この点において、アニメーションの提供によって、前述の2つのハイレベルな目的を果たすために、順調にスタートできたといっていいだろう。最先端の技術では、利害関係者が外部とのコミュニケーションに利用するためのアニメーションとしての品質はじゅうぶんに高い。それは、プロジェクトを促進するだけではなく、組織を育成するためにもよく活用されている。

もちろん、この機能を得るには、たいていは無料ではない。**利害関係者**のために適切なアニメーションを構築することは、利用するモデリングソフトウェアに依存する。特にアニメーションがモデルとは独立して構築される製品においては、大きく依存する。時々、アニメーションが完全に別のアニメーションソフトウェアで作成されるものがある。その場合、アニメーションはモデル実行の後に、**後工程**として実行されるだけかもしれない。そのような後工程での分析は、モデル実験の範囲をかなり制限する。

幸い、Simio がデフォルトで作成するアニメーションは、たいていは、かなり「利害関係者が意図するもの」に近い。Simio にはアニメーションが内蔵されており、実際に、モデルを構築する際に同時に作成される。そして、アニメーションを見ながら同時に、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

ここまで説明では、3D アニメーションについて述べていない。それがなぜ必要であるか、なぜそれを用いるのかと不思議に思われるかもしれない。10年前ならば、3D アニメーションはほとんどのプロジェクトで不要であり、努力する価値がないと伝えたかもしれない。しかし、この10年の間に状況は大きく変化した。まず、高品質なグラフィックスはありふれたものになり、日常生活にまで浸透してきた。今や、多くの人々が容易に3D アニメーションに関わるようになり、利用することのよさを分かっている。この新しい基準により、2D アニメーションは原始的に見えて利害関係者には魅力的に映らない。そして、3D と同等に彼らの理解と信用を得ることはできない。さらに、3D アニメーションを作成するために必要な労力が極めて少なくなった。モデル構築者はもはや過剰な描画技術や大規模なカスタムシンボルライブラリへのアクセスを必要としない。そして、複雑なグラフィックソフトの操作技術も必要としない。最近のソフトウェアでは、最小限の芸術的センスを持つモデル構築者であれば、特殊技能やツール、カスタムライブラリなしで、説得力のあるアニメーションを作成することができる。

多くの場合、アニメーションは重要であり、多くの人々にとってそれが楽しいものであり、それに取り組むことの価値をわかっている。しかし、どんなによいものでも、行き過ぎることはすべてを台無しにする。アニメーションを「非常に正確に」見せるようにすることで、モデル構築、妥当性の確認、分析、およびプロジェクトの他の重要な部分に対して、あまり時間を費やすことができないことに、すぐに気づくだろう。アニメーションは、ただモデリングの残りの部分として、実システムの近似であることを自覚しておいてほしい。プロジェクトの目的を満たすためにじゅうぶんなアニメーションを開発しようとしているだけに過ぎない。利害関係者と共にアニメーションに必要な適正水準を決定し、そして、そのプランに忠実であってほしい。プロジェクトの他の重要な側面がすべて完成した後に、アニメーションの構築時間を確保してほしい。

8.1.2 ナビゲーションと View のオプション

もし表示をまだ無効にしていないなら、Facility ウィンドウの上部にグレイの領域が表示されているだろう。すでに無効にしており、再度表示したいなら、Facility ウィンドウで H キーを押すことでオンとオフを切り換えられる。これらのナビゲーションコントロールについて、もう少し詳細に説明する。

- 2D と 3D の切り替えには、それぞれ 2 あるいは 3 のキーを押す。または View リボンで 2D もしくは 3D のボタンをクリックする（図 8.1 を参照）。
- 空いているスペースで左クリックし、Facility ウィンドウをドラッグすることによって、ビューウィンドウを動かす（パンする）ことができる。注意：この時、オブジェクト（たとえば、大きい背景オブジェクト）をクリックすると、ビューをパンする代わりにオブジェクトを動かしてしまう。これを避けるために、マウスの中ボタン²を利用してクリックやドラッグすることができる。または背景オブジェクトの上で右クリックして、Lock Edits オプションを選択することで、そのオブジェクトが動くのを防ぐことができる。
- スクロールホイール付きのマウス（強く推奨する）があれば、マウスホイールをスクロールすることによって、視点を内外へズームさせることができる。もしないならば、Control キーを押しながら右クリックして、上下にマウスを動かすことで同じ操作ができる。
- ハンドルを用いることで、オブジェクトのサイズ変更、または回転させることができる。ハンドルは、アイテムを選択するときに現れる緑色のドットである。ハンドルの 1 つをクリックしてドラッグすると、オブジェクトのサイズ変更ができる。ハンドルの上で Control キーを押しながら左クリックすると、オブジェクトを回転させることができる。
- View リボンの上の View All ボタンをクリックすると、モデルの全体が見えるように、Facility ウィンドウのサイズが変更される。これは特に、ズームやパンをしていて、モデルのオブジェクトを「見失った」場合に、見慣れた視点に戻るときに役立つ。
- View リボンの上の Background Color ボタンは 2D と 3D の両方の視点の背景色を変更する。いくつかの古いモデルでは 3D においては黒の背景であったが、2D と 3D で異なる背景色にすることはできない。
- SkyBox と Day/Night Cycle ボタンはアニメーションにおける背景の見え方や変化をコントロールする。これは、特に港や輸送ネットワークのような外のモデルで役立つだろう。



図 8.1 Simio View リボン

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、左右へドラッグして、3D における視点を回転させる。
- 右クリックしたまま、上下にドラッグして、内外へズームする。
- Shift キーを押したままオブジェクトを動かすと、床面に対して上下に動かせる。
- Control キーを押したままオブジェクトを動かすと、他のオブジェクトの上にそれを積み重ねることができる。
- 3D ビューで R キーを押すか、または 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 を押すと、回転は止まる。

図 8.2 で示す Visibility リボンは、アニメーションの見え方を微調整するのに役立つ：

- Visibility グループには、特定の構成要素の表示のオンとオフを切り替えるボタンがある。多くの場合、より現実的にアニメーションを見せようするために、大部分またはすべてのアイテムの表示を無効にするだろう。モデルを見直したいときには、ボタンを有効にしなければならないことに注意する。たとえば、Nodes アニメーションが無効であると、ノードは選

² マウスのスクロールホイールをクリックする操作と同じ。

択できない。

- Visibility グループの Direct Shadows と Diffuse Shadows ボタンは、オブジェクト周りに現れる影をより細かく制御することができる。
- Networks グループのボタンは、移動経路のネットワークを個々に、あるいはセットで表示することができる。
- 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、モデル内の別に定義されたレイヤに設置することができる。

これらの機能の使用方法の詳細は、Simio のヘルプを参照されたい。

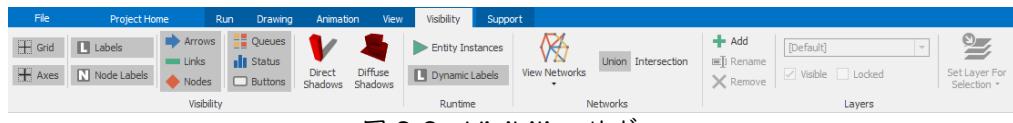


図 8.2 Visibility リボン

モデルの準備ができたとき、View リボンにおいてさらに役立つ機能がある：

- View (ビュー) は、2D や 3D、ズーム比や回転など、モデルを見るための方法を提供する。Named View グループの Add View ボタンで視点を加えることができる。これにより、現在スクリーンに表示されている視点に名前をつけることができる。Manage Views では、既存の視点の編集や削除をすることができる。Change View ボタンは 2 分割のボタンである。上の部分は、すべての命名された視点を順に巡回する。下のボタンは、どの視点を表示するか、リストから選択できる。
- Camera Tracking グループは、カメラのよいポイントを決定するためのアイテムが含まれている。特定のオブジェクトに対してカメラに焦点を当てる機能や、エンティティや乗り物のような動くオブジェクトのちょうど前方や後方に乗ったようなカメラを持つこともできる。
- Camera Sequence グループは、「ストーリーを伝える」ためのプレゼンテーションの準備に役立つ、複数のカメラの動きのタイミングと流れを決定することができる。
- そして最後に、Video グループは、アニメーションと関連するアクティビティのビデオ(avi)ファイルを録画することが可能である。

View と Visibility リボンのオプションに加えて、Project Home リボンに配置されているもう 1 つの表示オプションがある。Render to Oculus ボタンは、接続したアクティブ Oculus HMD デバイスに対して 3D ビューのレンダリングを有効または無効にする。この技術はまだまだ新しく高価であるが、アニメーションは魅力的なエフェクトを生み出すことができる。

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が動かないもの、つまり静的なアニメーションに関して説明しよう。Simio は、描画ツールであることを目的としていないが（無料で利用可能な Google Sketchup のような非常に強力なツールがある）、基本的な描画とラベリングツールを提供している。そして、シンボルをインポートする機能やインポートされたシンボルの見た目を変化させる機能も提供している。これらのツールを利用するることは、モデルの現実性と信憑性を一層高めるための、迅速で簡単な方法であるかもしれない。図 8.3 で示されている Facility ウィンドウの Drawing リボンを見てみよう。

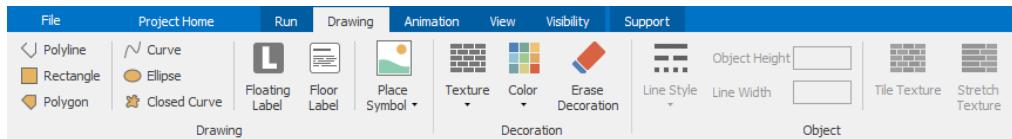


図 8.3 Facility ウィンドウの Drawing リボン

図 8.3 の左側の Drawing グループには、いくつかの基本的なオブジェクトを作成するための 6 個の描画ツールがある：直線（Polyline）、長方形（Rectangle）、多角形（Polygon）、曲線（Curve）、椭円（Ellipse）、および閉曲線（Closed Curve）である。これらの形になじみがないなら、それらがどう働くかを見るために試してみるとよい。図形を描いているとき、最後のポイントをクリックした後に、右クリックすると描画が終了する。Escape キーを押すと、試みが中止され、描かれたポイントは消される。図形の多くには、余分なポイントがあるように見えるのに気づくかもしれない。それは回転ポイントである。回転ポイントは、どんな位置にも動かすことができる。他のポイントのどれかを Ctrl キーを押しながらクリックして、ドラッグすると、全体のオブジェクトはその回転ポイントを中心にして回転する。

Drawing リボンの Decoration グループと Object グループを見てみると、色、テクスチャ（パターン）や、テクスチャのタイルや伸縮のオプション、線のスタイル、幅、オブジェクトの高さなど、基本的なオブジェクトの見た目を変更するツールがある。これらのツールすべての機能についての説明は省略するが、ここには、想像力をかき立てるようないくつかの応用とトリックがある：

- ・ 壁を作る：壁の長さの線を加える。Line Width を 0.1 メートルに、Object Height を 2 メートルに設定し、3D ビューに変更する。Texture ボタンをクリックする。そして、テクスチャ（おそらくレンガパターンがよい）を選択し、壁オブジェクトをクリックして、テクスチャを適用する。
- ・ 会社のロゴを作成する：希望するロゴの JPG ファイルを見つける。共有フォルダ ¥Simio¥Skins フォルダにそのファイルを保存する（オペレーティングシステムによって、このパスは異なる）。すると、このファイルが Texture ボタンの下に選択肢として自動的に現れる。ロゴと同じような形の長方形を描く。長方形の上か側面にロゴテクスチャを適用する。もう一つの方法は、長方形の背を高くし、細くするようにして、ロゴやテクスチャをサイドに適用する。それは、サイン、ビルボード、テレビモニタのように表示される。
- ・ 簡単な建物を作成する：長方形を作成し、高さを設定する。テクスチャを側面に適用する。その他の方法として、好きなビルの表面の写真を撮り、そしてそれを建物に適用することもできる（もちろん、Trimble 3D Warehouse か Google Maps からは、自分で撮ったものより、おそらくかなりよいビルの写真を得ることができる）。

マップについては、View リボンで、Simio はグラフィック情報システム（GIS）マップもサポートしている。最初にマップビューを選択し、住所または緯度/経度で地図の場所を設定する必要がある。インタラクティブな背景が画面に表示される。この上にノードを配置し、2 つのノードをハイライト表示し、表示された道路システムを使用して GIS システムにノードを接続させることができる。このシステムではアクティブなインターネット接続が必要である。そのため、後でインターネット接続なしでモデルをロードすると、バックグラウンドは空白になる。

Drawing リボンの説明で省いた 3 個のボタンがある。FloatingLabel は、空間中に「浮く」簡単なラベルを作成でき、それは視点の指示に関係なく、いつも正面を向いている。また、ズームレベルに関わらず、いつも同じサイズで表示される。Floor Label は、名前が含意するように、まるで床に塗装してあるかのように見える。それは多重線を持つことができ、色、サイズ、および形式オプションを選ぶことができる。ダイナミックに情報提供するラベルを作成できるように、テキストに式を埋め込むことができるということを見逃さないでほしい。

もっともよいボタンを最後に取っておいた。**Place Symbol** は、モデル背景の一部としてシンボルを配置するために、多くのオプションを提供している。ボタンの上の部分は、最後に選択したシンボル（もしあれば）と同じものを配置するための、簡単な手段を提供している。**Place Symbol** ボタンの下の部分を押すと、3つの主要な選択を提供するダイアログが表示される：

- 4.8 節でしたことと同様に、配置するシンボルを選択するために、内蔵の Simio シンボルライブラリをスクロールできる。
- 利用可能なローカルファイルがあれば **Import Symbol** を利用できる。おそらくこれは、Sketchup を用いて作成されたファイルだろう。または CAD プログラムからエクスポートされた DXF ファイルかもしれない。あるいは、JPG、BMP、または PNG ファイルのようなイメージファイルをインポートすることもできる。ここでの留意点は、DXF ファイルがかなり大きい場合があることである。それらはシミュレーションには過ぎるほど詳細に作成されていることがある（たとえば、建築構造を結合するナットとボルトのねじのようなものを含んでいる）。DXF ファイルを用いることを計画しているなら、DXF ファイルをエクスポートする前に、不要なアイテムを削除することを勧める。ヒント：JPG や PDF ファイルで見られるような 2D 建物のレイアウトは、アニメーションのよいベースとなる。それにより大きさと背景を効率的に配置できる。それから、深さを表現するために、3D オブジェクト（壁など）を加えることができる。
- Download Symbol** は、4.7 節で説明したように、Google 3D ギャラリーから検索することができる。ヒント：可能であれば、「単純」なシンボルを探してほしい。これはポリゴンの少ないシンボルを示す。いくつかの Google の 3D ギャラリーシンボルでも、DXF ファイルと同じ問題がある。シミュレーションアニメーションに、それらの複雑さと詳細さは不適当である。「完全な」フォークリフトを表現するために、非常によいピクチャを選択するかもしれないが、適所でマフラーブラケットを支えるボルトに至るまで詳細に作成されているので、それがモデルサイズを 10MB に膨らませているのがわかる。その場合、そのようなシンボルを Sketchup に読み込んで、不要な詳細部分を取り除くことができる。

8.1.4 Animation リボンの状態変数アニメーション

オブジェクトがスクリーンで動いているのを見ることに加えて、モデルのパフォーマンスにアクセスするのに役立つ他のタイプのビジュアルフィードバックや対話性がほしい。Simio はこのために 1 セットのツールを提供するだけではなく、それらをどのように表示するか (Console ウィンドウまたは Facility ウィンドウ) の選択も提供している。

8.1.4.1 Console ウィンドウ

すべてのモデルには、選択されたグラフィカルな状態変数を表示する場所として、そしてインタラクティブなボタンを表示する場所として設計された **Console** ウィンドウがある。すべてモデルの **Console** ウィンドウは、**Definitions** タブを用いて定義される。そのタブの下は、ライブラリオブジェクトを配置することができないことを除いては、Facility ウィンドウと同様に見える。本節で説明したアイテムだけ、Console ウィンドウに配置することができる。

Console ウィンドウへの状態変数アイテムの配置には 2 つの利点がある：

- Facility アニメーションとは別に、ビジネスグラフィックスを保持することができる。その時、ビジネスグラフィックスを隠したり、またはそれらを見るためにスクリーンの一部をとっておくことができる。
- もし後で、そのモデルがオブジェクトとして用いられるなら、このコンソールはそのオブジェクトの右クリックメニューから利用することが可能である。オブジェクトのユーザは、あ

なたが設計した状態変数の表示を見るだろう。埋め込んだオブジェクトからのコンソールには、更なる利点がある。それは、利用可能であるなら 2 番目のモニタ上など、Simio ウィンドウスペースの外部に表示できるということである。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を蓄積するにはすばらしい場所であるかもしれない。フロアラベルにそれを加えるだけである。

8.1.4.2 Facility ウィンドウ

状態変数アニメーションも Facility ウィンドウに配置することができる。それらを配置することの利点は、興味のあるオブジェクトに隣接してそれらを準備できることと、モデルの別の場所に配置して、名前をつけた視点と関連づけることができる。主な短所は、これらが主として 2D のビジネスグラフィックスであり、3D Facility ウィンドウでいつもきれいに見えるわけではないことである。

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を表示するために選択したウィンドウに関わらず、設置するための同じようなボタンがある。Console ウィンドウを表示すると、デフォルトリボンは 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 Animation リボンになる（図 8.4）。Facility ウィンドウに状態変数オブジェクトを加えたいなら、Animation リボン（図 8.5）をクリックしなければならない。

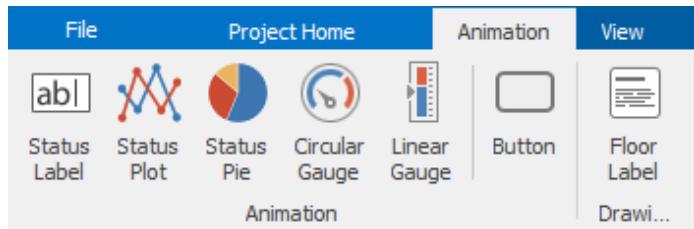


図 8.4 Console ウィンドウの Animation リボン

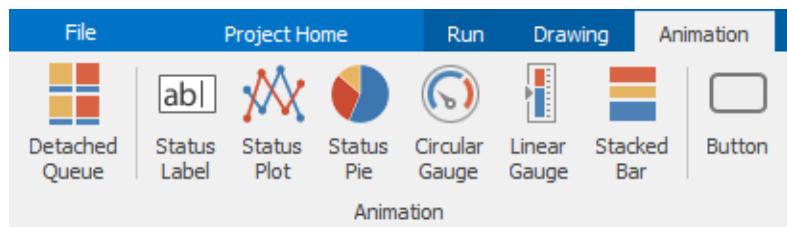


図 8.5 Facility ウィンドウの Animation リボン

これら 2 つのリボンのツールセットが非常に類似していることに注意する：

- **Status Label**（状態変数ラベル）：静的なテキストまたはどんな式の値でも表示する。
- **Status Plot**（状態変数プロット）：時間変化に伴って、1 つ以上の値を表示する。
- **Status Pie**（状態変数パイ）：合計の割合として 2 つ以上の値を比較する。
- **Circular Gauge** と **Linear Gauge**（円形のゲージと直線的なゲージ）：モデルの値の目視により説得力のある表示を提供する。
- **Button**（ボタン）：ユーザがモデルと対話する方法を提供する。ボタンがクリックされるたびに（プロセスにリンクされた）Event が発火される。
- **Floor Label**（フロアラベル）（Console 専用）：8.1.3 項で説明した Floor Label と同様。
- **Detached Queue**（独立待ち行列）（Facility 専用）：待ち行列に待機中のエンティティを示すためにアニメーションを加える。

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が自動的に Symbols リボン（図 8.6）に変わる。このリボンでは、シンボルを変更したり、アニメーション機能を加えたりすることによって、その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、新しいオプションが多くあるように見えるが、よく見ると、多くの見慣れた機能を発見するだろう。実際、4.8 節では新しい ATM Customer を選択するのにこのリボンの Project Symbols カテゴリを用いた。そして、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 ボタンのオプションと同じである。同様に、Decoration グループにおけるアイテムは、Drawing リボン Decoration グループのアイテムと同じである。

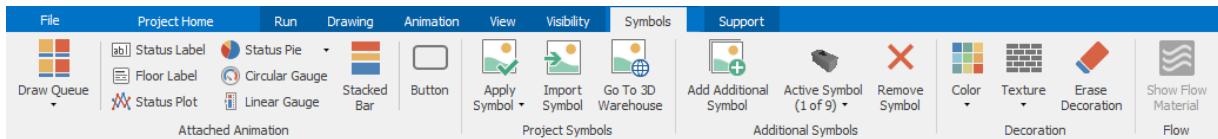


図 8.6 シンボルの編集のための Symbols リボン

8.1.5.1 Attached Animation (付属アニメーション)

Attached Animation グループは図 8.5 の Animation リボンとまったく同様に見えるが、「Attached」という単語によってまとめられることによって、1 つの非常に重要な違いがある。オブジェクトを選択することでこのリボンが開かれているので、Attached Animation グループから何かアイテムを置くと、それらのアイテムはその選択されたオブジェクトに付属される。Server のような固定されたオブジェクトのケースでは、後にスクリーンでオブジェクトを動かすと、付属のアイテムがオブジェクトと共に動くことを意味する。しかし、Entity、Vehicle または Worker のようなダイナミックなオブジェクトにアイテムを付属するとき、おもしろい状況が起こる。この場合、モデル内を移動するとき、付属アイテムはそれぞれのダイナミックなオブジェクトと共に移動する。このいくつかのおもしろい応用を調べよう：

- デフォルト Vehicle シンボルにある緑色の水平な線は、乗り物によって運ばれる（すなわち、乗る）エンティティを表示する RideStation.Contents をアニメートするために付属した待ち行列である。
- 同様に、Combiner オブジェクトを用いることでエンティティをグループ化しているなら、親エンティティオブジェクトに付属待ち行列を加えることによって、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が見られるだろう。SimBits の CombineThenSeparate と RegeneratingCombiner での例を見られる。
- また、エンティティに伴うテキストまたは数値情報を表示できる。SimBit の OverflowWIP では、それぞれのエンティティの生成時間について検証するために、エンティティの式 TimeCreated を用いることで付属の状態変数ラベルを追加した。また、待ち行列にあるとき、それを読むことができるようラベルを回転させた。

付属情報を加えることは、目視により注目させ、かつモデルを検証するのにとても役立つ。付属のアニメーションはオブジェクトの一部になるので、付属のアニメーションの範囲はオブジェクト自体のみである。この利点は、オブジェクトの視点からどんな式にも参照をつけることができるということである。このため、上述した箇条書きでは、たとえば、Vehicle1.RideStation.Contents ではなく、RideStation.Contents の参照をつけた。前者は、オブジェクトではなくモデル自体の範囲にあるから機能しない。

8.1.5.2 Additional Symbols (追加シンボル)

Symbols リボンに関する最後のグループは Additional Symbols グループである。これらの特徴は、あるオブジェクトに関連しているシンボルのセットから、シンボルを加えたり、編集したり、取り除いたりすることができる。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、オブジェクトに複数のシンボルがほしいと考えるだろう。いくつかの有用な例は：

1. 1 つのエンティティタイプではあるが、いくつかの多様性を示したいエンティティ (Person エンティティのように、5 ないし 10 の異なる人のシンボルを表示したい)。
2. モデル内の進行に応じて変化するエンティティ (たとえば、処理の進行につれて、または点検で不合格になったときに、ピクチャが変化する部品)。
3. それぞれの状態 (たとえば、Busy、Idle、OffShift) に変化した際に、異なるピクチャを示したい 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、プロパティに Current Symbol Index と Random Symbol などを持つ Animation (図 8.7) というカテゴリがある。少なくとも 1 つの追加シンボルをオブジェクトに加えていないと、これらのプロパティは無効になることに留意してほしい。これらのプロパティを用いることで、指定したシンボルから選択することができる。

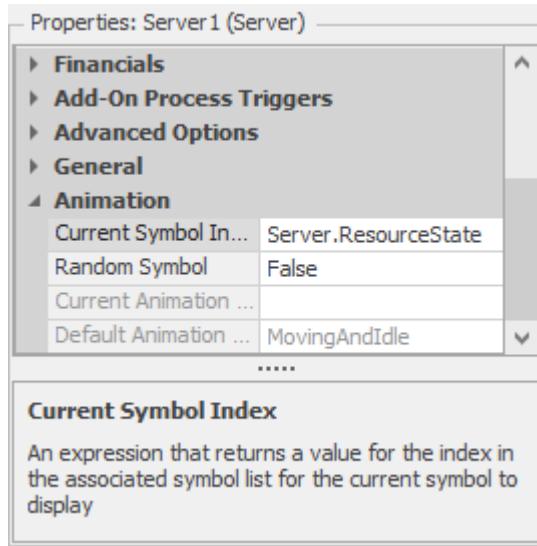


図 8.7 サーバに対するデフォルトアニメーションプロパティ

オブジェクトが最初に作成されると、Random Symbol プロパティが表示される。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 人のシンボルを定義したいなら (男性、女性および子供とする)、Random Symbol を True に設定すると、各エンティティはその結果、各シンボルを用いる機会が 3 分の 1 となる。

Current Symbol Index プロパティは、オブジェクトのシンボル番号を決定するためにどこを参照するかを Simio に設定するために用いられる。シンボルは 0 から番号がつけられるので、もし 5 つのシンボルがあると、それらには 0~4 が付番される。上のケース 2 をアニメートするために、ModelEntity.Picture に Current Symbol Index を設定する。次に、エンティティがモデル内で変化するとき、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 1 つのオブジェクト状態変数である必要はなく、式であってもよい。たとえば、SimBit の OverflowWIP では、部品がどの時間に作成されたかに関する明確

な視覚情報がほしいと思っていたので、エンティティの Current Symbol Index に式 Math.Floor (ModelEntity.TimeCreated) を用いた。時間が数時間にわたるため、1日のそれぞれの時間に新しいシンボルを用いたのである。

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、OffShift Processing、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 Simio のシンボルライブラリについて説明し、アニメーションの人物を含むフォルダについて簡単に触れた。図 4.34 の下段に示したフィルタ (Domain、Type、Action) を使うと、より素早く最適なシンボルを見つけることができるこことを思い出してほしい。特に Action フィルタには 2 つの選択肢、Animated と Static がある。ここでいう「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（たとえば、火花を散らす機械や、車輪の回る乗り物など）で、このような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しか提供されていない。しかし、これを利用して説得力のあるアニメーションを作成することができる（図 8.8）。



図 8.8 ATM を利用するアニメーションの人物

多彩な人物と服装・色が収録されている。たとえば、男性、女性、女の子、男の子、お年寄り、そしてアニメのキャラクタまで選ぶことができる。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に身を包んだ人たちが用意されている。そして、多くの人種・民族の背景も含まれている。これらのシンボルを使う一番のメリットは、よりリアルに「動く」ことができる所以である。もっともわかりやすいのは、歩く人である（例：移動すると脚や身体が自然に動く）。図 8.9 は、標準的な Simio people で利用可能な 22 の定義済みのアクションを示している。

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 8.9 利用可能な人物アニメーションのアクション

リストを見るとわかるとおり、かなり自由度が高い。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りたい場合、アクション 13 「Walk Pushing」を選ぶとよい。また、病院のベッドに寝ている人をアニメートしたい場合は、アクション 21 「Sleep」を選択すればよい。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が選択されたときに組み込まれたアクションである。たとえば、アニメーションの人物が待ち行列で待っているとき、リアリティを出すために、様々なスタンディングアクション（1～8）をランダムに繰り返す。

自動的な動作を制御する一つの方法は、図 8.10 に示されたプロパティを調整することである。特定のアクションを指定するには、`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」に割り当て、待ち時間に座るようにする。そして、待ち行列を出てから、`ModelEntity.Animation` に「」を割り当てる（割り当てられていたアクションを消す。つまり、特定のアクションを削除する）と、デフォルトに戻る。

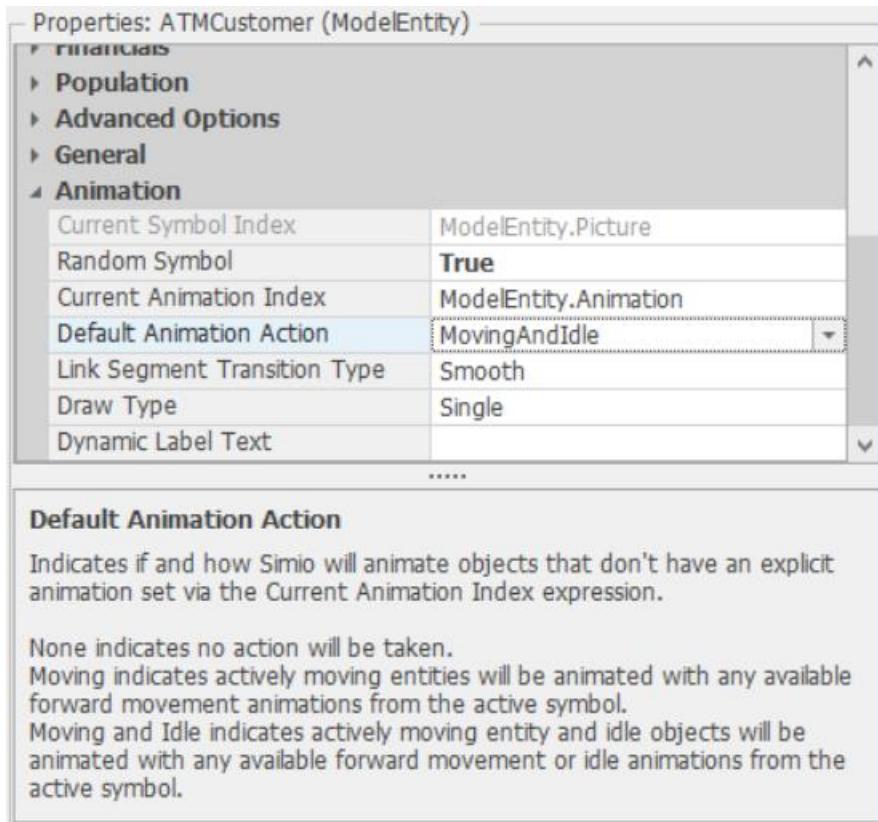


図 8.10 エンティティの Animation プロパティ

8.1.7 Model 8-1 : PCB 組立のアニメート

上述のいくつかの新しい知見を用いて、PCB モデルへの作業を続けることにする。しかし、開始する前に以前指摘した注意事項を念頭に置いて、プロジェクトの目的を満たすために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。しばしば、ちょうどよい適切なシンボルを見つけることができないことや、またはまったく適切に表示することができないことがあるかもしれない。しかし、それでじゅうぶんなのである。合理的な近似は、たいていはじゅうぶんよいことである。時間と指示が与えられた後に、アニメーションを改良すればよい。

5.4 節で説明した PCB Model5-3 を読み込むことから開始し、Model8-1 として保存しよう。これは作り物のシステムであるので、アニメートするなかで、いくつか自由なことをその中に入れることができる。最初に、PCB オブジェクトに用いるための、よりよいシンボルを見つけよう。

- PCB オブジェクトをクリックする。次に、Symbols ライブラリをクリックする。ライブラリから探して、PCB（これはコンピュータ・チップ上の小さい基盤であることを想起してほしい）として用いることができるシンボルがそこにあるかどうか確認する。残念ながら、あまり近いものがないので、探し続けよう。
- もう一度 PCB オブジェクトをクリックする。次に、Download Symbol ボタンをクリックする。もしオンラインであるなら、Trimble 3D Warehouse 検索画面が表示される。そこで PCB を検索してほしい。検索では、多くのシンボルが見つかり、いくつかのよい候補があった。Gumstix Basix R1161 というシンボルを選択し、Download Model（モデルをダウンロード）を選択する。ヒント：検索時に最初に表示される Nat's PCB のようなシンボルが適切なシンボルであると思うだろうし、それはおそらく適切である。しかし、それをダウンロードした後に、モデルサイズが数メガバイト増加したことがわかるだろう。これは、そのシンボルが複雑で非常にポリゴンが多いためである。たまたまそのような大きいシンボルをダウンロードしてしまい、モデルを膨らませたくないなら、シンボルをより小さいものに取り替え

- て、Project→Symbols ウィンドウから大きいシンボルを削除することができる。
- シンボルがダウンロードされた後に、Simio Import ウィンドウが表示される。ここでは、シンボルを回転し、リサイズし、記録することができる。それらをすべて行なってみよう。短い端が右になるように回転させるために Rotate ボタンを用いる。もっとも長い寸法を 0.3 メートルにしたいと考えているので、0.3 になるように Width を変更する（他の寸法が比例して変化することに注意する）。そして、最後に Name を PCB にし、Documentation を Gumstix Basix へ変更する。
 - シンボルはかなり小さいものが現れるが心配はいらない。後でそれに対処する。

部品配置機械を変更するために同様のプロセスを行う。

- 部品配置機械は、まったく Simio シンボルライブラリにはないので、そのステップをスキップする。Placement オブジェクトをクリックする。そして、次に、Download Symbol ボタンをクリックして、3D Warehouse 検索画面を表示させる。そこで、placement machine（部品配置機械）を検索してほしい。
- Fuji QP351 を選択し、それをダウンロードした。正しい方向とサイズでダウンロードされたので、Placement オブジェクトに適用するには、ただ OK をクリックするだけでよい。

3つの並列のステーションもまた部品配置機械であるので、同じシンボルを用いることができるだろう。しかし、実際には正確な部品配置ができるより高性能な機械を表すために、それらの機械には異なったシンボルを選択してほしい。上側の機械から、上述したプロセスを繰り返す。異なるマシン（高性能に見えた Fuji QP 242e を選んだ）を選択し、それをダウンロードする。選択したものを使切に回転させ、サイズを調整する。180 度回転させて、幅を 1.5 メートルに変えなければならなかった。

シンボルをダウンロードするたびに、それがプロジェクトライブラリに追加されることに気づくだろう。ここで、それぞれ他の 2 つの並列機械をクリックして、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。そして、検査、再加工の両方は手作業なので、適当な机のシンボルを用いて、それら両方をアニメートする。別のシンボルをダウンロードすることもできるが、ここでは Equipment カテゴリの下のシンボルライブラリの TableSaw が、要求を満たす作業台としてじゅうぶんに見えるので、それを両方の手作業のステーションに適用した。

指示された通りにされているなら、ここで、不愉快に感じられるかもしれない。シンボルは 2D における白地に対してあまりよく見えない。そして、事実、それらはかなり小さく見える。この問題は、モデルの詳細さのことである。以前のモデルでは、ステーション間は瞬時の移動として簡素化した仮定にしていたので、部品の細部、機械のサイズ、およびそれらの近似に関して関心がなかった。たいてい、簡素化した仮定は、プロジェクトの初期においては非常によい仮定であるが、通常、そのことが重要な問題となってくる。現在がまさにその段階である。

多くの場合は、背景画像として用いることができるシステムレイアウトの JPG、または他のイメージファイルを持っているだろう。そして、そのレイアウト画像を用いて、設備のシンボルを適切なサイズに調整して置くことができる。この例題の場合は、それがない。オリジナルモデルでは、約 10 メートル離して機械を置いた。しかし、実システムでは、それらは約 1~2 メートルだけ離れている。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仕上げよう：

- スクリーンいっぱいになるように、3 つの並列ステーションへズームする。
- （もっとも見やすい）スクリーンの下側から開始する。入力ノード（Input@FinepitchSlow Station）をクリックする。そして、部品配置機械シンボルの入力側に隣接するようにドラッグする。出力ノードでそれを繰り返す。3D ビューにするとよく見えるかもしれない。ただ

し、両方の視点でよく見えるように微調整するために、2D と 3D 間を切り替える必要があるだろう。アニメーションのノードとパスは今、図 8.11 のようになるだろう。

- Processing.Contents という名前の待ち行列（2D ビューにおける機械上の緑色のライン）は、現在機械に処理されているものを表示する。機械上（実際は機械の内側）にこれがあると、よりよく見えるだろう。2D ビューでは、それが部品配置機械の土台部分にあるように、待ち行列をドラッグする。この時点では、それはまだ床にある。3D ビューに移動し、それが土台部分の上にあるように、待ち行列を Shift キーを押しながらドラッグして持ち上げる。そして、図 8.11 で示されているように、土台部分の中心に待ち行列が置かれるように、線の両端を動かす。ヒント：大きいものの内部で小さいシンボルを「見失う」ことはよくあるので、一般に、それが正しい位置にあるのを確認できるまで、表示しているオブジェクトより長い待ち行列のままにしておくのがよい。そして、最後に長さを調整するとよい。
- Output Buffer の待ち行列の長さを短くし、出力側に隣接するように動かす。入力バッファの容量はゼロとしているので、入力側に待ち行列は用いない。混乱を避けるために、それを削除する。
- 次に、他の 2 台の部品配置機械で上の 3 ステップを繰り返す。
- Placement、Inspection および Rework でも、同じ過程に従って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のアニメーションは使用する可能性があるため、削除せずに残しておく。

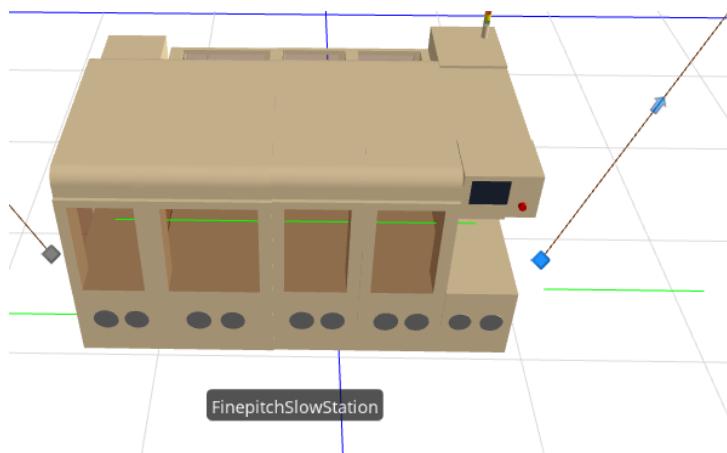


図 8.11 FinepitchSlowStation に関するアニメーションの編集

上述のすべてを完成した後も、モデルはまだ中途半端に見える。ここで実行すると、動き回っている PCB シンボルは、小さいドットに見えるだろう。シンボルとそれらの関連オブジェクトのスケールを固定したが、まだレイアウトのスケールを現実のものに調整していない。ここで、その対処をしよう。

- 2D ビューで、FinepitchSlowStation を選択して、FinepitchMediumStation から約 1 メートル離れたところに、平行になるように上側にドラッグする（グリッドの利用が役立つだろう）。次に、FinepitchFastStation を選択して、FinepitchMediumStation からの約 2 メートル離れたところで、平行になるように下側にドラッグする。待ち行列とノードが各オブジェクトと共に動くことに留意する。
- オブジェクト間に約 2 メートルの水平距離があるように、残りのオブジェクトを動かす。
- 現在、標準のライブラリシンボルを用いているオブジェクトは、かなり大きく見える。望むなら、それらを他のシンボルに取り替えることもできるが、ここでは適切なサイズへ縮小させるためにただ角をドラッグした。

最後の拡張として、床といくつかの壁を加えよう。

- 2D Facility ビューで、設備の床面積を完全にカバーする長方形を描くために Drawing リボン上の Rectangle ボタンを用いる。床をグレイにするために、Color ボタンを用いる。床をグリッドが透けて見えるようにしたいなら、3D ビューに行き、Shift キーと押しながらドラッグして、床を少し下げる。今度は、床で右クリックする。そして、Lock Edits を選択することによって、偶然それを動かしてしまうことを防ぐことができる。
- 2D ビューに戻り、床の左、裏および右端の周りに壁を表す polyline を描く。Drawing パネルで、壁の Width を 0.2 とし、Height は 1 メートルとする。3D ビューで、Texture ボタンから興味あるテクスチャを選択して、壁に適用する。右クリックして Lock Edits を選び、壁が偶然に移動するのを防ぐ。

本節では、アニメーションを少し改良して、それをおおよそ実寸図示したものにするために（図 8.12）、前節で学んだ簡単な技法のいくつかを用いた。

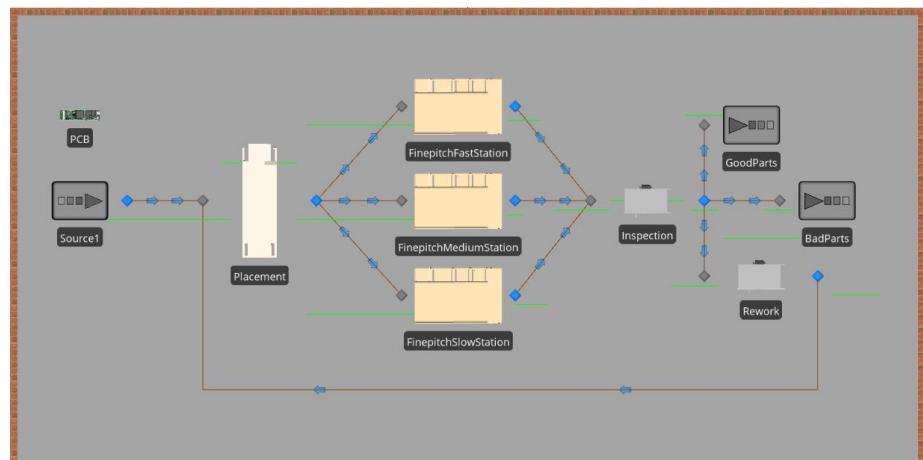


図 8.12 実寸図示した PCB の 2D アニメーション

ここでしたように、「事後に」実寸表示に修正できることは便利である。しかし、一般に初めから、少なくともおおよその実寸表示したものでモデルを構築するほうが簡単である（望ましくは、ある種の縮尺拡大図から始めるといい）。しかしいずれにしても、ほんの少しの労力で、かなり簡素なエンジニアリングツールで、図 8.13 のようにもう少し現実的なものに変換することができる。

さらに詳細に注目してアニメーションの改良をし続けたが（作業台をアップグレードさせるなど）、残りは課題としておこう。しかし、後でいくつかの追加拡張をするために戻ることにする。



図 8.13 実寸図示した PCB の 3D アニメーション

8.2 エンティティ移動

エンティティのある位置からある別の位置までの動きは、これまでのモデルの大部分に取り入れてきた。いくつかのモデルでエンティティが動いているものを見てきたが、他のモデルでは瞬時に起こる移動であった。実際、Simio でエンティティが動くためには多くの方法がある。図 8.14 はいくつかの移動手段を示している。もっとも簡単でもっとも直感的な方法は、Standard ライブラリに組み込まれたネットワーク機能を用いるものである。以下の項でそれについて詳細に説明する。しかし、まずはネットワークに依存しない移動について、非常に簡潔な紹介から始める。

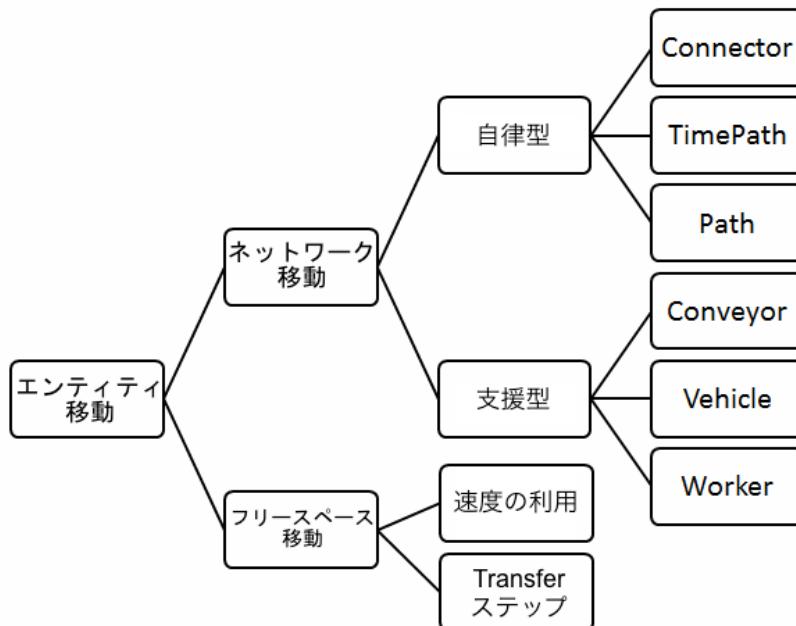


図 8.14 Simio でエンティティが移動できる方法

8.2.1 フリースペースを通るエンティティ移動

多くの場合、Standard ライブラリに組み込まれたリンクの移動サポートを利用したくなるが、より多くの柔軟性が必要となるケースがある。フリースペースとは、Simio がネットワーク上に存在しないモデルの領域（たとえば、オブジェクト間の「空間」）を説明するのに用いる用語である。エンティティが「物理的な位置」（図 8.15 に図示したステーション、ノード、またはリンク）に位

置していないとき、それはフリースペースに存在している。エンティティはフリースペースに存在でき、フリースペースを通って動くことができ、そこでの移動をアニメーションすることができる。主に、物理的な位置が関わるかどうかによって、図 8.14 のフリースペース移動の下の 2 つのオプションに分かれる。

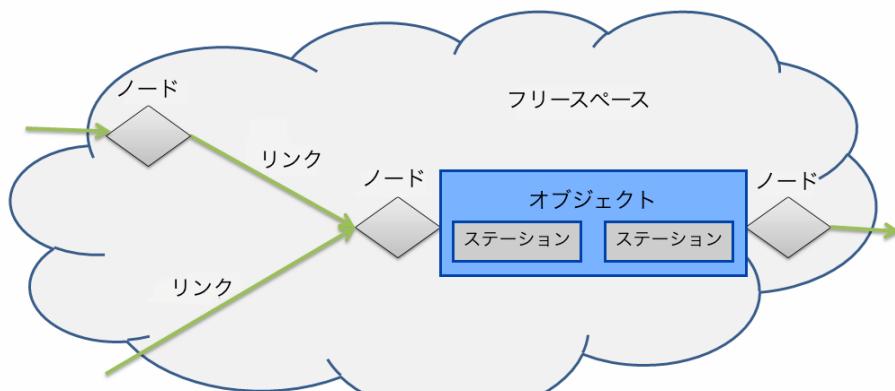


図 8.15 エンティティはノード、リンク、ステーションまたはフリースペースに存在可能

オフネットワークで瞬時に移動させたいのであれば、プロセスで Transfer ステップを用いることができる。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、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（おそらく瞬時に起こる）移動を開始する。この一般的な利用方法の 1 つは、Create ステップと組み合わせることである。エンティティが生成されるとき、それらはフリースペースで生成される。すぐに変更を加えなければ、デフォルト速度でフリースペースを通り、移動し始めるだろう。そして、Create ステップの後に、指定された位置への移動を指示する Transfer ステップが続くのが一般的である。

もちろん、エンティティが瞬時の移動先として、物理的な位置のないフリースペースを通って移動したい場合もあるだろう。これは、特に、エージェントベースモデリング (ABM: Agent-Based Modeling) で一般的なことである。エンティティの位置、方向、速度および加速度を指定するために移動状態変数を設定できる。具体的な設定は次の通りである：

- 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 および Movement.AccelerationDuration

これらのパラメータの大部分はフリースペースだけで働き、リンクでは機能しないことに注意する（リンク上にあるときは、たとえば、加速度を設定するのにこれらは用いない）。しかし、少し簡単に行うために、Simio は Travel ステップを提供している。それは、上述した機能と、より容易に設定できるように提供されている。提供される追加機能の 1 つは、任意の絶対座標または相対座標、あるいは指定した任意のオブジェクトに、フリースペースを介して直接移動させることである。Travel ステップは、Direct To Destination または Follow Network Path のいずれかの Steering Behavior プロパティが設定される。後者は、廊下を通る人々の自然な動きに類似した行動を生成するために使用することができる（たとえば、経路の幅にわたって分散させ、お互いを回避しようとする）。

設定を簡単にするために、エンティティ (ModelEntity, Vehicle および Worker など) の Initial Travel Mode プロパティには、次の 3 つの値がある。

- Network Only は、エンティティがネットワーク上を移動でき、フリースペースを自動的に移動しないことを示す。
- Free Space Only は、エンティティがフリースペースを通過することができ、ネットワーク上を自動的に移動しないことを示す。その名前が示すように、フリースペースを介してオブジェクトからオブジェクトへ直接移動する代わりに、そのエンティティはネットワーク上を移動するという制約から解放される。このことは、多数のロケーションがある（全国的な流通ネットワークがあるような）ときや、エンティティが特定の経路に従う必要がない場合に、特に役立つだろう。
- Network If Possible は、上の 2 つを組み合わせたデフォルトの設定であり、エンティティがネットワーク上を移動して目的地に達することができれば、これを実行するということを示している。それ以外の場合は、自動的にフリースペースを通過する。

3 番目のオプションはデフォルトのオプションで、たとえば、リンクが接続されていない Source と Sink を配置すると、フリースペースを利用してエンティティがそれらの間を移動できる。そのため、Network If Possible は非常に使いやすいが、慎重に扱う必要がある。これを用いると、エンティティはほぼ常に目的地に到達することができる。これは一般によいことであるが、エンティティをネットワークに制約させることが求められる場合や、必要なリンクを入れるのを「忘れた」場合は別である。その場合、モデルは機能するが、その動作は間違っており、このようなエラーは発見するのが難しいかもしれない。

5.4 節で習得したエンティティの目的地を選択することに対するほとんどの概念においても、フリースペースの移動が適用される。デフォルトを変更しない限り、エンティティは、常にもっとも近い目的地まで移動する。しかし、Entity Destination Type において、利用可能な一連のオプション（たとえば、By Sequence、Continue、Select From List、Specific）がある。

いくつかの例については、SimBit の FreeSpaceMovement を参照してほしい。そこでは、フリースペース移動の概念について、3 つのモデルで説明されている。SimBit の TravelWithSteering Behavior でもステアリング動作について説明されている。

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、多くの場合、Standard ライブラリによって提供されている機能を用いて、ネットワークを介してエンティティを動かしている。最初の選択が必要となることは、エンティティがそれ自体で動いているかどうか（たとえば、それ自身で動く人間、あるいは自律型のオブジェクト）、または動くために支援（たとえば、人、乗り物、コンベア、または他の装置）を必要とするかどうかということである。図 8.14 のネットワーク移動下のオプションは、次項で詳しく説明する。ここでは、最初にリンクが何であるか、そして、それらの何が共通しているのかを説明しよう。

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したオブジェクトとして定義されている。この定義には多くの概念が含まれている：

- リンクはオブジェクトである。それは、実行中に動くことができない固定オブジェクトである。
- リンクはノードを接続する。それ自体、または 1 つのノードだけでは存在できない。
- エンティティは、リンク上を移動する唯一のオブジェクトである。それには、より多くの制約があるようと思われる。エンティティから派生しているため、Vehicles と Workers もまた（追加のふるまいをする）エンティティである。

さらに、すべてのリンクに共通の概念がある：

- ・ リンクには、開始ノードと終了ノードがなければならない。
- ・ リンクは 1 つ以上のネットワークのメンバーであり、ネットワークはリンクの集合である。リンクは、常に「Global」ネットワークと呼ばれる特別なネットワークのメンバーである。
- ・ リンクオブジェクトには、等しく区切られた位置（セル）で規定される長さがある。
- ・ リンクの制御と位置はエンティティの先頭で決定される。エンティティの先頭がリンクに移動すると、エンティティの移動はそのリンクに組み込まれたロジックによって制御される。
- ・ Simio では、それぞれのエンティティの先頭と末尾を追跡し、そして衝突と追越イベントを検知する。これにより、プロセスにおけるリンクの挙動を定義し、カスタマイズすることが比較的簡単になる。

Standard ライブラリのリンクオブジェクトには、コネクタ (Connector)、パス (Path)、タイムパス (TimePath) およびコンベヤ (Conveyor) がある。次に、これらの最初の 3 つのオブジェクトについて説明しよう。

8.2.2.2 コネクタ(Connector)

コネクタはもっとも単純なタイプのリンクである。それらは直接、内部の時間遅延なし（ゼロ時間）で、2 つのオブジェクトを接続する。移動が瞬時に起こっているので、エンティティ間には、干渉がまったくない。事実上、一度に 1 つのエンティティだけがリンクを横断している。まるでエンティティが直接開始ノードから終了ノードまで、または直接終了ノードを持つオブジェクトまで、移動しているように機能する。

コネクタには、Selection Weight プロパティがある。複数の出力リンクがノードに接続されるときに、1 つの出力リンクを選択するのに用いられる。この機能は、5.2 節で説明し、Model 5-1 で用いた。

8.2.2.3 タイムパス (TimePath)

タイムパスには、上述した Selection Weight を含む基本的なリンク機能のすべてがある。しかし、コネクタのようなゼロ時間移動の代わりに、エンティティがタイムパスを移動する時間を指定する Travel Time プロパティがある。また、タイムパスを同時に移動するエンティティの数を制限できるように、Traveler Capacity がある。そして容量制限のための、Entry Ranking Rule がある。これは、潜在的な待ちエンティティのうち、次にどれに移動の許可を与えるか選択するために利用される。

また、タイムパスには、進行方向を決定する Type がある。これは、Unidirectional か Bidirectional のどちらかである。Unidirectional は開始ノードから終了ノードまでの一方の進行のみを許可する。Bidirectional は、どちらの方向の進行も許可するが、同時に一方の進行のみを許可する（つまり、複数のエンティティが同時に別方向に向かって進行し、それ違うことはできない）。完全に双方向の経路をモデル化するには、別々の单方向の経路として各経路をモデル化する必要がある。双方向のパスでは、パス状態変数の DesiredDirection に次のいずれかの値を割り当てることによって、ロジックで方向を制御することができる：

```
Enum.TrafficDirection.Forward
Enum.TrafficDirection.Reverse
Enum.TrafficDirection.Either
Enum.TrafficDirection.None
```

タイムパスが On Entering と Before Exiting において State Assignments をサポートして

いることに留意する。また、それらは、重要な場面でロジックを実行できるように Add-on Process Triggers もサポートしている。これらは、リンクの挙動と、関連オブジェクトとの相互関係をカスタマイズするために、非常に強力な機能である。

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。その代わりに、パスは個々のエンティティの DesiredSpeed とパスの長さに基づいて移動時間を計算する(たとえば、毎分 2 メートルの Desired Speed を持つエンティティは、6 メートルのパスを横断するために少なくとも 3 分を必要とする)。また、移動時間に影響を与える他のパラメータもある。Allow Passing³パラメータが False に設定されると、移動速度の速いエンティティが遅いエンティティに追いついたとき、速いエンティティが前に続くよう自動的に遅くなる(この例に関しては SimBit の VehiclesPassingOn Roadway を参照してほしい)。また、パス自体では、エンティティが Desired Speed で移動するのを妨げるよう Speed Limit を課すこともできる。

パスには、Drawn To Scale プロパティもある。Drawn To Scale を False に設定すると、描かれた長さとは異なる論理上の長さを指定できる。これは、正確な長さを必要とする状況、またはアニメーション目的のために描かれた長さを圧縮したい状況で役に立つ(たとえば、部署内外の移動をモデル化している際に、部署間の距離が長い場合)。

8.2.3 コンベアの利用

8.2.3.1 一般的なコンベア概念

コンベア (Conveyor) は、装置上または装置に沿ってエンティティを動かす、固定位置の装置である。コンベアは(パスのような)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を表すので、特別に説明することにする。コンベアには多くの異なるタイプがある。箱を扱うような動力つきまたは重力で動かすローラコンベアや、食品や石炭など運搬するベルトコンベアやパケットコンベア、器具や車体を扱うようなオーバーヘッドパワーコンベアやフリーコンベアがある。また、重工業が何十年間もコンベアに頼っている一方で、軽工業やその他の産業で目を見張るような応用例が目立つようになっている。食品と消費財産業では、瓶に入れたり、缶詰めしたり、パッケージ作業するのに、精巧な高速コンベアを用いている。通販やドラッグストアは、処方薬を調合する過程を自動化するのに役立つように、軽い負荷のコンベアを用いている。多くの大規模食料品店が、ベルトコンベアをレジに組み入れることもしている。

Standard ライブラリの Conveyor オブジェクトは、上述した Path オブジェクトと多くの機能を共有しているが、また重要な新機能も備えている。現在の Simio コンベアは反転しないので、Type プロパティがない。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動く。コンベアには Speed Limit プロパティがない。代わりに、作動時の(初期)コンベア速度を示す Desired Speed プロパティがある。そして、Allow Passing プロパティがない。動いているすべてのエンティティが同じ移動速度(コンベアリンクの速度)で移動するため、追い越すことはコンベア上では決して起こらない。

エンティティはコンベアの速度で動いているか、または停止している。(動いているか、または停止しているかに関わらず) 速度がまさにコンベアの速度と同じなら、エンティティは連動するという。エンティティが動いていないのに、コンベアが動いているなら、(たとえば、コンベアがエ

³ あるエンティティへ Allow Passing を許可すると、別のエンティティも通過することができる。これは、2つのエンティティが実際に同じ空間を占めることができるという点で、興味深い側面である。複数のエンティティが同時にパスに進入した場合、デフォルトの True では、すべてが同じスペースを占有し、単一のエンティティだけが移動しているように見えることを意味している。

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になる。それがどのように起きるのかについて、さらに説明する。

Entity Alignment は、エンティティにとって、コンベアに載る有効な位置を決定するプロパティである。Entity Alignment が Any Location であれば、エンティティはリンクに沿ったどんな位置でも連動できる。Entity Alignment が Cell Location であれば、コンベアが持っている Number of Cells を指定しなければならない。この場合、エンティティは、先頭がセルの境界に並んだときだけ連動できる。バケットコンベアの上のバケツやパワー&フリーシステム上のキャリアのような、コンベアに関する離散的な構成要素があるときに、このプロパティが使用される。

Accumulating プロパティは、エンティティがコンベアから非連動になることができるかどうかを決定する。Accumulating が False に設定されると、コンベアの上のあらゆるエンティティがいつも連動する。また、エンティティが停止しなければならない場合(たとえば、端に達したとき)、コンベアとコンベア上のすべてのエンティティも止まらなければならない。Accumulating が True に設定されると、エンティティがコンベアの端で停止するとき、そのエンティティはコンベアから非連動になる。一方、コンベアとコンベア上の他のエンティティは、動き続けるだろう。各エンティティがコンベアの端で止まっているエンティティに追いつくとき(または、「衝突」するとき)、そのエンティティもまた非連動になり、停止する。これらのエンティティはコンベアの端に蓄積したと呼ばれる。

停止していたエンティティがコンベアを去るとき、逆のプロセスが発生する。移動の障害が取り除かれるため、2番目のエンティティは再連動することを試みる。Entity Alignment が Any Location であれば、その試みはすぐに、いつもうまくいく。Entity Alignment が Cell Location であれば、先頭がセル境界に並ぶまで2番目のエンティティは待たなければならない。そのとき、エンティティは再び連動し、移動を再開する。

コンベアがパスと異なっている最後の点は、コンベアが装置を表すので、(5.3.4 項で説明した Server のものと同様に) Reliability プロパティを持っているということと、信頼性に関連した Add-on Process Triggers を持っていることである。

8.2.4 Model 8-2：コンベアがある PCB 組立

PCB モデルへの作業を続けて、いくつかの新しく得た知見を応用してみよう。部品配置のすべてがコンベアによって供給され、コンベアで配送される。そして、それらはインバウンド側およびアウトバウンド側にバッファ領域をまったく持っていない。つまり、部品はコンベアへと、あるいはコンベアから直接供給され、部品が移動できないときはブロックされる。

そこで、3つのステーションを選択し、Output Buffer の Buffer Capacity を 0 に変更する (Input Buffer はすでに 0 である)。6個すべてのコネクタを選択するには、Ctrl キーを押しながらクリックを繰り返せばよい。この操作で、グループとしてこれらのプロパティを変更し、構成することができる。

- グループのど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、6個のコネクタをコンベアに変換する。
- Desired Speed を 1 Meters per Minute に変える。
- コンベアのように見えるようにしたい。Simio 内蔵のコンベアパスデコレータを適用できるが、利用するにはそれらは大き過ぎる。そこで、より現実的に見えるコンベアパスを表す線を作ることにする。
 - General Category→Size and Location→Size→Width で、0.2 メートルに設定する。
 - Draw パネルで Texture ボタンの下半分をクリックし、Corrugated Metal パターンを選択する。ローラコンベアのように見える。コンベアの 1 つにそれを適用する。

- Texture ボタンの上半分をダブルクリックして、他の 5 つのコンベアにテクスチャを適用する。
- ・ コンベアの見た目をよくしたので、接続した機械の正しいポイントに部品を届けるために、エンドポイントの高さを調整する。適切な高さで機械に到着できるように、それぞれのノードの高さを動かすために、3D ウィンドウで Shift キーを押しながらドラッグする。

Inspector はテーブルに 3 つの部品しか蓄積できない。他のすべての部品は、入って来るコンベア上に残るようしなければならない。したがって、Inspector の InputBuffer を 3 に設定する。検査の入力バッファ待ち行列 InputBuffer.Contents を調整し、図 8.16 のように、検査員のテーブル上に 3 つの部品を表示し、配置するのにじゅうぶんな大きさの短い線にする。



図 8.16 ノードと待ち行列の位置を示す 3D ビュー

モデルを実行し、厳密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止められることに留意する。一方で、他の部品はコンベア上でまだ動いている。特にオフシフトのときに、Inspector に入ってきたコンベア上にエンティティが溜まり始めることに留意する。そして、FinepitchFastStation は、生産率がより速いため、より早く部品が滞留する傾向がある。しかし、それは、まさにブロックしたくない機械である。したがって、コンベア上のエンティティに、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によって、より速い機械で加工せるようにしよう。

- ・ 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、割り当てるために表示される。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この手順を繰り返し、ここでは New Value プロパティを 2 に設定する。
- ・ Inspection (Input@inspection) のエントリーノードを選択する。そして、Initial Capacity プロパティを 3、つまり各コンベアから 1 つずつに設定する。これは、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 から選択されることを意味する。それらのエンティティはまだコンベアを去っていないが、それらの先頭がノードにあることに留意する。

- Inspection を選択して、Ranking Rule を Largest Value First へ、Ranking Expression を Entity.Priority に変更する。これで、もっとも長い間待っているものを選択するよりも前に、まずコンベアから最優先するエンティティを選択できる。

実際には、検査員がオフシフトのとき、部品は優先権に関わらずまったく加工されないので、この変更はモデル挙動においてあまり効果がない。しかし、一番上のコンベアが空のままで残っているのに役立ち、もっとも速い機械のブロックも最小限となり、できるだけ早く空に戻る。

これまで、その他のコンベアの機能についてほとんど触れなかった。多くの異なるタイプのコンベアをモデル化するために、様々なプロパティを組み合わせができる。また、複雑な合流や分岐のある状況では、複雑なシステムをモデル化するために、コンベアまたは周囲のオブジェクトのアドオンプロセスで補うことができる。そして、Conveyor は変更可能な Process ロジックですべて制御できるので、独自の挙動をする Conveyor オブジェクトを作成することもできる。

8.2.5 Worker の利用

前節では、エンティティがいろいろと移動できる様々な方法について説明した。エンティティ移動の補助にはかなり広い種類がある。それには、エンティティが目的地に着くのを補助するための治具、カート、人、またはバスのような何らかの制約付きリソースも含まれる。もっとも簡単なケースとして、固定リソースを考えてみよう。リソースの現在位置からの移動時間や、動くアニメーションが重要でないときは、必要なときに固定リソースを占有し、完了したときに解放することでじゅうぶんであるだろう。必要なことが本質的に移動の許可だけである場合に、このアプローチがとられるだろう（たとえば、あなたを指さして、通行の許可を与える交通整理員）。

次に説明するより一般的なケースは、運搬者が必要で、その現在位置と他のプロパティや状態変数が重要な場合である。Simio の Standard ライブラリはまさしくその目的のために、2 個のオブジェクトを提供している。Worker と Vehicle である。2 つは Entity から派生したオブジェクトであり、エンティティのすべての機能と同じ挙動をし、さらにそれ以上の能力もある。ノードにおいて挙動が少し異なることを除けば、それらはネットワークを介してエンティティと同様に動くことができる。

8.2.5.1 Worker

Worker は他のエンティティを運ぶ機能を持ったエンティティである。Worker は動的であり、実行中にランタイムオブジェクト (RunSpace) 内に作成される。Initial Number in System で指定することによって、いくつの Worker のコピーが利用可能かを決定できる。各コピー（別々のオブジェクト）が個人を表すため、Worker には Initial Capacity プロパティがない。つまり、1 以上を割り当てる事がない。Worker が主として人を表すことを意図するので、Reliability のような装置特有のプロパティはないが、通常、作業員が従うことになる Work Schedules をサポートしている。Worker のいくつかの主要なプロパティは、図 8.17 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、可能であれば、フリースペースを使用する。Routing Logic Category では、Worker が開始するネットワーク上の場所と、「Home」ロケーションに向かう場合に戻る場所を指定するために、Worker に Default Node (Home) の値を設定する必要がある。指定する必要があるアクションは 2 つある。Idle Action は、Worker が作業および作業要求がないときに取る動作を指定する。OffShift Action は、Worker がオフシフト（容量がゼロ）になるときに取る動作を指定する。これらの動作の両方には、2 つの選択肢があ

る。最初の選択は、現在のノード位置に残るか、または指定されたホームノード位置に戻るかどうかである。2つ目の選択は、Park（アニメートされるかどうかはわからないが、ネットワークから出て駐車場に移動）するか、ネットワークに留まるかどうかである。後者では、通過したい交通を妨げる可能性がある。

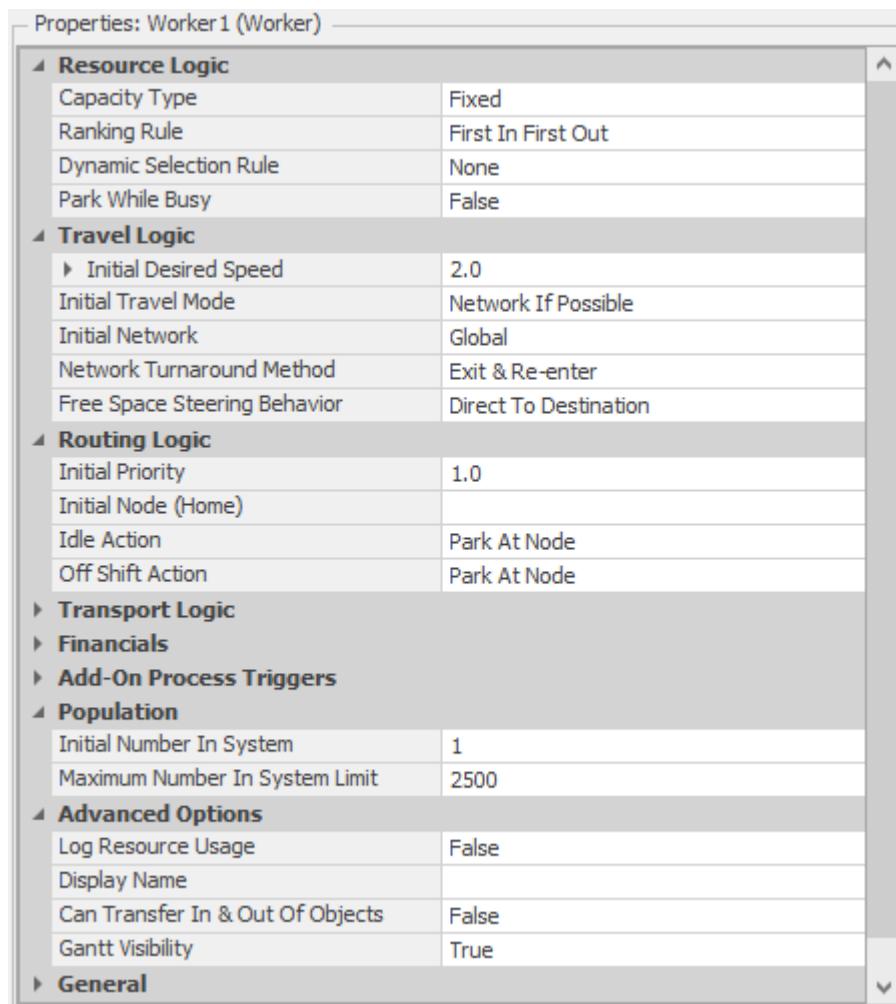


図 8.17 Worker オブジェクトのデフォルトプロパティ

Transport Logic カテゴリでは、Worker が同時にいくつのエンティティを運ぶことができるかを指定する Ride Capacity がある。また、Load Time と Unload Time もある（エンティティ単位で必要な積み降ろしの時間を決定する）。

まさしくエンティティと同じく、Worker には、Population に Maximum Number in System Limit がある（それは検証ツールとして提供される）。そこで、予想される Worker の最大数を設定できる。そして、この限界を超えて Worker が生成されると、Simio はエラーを起こす。また、Advanced Options には、Worker が外部ノードを通してオブジェクトに入ることができるかどうかを制御する Can Transfer In & Out Of Objects プロパティがある（たとえば、サーバに入るなどを許可するかどうか）。多くの場合、Worker がエンティティや他のオブジェクトに対してサービスを提供するので False をデフォルトとするが、通常 Worker は、サーバでエンティティが行うのと同じプロセスロジックを実行しない。

Worker には、2つのモードがある。1つは可動リソースとして、占有され、解放され、モデルの位置の間を移動するモードである。もう1つは、トランスポータとして、積み込み、運搬し、別の場所にエンティティを降ろすことができるモードである。

8.2.5.2 リソースとしての Worker の利用

Worker は、まさしく他のリソースオブジェクトと同様に、占有され、解放されることが可能である。Server、Combiner、そして Separator オブジェクトでは、Secondary Resources プロパティを用いて、直接規定できる。すべての Standard ライブラリオブジェクトでも、アドオンプロセスで Seize ステップと Release ステップを用いて、リソースを占有し、解放できる。

いずれの場合でも、最終的には図 8.18 のような Repeating Property Editor が表示される。もっとも簡単な利用法は、Resource Type を、どのリソースが必要であるか既知であることを意味する Specific に設定したままにすることである。そして、必要なオブジェクトの名前（たとえば Worker1）を Resource Name プロパティに入力する。それがトランスポータ（たとえば Worker または Vehicle）であれば、Specific リソースを指定しても、同じ名前の複数のユニットがあるかもしれないことに留意する。リソースのセットから交互に選択したいのであれば、Resource Type に FromList を選択する。これで、選択したいリソースのリストの名前を指定することができる。

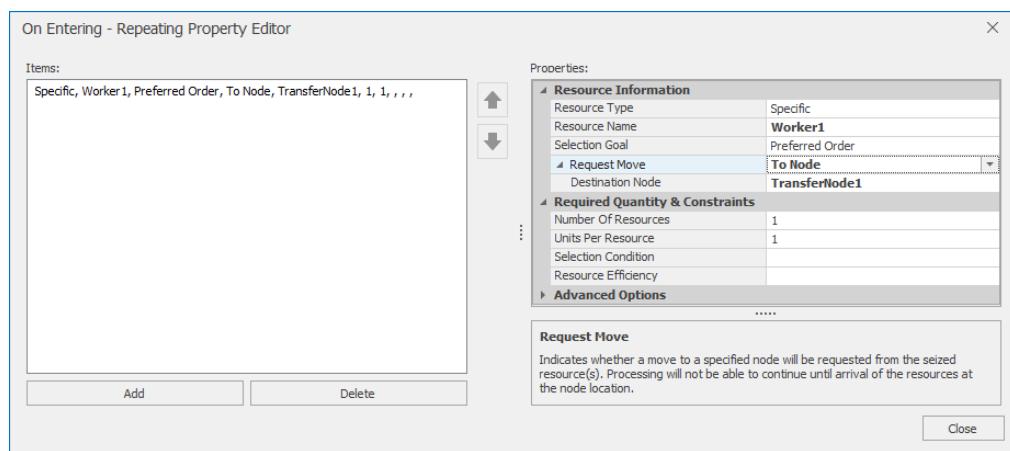


図 8.18 可動オブジェクトを占有するためのプロパティ

複数のリソースが選択可能な場合には、Advanced Options カテゴリで選択肢から選択する規則がある (Selection Goal と Selection Condition)。また、Advanced Options カテゴリで、必要な数とその他のオプションも指定できる。

リソースを任意の位置に動かすことが重要であるなら、Request Move プロパティに ToNode を指定して、Destination Node プロパティでリソースが訪問する位置を指定できる(図 8.18 参照)。目的地としてどんなノードでも用いることができるため、固定オブジェクトに隣接するようにノードを追加し、可動リソースの目的地としてそのノードを利用するように設定することはさらに簡単である。

Request Move は、可動リソースに対してだけ機能する。つまり、Worker や Vehicle のようなエンティティから派生したオブジェクト、もしくは Entity のみである。Request Move を指定するとき、リソースが占有され、指定されたノードに動かされるまで、トーカンは Seize ステップを出ない。リソースが最終的に解放されるとき、自動的に別のタスクに再割り当てされるか、もしくは Idle か OffShift になる。

8.2.5.3 トランスポータとしての Worker の利用

Worker は、手軽な Vehicle として考えることができる。Worker は、位置間で 1 つ以上のエンティティを運ぶことのできる基本的な輸送能力を持っている。Worker の利用設定は、Vehicle や他の Transporter の利用設定と同様である：

- TransferNode を選択する（ほとんどの Standard ライブラリオブジェクトは出力側にトラ

ンスファーノードがある）。

- Transport Logic カテゴリでは、プロパティ Ride On Transporter を True に設定する。いくつかの追加プロパティが表示される。どのトランスポータを用いたらよいかを指定するまで、モデルはエラー状態にあり、実行されない。
- Transporter Type プロパティは上述したリソースの Object Type プロパティと同様に働く。名前から特定のトランスポータを要求し、または適切な Selection Goal と Selection Condition を用いてリストから選択することで、それを用いることができる。
- Reservation Method は、どのようにトランスポータを選択して、予約したいかを指定する。3つの選択肢がある：
 - Reserve Closest：現在もっとも近くにあるトランスポータを選択し、同順位の場合は Selection Goal を用いる。そして、トランスポータは Reserve（予約済み）となる。予約は双方向の義務である。トランスポータは（即座ではないにしろ）そこに行くことに責任を持ち、エンティティは予約されたトランスポータを待つことに責任を持つ。
 - Reserve Best：Selection Goal を用いて、もっともよいトランスポータを選択し、同順位の場合はもっとも近いものを利用する。そして、トランスポータは Reserve となる。
 - First Available at Location：予約をしないが、最初の適切なトランスポータがピックアップのために到着するまで、代わりにただ待っている。トランスポータが（バス運行表のような）順序に従うか、またはカスタムロジックで指示されない限り、トランスポータがまったく届かない可能性があることに留意する。

8.2.6 Vehicle の利用

Vehicle は、Worker とほぼ同様の利用方法である。それは、占有され、解放され、位置間を移動することや、そして、エンティティを載せ、運搬し、降ろすというトランスポータとして挙動するという、同じ機能を持っている。実際に、それには多くの同じ機能があるが、主な違いは、意図する使用目的が人ではなく装置をモデル化することである。また、いくつかの付加機能がある。

Vehicle には、そのルートの挙動を指定するための Routing Type がある。On Demand として指定すると、Vehicle は要求に応じて訪問するために知的に対応し、適切に乗車と降車の計画をする。Fixed Route として指定すると、Vehicle はノード訪問の順序（7.1.3 項）に従う。都市交通用語では、On Demand はタクシーのサービスのように働くことであるが、Fixed Route はバスや地下鉄のように働くことである。

Vehicle には Reliability Logic がある（装置はしばしば故障するものと認識される）。Vehicle に対しての故障は、Calendar Time Based か Event Count Based で設定できる。後者は非常に柔軟性に富んでいる。外部のイベントに反応するだけではなく、移動した距離、充電状態、またはその他のパフォーマンス基準など、追跡したものに基づいた内部のイベントをトリガーとすることができます。上記のような違いはあるが、Vehicles と Workers は似たように挙動し、用いられる。

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に、学んだことを用いよう。ED Model 7-5 から始め（Model 8-3 とする）、Worker を用いてスタッフ配置を拡張する。しかし、ED の機能を向上させる前に、若干のアニメーションを修正してみよう。Trauma（検査室）、Treatment（治療室）、および Exam（診察室）のオブジェクトは、個々のベッドを表してはおらず、複数の患者を扱う領域を表している。たしかに、個別に各ベッドを表すようにモデルを充実させることもできるが、まだ、そのレベルの詳細さは必要でない。代わりに、複数のベッドのある部屋を表すために、いくぶん改良したアニメーションシンボルを作成しよう。

8.2.7.1 新しいシンボルの作成

Project 名はデフォルトで、ファイル名（たとえば Model_08_03）と同じである。ナビゲーションウィンドウにおいて Project 名を選択すると、プロジェクト構成要素のすべてを含むウィンドウが表示される。このウィンドウで、プロジェクト構成要素を加えたり、削除したり、編集したりすることができる。アクティブなシンボルを見るには、左の Symbols アイコンをクリックする。ここでは、シンボルを編集したり、またはプロジェクトから未使用のシンボルを削除したりすることができる。このケースでは、シンボルを追加したい：

- Create New Symbol をクリックする。Facility ウィンドウと同様の描画ウィンドウが開き、2D と 3D のビューといくつかのおなじみのリボンがある。
- 約 10 メートル × 3 メートルの部屋を作成したい。Polyline ツールを用いて、1 つの線で四方の壁となる 4 セグメントを作成する。この説明では行わなかったが、ドアを表すために 1 つの線ではなく、途中に切れ目を入れて描いてもよい。Drawing リボンで、Object Height を 1、Line Width を 0.05 に設定し、任意の Texture を適用する（ここでは Banco Nafin を用いた）。
- 各患者間に仕切りを設置したい。再び Polyline を用いて、最初の仕切りを作成する。Drawing リボンで、Object Height を 1、Line Width を 0.04 に設定する。そして、任意の Texture を適用する（Dark Green Marble を用いた）。
- 1 メートルずつ離れて設置された、合計 9 つの仕切りを作成するために、最初の仕切りをコピーして、貼り付ける。
- Properties ウィンドウでは、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、シンボルは図 8.19 のように表示されるだろう。
- 完了したときに、Project ウィンドウか Model ウィンドウに戻るには、Navigation ウィンドウを用いる。その際、シンボルは自動的に保存される。

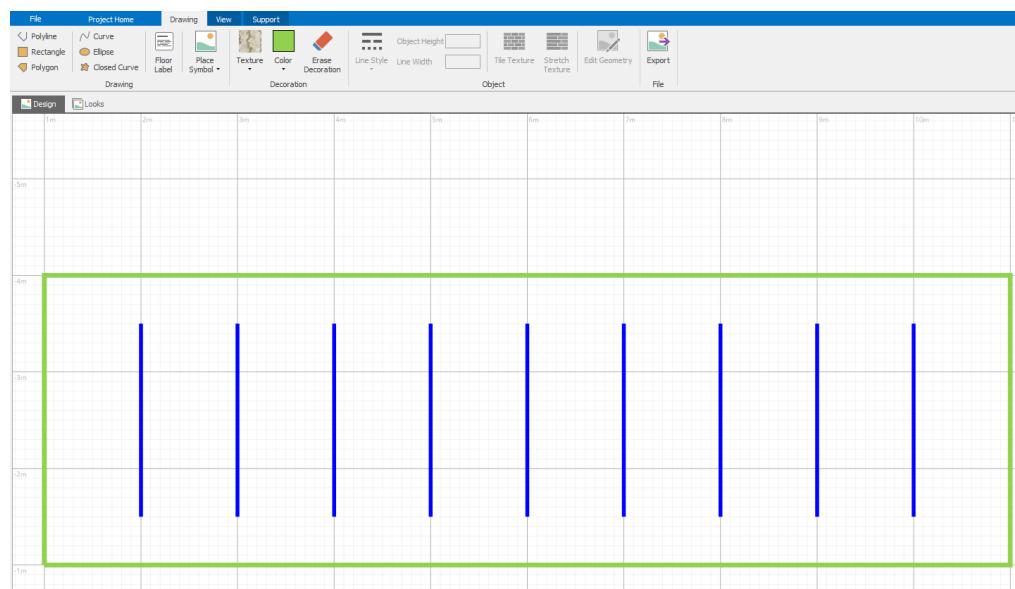


図 8.19 治療室の簡単な描画

いま、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ている。これを、ExamRooms、TreatmentRooms および TraumaRooms への新しいシンボルの適用に用いる。これらはオリジナルのシンボルとはかなり異なるサイズであるため、おそらくモデルの実行を一時停止して、関連する部分を調整するべきであろう（調整の例は図 8.20 を参照）：

- ・レイアウトが自然に見えるようにオブジェクトの位置を調整する。
- ・ノード位置を調整して、オブジェクトの外側に配置する。ドアを作成していたなら、調整は必須だろう。
- ・それらのオブジェクトを接続している元のパスを削除し、適切なマルチセグメント(折れ線)のパスに取り替える。
- ・それぞれの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動かす。そして、出力ノードの近くへ OutputBuffer.Contents を移動させる。さらに、部屋の長さに合わせて Processing.Contents を伸ばす。これは、それぞれの部屋に「占有者」(つまり患者)を表示するための簡便な方法である。

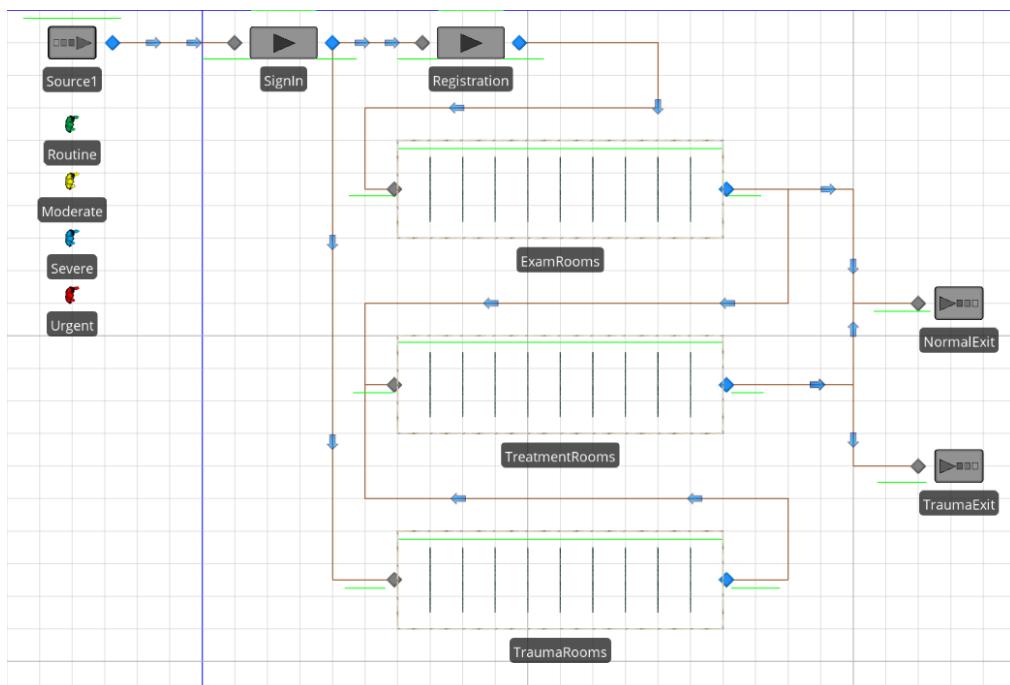


図 8.20 複数のベッドのある部屋を新しいオブジェクトシンボルでモデル化した
Model 8-3

8.2.7.2 モデルの更新

ED のレイアウトを更新したので、スタッフ配置を加えよう。2 つのタイプのスタッフをモデル化する。看護師 (Nurse) と看護助手 (Aide) である。表 8.1 で指定されるように、診察室 (Exam)、治療室 (Treatment)、および検査室 (Trauma) において、すべての患者に対して介助役をつけたいと考えている。Routine (定期検診) の患者は、Aide が利用可能になるまで待つ。Moderate (中程度) の患者と Severe (重度) の患者は、可能であれば Aide によって介助される。そうでなければ、利用可能になる最初の Aide か Nurse に付き添われる。Urgent (緊急) の患者は、可能であれば Nurse によって付き添われ、そうでなければ Aide によってなされる。すべての患者が出口まで、可能であれば Aide によって付き添われるが、そうでなければ Nurse によってなされる。

それでは、ホームと呼ぶ Worker のための場所を作成することから始めよう。

- ・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。1 メートルに壁の高さを設定する。そして入口を表す BasicNode を加えて、NurseStation と命名する。
- ・ノードの自動的な駐車待ち行列の機能をオフにして、ナースステーションの内部に駐車待ち行列を描きたい。NurseStation を選択して、自動的な駐車待ち行列の機能を解除するため Appearances リボンの Parking Queue ボタンを押す。Draw Queue ボタンを用いて、

ParkingStation.Contents 待ち行列を選び、部屋の中に待ち行列を描く。必要なら、多くの看護師が並ぶように、部屋の周りに待ち行列を描くこともできる。必要に応じて、看護師「たち」をうまく並べられるように待ち行列線を部屋に巻きつけることができる。

- 図 8.21 のようにナースステーションを作成する。
- 上の 3 つのステップを繰り返して、Aide のホームになる AideStation を作成する。

表 8.1 目的地までの移動に必要とされる介助のタイプ

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

次に、Worker を生成して、構成しよう。

- Worker をモデルに配置し、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 と設定する。Idle Action と OffShift Action を Park At Home に設定する。Population カテゴリでは、モデルで 4 人の看護師 (Nurse) から始めるために Initial Number in System に 4 と設定する。
- 別の Worker をモデルに配置し、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 と設定する。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 に設定する。これはモデルで 6 人の看護助手(Aide) から始めるることを示している。

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、選択が行われる際に参照されるオブジェクトの集合である。たとえば、目的地（ノード）の選択やリソース（オブジェクト）を選択する機会が多いだろう。リストからの選択は、たいてい Random や Smallest Value のようないくつかの選択メカニズムをサポートするが、一般的にデフォルトは Preferred Order である。Preferred Order は、リストの先頭から、利用可能な最初のものを選択することを意味する。

再度、表 8.1 を見ると、ときには特定のタイプの Worker の付き添い（例：Aide）を必要とし、ときには Preferred Order リストから選択する（たとえば、利用可能であるなら Aide を用い、不可能なら Nurse を利用する）ということがわかるだろう。移動に関してすでにシーケンス表を用いているので、そのテーブルに新しいプロパティ（列）としてこの情報を加えることは、とても簡単である。Worker がトランスポータの一種であることを想起してほしい。この場合、プロパティタイプは、トランスポータもしくはトランスポータリストのどちらかである。各プロパティは、特定の 1 つのデータ型でなければならないため、すべてをトランスポータリストとし、選択の余地がないケースにはリストにただ 1 つのメンバーを加えることとする。では、表 8.1 を再び確認してから、リストから作成しよう。

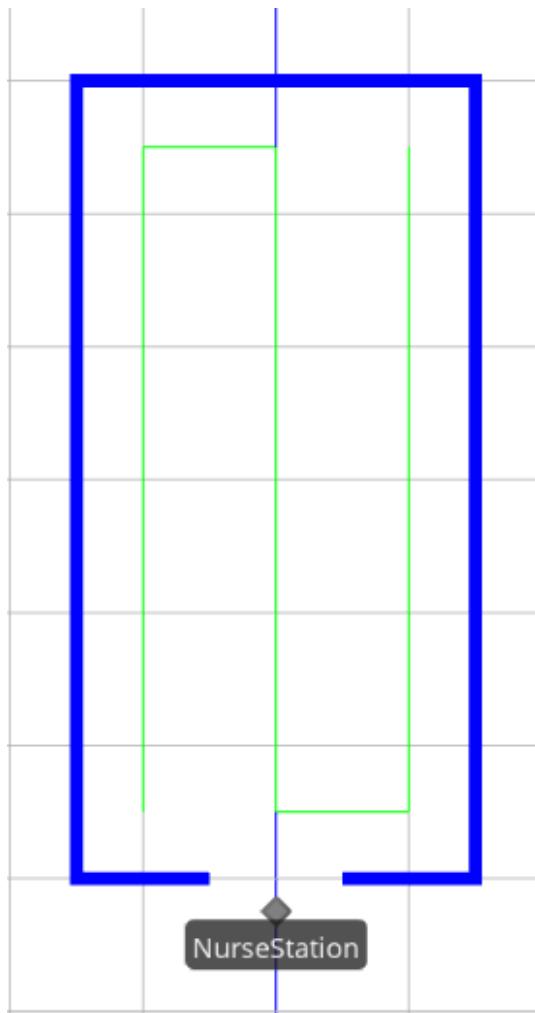


図 8.21 ノードと駐車の待ち行列のある Nurse ステーション

- リストは、Definitions ウィンドウの Lists パネルを用いて定義される。新しいリストを加えるためには、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。再び表 8.1 を参照すると、付き添いとして選ぶ必要がある 3 つのリストが存在する：Aide のみ、Aide 選好、および Nurse 選好である。3 回 Transporter ボタンをクリックして 3 つのリストを作成し、それらを AideOnly、AidePreferred および NursePreferred と命名する。
- AideOnly リストを選択すると、スクリーンの下部には、そのリスト内のトランスポータの一覧が表示される（現在は空である）。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 つの選択肢でよいため、これで定義完了である。
- AidePreferred リストについても、上述のプロセスを繰り返す。1 行目に Aide を選び、2 行目に Nurse を設定する。このリスト順序は、Preferred Order ルールを用いてこのリストから選択するとき、利用可能であるなら Aide を選択し、そうでない場合に利用可能であれば Nurse を選択することを意味する（どちらも利用可能でない場合は、利用可能になる最初のものを選択する）。
- もう一度、上述のプロセスを NursePreferred リストで繰り返す。1 行目に Nurse を選び、2 行目に Aide を選ぶ。

これで、リストが作成できたので、Treatments テーブルにデータを加えることができる。Data ウィンドウの Tables パネルで、Treatments とラベルされたタブを選択する。このテーブルは、患者が次に訪問するべき場所を選ぶときに、またその場所へ移動する間、あるいはそこへ着いたときに用いるプロパティを決定する際に、患者によって参照されることを思い出してほしい。このテーブルでは、診察室 (Exam) と検査室 (Trauma) の場所だけではなく、訪問される場所のすべてがカバーされている。では、ここで移動手段のためのプロパティを加える。

- Object Reference ボタンをクリックする。そして、Transporter List タイプを選び、そのプロパティを EscortType と命名する。各地点に付き添いをするために、どの Worker グループから選んだらよいかを指定するためにこれを用いる。
- 新しい列のデータを完成するために、表 8.1 のデータを用いる。完成すると、図 8.22 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 8.22 介助スタッフを加えた 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することによって、このモデルの拡張を終える。ネットワークへの追加を開始する前に、1 件の若干の修正をする必要がある。SignIn (受付) を離れるほとんどの患者は、付き添いなしで直接 Registration (診療受付) に行く。これらの患者では、Ride On Transporter プロパティを False にする必要がある。しかし、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 True に設定する必要がある。このジレンマを解決する 1 つの簡単な方法は、SignIn の隣に新たに TransferNode を加え、Urgent の患者はそちらのノードに向かわせることである。すなわち、すべての患者が付き添いなしで SignIn を離れるが、Urgent の患者はすぐに介助を待つためのノードに行くことになる。この変更のために、SignIn から TraumaRooms までの既存のパスを削除しなければならない。そして、SignIn に隣接して新しいノードを加えて (WaitUrgentEscort と命名する)、SignIn

から WaitUrgentEscort まで、および WaitUrgentEscort から TraumaRooms までのパスを加える。

これで、ネットワークを拡張する準備が整った。スタッフは既存のネットワーク上で患者に付き添うことになるが、スタッフもまた位置間を移動するのに彼ら自身のパス上を移動している。そして、すべての入力・出力ノードから別の入力・出力ノードまで、自由に移動できるようにする必要がある。そのために、スタッフ移動を円滑にするためのパスを加えなければならない。

このネットワークの設置にはいくつかの方法がある。1つの極端なアプローチは、すべての組のノード間に2つのパス（各方向へ1つ）を引くことである。これは、明らかにかなり退屈な作業であるが、起こりうる移動経路を完全に表現できるだろう。別の極端なアプローチは、パスを一方向への巡回（たとえば、時計回り）としてしまうことである。これは、おそらくパスの数がもっとも少なくなるが、効率が悪くて非現実的な移動ももたらすだろう。ここでは、中庸のアプローチを取る。つまり、一方向への巡回のパスとするが、移動をより効率的にするためにいくつかの近道ができるように、特定の目的地ノードを接続するパスを引けるように追加的なノードを加える。

中庸のアプローチとして、図8.23のハイライトしたパスで示されるノードとパスを加えた。

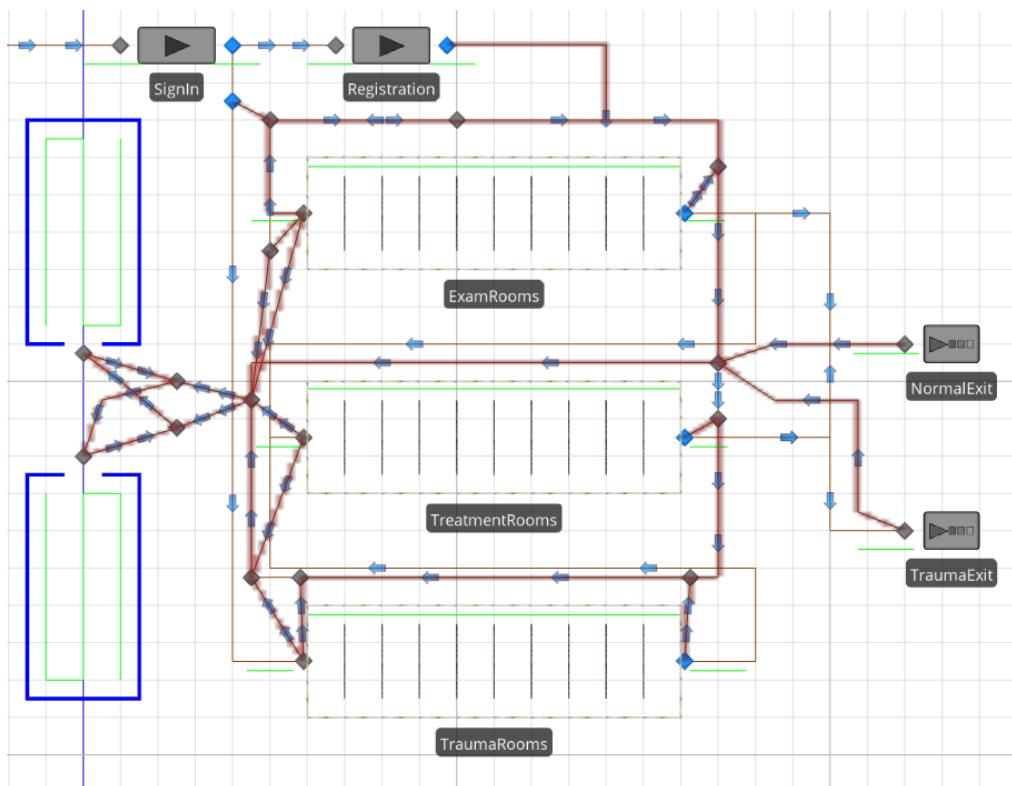


図8.23 スタッフネットワークのパス（ハイライト部分）

図を参考に、パスとノードをモデルに追加してほしい。具体的には、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。特定のエンティティのアクセスを制限するためにネットワークを用いることもできるが、ここではその機能は必要としない。しかし、ネットワークが複雑になると、可視化とデバッグに際して、命名されたネットワークにそれらを割り付けすることが便利である。この方法を用いるには、パスをハイライトさせ、View リボンで View Networks ボタンを選択する。

これまで、スタッフを生成し、選択をサポートするためのリストを作成し、選択すべきものを参照できるようにテーブルにデータを追加した。そして、介助を表すためにパスを設定した。すべての土台が整ったので、いつ付き添いを待つかをエンティティに設定する必要がある。トランスマスターノードでそれを設定する。

新しいノードの WaitUrgentEscort から始めよう。それを選択して、Ride On Transporter を

True に設定する。Transporter Type に From List を設定する。テーブルからこの患者と場所のリストを得たいので、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 (つまり、どのように Best を定義するか) は Preferred Order のままにする。

介助が必要な他の 3 つの出力ノードを選択する (Registration、ExamRooms および TraumaRooms)。そして、それらのプロパティの変更も同様に行う。個別に設定することもできるし、グループ選択して一括して行うこともできる。

まだモデルの保存をしていないのであれば、ここでモデルを保存して実行しよう。すべての設定を正しくできているなら、このモデルの旧バージョンのように患者が移動する様子を見られるだけでなく、本項の設定により、いくつかの場面では患者にスタッフが付き添って移動しているだろう。患者がスタッフの正面ではなく Worker の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、おそらく次のように思われるのではないだろうか。つまり「実際の病院と比べるとどうだろうか？患者が自ら治療をするのか？」このモデルでは、スタッフは患者に付き添って移動しているが、患者が治療を受けているときにはいなくなっているではないか！」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関しては、様々な拡張方法があるため、本章で解説した以外の方法は読者自身で確かめてみてほしい。

8.3 まとめ

本章では、2D・3D アニメーションの価値や、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、より優先すべきプロジェクト業務や他の分析技術を追いやつてまで、アニメーションにあまりにも多くの力を注ぐことの落とし穴についても説明した。

モデルをアニメートするための基礎知識を身につけるために、多くの基本的なアニメーションや表示オプションについて簡単に述べた。そして、動的なオブジェクト（たとえば、エンティティ）が、フリースペースやリンク、コンベア上を移動したり、また Standard ライブラリの Worker や Vehicle のような別の動的オブジェクトと共に移動したりする方法について議論した。

上記の内容に加えて、他にも多くのモデリング状況やアプローチがあるが、それは第 10 章で説明する。

8.4 問題

最初の 4 つの問題の目標は、必ずしも有用な分析を行うことではなく、アニメーションの問題をより理解することにある。この目的を達成するために、別途指定されない限り、問題を解くために妥当なモデルパラメータを自ら考えて作成してほしい。

1. Model 4-3 と同様の新しいモデルを構築しなさい。このモデルでは主としてアニメーションが重要であるので、それをできるだけ現実的にするために、本章で学んだことを用いる。顧客は少なくとも 2 人の男性、2 人の女性、および 1 人の子供を含む複数のアニメーションされた（徒歩で移動しているような見た目の）人々を用いる。現実感を増すために Trimble 3D Warehouse からシンボルを用いる。
2. 問題 1 の銀行では、ATM を待つ顧客のために、ベンチシートを追加することにした。問題 1 の解決策を拡張してベンチを追加し、その後、向きを決められる待ち行列点とアニメーション・インデックスを使用して、ベンチに座って待っている顧客をアニメーションで表示する。その待ち行列で Keep In Place オプションを選択すると、どのような影響があるか？ATM 処理の待ち行列を変更して、処理中に顧客が ATM の方を向くようにし、そのアニメーション動作を改善するために適切なアニメーション・インデックスを追加しなさい。
3. Model 7-2 を拡張して、各エンティティインスタンスにフロアラベル (floor label) を追加

- し、サービス水準 (Level of Service。分単位のシステム内時間とする) を表示すると共に、患者の種類を示すために背景色を用いなさい。また 1 つ以上のフロアラベルをモデルに追加し、各患者タイプについて次の 3 つの情報を表形式で表示しなさい：Number In-Process (処置中人数)、Number Processed (システム離脱人数)、システムから離脱した患者の Average Level of Service。
4. 3 組の Source-Path-Sink を使って、Source から対応する Sink へのエンティティの移動に Vehicle を利用するモデルを構築しなさい。既存のライブラリから乗り物のシンボルを用いなさい。それぞれの組で別々の Network Turnaround Methods (Vehicle プロパティ) を設定し、各アプローチを比較しなさい。
5. Speedee Car Rentals は 24 時間営業で、2 つのクラスの顧客がいる。25%はプレミアム顧客であり、入店してから 5 分未満の待ち時間であることが保証されており、それを超えればレンタル代金から\$15 ドルが割り引かれる。他の顧客は、自分の順番が来るまで待フレギュラー顧客である。顧客は 2 分毎に（指數分布に従って）到着し、プレミアム窓口（1 人の係員）もしくはレギュラー窓口（2 人の係員）へと、時速 1 マイルの速さで 4 メートルの距離を移動する。すべての顧客は、2 ないし 7 分の一様分布に従うサービス時間がかかる。各係員には 1 時間当たり\$55 の費用がかかり（間接費含む）、各顧客はサービスを完了すると収入に\$8 が計上される（ただし、割り引かれた場合は、ここから差し引かれる）。リンクを利用せずにシステムをモデル化しなさい。10 日間の反復実行を行いなさい。各シナリオにおいて、各タイプの係員に対する稼働率と、各タイプの顧客に対する待ち時間を報告しなさい。画面上および出力結果として、割引を受けたプレミアム顧客の人数、総割引額、およびシステムの純利益（割引額と費用を考慮した後の額）を報告しなさい。
- ・ シナリオ a) 各係員は、指定された種類の顧客しか応対しない。
 - ・ シナリオ b) 上の条件を緩和する。同じ係員に対して、別の就業規則（どのような条件でどの顧客に誰が応対するか）を考える。評価基準への影響から、各シナリオを評価しなさい。
 - ・ シナリオ c) システムを改善するために、その他のソリューションをモデル化しなさい。リンクを用いるとよいかもしれない。

第9章 Simio の高度なモデリング

本章の目標は、これまでの4つの章で扱われなかつたいくつかのSimioの高度なモデリング概念について説明することである。本章では3つのモデルを紹介し、それぞれ1つ以上の発展的な概念を示す。本章中で共通のテーマは、包括的な実験と、OptQuest®アドインによるシミュレーションを用いた最適化の活用である。9.1節では、8章で紹介した救急診療のモデルを再考し、その問題における意思決定の側面に焦点を合わせる。意思決定機能の一部として、1つの**総費用**(total cost)評価基準を設定し、代替システム構成を評価するためにその評価基準を用いる。この節では、OptQuestのシミュレーションベースの最適化アドインを導入し、このアドインを用いる方法を解説する。9.2節では、ピザのテイクアウトレスポンスマネジメントモデルを導入し、プールされたリソースシステムを実証するためにWorkerオブジェクトを組み込む。そして、最初に手動で定義した実験を行い、次にOptQuestを用いて、最適なリソースレベルを探索するために、このモデルを用いる。最後に、9.3節では、ステーション間に有限容量バッファがある状態での組立ラインモデルを開発し、異なるバッファ配置のラインスループットを見積もるための設定や実験の方法を説明する。もちろん、ここで説明しきれない多くの高度な機能や概念があるので、本章をさらに進んだ研究における出発点としてほしい。多くの他のシミュレーションやSimio特有の概念は、SimioのSimBitsや、年に一度のWinter Simulation Conference (www.wintersim.org) や他のシミュレーション関連会議の論文集などで紹介されている。

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。しかしながら、実際の患者の治療については説明しなかったし、モデルでの実験もまったく実行しなかった。この節の目標は、主として**意思決定**を支援するためにどのようにModel 8-3を変更するかを示すことである。特に、リソース配置とスタッフ配置の決定に役立つようにモデルを活用することに関心がある。患者の治療と評価基準の要素をモデルに加えることによって、Model 8-3を拡張することができるが、ここでは、Workerオブジェクトと移動経路を取り除き、エンティティ移動にはConnectorオブジェクトを用いて、モデルの抽象的なバージョンを作成することにした。このことに関する動機は2つある。1つ目は、抽象的なモデルは、より小さくより簡単であるため、本書で示しやすいためである。そして、2つ目は、プロジェクトにおける「もっともよい」シミュレーションモデルは、プロジェクトの需要を満たすもっとも簡単なモデルであるという概念を補強するためである。Model 8-3での目標は、StandardライブラリのWorkerオブジェクトを組み込んだSimioにおける移動とアニメーションの側面を実証することであった。Model 9-1での目標は、Simioを用いて意思決定に活用可能なモデルを示して、シミュレーションベースの最適化を実証することである。

図9.1はModel 9-1のFacilityビューを示している。モデルがSignIn、Registration、ExamRooms、TraumaRoomsおよびTreatmentRoomsサーバオブジェクトと、4つの患者エンティティオブジェクト(Routine、Moderate、SevereおよびUrgent)を持つことに留意してほしい。そして、NurseとAide WorkerオブジェクトをDoctorとNurse Resourceオブジェクトに取り替えた。また、移動ネットワークとPathオブジェクトは、Connectorオブジェクトを用いて簡易型のネットワークに取り替えた。そして、物理的な患者の動きをモデル化することにそもそも関心がないため(そして、モデルを簡素化したいため)、患者エンティティと様々な部屋のアニメーションシンボルを取り除いた。

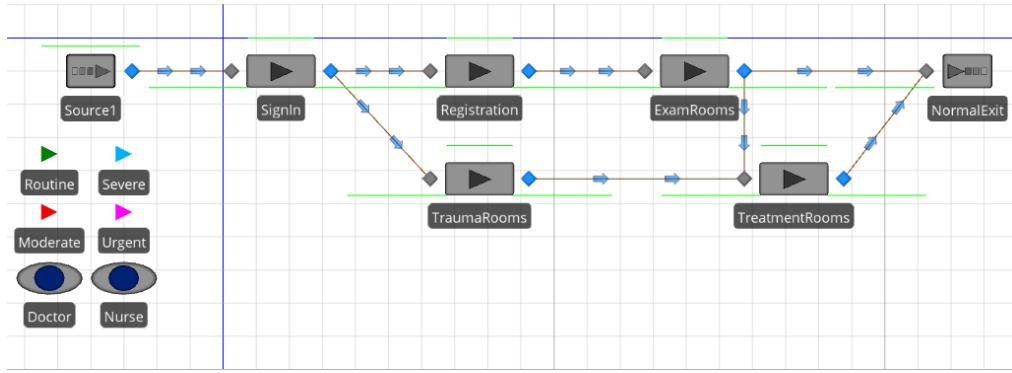


図 9.1 Model 9-1 の Facility ビュー

Model 9-1 は、4 つの基本型の患者が到着し、シーケンス表を利用して指定されたルートでシステムの異なった経路に従うという、同じ基本的問題構造を維持する。患者ルーティングの唯一の違いは、Urgent の患者は、ここでは NormalExit オブジェクトを通って終了するが、Model 8-3 では、TraumaExit オブジェクトを通って終了したということである。図 9.2 は Model 9-1 のために更新された PatientData データテーブルと Treatment Sequences テーブルを示している。WaitingTime 列が PatientData テーブルに加えられ、EscortType 列が Treatments テーブルから取り除かれていることに留意してほしい。リソース配置モデルの一部として、患者の待ち時間を追跡する。つまり、患者が、医者や看護師に会うのを待つ時間（患者が最初に到着してから、最初に医者や看護師に診てもらうまでの間隔を含む）が定義されている。患者サービスレベルを決定することと、リソースとスタッフ配置費用を最小にするという要求のバランスをとるために待ち時間用いる。患者タイプごとの待ち時間は、PatientData テーブルの Waiting Time エントリで追跡できる。ここでは患者とスタッフの物理的移動はモデル化していないので、EscortType 列を取り除いた。

Patient Data					Treatments			Patient Data			Treatments		
	Patient Type	Priority	Patient Mix	Waiting Time		Sequence	Treatment Type	ServiceTime (Minutes)		Sequence	Treatment Type	ServiceTime (Minutes)	
1	Routine	1	40	RoutingWaiting		1	Routine	2		1	Routine	2	
2	Moderate	2	31	ModerateWaiting		2	Routine	Random.Uniform(3,7)		2	Moderate	2	
3	Severe	3	24	SevereWaiting		3	Routine	Random.Triangular(5,10,15)		3	Moderate	Random.Uniform(3,7)	
4	Urgent	4	5	UrgentWaiting		4	Routine	0.0		4	Moderate	Random.Triangular(10,15,20)	
*						5	Routine	0.0		5	Moderate	Random.Triangular(5,8,10)	
						6	Routine	0.0		6	Moderate	Random.Uniform(3,7)	
						7	Routine	0.0		7	Moderate	Random.Triangular(10,15,20)	
						8	Routine	0.0		8	Moderate	Random.Triangular(5,8,10)	
						9	Routine	0.0		9	Moderate	0.0	
						10	Routine	0.0		10	Severe	1	
						11	Routine	0.0		11	Severe	2	
						12	Routine	0.0		12	Severe	Random.Triangular(15,20,25)	
						13	Routine	0.0		13	Severe	Random.Triangular(15,20,25)	
						14	Routine	0.0		14	Severe	Random.Triangular(15,20,25)	
						15	Routine	0.0		15	Severe	0.0	
						16	Routine	0.0		16	Severe	Random.Triangular(15,20,25)	
						17	Routine	0.0		17	Severe	Random.Triangular(15,20,25)	
						18	Routine	0.0		18	Severe	0.0	
						*				*			

図 9.2 Model 9-1 の PatientData と Treatments データテーブル

また、ExamRooms、TreatmentRooms および TraumaRooms リソースに加えて、Model 9-1 には患者の診察と処置のためのリソース要件も組み込む（Model 8-3 は、患者に付き添いするのに Nurse と Aide Worker オブジェクトを用いたが、それぞれの部屋に到着したとき、患者の処置については明確にモデル化しなかった）。表 9.1 に診察／処置要件を示す。

（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、（本当

の救急治療室で行われているように) 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つ。そして、それぞれの異なる数のシステム構成要素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、医師の所属人数より多くの診察／処置室を持つ典型的な病院を考えている。

表 9.1 患者の診察／処置リソース要件

部屋	診察／処置で必要となるリソース
Exam Room	Doctor もしくは Nurse (Doctor が優先)
Treatment Room	Doctor および Nurse
Trauma Room	Doctor および Nurse もしくは Doctor (Nurse が優先)

救急治療室は、(Model 8-3 でしたように) PatientData テーブルで指定された優先順位に基づいて、Doctor と Nurse リソースが優先される。したがって、たとえば、医師を待っている Routine 患者と Moderate 患者がいて、Urgent 患者が到着すると、Urgent 患者は医師を待つ待ち行列の前方に移る必要がある。同様にまた、待っている患者が Routine 患者しかいない場合は、Severe 患者か Moderate 患者が到着したなら、それらの患者は医師を待つ待ち行列の前方に移るべきである。これは、Doctor と Nurse リソースに RankingRule プロパティを用いることで実装できる。図 9.3 は Doctor リソースのプロパティを示している。Ranking Rule プロパティを Largest Value First に、Ranking Expression プロパティを PatientData.Priority に設定する。エンティティが PatientData.Priority フィールド⁴に保存された値に基づいて、リソースを待つ待ち行列に命令する（または順位付けする）ように Simio に設定する。Urgent 患者が Routine 患者から医師や看護師を先取り (preempt) するのを許容することも考慮できるが、それは読者のための練習問題として残すことにしてよう。

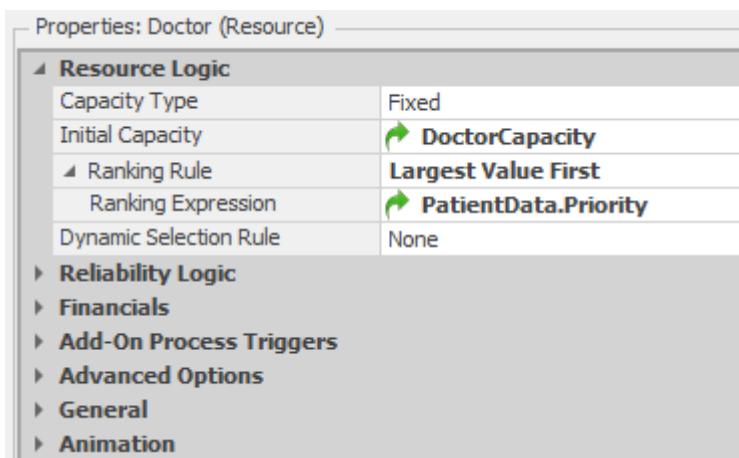


図 9.3 Doctor リソースのプロパティ

診察／処置リソースの患者への配置をモデル化するために、それぞれの部屋のリソースに関連づけられたアドオンプロセスを用いる (5.1.4 節を参照)。アドオンプロセスが既存の Simio オブジェクト内のロジックを補うためのメカニズムを提供することを想起してほしい。すなわち、現行モデルに、患者が部屋（診察室、治療室、または検査室）に到着するとき、処置を行う医師および／

⁴ リソースがモデルの別々の場所から占有されるとき、Simio には待ちエンティティのためのただ 1 つの内部待ち行列があることに留意してほしい。この待ち行列は、本モデルのようなケースで必要なグローバルな優先順位の実装を簡略化できる。

または看護師を「呼び出し」、そしてその「処置を施すリソース」が到着するまで部屋で患者が待つためのロジックを加える必要がある。これまで見てきた Server オブジェクトロジックは、処置を施すリソースではなく、部屋の容量の割当を扱っている。したがって、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ぶプロセスをアドオンする必要があるだろう。

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。

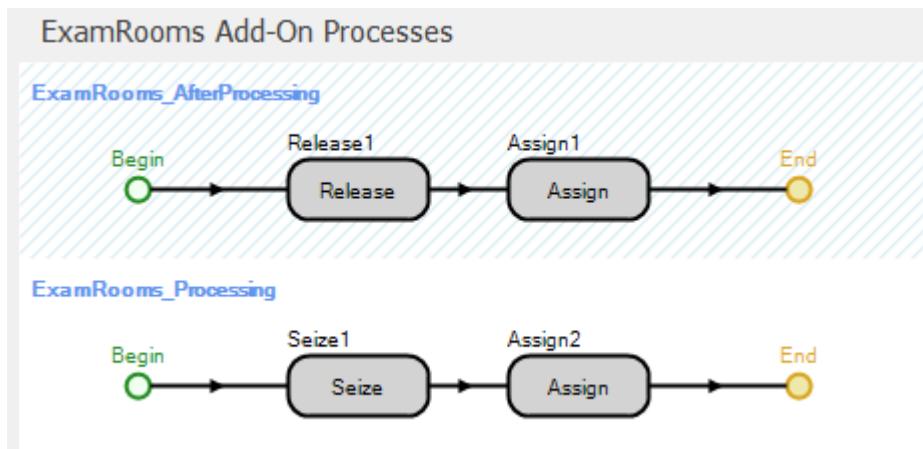


図 9.4 ExamRooms オブジェクトのアドオンプロセス

患者が ExamRooms サーバに入るとき、2 つのことをする。診察あるいは治療リソースを占有し、リソースが患者に割り当てられた際に待ち時間を記録する。表 9.1 に示されているように、診察室の患者は医師か看護師のどちらかを必要とし、もし両方が利用可能であるなら、医師を優先する。8.2.7.3 では、エンティティが複数の選択肢から好みの順番で選択するために、Simio のリストと PreferredOrder オプションの用い方を示した。ここで同じメカニズムを用いる。ここでは、Doctor と Nurse オブジェクトを含んでいるリスト (DoctorNurse。リストの最初は Doctor オブジェクト) を定義して、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 は DoctorNurse リストと Seize ステップの詳細を示している。Doctor は、より高い優先度で選択されるため、Selection Goal プロパティに Preferred Order を指定したことに留意してほしい。また、これと同じ機能性を達成するのに Server オブジェクトの Secondary Resources プロパティを用いることもできたが、これらのセカンダリリソースの割当は「内部」でどう働いているかについて明確に議論したいためであることに留意してほしい。

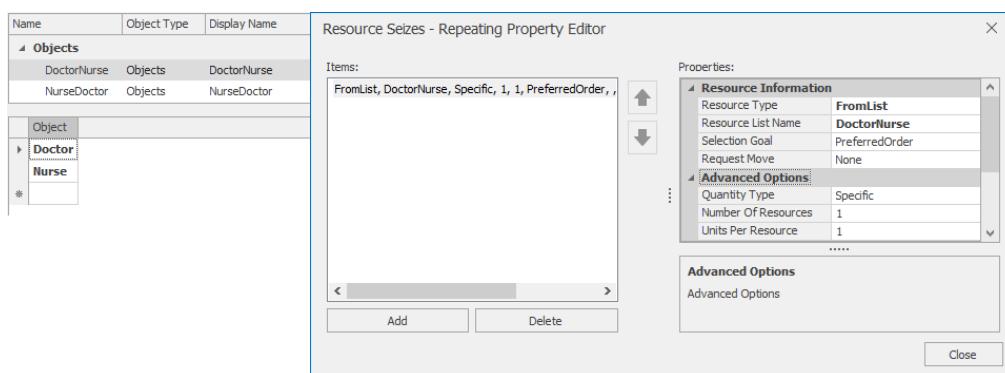


図 9.5 ExamRooms のアドオンプロセスのリストと Seize の詳細

Assign ステップにおいて、Assign2 は患者の待ち時間の初期構成要素を記録するために用いら

れ、Assign1 は患者が診察室を離れる時間を記録するために用いられる（後で詳述する）。医師または看護師リソースをエンティティに割り当てたとき、Assign2 ステップが実行される。特定の患者が医師もしくは看護師を待つ時間のすべてを追跡したかったことを想起してほしい。患者が複数の場所で待つ可能性があるため、各患者の総待ち時間を蓄積するのに、エンティティ状態変数（WaitingTime）を用いる。図 9.6 は 2 つの Assign ステップのプロパティを示している。

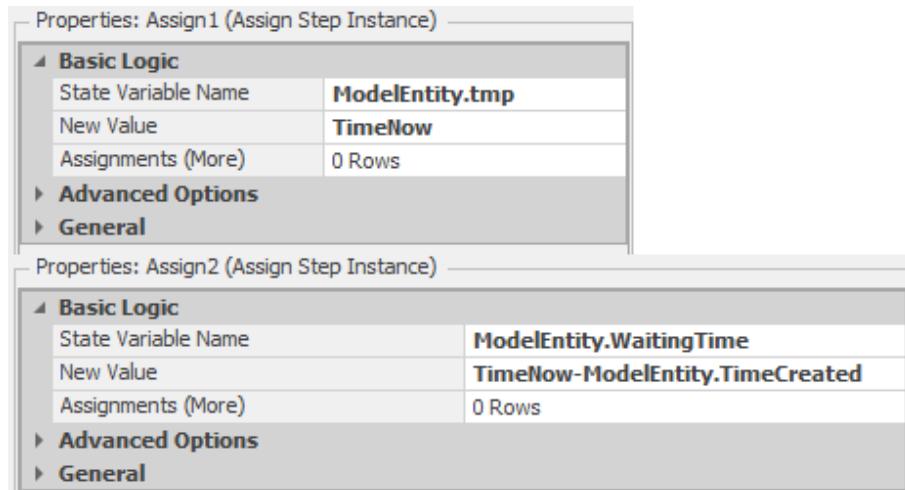


図 9.6 Assign ステップのプロパティ

Assign2 ステップでは、 $\text{TimeNow} - \text{ModelEntity.TimeCreated}$ の値を状態変数 `ModelEntity.WaitTime` 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（モデル状態変数の `TimeNow` で蓄積される）からエンティティが生成された時刻（患者が到着した時刻）を差し引き、エンティティ状態変数の `ModelEntity.WaitTime` にその値を代入する。もしエンティティが `Routine` の患者を表しているなら、これは患者の総待ち時間になる。その他のエンティティタイプでは、患者が `TreatmentRooms` に到着した後も待ち時間を追跡する。この追加的な待ち時間を追跡するために、Assign1 はそのエンティティを記録する。現在のシミュレーション時刻をエンティティ状態変数の `ModelEntity.tmp` に蓄積する。この時間は、エンティティが治療室に到着したときに利用され、必要なリソースが割り当てられる。`ExamRooms_Processed` ステップにおける Release ステップでは、前のプロセスで占有された医師もしくは看護師リソースを解放する。

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。プロセスは `ExamRooms` オブジェクトと同様である。唯一の違いは、`TraumaRooms_Processing` プロセスには 2 つの Seize ステップが含まれるということである。1 つは `Doctor` オブジェクトの占有であり、もう 1 つは `Nurse` オブジェクトもしくは `Doctor` オブジェクトの占有である。ここでの優先順位は、2 人の医師ではなく、医師に加えて看護師を必要とするので、`Doctor` オブジェクトの前に `Nurse` オブジェクトが記載された `NurseDoctor` というリストから占有する（図 9.5 で示され、`ExamRooms` オブジェクトで用いられた `DoctorNurse` リストの正反対の順序）。

図 9.8 は、`TreatmentRoom` 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）もしくは診察室（Moderate 患者および Severe 患者）から治療室に来るため、（以前したように）患者の待ち時間を追跡するためにエンティティ生成時刻を用いることができない。代わりに、前に蓄積した時間（`ModelEntity.tmp`）を用いる。そして、患者の総待ち時間に治療室での待ち時間を加えるために、Seize 直後に第 2 の Assign を用いる（リソースが利用可能になるまでエンティティ（技術的に正確には Seize ステップで待つトークンであるが、この場合トークンはエンティティを表す）が Seize ステップに残っていることを思い出してほしい。ここでの待ち行列時間を加える）。

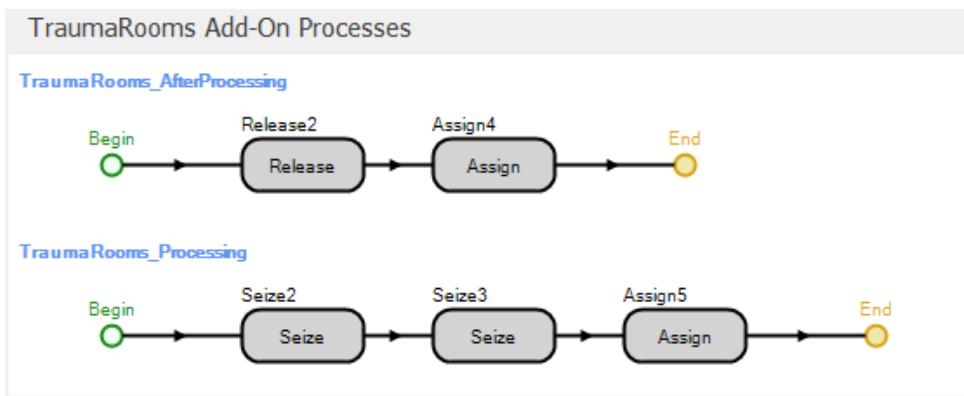


図 9.7 TraumaRooms オブジェクトのアドオンプロセス

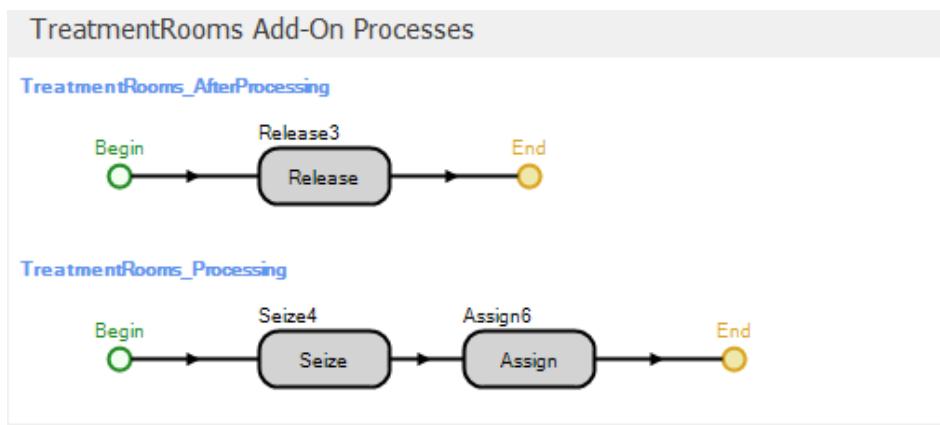


図 9.8 TreatmentRoom オブジェクトのアドオンプロセス

図 9.9 は、TreatmentRooms_Processing アドオンプロセスでの第 2 の Assign ステップのプロパティを示している。モデルの任意の時点のシミュレーション時刻でエンティティをマークして、次に、タリー統計量でマーク時刻と後の時刻との時間間隔を記録するという基本的概念は、パフォーマンス測定基準として時間間隔を記録したいと考える状況とまったく同じである。Simio やその他のシミュレーションパッケージでは、いくつかの間隔（たとえば、エンティティが生成されるときと、それが破棄されるときの間隔）は容易に追跡することができるが、パッケージには、個別の問題においてどれが重要な間隔であるかを知る方法がまったくない。そのため、これらのタイプのユーザ定義統計量を追跡するために、明確にモデルに設定する方法を理解しておきたい。

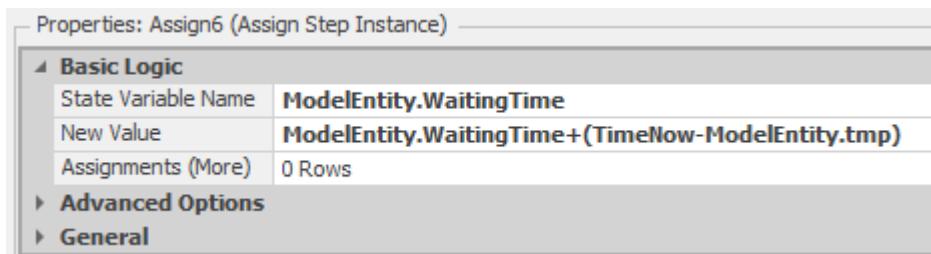


図 9.9 2 番目のエンティティ待機を追加する Assign ステッププロパティ

図 9.10 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（プロパティに関しては図 9.11 を参照）は、（WaitingTime エンティティ状態変数で蓄積された）患者の待ち時間を記録する。

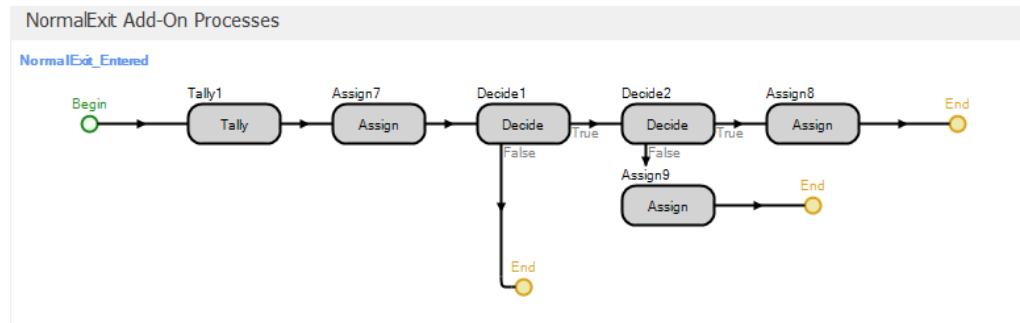


図 9.10 NormalExit オブジェクトのアドオンプロセス

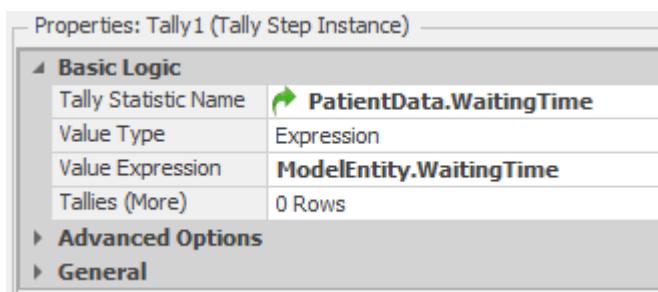


図 9.11 Tally ステップのプロパティ

TallyStatisticName 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す（データテーブルは図 9.2 に示されていた。図 9.12 はモデルのタリーと出力統計を示している）。このようにして、待ち時間は患者タイプごとに記録される。これは Simio データテーブル機能の利用例の 1 つである。Simio は、テーブルでタリー統計量を検索する。そしてさらに重要なことは、将来もし患者タイプを加えても、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）エンティティの待ち時間にモデル状態変数 TotalWait を加えるだけで、すべてのエンティティの総待ち時間の追跡ができる。

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 9.12 Model 9-1 の Model Elements

このプロセスの最後のステップは、患者サービスレベルを追跡するために用いられる。ED の例では、サービスレベルを総待ち時間が 30 分以下である Routine 患者の割合と定義する（任意にデフォルト値として 30 分を選んだ）。最初にすべきことは、Routine 患者を特定することであり、最初の Decide ステップで特定する（ステップのプロパティに関しては図 9.13 を参照）。

次に、その患者の待ち時間に基づいて、現在の患者が「満足」しているかどうか決定する必要がある。第 2 の Decide ステップで判定する（ステッププロパティに関しては図 9.14 を参照）。

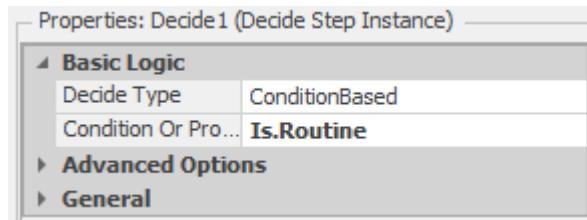


図 9.13 Routine 患者オブジェクトを特定する Decide ステップのプロパティ

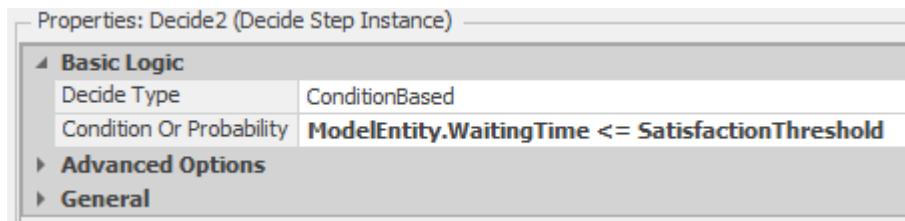


図 9.14 患者が待ち時間に満足しているか否かを判定する Decide ステッププロパティ

異なるしきい値での実験を簡素化するために、固定値（30 分間を表す 0.5）ではなく、プロパティ参照（SatisfactionThreshold）を用いたことに留意する。患者が満足しているならば、満足した患者（SatisfiedRoutine）を数えるために、モデル状態変数を増加させる。不満足ならば、満足していない患者（UnsatisfiedRoutine）を数えるために、モデル状態変数を増加させる。2 つの Assign ステップで、これらを設定する。実行における総合的なサービスレベルを計算するために、出力統計量を用いる。図 9.12 はモデル出力統計量を、そして図 9.15 は満足している患者の割合を計算する Satisfaction 出力統計のプロパティを示している。

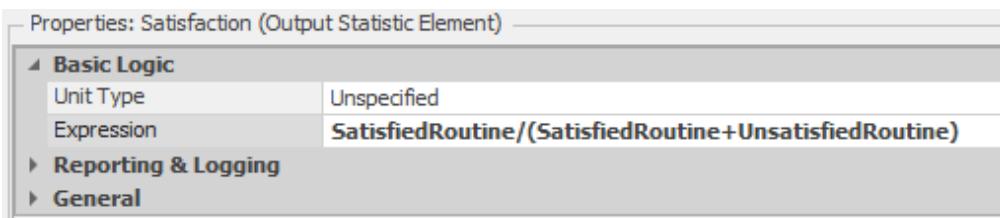


図 9.15 Satisfaction 出力統計量のプロパティ

最終的に、リソース配置分析をサポートするために、いくつかのプロパティ参照を用いる。救急診療において必要となる、診察室、治療室および検査室の数と、医師および看護師の数を決定することに関心があったので、それぞれの項目へのプロパティ参照を定義した。図 9.16 はモデルプロパティを示している。そして、図 9.3 で DoctorCapacity プロパティが Doctor リソースの初期の容量を指定するためにどのように用いられるかを示したように、Nurse オブジェクトの容量は、ExamRooms、TreatmentRooms および TraumaRooms サーバオブジェクトの容量を指定した。

図 9.17 は 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** に表示されることに注目してほしい。このように、実験では個々のシナリオでこれらの値を変えることができるので、モデルを変更する必要はなく、異なった実験シナリオを用いることで、容易に代替案の配置構成を比較できる。

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 9.16 Model 9-1 の Model プロパティ

Design	Response Results	Pivot Grid	Reports	Dashboard Reports	Input Analysis					
Scenario	Name	Status	Replications	Controls						
	<input checked="" type="checkbox"/> Scenario1	Idle	10 0 of 10	5 6 6 3 1 0.5	DoctorCapacity	ExamCapacity	TreatmentCapacity	TraumaCapacity	NurseCapacity	SatisfactionThreshold
*										

図 9.17 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、配置された様々なリソースの各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、医師および看護師の数に関心を持っている。表 9.2 は、ここで利用する週当たりのリソース費用を示している。また、リソースとスタッフ配置費用を補正するために、患者の待ち時間の 1 時間につき\$250 の「待ち時間の費用」を割り当てる。これで、リソース費用を最小にして患者の満足度を最大にするという、明らかに相反する目的のモデル化ができるだろう。

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、すでに総待ち時間について一覧にしているので、反復実行の長さを 1 週間に設定し、出力統計量として設定された「総費用」について計算できる。TotalCost 出力統計量に用いる式は、以下の通りである：

$$\begin{aligned}
 & 2000 * \text{ExamCapacity} \\
 & + 2500 * \text{TreatmentCapacity} \\
 & + 4000 * \text{TraumaCapacity} \\
 & + 4000 * \text{NurseCapacity} \\
 & + 12000 * \text{DoctorCapacity} \\
 & + 250 * \text{TotalWaitingTime.Value}
 \end{aligned}$$

これで、システム構成を評価するために 2 つの出力統計を持つことになった。総費用とサービスレベルである。総費用はできる限り低くしたいし、サービスレベルは利害関係者が決めた最小値を満たしたい。図 9.18 は、3 つのシナリオと実験の Responses として定義された 2 つの出力統計（総費用(Total Cost)の TC とサービスレベルの Satisfaction）がある実験例(InitialExperiment)を示している。ここでは、モデルの分析対象として関心があるすべての費用を恣意的に設定したことに留意してほしい。現実には、これらは決定／特定することが難しい値である。実際に教育環境

において学生が直面するもっとも重要な事柄の一つは、正確なモデルデータの発見／収集に関する困難さであるだろう。

Scenario	Replications	Controls							Responses		
		Required	Completed	DoctorCapacity	ExamCapacity	TreatmentCapacity	TraumaCapacity	NurseCapacity	SatisfactionThreshold	TC	Satisfaction
Scenario1	Complete	10	10 of 10	3	6	2	8		0.5	135.274	0.51335
Scenario2	Complete	10	10 of 10	5	6	2	10		0.5	24.1954	0.970648
Scenario3	Complete	10	10 of 10	4	7	3	9		0.5	39.8756	0.855517
Scenario4	Complete	10	10 of 10	2	5	1	1	6	0.5	471.506	0.951079
*											

図 9.18 Model 9-1 の実験例 (InitialExperiment)

9.1.1 OptQuest を用いた Model 9-1 の最適なリソースレベルの探索

病院の経営者側は、総費用 (TC) を最小にする 5 つの能力レベル (Doctor, Exam, Treatment, Trauma、および Nurse) と同時に、提供している「適切」なサービスレベル (Routine 患者の総待ち時間が 30 分以下である割合) において、まさしく適正値を選択したいと考えている。経営者側は、「適切」なサービスレベルについて、割合が少なくとも 0.8 あることに意味があると感じている。換言すると、30 分を超える総待ち時間は Routine 患者の 20% 未満に抑えなければならない（もちろん、これは全体的に主観的であり、しきい値の待ち時間や満足度の割合として用いられる「正しい」値は、利害関係者によって決定されるべきだろう）。それぞれの 5 つの能力レベルは整数で、2 以上（検査室 (Trauma) は 1 であるため除く）でなければならぬ。また、経営者側は、医師 (Doctor) の人数は最大 10、看護師 (Nurse) の人数は最大 15、診察室 (Exam) および治療室 (Treatment) の能力をそれぞれが最大 10、そして検査室 (Trauma) の能力を最大 5 に制限したいと考えている。

もう少し、これらが意味することについて考えよう。ここでは、Simio モデルからの出力レスポンスである総費用を最小にしようとしている。一見、その単独の観点から最良であることは、それぞれ 5 つの能力レベルを許容されている最小値（この場合、検査室の 1 を除いてそれぞれ 2）に設定することであるように思えるかもしれない。しかしながら、総費用は、患者の待ち時間に対して 1 時間当たり \$250 のペナルティを含んでいることを思い出してほしい。したがって、最小の能力レベルに設定すると、待ち行列が長くなり、総費用が上昇するだろう（総費用の観点からは、おそらく、名案でないだろう）。また、少なくとも 0.8 であるというサービスレベルの満足度の必要条件を満たす必要がある。それは、最低限の値を上回る容量レベルの人員が含まれていることおそらく意味している。

目標関数がシミュレーションから得られる出力レスポンスであり、（正確に測定できないサンプリング誤差を含んでおり）確率的であるため、たとえ標準的な数理計画的手法（線形計画法や非線形計画法、整数計画法など）では実際に解くことが難しいとしても、最適化問題としてこれらの条件を定式化することは状況を理解するのに役に立つだろう。数理計画的に定式化すると次のように表現できる：

$$\begin{aligned}
 & \text{min} && \text{TotalCost} \\
 & \text{Subject to:} && \\
 & && 2 \leq \text{DoctorCapacity} \leq 10 \\
 & && 2 \leq \text{ExamCapacity} \leq 10 \\
 & && 2 \leq \text{TreatmentCapacity} \leq 10 \\
 & && 1 \leq \text{TraumaCapacity} \leq 5 \\
 & && 2 \leq \text{NurseCapacity} \leq 15 \\
 & && \text{上記 5 つの能力は整数} \\
 & && \text{Satisfaction} \geq 0.8
 \end{aligned}$$

ここで、DoctorCapacity、ExamCapacity、TreatmentCapacity、TraumaCapacity および

`NurseCapacity`について、最小化が図られる。最小にしたい目標関数は、`TotalCost`である。そして、「`Subject to`」以下の行は制約条件である（最後のものは、非目標関数の出力）である。最初の 5 つの制約条件および 6 番目の整数制約は、シミュレーションにおける入力制御に対するものであることに注意してほしい。したがって、それらのすべての範囲と整数に従っているか否かは実験を設定するときにわかっている。しかし、`Satisfaction`は Simio モデルの（総費用目標とは異なる）別の出力レスポンスであるので、少なくとも 0.8 であるという必要条件に従っているかは、モデルを実行するまでわからない（入力ではなく出力であるので、これは制約ではなく必要条件と呼ばれる）。

さて、上述の定式化された最適化問題を実際に解くには、どのようにすべきだろうか？最初に、容量レベルに関するオプションをさらに（可能であればすべて）探索するために、より多くの Scenario 行を設定して、図 9.18 のサンプル Experiment を広げる。次に、TC（総費用）レスポンスを昇順にソートし、最初の「適合」シナリオ、すなわちサービスレベル Satisfaction 必要条件を満たす最初のシナリオ (`Satisfaction` レスponses が少なくとも 0.8) が表示されるまで下にスクロールすればよいだろう。スタッフ配置 Capacity レベルにおける範囲と整数制約を見ると、シナリオ数は $9 \times 9 \times 9 \times 5 \times 14 = 51,030$ であり、Experiment に入力し、実行するにはかなり多い Scenario 行になる。そして、（じゅうぶん正確な）`Total Cost` と `Satisfaction` のよい推定値を得るために、各シナリオを何回か反復実行する必要がある。合理的に正確な `Total Cost` の推定値（たとえば、95%の信頼区間の半幅が平均の 10%未満）を得られる初期の実験計画によれば、スタッフ配置能力の組み合わせを検討するために、各シナリオにつき 10 日間、約 120 回の反復実行が必要である。クアッドコア 3.0GHz のノート PC では、それぞれの反復実行時間は約 1.7 秒である。したがって、（Experiment Design に 51,030 の Scenario 行を設定できたとしても）この実験を実施するには $51,030 \times 1.7 \times 120$ 秒かかることになり、それは約 120 日、または約 4 ヶ月である。ただし、これは説明用の例題にすぎない。実際には、この種の徹底的な完全列挙の方針が、実行時間に関してまったく役に立たず、たいていは非常に深刻な問題となる。

このような方針では、（高い）`Total Cost` で大いに劣るか、または必要条件の 0.8 を下回る低い `Satisfaction` の割合のために容認できない（すなわち、実行不可能な）シナリオをシミュレートすることに 4 ヶ月の時間を費やすことになるため、よりスマートな方法を考える必要がある。入力の容量レベルの観点から定式化された最適化問題を考えるなら、5 次元の箱の整数格子点を隅々まで探索することになる。そして、おそらくいずれかの格子点から探索を開始し、近接する複数の点をシミュレートして、`TotalCost` を低下させ、かつ `Satisfaction` の必要条件を満たす方向へと探索点を移動する。そのような最適を検索しようとするヒューリスティック探索法で行われた多くの研究があるが、ほとんどの研究は最適なシミュレーションモデル構成を見つけることより、複雑な確定関数の最適化に向けられている。そして、これらの研究に裏付けられたソフトウェアパッケージが開発されている。それらの 1 つである OptQuest (OptTek Systems, Inc. (www.opttek.com)) は、Simio やその他のシミュレーションモデリングパッケージで利用するのに適している。たとえば、ここで検討している病院の能力問題のような問題の最適に近い実現可能解を探索できる。このすべてに関する包括的な文献がある。オープンアクセスの Winter Simulation Conference チュートリアル (Glover et al. 1999) で開始するのがよい。詳細はハンドブックを参照してほしい (Glover and Kochenberger 2003)。

この研究の結果と推奨事項を活用し、OptQuest は内部的に 5 次元の箱内の特定のポイント（ユーザが出発点、すなわち問題の 5 つの初期容量を提供しなければならない）からどの方向に移動するか（この例では総費用を削減）を決定し、入力パラメータ（5 つの容量の範囲）の制約に従いながら、他の出力の制約/要件 ($Satisfaction \geq 0.8$) に従う。この例では含まれていないが、OptQuest では、5 つの容量を線形の式で制約を指定することもできる。たとえば、

$$5 \leq \text{DoctorCapacity} + \text{NurseCapacity} \leq 20$$

この式では、それぞれに 1 つの不等式を持ち、2 つの制約を指定する必要がある。OptQuest は、5 次元の箱の特定のポイントから最適のポイント（これらの手法が最適解を見つけるという絶対的な保証はできないので、少なくとも最良のポイント）に向かって、より効率的なルートによって探索するために、いくつか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）を組み合わせて利用している。Glover et al. (1999) によると、

…最適化とシミュレーションコミュニティの長年にわたる目標は、高品質のソリューションを生み出す一連のシミュレーションを導く方法を作り出すことであった…最適化手順では、モデルに入れられた入力の結果を評価するシミュレーションモデルの出力を使用する。この評価に基づいて、そして現在のシミュレーション出力と統合され分析された過去の評価に基づいて、最適化手順は新しい入力値セットを決定する…最適化手順は、連続的に生成された入力が様々な評価を生成し、すべてが改善されるわけではないが、時間の経過とともに最良の解決策に効率的な軌道を提供する、特別な「非単調探索」を実行するよう設計されている。このプロセスは、適切な終了基準が満たされる（通常、ユーザの好みによって与えられる探索に費やす時間の長さ）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について、OptQuest がそれらをどのように使用するかについては、Glover et al. (1999) および Glover and Kochenberger (2003) を参照してほしい。

OptQuest を Simio で利用するには、まず Simio の Experiment に定式化された最適化問題を設定する。最小または最大にしたい項目（この例題では、総費用を最小にしたい）、入力制約（この例題では、容量の範囲と整数制約）、その他の必要条件や非目標関数出力（この例題では、Satisfaction の下限が 0.8）を指定する。それから、Experiment の実行をやめて、OptQuest をモデル化する。そこで、実行するシナリオやその順番などを決定することによって、4 か月間ノンストップで計算し続けるよりも非常に少ない時間で、（たとえ最適と証明できないものであっても）最良の解を求めることができる。OptQuest が用いている時間節約戦略の 1 つは、特定のシナリオがすでにシミュレートされたものより劣っていることが判明した段階で、反復実行を止めることである（劣っているシナリオに対して高い精度の推定を得る必要はない）。

ここで、どのように Simio の実験で OptQuest を設定するのかについて説明する。最初に、OptQuest 利用の有無に関わらず両方で実験できるように、新しい Simio Experiment を作成する（Project Home リボンから New Experiment ボタンを選ぶ）。この Experiment は、OptQuest1 と名づけた。次に、Simio アドインで OptQuest を加える。実験アドインは、設計や実行に関する追加的な能力を実験に加える Simio の拡張である（アドインは、ユーザ自身や他の誰かによって記述されている。アドイン作成に関する詳細については、Simio の説明書を参照してほしい）。これらのアドインは、（もしかすると外部のデータから）新しいシナリオを設定したり、シナリオを実行したり、またはシナリオ結果を解釈するために用いることができる。これらの 3 つの技術を組み合わせると、Simio アドインの OptQuest のような強力なツールを提供することができる。アドインを加えるために、Experiment の Design リボン上で Select Add-In ボタンを選択する。そして、リストから OptQuest for Simio を選択する（図 9.19 を参照）。

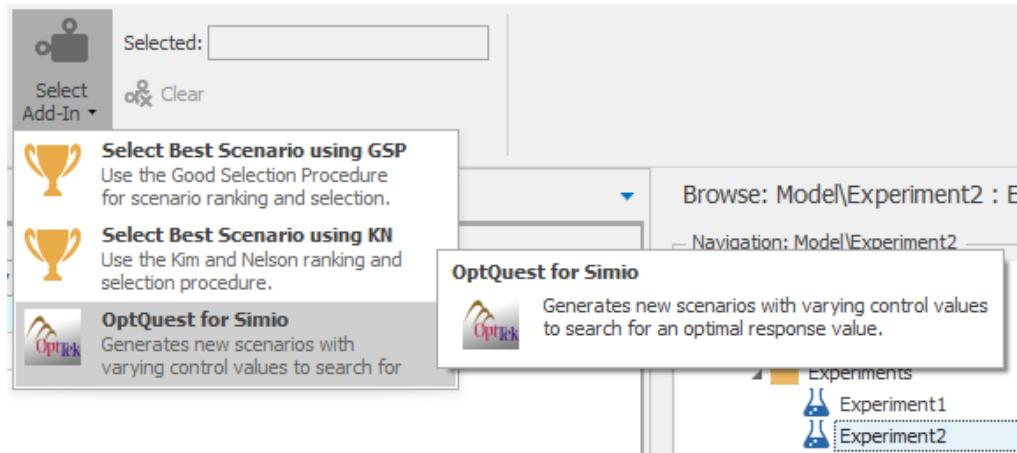


図 9.19 OptQuest アドインの選択

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、元に戻すことができない。アドインを設定すると、現在のアンドウ履歴は消去される。継続してもよいか?) という警告が表示される。アンドウ履歴を保持する必要があるなら「いいえ」と答え、モデルのコピーを作って、OptQuest を設定するためにそのファイルを利⽤する。そうでなければ、「はい」と答えると、Simio はアドインを読み込む(アンドウ履歴は消去される)。OptQuest アドインが追加されると、実験には OptQuest がどう働くかを制御するためのいくつかのプロパティが増える(図 9.20 を参照)。

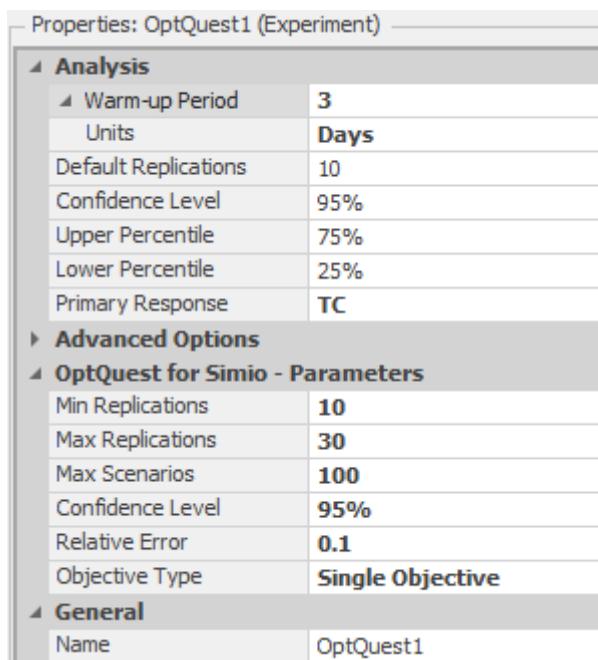


図 9.20 OptQuest アドイン追加後の「OptQuest1」の Experiment プロパティ

OptQuest 関連のプロパティは以下の通りである：

- Min Replications : OptQuest によって定義された各シナリオで Simio が実行する反復実行の最小数。
- Max Replications : OptQuest によって定義された各シナリオで Simio が実行する反復実行的最大数。
- Max Scenarios : Simio が実行するシナリオの最大数。基本的に OptQuest は、目標関数の

よりよい値（この場合は、Total Cost が小さい値）を見つけ、また Satisfaction が少なくとも 0.8 という必要条件を満たす決定変数（この場合は、Doctors、Exam Rooms、Treatment Rooms、Trauma Rooms および Nurse の数）の異なる値をシステムチックに探索する。そして、このプロパティは、OptQuest がテストするシナリオ数を制限する。OptQuest がいつもここで設定する数のシナリオを必要とするわけではないが、この制限により強制的な中止ポイントが提供されることに留意する。中止ポイントは、非常に多数の解空間を持つ問題において役に立つだろう。しかし、このプロパティを小さくし過ぎると、OptQuest が見つけられる解の質が制限される可能性があると理解すべきである。

- Confidence：2つのシナリオのレスポンス値に関して統計的な比較をするとき、OptQuest で利用したい信頼水準。
- Error Percent：信頼区間の半幅が下回っている必要のある標本平均の割合。たとえば、標本平均が 50 である場合に、Error Percent が 0.1 に設定されていると（つまり、信頼区間の半幅を大きくても平均の 10% 程度に抑える、大まかにいえば約 10% の誤りにしたい）、信頼区間は少なくとも 50 ± 5 ($5 = 0.1 \times 50$) 程度の正確さになる。デフォルトの反復回数を実行後に、OptQuest は、この条件を満たしているかどうか判断する。満たしていない場合、（Max Replications プロパティ値で指定された反復回数に達するまで）Simio はこの目標を達成するため、変動性を減少させるために追加の反復実行を行う。

また、OptQuest アドインを加えると、実験コントロールにプロパティが加わる。OptQuest アドインで、実験コントロールか Constraints のどちらかを用いることで最適化の制約を指定できる。ここでは、制約のそれぞれで、**单一のコントロール**（モデル内の Reference Property）の値を制限する必要があるため、現行モデルに実験コントロールを用いる。医師や看護師の総数を制限するような、前述の線形の式を組み合わせた制約を用いる場合は、実験 Constraints を用いる。

図 9.21 は、OptQuest アドインの追加後の Doctor Capacity 決定変数のプロパティを示している（アドインを追加する前には、実験コントロールにおいて編集可能なユーザプロパティがまったくなかったことに注意する）。

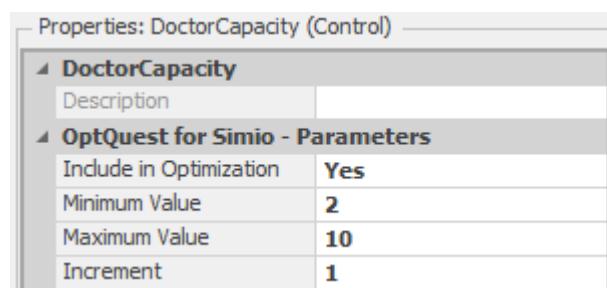


図 9.21 OptQuest 追加後の Doctor Capacity コントロールのプロパティ

「新しい」コントロールプロパティは、探索可能範囲（Include in Optimization）、決定変数の「取り得る」値（Minimum Value と Maximum Value プロパティ値を含んだ間の値）、そしてある点から次の点に範囲を検索するとき用いる増分（Increment プロパティ値）を OptQuest に設定する。図 9.21 に示されているコントロールプロパティは、最適化モデルから Doctor Capacity の制約を指定している（すなわち、2 ないし 10 の整数）。Exam Capacity、Treatment Capacity、Trauma Capacity および Nurse Capacity 決定変数の対応するプロパティも同様に設定した。ここで最適化において、Satisfaction Threshold 決定変数を用いていないので（しきい値を指定するのにプロパティ参照を用いたことを想起してほしい）、その Include in Optimization プロパティには No を設定する。最適化で用いられない決定変数に対して、Simio はすべてのシナリオのプロパティでデフォルト値（この場合、0.5）を用いる。

これまでの最適化において唯一残っている「制約」は、Satisfaction レベルに関する必要条件である (Satisfaction は入力ではなく出力であるため、これを制約条件というより必要条件としたことを想起してほしい)。図 9.22 は Satisfaction レスポンスのプロパティを示している (Lower Bound プロパティを 0.8 に設定した以外は、このレスポンスは前の実験の Satisfaction レスポンスと同じである)。

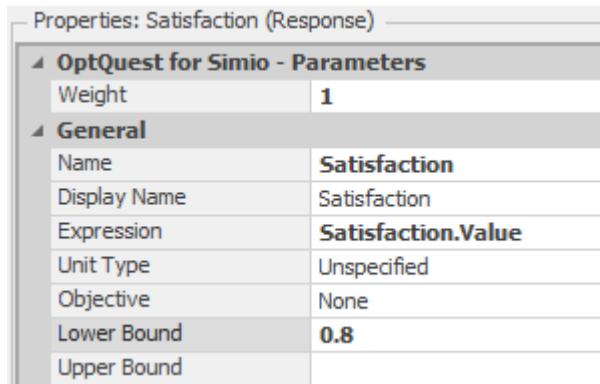


図 9.22 Satisfaction レスポンスのプロパティ

OptQuest を設定するための最終段階は、目標関数を定義することである。すなわち、最適化のための評価基準を OptQuest に設定することである。総費用を最小にしたいので、(以前したように) TC レスポンスを定義して、Objective プロパティを Minimize として指定する (図 9.23 を参照)。

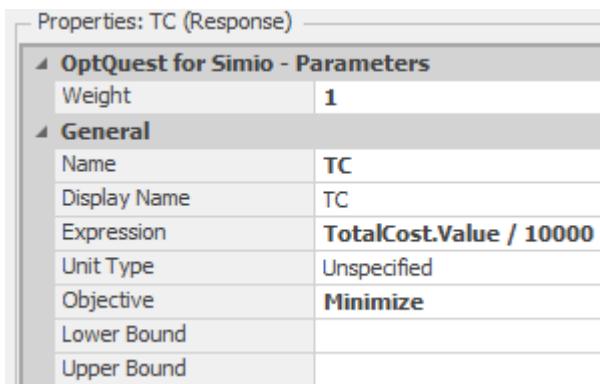


図 9.23 TC レスポンスのプロパティ

目視により実験テーブルで結果を比較しやすいように、TC レスポンスの定義において TotalCost モデル状態変数が 10,000 で割られていることに注意してほしい (これは、シナリオ間の比較関係は変更せず、測定単位だけを変えるものである)。最後に、TC レスポンスが最適化の目的であること (この場合 TC を最小にすること) を OptQuest に設定するために、TC に実験プロパティ Primary Response を設定する。

ここで Run ボタンをクリックすると、OptQuest の実行の様子をじっくり見ることができる。図 9.24 は、OptQuest の実行から、上位のいくつかのシナリオを示している (Total Cost の昇順に並び替えたため、一番上のシナリオは実行したシナリオの中で最小総費用のものである)。注：実行している Simio のバージョンによって、リストに異なったシナリオが表示されるかもしれないが、同一のシナリオ構成においては類似のレスポンス値が得られるだろう。これらの違いはプロセッサ/コア数とプロセッサ速度の違いによっても引き起こされる場合がある。

Scenario			Replications		Controls						Responses	
	Name	Status	Required	Completed	Doc...	Exam...	Trea...	Trau...	NurseCa...	SatisfactionThr...	TC	Satisfaction
051	Completed	✓	10	10 of 10	6	8	7	2	10	0.5	23.5...	0.995066
074	Completed	✓	10	10 of 10	6	9	6	2	10	0.5	23.6...	0.99375
049	Completed	✓	10	10 of 10	5	10	6	2	10	0.5	23.7...	0.99186
069	Completed	✓	10	10 of 10	6	7	7	2	10	0.5	23.7...	0.987747
034	Completed	✓	10	10 of 10	6	10	6	2	10	0.5	23.9...	0.99579
101	Completed	✓	10	10 of 10	6	8	7	3	10	0.5	24.0...	0.992552
099	Completed	✓	10	10 of 10	6	8	7	2	9	0.5	24.0...	0.978905
068	Completed	✓	10	10 of 10	6	9	7	2	10	0.5	24.0...	0.988275
070	Completed	✓	10	10 of 10	7	7	7	2	10	0.5	24.139	0.997348
056	Completed	✓	10	10 of 10	7	9	6	2	10	0.5	24.2...	0.999736
098	Completed	✓	10	10 of 10	6	8	7	4	10	0.5	24.4...	0.991313
048	Completed	✓	10	10 of 10	7	10	6	2	10	0.5	24.5...	0.999297
055	Completed	✓	10	10 of 10	7	9	7	2	10	0.5	24.5...	0.996495
053	Completed	✓	10	10 of 10	7	10	7	2	10	0.5	24.8...	0.996508
086	Completed	✓	10	10 of 10	6	8	7	4	9	0.5	24.9...	0.979897
033	Completed	✓	10	10 of 10	8	8	7	2	10	0.5	25.3...	0.999384
060	Completed	✗	30	30 of 30	2	9	7	5	4	0.5	25.4...	0.767915
050	Completed	✓	10	10 of 10	8	10	6	2	10	0.5	25.4...	0.999911
024	Completed	✓	10	10 of 10	8	8	8	2	10	0.5	25.6...	0.997771
084	Completed	✓	10	10 of 10	6	8	8	7	10	0.5	26.2...	0.985029
085	Completed	✓	10	10 of 10	6	5	10	2	10	0.5	26.2...	0.945934
072	Completed	✓	10	10 of 10	7	7	8	7	7	0.5	26.7...	0.967756

図 9.24 最初の OptQuest 実行の結果

リストの一番上のシナリオはシナリオ 051 であり、その構成は、Doctor は 6、Exam Rooms は 8、Treatment Room は 7、Trauma Room は 2、そして Nurse は 10 であり、Total Cost は 23.592 (万)、および Satisfaction は 0.995 であった。ここで留意すべき点は、すべての Satisfaction 値がかなり高いことである（実行不可能な解（80%の満足度の必要条件が満たされないもの—Satisfaction の値は赤い背景で、シナリオはチェックされず、実行不可能であることを示している）を見つけるには、リストをシナリオ 060 より下にスクロールさせなければならない）。当然ながら、次に湧き上がる疑問は、「これが本当に最適解であるか？」ということであろう。残念ながら、答えは「確信できない」という他ない（実際にこれがこの問題の最適解でないことはわかつており、以下で述べるが、一般に答えはわからない）。先に述べたように、OptQuest はヒューリスティックな探索手法を組み合わせて用いるので、発見した解が本当に最適であるかどうかを断定する方法は（一般的に）まったくない。このシステムにおける 51,030 のシナリオ構成を示し、OptQuest ではこれらの内 100 のシナリオをテストすることに制限したことを見出してください。この場合、OptQuest はかなりうまく解を発見したようだ。その後、追加実験（OptQuest2）として OptQuest が 1,000 シナリオをテストできるように設定したところ、Total Cost が 22.865（少しよい）、Satisfaction が 0.998 の構成を発見した（この場合の構成は、Doctor は 5、Exam Rooms は 8、Treatment Room は 5、Trauma Room は 2、そして Nurse は 10 である）。10 倍のシナリオを実行したが、新しい「最良」解が実際に真の最適解に対してどれくらいよいかは、残念ながらまだわからない。混合最適化問題におけるヒューリスティック解の本質はそのようなものである。よい解であったと確信しているが、唯一の最適解を確実に得られる方法はわからないため、これ以上の改善は難しい。

OptQuest で最適解を確実に発見できるとはいえないが、ユーザがただ恣意的に探索可能な範囲を隅々まで検索しようとするより、OptQuest のほうがうまくできそうである（37 番目くらいのシナリオに達して、コンピュータを見るのに飽きた頃にそう思うだろう）。OptQuest が本当に役に立つケースは、探索可能な領域の範囲を制限できるケースである。たとえば、問題では Trauma Rooms の数が 1 ないし 5 であるように制約したが、Urgent 患者（検査室を必要とする唯一の患者）の期待到着率はかなり低い（毎時約 0.6 人）ので、3 つ以上の検査室が必要となることはそうもない。したがって、5 から 2 に Trauma Room 決定変数の上限を変えると、探索可能な領域

のサイズが 51,030 から 20,412 まで減少し、少ない数の許容シナリオで OptQuest のパフォーマンスをおそらく向上させることができる。これは最適解を求めるプロセスにおけるユーザの関与の重要性を強調している。OptQuest は、実行中のモデルやモデル化された実システムの特性に関して何も理解していない。しかし、ユーザはそれを知っており、その知識を利用することでコンピュータの負担をかなり減少させ、決定する最終的な「解」を改善できるだろう。それでは、OptQuest が特定したシナリオを評価すること、できれば「もっともよい」シナリオを特定することに話題を戻そう。

9.1.2 サブセット選択と KN を用いた Model 9-1 における代替シナリオのランキングと選択

図 9.24 に示されているレスポンス値は、それぞれのシナリオの 10 回の反復実行から得られた標本平均であるので、サンプリング誤差の影響下にある。このように、OptQuest 実験から得られた標本平均に基づいた順序が、この問題の真の最適であるか(つまり、最小総費用という観点から、リソースの最良の構成であるか)は定かではない。図 9.25 は、OptQuest の実行による上位 5 つのシナリオを SMORE プロットに示している。他のシナリオと比べて、シナリオ 051 の総費用 (TC) の標本平均がもっとも低いことは明らかであるが、SMORE プロットの目視比較で信頼区間を考慮すると明確でなくなる。実際、図 9.25において、5 つの標本平均の間に統計的な有意な差異があるかを判断することは難しい (SMORE プロットの目視検査は、正確に真の統計的な有意な差異を判断できないが、さらなる分析をまだ模索することができる)。この不確実性は、サンプリングの不確実性に起因し、ここがまさに、Subset Selection Analysis の出番である。この分析方法は、小さな例を対象に 5.5 節ですでに解説している。

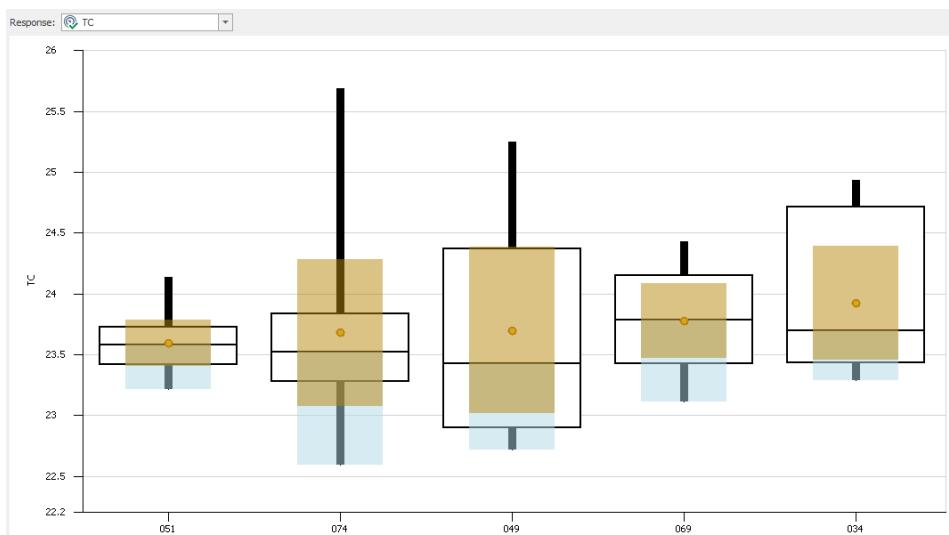


図 9.25 トップ 5 シナリオに対する Model 9-1 の OptQuest 実行の SMORE プロット

図 9.26 は、Trauma Room の上限を 2 に変更し、Subset Selection Analysis 機能を実行 (Design リボンの上の Subset Selection Analysis アイコンをクリック) した後の結果を示している。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：レスポンスの背景色が濃い茶色のシナリオであり、そのサンプル平均が最善のサンプル平均と統計的に差がないシナリオで構成される「おそらく最善」グループ；対応するレスポンスのセルが薄茶色の背景色で表現されるシナリオで構成された「棄却」グループである。シナリオごとに 10 回の反復実行による初期実験に基づいて、「おそらく最善」グループにおけるシナリオはさらに検討する価値があるが、「棄却」グループは統計的に「より悪く」、(特定のレスポンスについて) これ以上の検討に値しないと判断される。「おそらく最善」グループのシナリオは、それらが「良い」

と「実証」されていると解釈するべきではない。それは棄却されていないということだけである。この非対称性は、帰無仮説を棄却することが真実の如何なる「証拠」も暗示しないという古典仮説検定に類似している。それを拒否する証拠を持っていないが、一方で、帰無仮説を棄却することは、それが偽であるという確かな証拠を示唆している。もし実験におけるシナリオがいつも「棄却」とされないなら、その実験は Subset Selection Analysis が統計的に異なるグループを特定するためには、じゅうぶんな反復回数ではないことに注意してほしい。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限り) 追加の反復実行を行うことで問題を解消できる。

	Scenario		Replications		Controls							Responses		
	<input type="checkbox"/>	Name	Status	Required	Completed	Doct...	Exam...	Trea...	Trau...	NurseCa...	SatisfactionThr...	TC	Satisfaction	
②.	<input checked="" type="checkbox"/>	099	Comple...	10	10 of 10	6	8	6	2	9		0.5	23.181	0.991554
	<input checked="" type="checkbox"/>	037	Comple...	10	10 of 10	6	9	5	2	10		0.5	23.4374	0.998439
	<input checked="" type="checkbox"/>	097	Comple...	10	10 of 10	6	8	5	2	10		0.5	23.4804	0.997856
	<input checked="" type="checkbox"/>	062	Comple...	10	10 of 10	6	9	6	2	10		0.5	23.6803	0.99375
	<input checked="" type="checkbox"/>	042	Comple...	10	10 of 10	6	9	6	2	9		0.5	23.9046	0.981898
	<input checked="" type="checkbox"/>	043	Comple...	10	10 of 10	5	9	4	2	10		0.5	23.9209	0.999323
	<input checked="" type="checkbox"/>	075	Comple...	10	10 of 10	6	9	7	2	10		0.5	24.0881	0.988275
	<input checked="" type="checkbox"/>	086	Comple...	10	10 of 10	6	9	7	2	9		0.5	24.0888	0.980461
	<input checked="" type="checkbox"/>	091	Comple...	10	10 of 10	5	7	6	2	9		0.5	24.134	0.959001
	<input checked="" type="checkbox"/>	050	Comple...	10	10 of 10	6	9	4	2	10		0.5	24.136	1
	<input checked="" type="checkbox"/>	107	Comple...	10	10 of 10	7	8	7	1	7		0.5	24.2346	0.972244
	<input checked="" type="checkbox"/>	049	Comple...	10	10 of 10	7	8	7	2	9		0.5	24.4391	0.987966
	<input checked="" type="checkbox"/>	063	Comple...	10	10 of 10	6	9	10	2	10		0.5	24.7969	0.987193
	<input checked="" type="checkbox"/>	074	Comple...	10	10 of 10	7	8	9	1	7		0.5	24.9535	0.974061
	<input checked="" type="checkbox"/>	022	Comple...	10	10 of 10	8	8	8	2	7		0.5	25.0086	0.989396
	<input checked="" type="checkbox"/>	005	Comple...	10	10 of 10	8	8	8	2	8		0.5	25.0396	0.993925
	<input checked="" type="checkbox"/>	023	Comple...	10	10 of 10	8	8	7	2	7		0.5	25.108	0.981907
	<input checked="" type="checkbox"/>	045	Comple...	10	10 of 10	8	8	5	2	10		0.5	25.1131	1
	<input checked="" type="checkbox"/>	040	Comple...	10	10 of 10	8	8	4	2	6		0.5	25.1744	0.996537
	<input checked="" type="checkbox"/>	021	Comple...	10	10 of 10	8	8	8	2	9		0.5	25.367	0.996235

図 9.26 Subset Selection Analysis 実行後の最初の OptQuest 実行結果

この例では、濃い茶色の「おそらく最善」グループには 11 のシナリオが含まれる（図 9.26 参照）。表のかなり下にあるシナリオ 090 は濃茶色の影付きのレスポンスを示しているが、Satisfaction の列の背景色は赤色となっており実行不可能である。なぜなら、この出力は問題の定式化で少なくとも 0.8 である必要があったためである。明らかに、ここで「さて、どうしようか？」と思われるだろう。2 つの基本的な選択肢がある：11 の「よい」構成があるとするか、追加実験をするかである。唯一の最善の構成（または可能な限り少数の可能な最善のシナリオ）を見つけるとすると、現在の「おそらく最善」グループの 11 のシナリオで追加の反復実行することになるだろう。そして、より少ない「おそらく最善」グループ（または、運がよければ唯一の「最善」のシナリオ）を特定するならば、判断するために、Subset Selection Analysis を再実行することになる。実験から「棄却」グループのシナリオは選択せず、残りの 11 シナリオについて、それぞれ 40 回の反復実行を追加する（合計 50 回の反復実行を行う）。そして、結果として得られる新しい「おそらく最善」グループは 4 つのシナリオが含まれることになる（結果は示さない）。ここでは、追加の反復実行を行う前に、リボンの Add-ins にある Clear アイコンをクリックして、OptQuest アドインを「消去」しなければならないことに注意してほしい。ここで以前と同様に、4 つの「よい」構成が存在するとするか、追加実験をすることができる。「最善」の構成を追求したいと思うなら、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削除し、残っているシナリオで追加の反復実行を行う）。ここでは、その代わりに、このプロセスを支援する別の Simio アドインを用いることにしよう。

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 節でも述べたが、所与の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。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が終わるまで、1 度に 1 反復で、候補シナリオの追加反復を実行し続

ける。以前に OptQuest を用いたときと同様に（図 9.19 参照）、この KN アドインを実行するには、Design リボンで Select Add-In アイコンを用いて Select Best Scenario using KN を選択する。この操作により、アドインに関連するいくつかの実験プロパティが新たに表示される（図 9.27 を参照）。

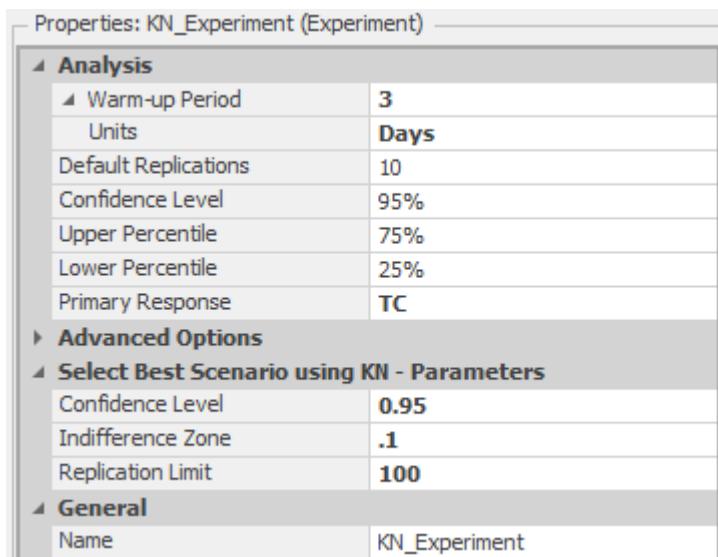


図 9.27 Select Best Scenario using KN アドインによる Experiment プロパティ

Confidence Level プロパティ（信頼区間に用いる水準を指定する）と、Indifference Zone と Replication Limit を指定する。Indifference Zone（無差別領域）は検出されるもっとも小さい「重要な」差である。たとえば、21.65 と 21.68 は数の上では異なっているが、特定の問題においては 0.03 の差は重要でないかもしれない（無差別領域の値に対する適切な選択は、レスポンスが測定している事柄や研究の文脈に、明らかに依存している）。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ことになる。もしもあなたが完璧主義者で、非常に小さな領域を宣言しようとする場合、非常に小さな準最適性を気にすると、きわめて長い計算時間を必要とする（計算量は、無差別領域が狭くなるにつれて超直線的に成長する）。Replication Limit（反復の限界値）は、最善のシナリオを見つけることを「断念する」（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最大値をアドインに設定する。

図 9.28 は、アドイン実行後の実験の様子を示している。最左端のチェックボックスにチェックされたものごとに、100 回の反復（ユーザではなく、KN の手続きがそれらの反復回数を決定する）で実行した後に、シナリオ 037、097 および 099 が「最善」の構成であると示している。複数のシナリオがチェックされている状態で終了する場合、アドインは、現在の反復実行の最大数（例では 100 回とした）および無差別領域に基づいて、チェックされたシナリオに差を見いだせないと意味する。これは、OptQuest の 2 番目の実行（図 9.26）後の平均総コストが最小であったのと同じシナリオであるが、追加の反復実行した後では、シナリオ 099 の総コストが高くなる（23.759（追加実行後）対 23.181（追加実行前））。SMORE プロット（図示なし）を見ると、サンプル平均の周囲にはかなり小さい信頼区間が示されており、OptQuest 実験での 10 回の反復による初期結果はサンプル平均の値が低い可能性が高いことを示している。これは、実験結果から意思決定を行う前に、信頼区間に細心の注意を払い、必要に応じてより多くの反復実行をする必要があることを示している。

Scenario			Replications		Controls						Responses		
	<input type="checkbox"/>	Name	Status	Required	Completed	Doct...	Exam...	Trea...	Trau...	NurseCa...	SatisfactionThr...	TC	▲ Satisfaction
097	<input checked="" type="checkbox"/>	Comple...	Completed	100	100 of 100	6	8	5	2	10	0.5	23.4291	0.998673
037	<input checked="" type="checkbox"/>	Comple...	Completed	100	100 of 100	6	9	5	2	10	0.5	23.5612	0.998875
099	<input checked="" type="checkbox"/>	Comple...	Completed	100	100 of 100	6	8	6	2	9	0.5	23.7533	0.985286
062	<input type="checkbox"/>	Comple...	Completed	90	90 of 90	6	9	6	2	10	0.5	23.7588	0.994729
042	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	6	9	6	2	9	0.5	23.9412	0.981851
075	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	6	9	7	2	10	0.5	24.2144	0.988033
043	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	5	9	4	2	10	0.5	24.3372	0.999251
050	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	6	9	4	2	10	0.5	24.359	0.99977
086	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	6	9	7	2	9	0.5	24.393	0.977575
049	<input type="checkbox"/>	Comple...	Completed	10	10 of 10	7	8	7	2	9	0.5	24.4391	0.987966
107	<input type="checkbox"/>	Comple...	Completed	50	50 of 50	7	8	7	1	7	0.5	24.6266	0.96893
063	<input type="checkbox"/>	Comple...	Completed	10	10 of 10	6	9	10	2	10	0.5	24.7969	0.987193

図 9.28 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正当な理由は、ここでの範囲を超えており、簡単に言えば、以下のように動作する。最善のものとなるように、競合しているすべてのシナリオについて、初期値で反復実行される（最初はすべてのシナリオが「まだ」競合している）。そして、シナリオの可能なすべてのペアにおいて、平均の差のサンプル分散が計算される。これらの平均値の差のサンプル分散に基づいて、ある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十分な確信が得られるまで、生存シナリオを一度に 1 つずつ繰り返し実行する。その手順は、最善の候補ではないと思われるシナリオを排除し、それ以上の反復実行はされないため、競合状態にあるシナリオセットのサイズが縮小される（それらの計算時間を省いて効率を向上させる）。最善の決定が下される前に、ユーザ指定による反復数（この例では 100）の制限に達すると、まだ競合している生存シナリオはすべて最善のものとみなされる。この KN を用いた最善シナリオの選択メソッドとサブセット選択との間のアプローチの根本的な相違に注意してほしい。サブセット選択では固定数の反復実行を行い、サブセット選択はシナリオをこの「サンプルサイズ」に基づいて、「おそらく最善」と「棄却」に分けるために最善を尽くす。一方、KN の最善シナリオは、単一の最善シナリオを特定するというユーザの希望を叶えるため、シナリオに必要な数の反復を必要とする。これまでの方法と比較して、KN には少なくとも 2 つの重要な利点がある。第 1 に、それは完全に逐次であり、すなわち、2 段階サンプリングではなく、一度に 1 回の追加の反復実行が行われる。2 段階サンプリングは、初期の反復回数のあと、単一の 2 段階で複数の反復がなされる。本当に必要な反復回数がおそらく「オーバーシュート」する（単一の反復を作成するには数分から数時間かかる場合がある大規模な複雑なモデルでは大変である）。第 2 にシナリオ間の共通乱数が同期されているかどうかによって、標本平均間の差異の根底となる真の分散を減らすことができる。KN の詳細については、Kim and Nelson (2001) を参照し、Simio での実装の詳細については、Books の Simio Support タブの Simio Reference Guide の「Select Best Scenario Using KN Add-In」を参照してほしい。

さて、この例題で行った手順をまとめると、ここでの最適化問題に基づいて、潜在的に有望なシナリオ（プロパティ参照に対して特定の値を持つ構成）を特定するために、OptQuest アドインを活用することから始めた。OptQuest を実行した後、Subset Selection Analysis 機能を用いて、「おそらく最善」と「棄却」グループにシナリオの集合を区分した。そして、「おそらく最善」の部分集合に含まれるシナリオに対して追加実験を行った。まず、この部分集合の大きさを削減するために反復回数を追加して実行し、最後に「最善」の構成を特定するために Select Best Scenario using KN アドインを用いた。

ヒューリスティックな最適化を利用しておらず、一般に実験空間のすべての構成を評価できるわけではないので、この手順に従うことでのっともよい構成（あるいはかなりよい構成でさえも）を確実に発見できるとは保証できない。しかし、上述の実験は、行き当たりばったりで行う実験よりも、この手順がかなりうまくいくことを示している。

9.2 Model 9-2：ピザのテイクアウトモデル

Model 9-2においても、リソース配置に関する問題を考える。ここでモデル化している環境は、電話によって注文を受け付け、顧客が受け取りに来るというテイクアウトのピザを提供しているピザレストランである。顧客からの電話は毎時 12 本の割合で着信し、着信時間間隔は平均 5 分の指数関数に従う。ピザ店には、現在 4 つの電話回線がある。そして、4人の顧客が通話中の場合、次に電話をしてきた顧客は、話し中の通話音を聞くことになる。顧客の注文の種類は、ピザの枚数が 1 枚 (50%)、2 枚 (30%)、3 枚 (15%)、4 枚 (5%) である。そして、従業員が顧客の電話に対応する時間は、1.5 ないし 2.5 分の一様分布に従う。従業員が注文を受けると、店のコンピュータシステムは、「顧客チケット」を印刷する。そして、注文のピザができあがるまでレジの近くに置かれる。この簡易型システムでは、それぞれのピザは注文してきた顧客のために作成され、調理され、箱に詰められる。従業員は作成と箱詰めの作業をする。そして、半自動化されたオープンがピザを調理する。

作成のステーションでは、従業員たちが同時に 3 枚のピザを作成するスペースがあるが、各従業員は一度に 1 枚のピザを作ることができる。ピザを作る時間は 2、3 および 4 分のパラメータを持つ三角分布に従う。従業員がピザを作り終えると、ピザは調理のためにオープンに置かれる。調理時間は 6、8 および 10 分のパラメータを持つ三角分布に従う。オープンには、8 個のピザを同時に調理するスペースがある。ピザを調理し終わると、ピザが自動的にオープンから取り出され、箱詰めを待つ場所に移動される（これはオープンが「半自動化している」部分である）。箱詰めステーションには、従業員が 3 人まで同時にピザを扱うスペースがある。作成のプロセスと同様に、各従業員は一度に 1 枚のピザを扱うことしかできない。箱詰めの時間はパラメータ 0.5、0.8 および 1 分の三角分布に従う（箱詰め前に、従業員がピザをカットする時間も含む）。箱詰めが完了し、注文のピザの準備ができると、その注文は完了したものとみなす。顧客の受け取りの過程もモデル化でき、またデリバリーの過程をモデル化することもできるが、ここでは、それらに興味のある読者に課題として残すことにする。

ピザ店は、現在、受注、ピザ作り、およびピザ箱詰めのプロセスに従業員をプールする方針を用いている。これは、従業員が全員複数の仕事ができ、タスクのいずれかを実行できることを意味する。作成と箱詰め工程のために、従業員はタスクを実行する各ステーションに物理的に位置する必要がある。店にはコードレス電話とノートパソコンがあるので、従業員は店のどこからでも注文を取ることができる（すなわち、物理的に注文を受けるステーションを設置する必要はない）。

図 9.29 は Model 9-2 の完成した Facility ビューを示している。Customer と Pizza エンティティタイプを定義して、Floor Label を用いてエンティティにラベルを付いた。Floor Label により、ラベル上にモデルとエンティティ状態変数を表示できる。エンティティに付いた Floor Label は、エンティティと共に移動する。

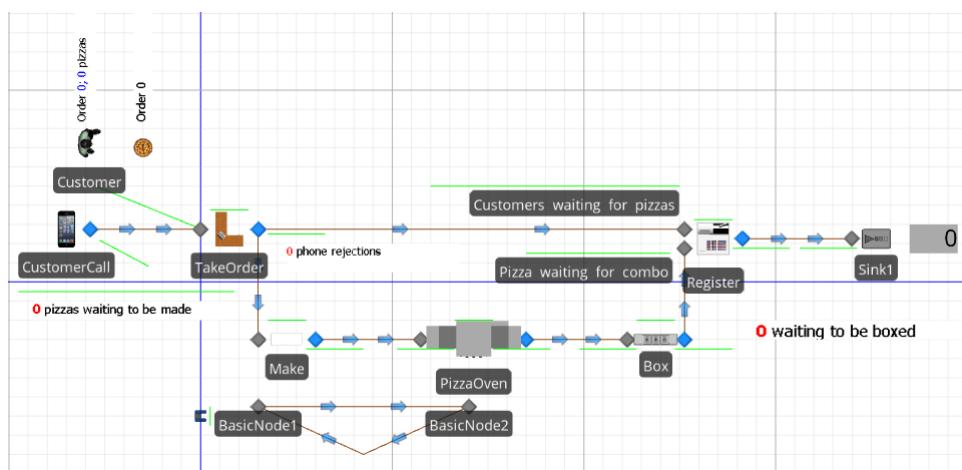


図 9.29 Model 9-2 の Facility ビュー

エンティティの Floor Label を作成するためには、Facility ビューでエンティティを選択し、Symbols リボン上の Floor Label アイコンをクリックする。そして、Facility ビューのラベルをつけたいところでクリックし、サイズを決めるためにドラッグし、最後にまたクリックしてラベルを配置する。これで、ラベルの書式に関する説明が書かれたデフォルトテキストが表示されたラベルが作成される。Appearance リボンで Edit アイコンをクリックすると、編集ダイアログボックスが現れる（Pizza エンティティの Floor Label の編集ダイアログボックスを示した図 9.30 を参照してほしい。カッコ内に OrderNumber エンティティ状態変数を利用したことに留意する。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）。また、モデルには、Source オブジェクト（CustomerCall）と、受注（TakeOrder）、作成（Make）、箱詰め（Box）プロセス、およびオープン（PizzaOven）を表す Server オブジェクトがある。モデルは、Customer オブジェクトと注文のピザを表す Pizza オブジェクトを結合する Combiner オブジェクト（Register）と、エンティティが離脱する唯一の Sink オブジェクトが含まれている。最後に、Customer と Pizza エンティティの流れを表す Connector オブジェクトを加えて、さらに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すために、2 個の Node オブジェクト（BasicNode1 および BasicNode2）間に Path のループを追加する。ここで、現在の Facility ビューがおおよそ図 9.29 と同様になるように、オブジェクトの配置を調整するとよいだろう。

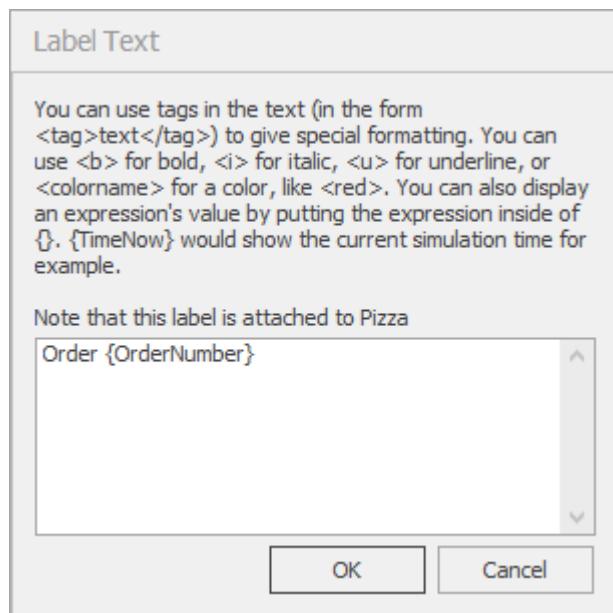


図 9.30 Pizza エンティティの Floor Label ダイアログボックスの編集

Source オブジェクト CustomerCall は上述した顧客着信過程に基づく顧客の着信を生成する（Interarrival Time プロパティは Random.Exponential(5) 分に設定される）。モデルが Customer および Pizza エンティティタイプを持つように、Source オブジェクトの Entity Type プロパティは Customer に設定する。それから、顧客の電話は必要な処理のために Server オブジェクト TakeOrder に送られる。システムに 4 人の通話中の顧客がすでに存在する場合、次の顧客は通話中の発信音を聞き、受話器を置くことを思い出してほしい。アドオンプロセスを用いて、この能力制限をモデル化する。顧客注文を処理する前に 2 抹の決定をする必要があるので、TakeOrder オブジェクトの Input ノードにアドオンプロセスを用いて、電話システムが容量の限度を超えている場合に通話を拒否する。エンティティがノードに入るときにそのプロセスが実行されるように、Entered トリガーを用いる。図 9.31 に Input_TakeOrder_Entered アドオンプロセスを示す。

Input@TakeOrder Add-On Processes

Input_TakeOrder_Entered

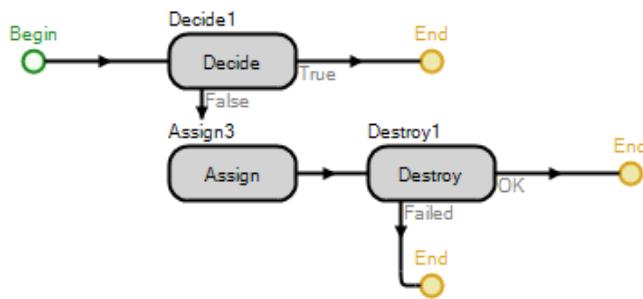


図 9.31 TakeOrder オブジェクトの入力ノードに関するアドオンプロセス

Decide ステップは利用可能な電話回線があるかどうかを決定するという条件に基づく判定を実施する。TakeOrder オブジェクトの容量と、対応する InputBuffer 容量を用いて、電話回線の数をモデル化する。図 9.32 は TakeOrder オブジェクトのプロパティを示している。

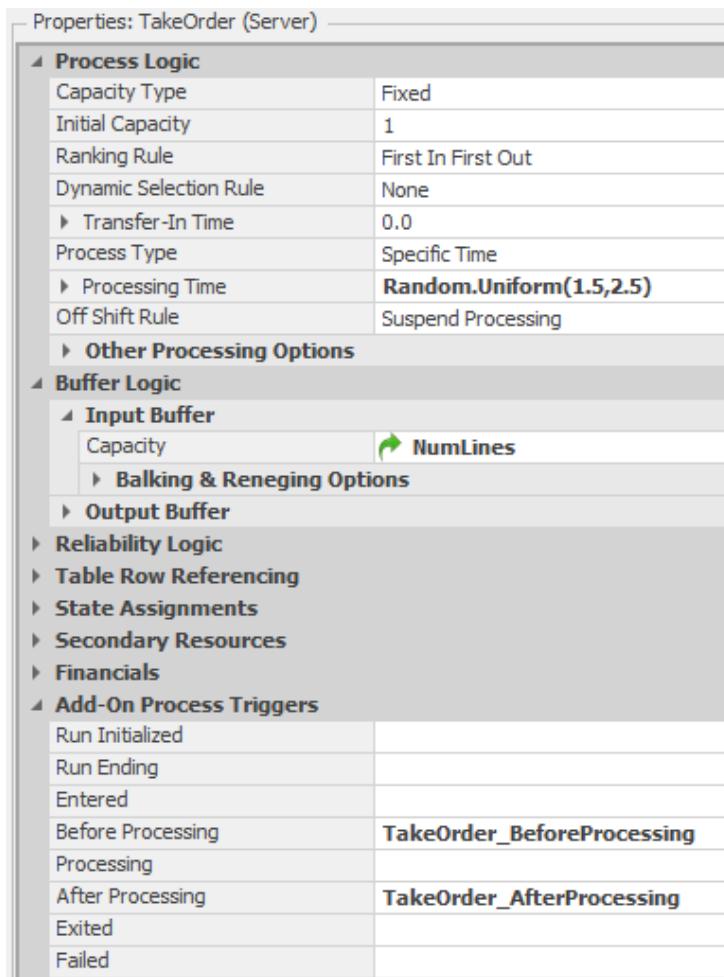


図 9.32 TakeOrder オブジェクトのプロパティ

Initial Capacity プロパティを 1 にし、Input Buffer プロパティを NumLines のプロパティ参照として設定したことに留意する。したがって、Decide ステップでは、通話を受け入れるかどうかを決定するために次の式を用いる：`TakeOrder.InputBuffer.Capacity.Remaining > 0`。この条件が真であるなら、通話は TakeOrder オブジェクトにつながる。そうでなければ、拒否された通話

数の動向を記録しておくために、ユーザ定義のモデル状態変数 PhoneRejections を増加させ、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基づいて、常に 1 人の従業員が 1 つの注文を受けることができ、そしてプロパティ参照 NumLines の値によって、「保留」で待っている顧客の数が決定される。入力バッファ容量を指定するためのプロパティ参照を活用することで、さまざまな電話回線数による実験を簡素化できることに留意してほしい。上述のシステム基本構成に対しては、通話中の顧客数を合計 4 に制限するために、プロパティ参照の値を 3 に設定することになる。

顧客がいったんシステムに入ると、従業員は電話を取り、注文の処理をする。プールされた従業員をモデル化するために、Worker オブジェクト (Worker1) を用いる。図 9.33 は、Worker1 オブジェクトのプロパティを示している。

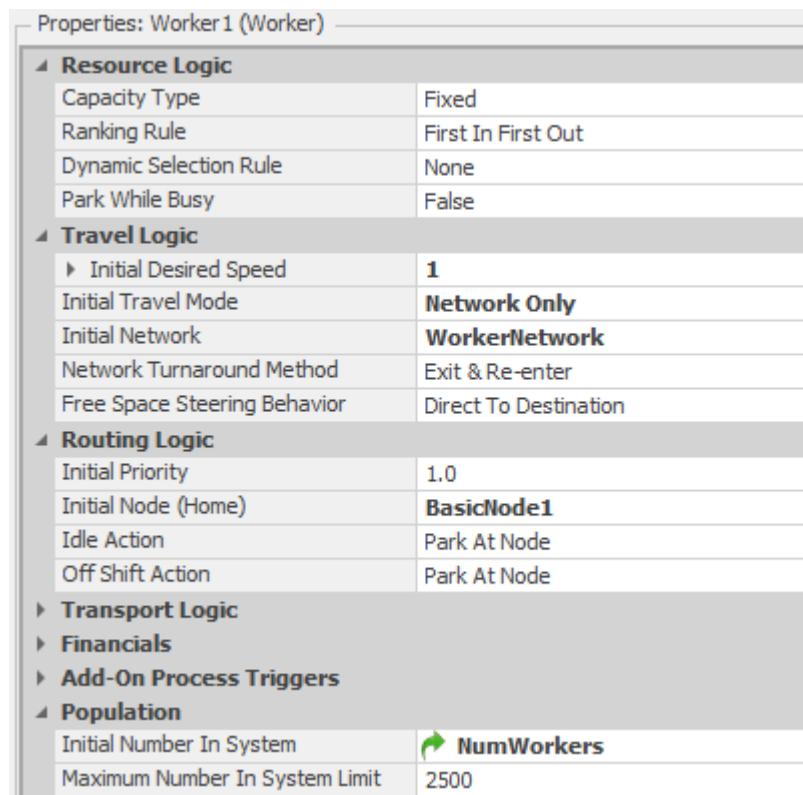


図 9.33 Worker1 オブジェクトのプロパティ

Worker オブジェクトはモデル内で対応する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、作成および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、そして Initial Node (Home) を BasicNode2 に設定する (Worker オブジェクトが 2 つのステーション間を動くのに用いるループを構成している Path オブジェクトに沿う、箱詰めステーションの下の BasicNode2 と作成ステーションの下の BasicNode1 を用いて、WorkerNetwork を定義した)。また実験中、容易に従業員の数を操作できるように、Initial Number In System プロパティに、プロパティ参照 (NumWorkers) を用いたことに留意してほしい。Model 9-1 で用いたように、TakeOrder オブジェクトの Processing アドオンプロセスにおいて、Worker1 リソースを占有するのに Seize ステップを用いる (図 9.32 を参照)。Worker1 リソースが利用可能でないなら、エンティティは Seize ステップで待つことになる。従業員が店のどこでも注文を受けることができる所以、実際にどこかへ動くよう Worker1 オブジェクトに要求する必要はまったくなく、たんに受注タスクを割り当てるためにリソースが必要とされる。これは、本当のピザ店において、コードレス電話で応答し、ノートパソコンを利用してどこからでも注文を入力

している従業員と同じだろう。ただし、Make と Box ステーション下では、その限りではない。

Worker1 リソースが割り当てられると、実際に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。ここで、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、完成した Pizza エンティティを待つために Customer エンティティを送る必要がある。TakeOrder_at_AfterProcessing アドオンプロセスでこの処理を行う（図 9.34 を参照）。

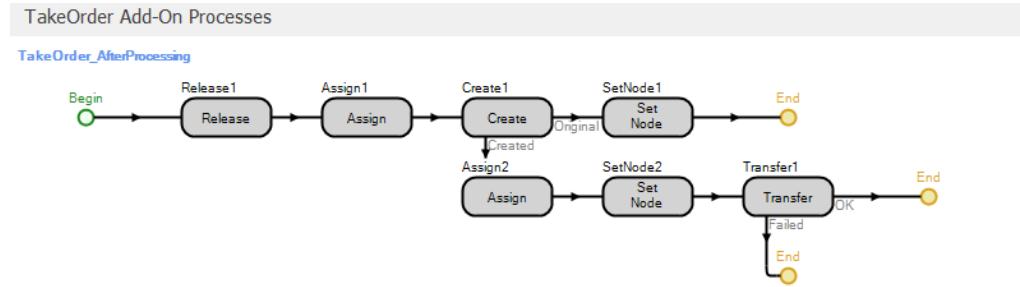


図 9.34 TakeOrder オブジェクトの AfterProcessing アドオンプロセス

（このプロセスが実行されるときに通話が終了するため）プロセスの最初のステップで、Worker1 リソースを解放する。最初の Assign ステップは以下の状態変数を割り当てる：

- HighestOrderNumber=HighestOrderNumber+1：モデル状態変数の HighestOrderNumber を増加し、現在の注文の到着を反映する。
- ModelEntity.OrderNumber=HighestOrderNumber：現在の注文番号を Customer エンティティに割り当てる。
- 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 Number Ordered に値を保存する。
- NumberOrdered[ModelEntity.NumberOrdered]=NumberOrdered[ModelEntity.NumberOrdered]+1：この割付は各サイズ（注文のピザ枚数）別に注文数を追跡するモデル状態変数を増加する。図 9.35 に NumberOrdered モデル状態変数のプロパティが示されており、サイズ 4 のベクトルであることに注意してほしい。つまり、4 つの注文サイズのそれぞれに対して 1 つの状態変数が対応する。シミュレーション時刻の任意な点で、その値がそれぞれの注文サイズで受けた注文数を表す。図 9.36 は Status Pie チャートとチャートに用いたプロパティおよび反復グループを示している。

Properties: NumberOrdered (Real State Variable)	
Value	
Dimension Type	Vector
Rows	4
Unit Type	Unspecified
Initial State Value	0
Auto Reset When Statistics Clea...	False
Display Format	
Advanced Options	
General	
Name	NumberOrdered
Description	
Public	True

図 9.35 NumberOrdered モデル状態変数のプロパティ

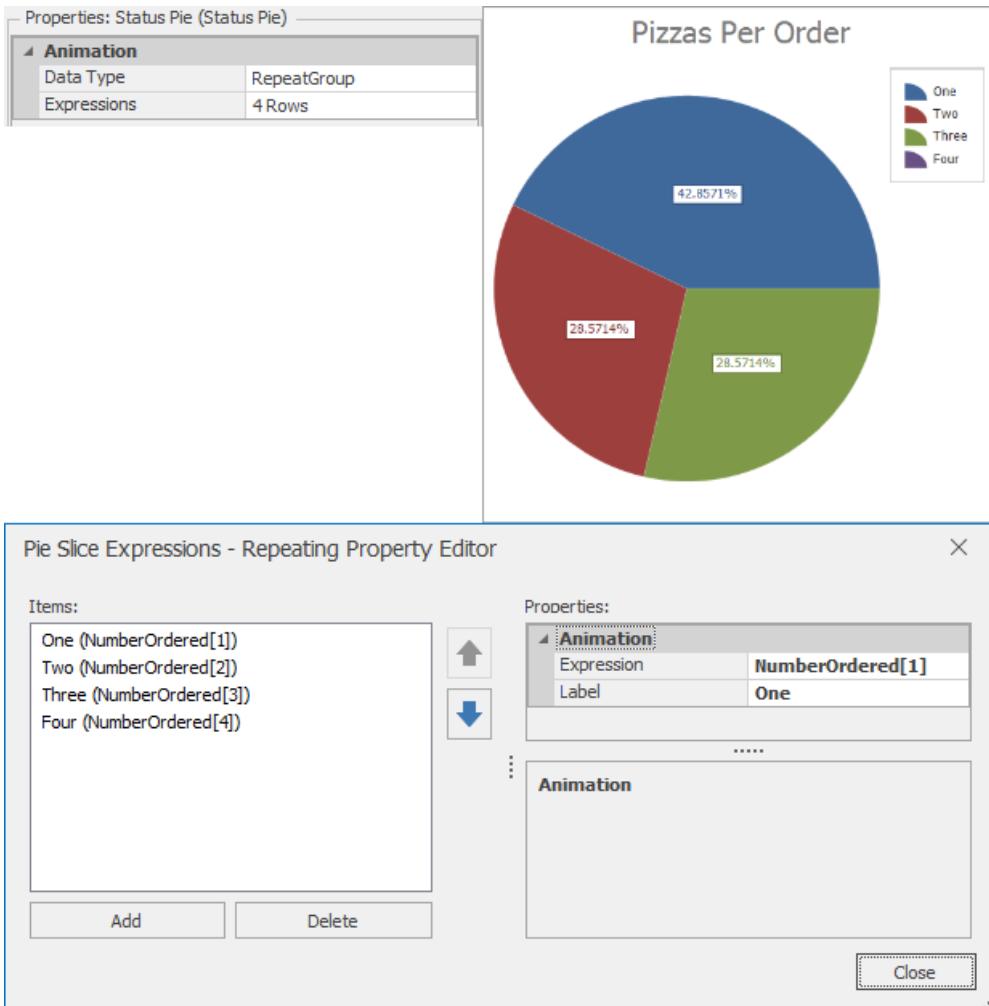


図 9.36 Status Pie チャートとそのプロパティおよび反復グループ

Create ステップは、いくつのエンティティを生成するかを決定するために、NumberOrdered 状態変数を用いて Pizza エンティティオブジェクトを生成する (Create ステップのプロパティに関しては図 9.37 を参照)。

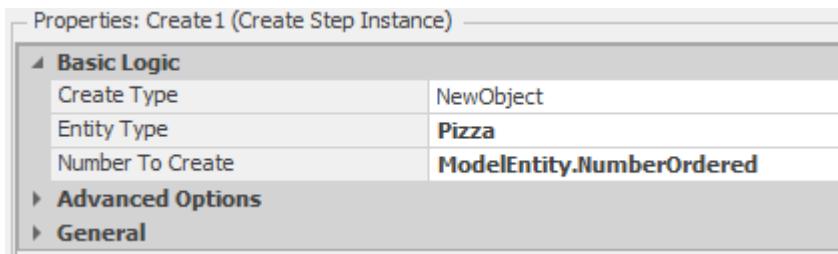


図 9.37 Create ステップのプロパティ

Create ステップを出ると、オリジナルのエンティティは Set Node ステップに移動し、そして新たに生成された Pizza オブジェクトは Assign ステップに移動する。Set Node ステップは目的地の Node Name プロパティに ParentInput@Register を割り当てる。オリジナルのエンティティオブジェクトが TakeOrder オブジェクトを離れるとき、その注文に関する Pizza エンティティオブジェクトを待つために Register オブジェクトに移される (Customer オブジェクトは上述の「顧客チケット」を表す)。Assign ステップは HighestOrderNumber の値を ModelEntity.OrderNumber 状態変数に割り当てる。Customer オブジェクトエンティティ状態変数の OrderNumber にも同じ値が割り当てられたことを思い出してほしい。これにより、OrderNumber

状態変数は、Customer オブジェクトと、顧客が注文したピザを表す Pizza オブジェクトを結合するためには用いることができる。新たに生成された Pizza エンティティオブジェクトの SetNode ステップは、Make オブジェクトにオブジェクトを移動させるように、目的地の Node Name プロパティを Input@Make に設定する。最後に、新たに生成されたオブジェクトをどこに置くかを Simio に設定しなければならない (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だけである)。したがって、Transfer ステップで、Node Name プロパティで指定されたノード (この場合、Output@TakeOrder) にオブジェクトをすぐに移動させる。ここまでで、Pizza オブジェクトを待つために Customer オブジェクトをレジスタに送り、Pizza オブジェクトを作成ステーションに送って、それぞれのエンティティ状態変数 OrderNumber を用いて、顧客注文に対応するオブジェクトのすべてを特定し、結合することができる。

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 9.38 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、いくつの「メンバー」が「親」に結合されるのかを Combiner オブジェクトに設定する。このケースでは、親は Customer オブジェクトであり、メンバーは対応する Pizza オブジェクトである。ModelEntity.NumberOrdered エンティティ状態変数を利用して、何枚のピザが 1 つの注文に属すかを指定する (生成する Pizza オブジェクト数を指定するのに、同じエンティティ状態変数を用いたこと思い出してほしい)。Member Match Expression と Parent Match Expression プロパティは、オブジェクトを結合するために、メンバーと親オブジェクトを対応させる式を示す。

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 9.38 Combiner オブジェクト Register のプロパティ

図 9.39 は、親待ち行列で待っている「Order 3; 1 pizzas」のラベルがついた Customer オブジェクトと、メンバー待ち行列で待っている Order 3 のラベルがついた 1 つの Pizza オブジェクトを示している。Floor Label が、状況を特定するのにどれだけ役立つかに注目してほしい。つまり、顧客注文 3 は 1 枚のピザから成り、箱詰めを待っている。Pizza オブジェクトが Combiner オブジェクトに着くと、他の 2 個の Pizza オブジェクトと Customer オブジェクトと共に結合され、注文が完了する。結合されると Customer オブジェクトは Combiner オブジェクトを離れ、Sink オブジェクトに移される。

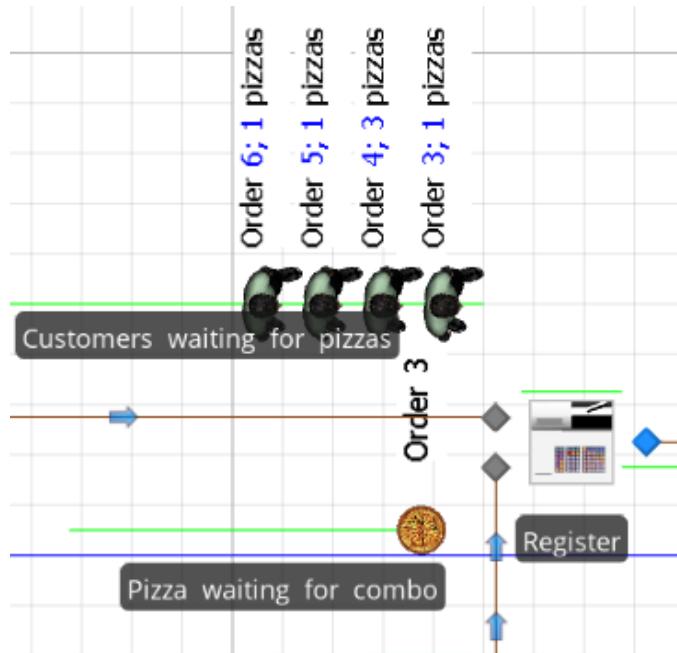


図 9.39 待ち行列に 4 つの親と 1 つのメンバー オブジェクトを持つ Combiner

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。作成ステーションには 3 人の従業員が同時にピザを作るスペースがあるので、Initial Capacity プロパティを 3 に設定する。また、Processing Time プロパティを Random.Triangular(2, 3, 4) 分に設定する。さらに、作成プロセスは従業員を必要とするので、Worker オブジェクトを占有するために Processing アドオンプロセスを用い、TakeOrder オブジェクトでしたように、従業員を解放するために Processed アドオンプロセスを用いる。ここでの違いは、作成プロセスは、従業員がピザを作るために作成ステーションに物理的に存在している必要があるということである。したがって、Worker オブジェクトが BasicNode2 (作成ステーションに指定された WorkerNetwork のノード) に移動するよう要求する必要がある。Worker オブジェクトを占有するとき、運よく、直接この要求をすることができる。図 9.40 は Make_Processing アドオンプロセスにおける Seize ステップの Seizes プロパティエディタを示している。

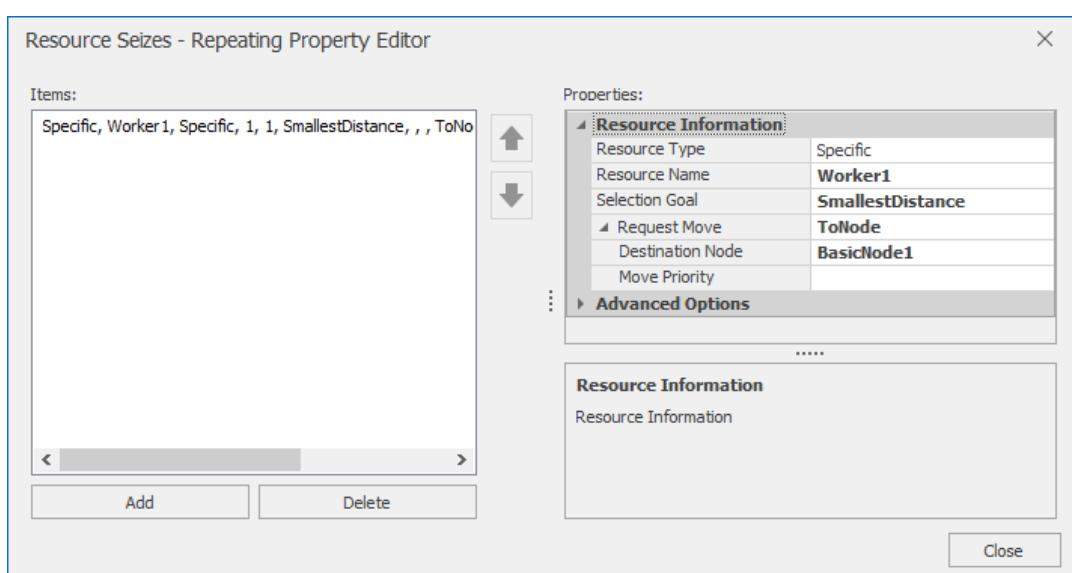


図 9.40 Make_Processing アドオンプロセスにおける Seize ステップの Seizes プロパティ

ここでは、Object Name プロパティ (Worker1) を指定することに加えて、オブジェクトを占有するとき、オブジェクトが BasicNode1 ノードに送られることを表すために、Request Visit プロパティを ToNode、Node Name プロパティを BasicNode1 に設定する。アドオンプロセスが実行されるとき、Worker オブジェクトが BasicNode に到着するまで、トークンは Seize ステップに残っている。最後に、同様に Server オブジェクト Box を設定する（ただし、Processing Time プロパティは Random.Triangular(0.5, 0.8, 1) と設定し、Worker オブジェクトは BasicNode2 を訪問するように指定する）。

これで、モデルは完成である。顧客からのコールが着信するとき、電話回線が利用可能であるなら、最初の手が空いている従業員が、電話を取り、注文を受ける。そして、対応する Customer Order エンティティがその注文に関連づけられた Pizza エンティティを生成する。次に、注文を構成する Pizza エンティティを待つために Customer Order エンティティが送り出され、そして、Pizza エンティティは、Make、Oven および 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 Order エンティティと Pizza エンティティが結合され、注文が完了する。さらに、実験をサポートするために、電話回線数 (NumLines) と従業員数 (NumWorkers) の指定にプロパティ参照を用いた。

9.2.1 Model 9-2 の実験

図 9.41 は、Model 9-2 に対して設定し、実行した簡単な 15 のシナリオ実験を示している（実行期間は 1,000 時間であり、ウォームアップ期間はなし。この実験条件の選択については、章末問題 3 で検討する）。電話回線数 (NumLines) と従業員数 (NumWorkers) が実験の決定変数である。また、3 つのレスポンスを定義した：

Scenario		Replications		Controls	General - Controls	Responses			
	Name	Status	Required	Completed	NumLi...	NumWorkers	CustTIS	OvenUtil	RejectionsPerHr
	Scenario13	Completed	10	10 of 10	1	1	16.0598	18.4577	5.7071
	Scenario14	Completed	10	10 of 10	1	2	15.0956	28.2782	2.2832
	Scenario15	Completed	10	10 of 10	1	3	14.3219	31.6468	1.2174
	Scenario10	Completed	10	10 of 10	2	1	24.8132	19.2435	5.4293
	Scenario11	Completed	10	10 of 10	2	2	19.3476	31.2318	1.2953
	Scenario12	Completed	10	10 of 10	2	3	16.2252	34.0222	0.3726
	Scenario1	Completed	20	20 of 20	3	1	30.5083	19.3529	5.386
	Scenario2	Completed	20	20 of 20	3	2	22.0499	32.4708	0.8731
	Scenario3	Completed	20	20 of 20	3	3	16.9789	34.8615	0.1262
	Scenario4	Completed	10	10 of 10	4	1	35.6134	19.3891	5.3914
	Scenario5	Completed	10	10 of 10	4	2	24.1352	33.186	0.6438
	Scenario6	Completed	10	10 of 10	4	3	17.24	34.9681	0.0438
	Scenario7	Completed	10	10 of 10	5	1	40.5784	19.3931	5.3737
	Scenario8	Completed	10	10 of 10	5	2	25.9701	33.5753	0.528
	Scenario9	Completed	10	10 of 10	5	3	17.3653	34.884	0.0214

図 9.41 Model 9-2 の簡単な実験結果

- CustTIS: 顧客注文のシステム内時間(分)。Customer.Population.TimeInSystem.Average * 60 と定義される。
- OvenUtil: オーブンのスケジュール稼働率。PizzaOven.Capacity.ScheduledUtilization と定義される。
- RejectionsPerHr: 毎時の顧客拒否数（電話線の容量制限による）。ユーザ定義の出力統計量 NumRejections を用いて定義される。NumRejections のプロパティには図 9.42 を参照してほしい。出力統計値は反復終了時に評価され、現在時刻 (TimeNow) で電話拒否の総数（モデル状態変数の PhoneRejections で蓄積）を割ると、必要な測定基準が得られる。

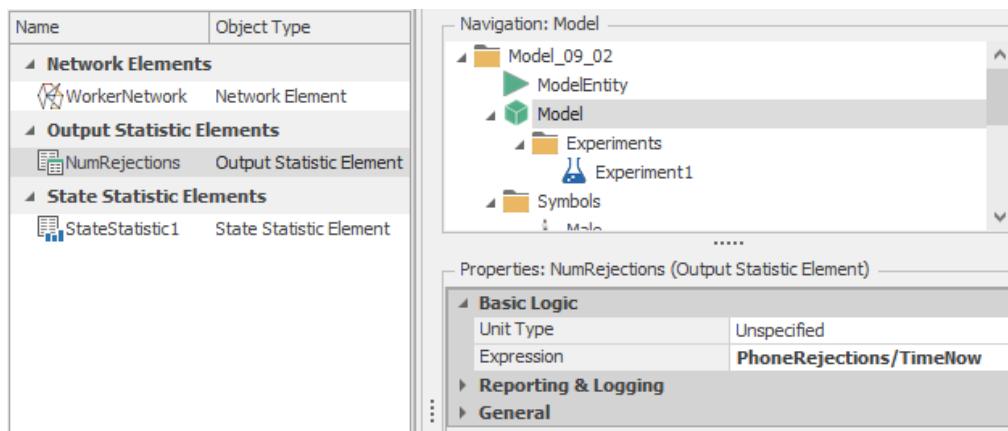


図 9.42 毎時の電話拒否数のユーザ定義出力統計

設定し終えたら、実験では「ないし 5 の電話回線と「ないし 3 人の従業員（合計 15 のシナリオ）を含むシナリオを評価し、CustTIS の値の昇順にシナリオをソートした。表の一番上のシステムが、顧客のシステム内時間について「最適」な構成である。また、（図 9.41 でしたように）手動で領域の中のすべてのシナリオを列挙するのではなく、「最適」な構成の検証をサポートするために OptQuest アドインを用いてもよい（9.1.1 節で述べた）。図 9.43 は Model 9-2 で OptQuest を用いた結果を示している。この最適化のために、実験を制御する 2 つの決定変数（Controls）と 2 つのレスポンス（Responses）を用いた（より厳密にいえば、実験を制御する方法を OptQuest に設定した）。これらの決定変数とレスポンスは以下の通り定義される：

Scenario		Replications		Controls		Responses		
	Name	Status	Required	Completed	NumLines	NumWorkers	Cust... ▲	RejectionsPerHr
▶	Scenario15	Comple...	5	5 of 5	1	3	14.3027	1.2296
▶	Scenario14	Comple...	5	5 of 5	1	2	15.0992	2.2776
▶	Scenario13	Comple...	5	5 of 5	1	1	16.0416	5.7536
▶	Scenario12	Comple...	5	5 of 5	2	3	16.2278	0.3534
▶	Scenario3	Comple...	5	5 of 5	3	3	17.0486	0.1154
▶	Scenario6	Comple...	5	5 of 5	4	3	17.2005	0.0368
▶	Scenario9	Comple...	5	5 of 5	5	3	17.3294	0.0204
▶	Scenario11	Comple...	5	5 of 5	2	2	19.3426	1.2964
▶	Scenario2	Comple...	5	5 of 5	3	2	22.0698	0.877
▶	Scenario5	Comple...	5	5 of 5	4	2	24.1491	0.634
▶	Scenario10	Comple...	5	5 of 5	2	1	24.6978	5.4708
▶	Scenario8	Comple...	5	5 of 5	5	2	25.9104	0.5248
▶	Scenario1	Comple...	5	5 of 5	3	1	30.4522	5.4052
▶	Scenario4	Comple...	5	5 of 5	4	1	35.5564	5.3998
▶	Scenario7	Comple...	5	5 of 5	5	1	40.6555	5.3564

図 9.43 OptQuest を用いた Model 9-2 の結果

- **NumLines**（決定変数）：利用できる電話回線の数。最低 1 回線から最大 5 回線を指定する。
- **NumWorkers**（決定変数）：従業員の数。最低 1 人から最大 3 人の従業員数を指定する。
- **CustTIS**（レスポンス）：顧客のシステム内時間。この出力値を最小にしたい（これにより、顧客満足度を高めることができる）。
- **RejectionsPerHr**（レスポンス）：時間当たりに拒否された顧客の電話数。この出力値を 4 未満で維持したい（任意にこの値を選んだ）。このサービスレベルの制約がなければ、最適化は、システムで電話回線の数を減少させることのみによって、顧客システム内時間を最小にするかもしれない。その結果、実際に注文を得られる顧客の数が制限される（ピザ店のマネージャがおそらく反対するだろう）。

図 9.44 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) に最小と最大の値を設定し、CustTIS レスポンスに目標 (Minimize) を設定し、RejectionsPerHr レスポンスに最大値 (4) を設定することに留意してほしい。

The screenshot displays four property windows for OptQuest parameters:

- Properties: NumLines (Control)**
 - NumLines**
 - OptQuest for Simio - Parameters**
 - Include in Optimization: Yes
 - Minimum Value: 1
 - Maximum Value: 5
 - Increment: 1
- Properties: NumWorkers (Control)**
 - NumWorkers**
 - 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 9.44 Model 9-2 の OptQuest 実験に対する決定変数とレスポンスのプロパティ

結果（図 9.43）から、1 回線で 3 人の従業員の構成が最小の顧客システム内時間 (14.3027) で、毎時の距離率の平均も受け入れられる水準 (1.2296) である。1 回線 1 人の従業員の構成では、若干大きな顧客システム内時間 (16.0416) となるが、毎時の拒否数の平均が制約に違反している ($5.7536 > 4$)。この構成の Rejections/Hr セルが赤でハイライトされ、実行不可能を示していることに注意する。分析における次の論理的な手順は、「最善」の構成を特定するために、

Subset Selection Analysis および／あるいは Select Best Scenario using KN を利用することだろう。

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である。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、ステーションからステーションへと連続的に移動する。ここで n はラインのステーション数である。ライン内で隣接しているステーションは、決められた数の製品を保持できるバッファ（バッファ容量と定義される）によって分割されている。2つのステーション間にバッファがまったくなければ、ゼロ容量のバッファを割り当てる（表現を一貫させるため）。ここで、2つの隣接ステーション i および $i+1$ を考える（図 9.45 を参照）。

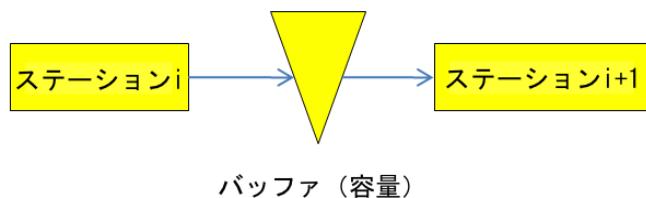


図 9.45 バッファで分割された 2 つの隣接ステーション

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）であるならば、ステーション i はブロックされて、次の製品を処理し始めることができない。他方、ステーション $i+1$ で作業が終わっていて、バッファが空であるなら、ステーション $i+1$ は部品がない（部品不足）ので、次の製品の処理を同様に開始できない。両方のケースにおいて、ラインで生産ロスが発生する。処理時間にかなり変動性があり、かつ／またはステーションの信頼性が低い場合、バッファ容量を追加することで、ステーションでのブロックや不足を明らかに減少させ、プロセスのスループットを向上させることができる。バッファ配置問題はバッファの配置場所とバッファの容量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（より具体的にいうと、バッファ配置問題を解くために用いるスループット分析を提供する）。連続生産ラインやそれらの分析に関する詳細な記述は Askin and Standridge (1993) で見られるが、上述の説明でここでのニーズ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は、単位時間当たりに生産された製品数と定義される。簡単にするために、このモデルでは、ステーションが同じで（すなわち、処理時間の分布は同じで）、また隣接ステーション間のバッファがすべて同じサイズであるとみなすので、バッファ構成は 1 つの数値で定義できる。もともと Conway et al. (1988) で示され、Askin and Standridge (1993) で解説された分析を再現するために Model 9-3 を用いる。図 9.46 は Model 9-3 の Facility ビューを示している。

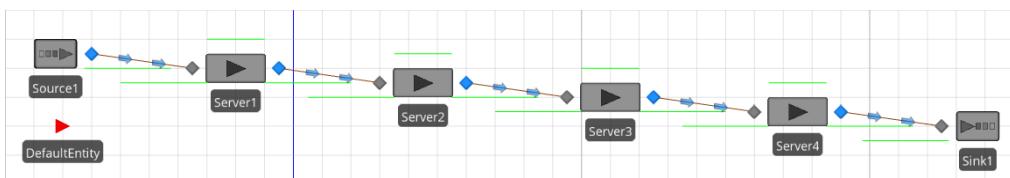


図 9.46 Model 9-3 の Facility ビュー

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加えたように見える。しかしながら、固定容量バッファ組立ラインをモデル化するの

に必要となる、いくつかの微妙な違いがある。以下に、これらの変更点について説明する。またさらに、プロパティ参照と、実験とシナリオ比較のためのツールとしての Simio Response Charts の活用について示す。

デフォルトでは、Connector または Path オブジェクトで 2 つの Server オブジェクトを接続すれば、いったん最初のステーションで処理が完了すると、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まで自動的にエンティティが移動する。もし 2 番目のステーションが稼働しているなら、移動したエンティティは待ち行列で待つ。2 番目のサーバが利用可能であるとき、待ち行列で待つ最初のエンティティを取り除いて、処理し始める。これは、実質的に 2 つのステーション間の無限容量バッファをモデル化しており、有限容量バッファに関するブロッキング効果はモデル化していない。リソースとしてモデル化する 2 つのステーション間にゼロ容量バッファを持っている（つまり、バッファがない）簡単なケースを考えてほしい。デフォルトの挙動に代わって必要なことは、現在のリソースを解放する前に、後続のリソースに利用可能な容量があることを確認することである（すなわち、エンティティが移動を開始する前に、行き先に場所があることを確認する）。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。2 つのステーション間が有限容量バッファであるなら、明らかに類似の問題があり、後続リソースの代わりに、バッファに残存容量があるのを確認することが必要である。

幸運なことに、Simio Server オブジェクトでは、重複リソースをモデル化するのはとても簡単である。図 9.47 は Model 9-3 の Server1 オブジェクトのプロパティを示している。ここでの関心は、特にプロパティの Buffer Capacity セクションにおける、Input Buffer と Output Buffer プロパティである。Server オブジェクトの Input Buffer は、じゅうぶんな容量がサーバリソースにないとき、サーバに入るエンティティを保持するのに用いられる。同様に、Output Buffer は、サーバでの処理を完了した後、まだ行く場所のないエンティティを保持するのに用いられる。これらのプロパティの値はデフォルトが Infinite であるので、Input Buffer サイズを指定するために BufferCapacity というプロパティ参照を用い、Output Buffer を 0 に設定する。すべてのステーションをこの構成とし、後続ステーションの入力バッファを表す内部待ち行列において、エンティティ数がプロパティ参照 BufferCapacity の値より少ない場合だけ、ある Server から次の Server までエンティティを移動させることができる。また、ProcessingTime プロパティにプロパティ参照 ProcessingTime を指定したことに留意する。バッファ容量と処理時間分布にプロパティ参照を用いると、異なるライン構成での実験を容易にできる（以下で説明する）。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、多くのオプションで退去と放棄をサポートしている。これらの機能は、Balking および Reneging Options プロパティ（図 9.47 参照）を用いてアクセスできる。10.2 節で示す Model 10-3 では、この詳細を用いたモデルである。

ラインの最大処理量を見積もるのにモデルを用いるので、エンティティ到着過程も変更する必要がある。周知のごとく、標準の Source オブジェクトは、到着時間間隔分布もしくは到着スケジュールを用いてエンティティを生成する。この最大処理量モデルでは、最初のステーション(Server1) を待っているエンティティが常に存在することを確かにすることである。つまり、最初のステーションでは決して不足しない（これは、ラインに原料の無限供給があるのと同じである）。このロジックを実行するために、Source1 の Maximum Arrivals プロパティを 1 に設定して、時刻 0 で 1 つのエンティティを生成する。そして、Server1 で処理が完了したとき、エンティティを複製して Server1 の入力に複製を送り返す。Server1 の Server1_Processing アドオンプロセスを用いて、この複製とルーティングを実行する（図 9.48 を参照）。

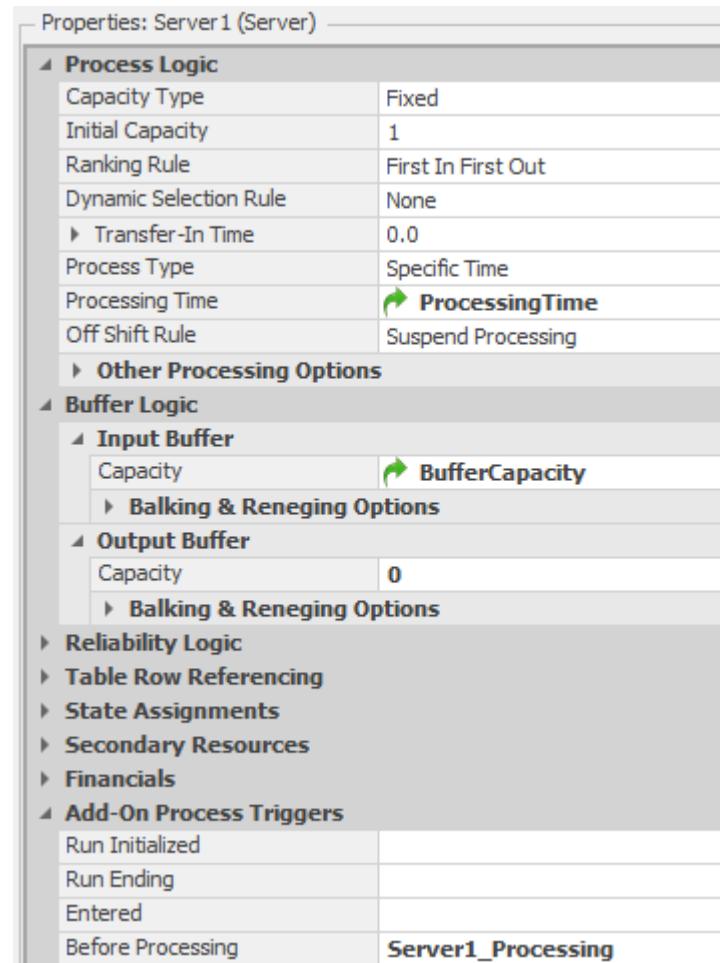


図 9.47 Model 9-3 の Server1 オブジェクトのプロパティ

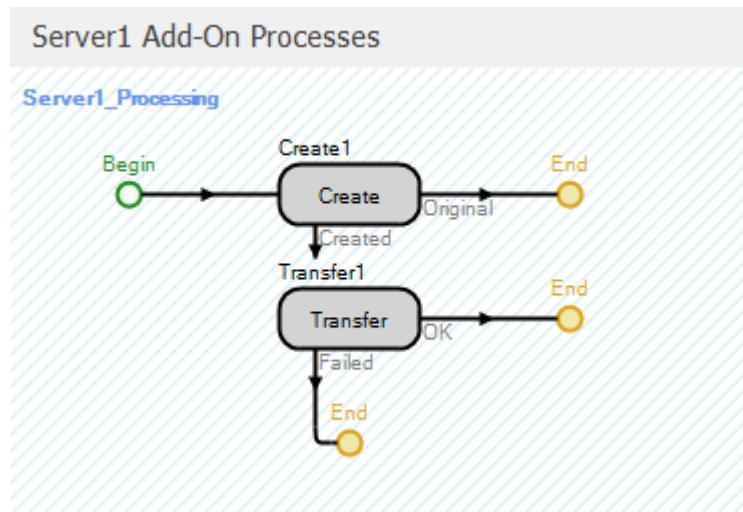


図 9.48 Server1 の Processing アドオンプロセス

Create ステップでは、エンティティを複製する。そして、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。Original 出力がプロセスの End に行くとき、オリジナルのエンティティは通常の経路を通り、Server2 あるいは Server1 と Server2 の間のバッファに利用可能な容量があれば 2 番目のステーションに移動する。そうでなければ、Output Buffer 容量が 0 に設定されているため、Server1 に留まる。最初のステーションで無限の原料供給を維持する別の方法に関しては、章末の問題 8 を参照してほしい。

ここで、初期モデルが完成したので、異なる構成を実験するためにこのモデルを用いることができる。図 9.49 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ので、これらが Controls として実験に現れる。この実験では、処理時間が平均 10 分の指数関数に従い、バッファ容量が 1 ずつ 0~10 まで増える 11 のシナリオを定義した。また、30,000 分の実行期間と 20,000 分のウォームアップ期間を設定した。10,000 分の実行時間で平均 10 分の処理時間が各ステーションで必要となる状態では、処理時間に変動がなければ 1,000 個の製品生産が期待できる。テストしたシナリオの 50 回の反復実行の平均スループットは、バッファ 0 の 517.5 から 10 バッファの 897.78 に変化した。

図 9.50 は、図 9.49 における実験の Response Chart を示している。図は明らかに増加するバッファ容量に伴い、スループットの増加が期待される傾向を示しているが、しだいに遞減していく。

	Scenario		Replications		Controls		Responses
	Name	Status	Required	Completed	ProcessingTime (Minutes)	BufferCapacity	
▶	Scenario1	Completed	50	50 of 50	Random.Exponential(10)	0	517.5
	Scenario2	Completed	50	50 of 50	Random.Exponential(10)	1	632.7
	Scenario3	Completed	50	50 of 50	Random.Exponential(10)	2	699.74
	Scenario4	Completed	50	50 of 50	Random.Exponential(10)	3	745.64
	Scenario5	Completed	50	50 of 50	Random.Exponential(10)	4	777.68
	Scenario6	Completed	50	50 of 50	Random.Exponential(10)	5	803.3
	Scenario7	Completed	50	50 of 50	Random.Exponential(10)	6	823.68
	Scenario8	Completed	50	50 of 50	Random.Exponential(10)	7	839.48
	Scenario9	Completed	50	50 of 50	Random.Exponential(10)	8	851.08
	Scenario10	Completed	50	50 of 50	Random.Exponential(10)	9	861.68
	Scenario11	Completed	50	50 of 50	Random.Exponential(10)	10	871.78

図 9.49 Model 9-3 の指數関数の処理時間分布による実験

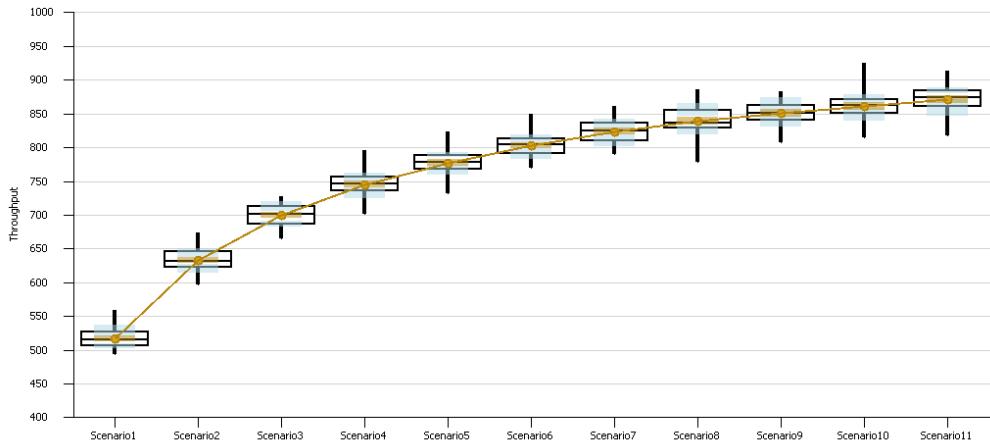


図 9.50 Model 9-3 の指數関数の処理時間分布のレスポンスチャート

以前に説明したように、Simio はリソースと Server オブジェクトにおいて、自動的に稼働／遊休の状態変数を追跡する。実際に、Simio はデフォルトで Server オブジェクトにおいて、以下の状態変数を追跡する：

- Starved (不足) : サーバは遊休であり、エンティティを待っている。
- Processing (処理) : サーバは 1 つ以上のエンティティを処理している。
- Blocked (ブロック) : サーバは処理を完了したが、エンティティがサーバを解放できない。

- Failed (故障) : サーバが故障している。
- Offshift (オフシフト) : サーバはスケジュールを用いており、オフシフトである。

ここで検討している連続生産ラインでは、最初のサーバには原料の無限供給があり、サーバ故障がなく、仕事のスケジュールもないため、サーバは製品を処理している（稼働）か、部品が不足している（遊休）か、またはブロックされている。連続生産ラインにおける4つのサーバの状態変数を追跡するために、SimioのStatus Pieチャート機能を用いることができる（図9.51を参照）。

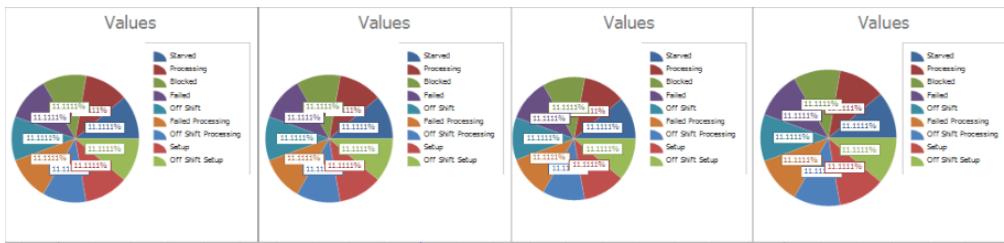


図9.51 Model 9-3 のサーバ状態変数の円グラフ

Status Pieチャートは、特定のサーバがそれぞれの状態変数であった時間の割合を示している。図9.52はServer1とServer3のStatus Pieチャートのプロパティを示している。双方の図で、Data TypeプロパティがListStateに設定され、List StateプロパティはサーバのResourceStateとなっている。2つの図の違いは、Server3のチャート（右）のAttached ToにServer3が設定されていることである。したがって、Server3を参照するためにListStateプロパティ値は必要としないが、Server1のチャート（左）にはサーバに付属されていないので、ListStateプロパティ値に明確にServerオブジェクトの参照(Server1.ResourceState)をつけなければならない。さらに、Server3のチャートは、Serverオブジェクトに付属されているので、FacilityビューにあるServerオブジェクトに伴って図も「動く」。ラインにおける最初のサーバは常に不足(Starved)せず、また最後のサーバは常にブロック(Blocked)されないことに注意する。まさにこれは、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たことである。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はブロッキングが多く、3番目のステーションは不足が多い。より工程数の多いラインであっても、このパターンが続くだろう。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生じ、ラインの最後に近いステーションではより多くの不足（スタービング）に見舞われる傾向がある。

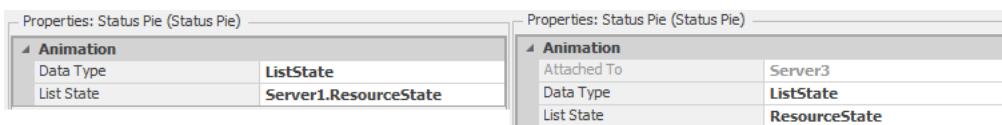


図9.52 Server1（左）とServer3（右）のStatus Pieチャートのプロパティ

処理時間分布を指定するためにプロパティ参照を用いたので、実験で容易にこれらを変えることができる（モデルに戻る必要がない）。図9.53は、実験のControlsセクションのプロパティ参照の値を変え、処理時間を三角分布(5, 10, 15)分に変更した以外は、図9.49に示されていたものと同様の実験のResponse Chartを示している。指数分布より三角分布はかなりバラツキが少ないので、三角分布のケースでは、ブロッキングと不足の影響がより少ないことが期待される。そして、バッファを追加することによるスループットの増加の効果がより早く現れることが期待できる（すなわち、少数のバッファでより多くのスループットが得られる）。これらの分析結果のすべては、Askin and Standridge (1993) と Conway et al. (1988) で説明された手法を用いた結果と同様であり、予想と合っている。

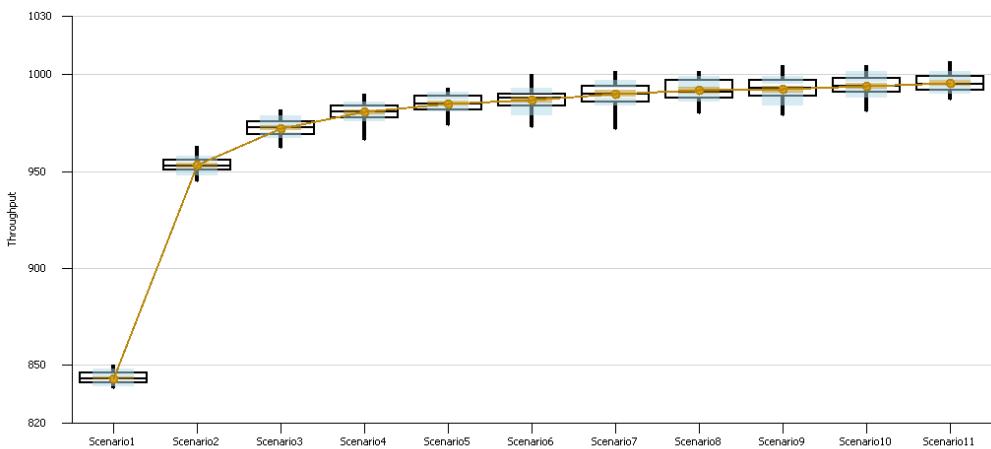


図 9.53 Model 9-3 の三角分布の処理時間のレスポンスチャート

9.4 まとめ

本章では、3つのシステムを対象に、それぞれ1つ以上のSimioモデルにおける高度なシミュレーション概念が含まれているものを概観した。モデリングの側面に加えて、これらのモデルそれぞれで実験にも焦点を合わせ、OptQuestのシミュレーションベース最適化のアドインを紹介した。ユーザが適切な目標、制約条件および必要条件を定義すると、OptQuestはシステムの最適構成に向かってSimio実験を「駆動する」。OptQuestによって特定された複数のシナリオから「最善」の解を選択する手順⁵によって、ランキングと選択のプロセスを解説し、SimioのSubset Selection Analysis機能とSelect Best Scenario using KNの活用方法を示した。これらのツールは、一般的な出力分析のプロセスをかなり容易にする。

もちろんここでは、一般的なシミュレーションの応用と、特にSimioの能力の表面をなぞっただけである。したがって、これがシミュレーションとSimio活用の習得のための探求の終わりではなく、むしろはじまりだとみなすべきである。さらに、一般的な出力分析について、特にシミュレーションベースの最適化については、極めて入門的に紹介したため、他の文献資料や実験の中から学び得るもののがたくさんあるだろう。

9.5 問題

- Model 8-3を元に、患者の物理的な移動とNurse/Aideによる付き添いの機能を含むように、Model 9-1を変更しなさい。そしてModel 9-1とリソース要件を比較しなさい。このレベルの詳細さを含むことが、結果の妥当性を「改善」するか？理由は何か？
- Model 9-1を変更して、Urgent患者がDoctorリソースを先取りするように変更しなさい。特に、Urgent患者が到着するとき、すべての医師が忙しいのであれば、Urgent患者は他のUrgentではない患者から医師を先取りするべきである。そして、モデルの結果を比較しなさい。結果から、救急診療の実システムにおいて類似する変更を行うようにアドバイスできるか？
- Model 9-2では、ウォームアップ期間を持たない1,000時間の実行期間を用いた（これらの設定に対する説明をまったくしなかった）。これは妥当か？理由は何か？より短い実行期間で同じ結果を得ることは可能か？より長い実行期間は結果を改良することになるか？
- 9.2.1節では、Model 9-2を対象に実験（手動とOptQuest）を行ったが、特定のシナリオ集合から「最善」の構成を選択するために、Subset Selection Analysis機能、および／またはSelect Best Scenario using KNアドインを活用しなかった。ここで、これらの機能を利用

⁵ 解説したランキングと選択のプロセスは、シナリオの特定方法に関わらず、あらゆるシナリオの集合に対しても機能する。つまり、必ずしもOptQuestによってシナリオを生成する必要はない。

- し、「最善」の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. 最近の多くのピザティクアウト店では配達をしている。顧客注文の配達を支援するために、Model 9-2 を変更しなさい。90%の顧客がピザの配達を希望すると仮定する (Model 9-2 で仮定したように、残りの 10%はピザを取りに来る)。ピザ店は顧客配達区域を 4 つのゾーンに分割した。そして、注文が同じゾーンから来ると、配達員は複数の注文を同時に配達することができる。表 9.3 は各ゾーンの配達時間の分布と各ゾーンからの顧客の割合を示している。簡単のため、配達員が複数の注文を届ける場合には、各注文のサンプル配達時間に 4 分を追加する。たとえば、配達員が 3 つの注文を届けていて、元々抽出された配達時間が 8 分であれば、代わりに $8 + 4 + 4 = 12$ 分を総配達時間として用いる。顧客到着、処理時間、そして従業員の特性は、Model 9-2 と同じデータを用いる。実験を設定して、何人の配達員が雇われるべきであるかを決定しなさい。解決策を必ず裏付けること (すなわち、配達員の適切な人数をどのように決めたのかを述べること)。

表 9.3 問題 5 の各ゾーンの顧客割合と配達時間分布（配達時間は往復。単位は分）

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。結果を前のモデルと比較しなさい。これは優れた方針か？少なくとも 3 つの注文を必要とする案はどうだろうか？
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 と同様に、それぞれのペアのステーション間には、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、また最後のステーションは必ず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、ステーションは処理時間分布が同じではなく、ランダムな故障を被りやすいということである。表 9.4 で処理時間分布と故障データを与える (Station 5 の対数正規 (Lognormal) 分布パラメータ(4, 3) は、生成された確率変量の平均と標準偏差である一方、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ことに注意すること。必要な変換の詳細については Simio ヘルプを参照)。連続生産ラインの Simio モデルを発展し、総バッファ容量 30 の「よい」配置を見つけるための実験を行なさい。総バッファ容量の配置例を 1 つあげると、[6, 6, 6, 6, 6] である (それぞれのペアのステーション間に 6 つ)。目標は、ラインスループットを最大にすることである。実験によって、最適なバッファ配置を発見できるか？理由と共に述べなさい。

表 9.4 問題 7 の処理時間と故障／修理分布（単位は分。、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、Server オブジェクトに対応するアドオンプロセスを用いるのではなく、新しいエンティティを最初のステーションで生成するために Source オブジェクトの On

Event Arrival Mode プロパティを用いて、Model 9-3 を変更しなさい。

9. 9.2.1 節で述べた 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にした。しかしながら、スタッフ配置費用を考えていなかった。代替手段として、顧客待ち時間、従業員の人物費、および顧客サービスを組み込んだ、単一の「総費用」測定基準を考えなさい。出力統計としてその総費用測定基準を加えて、最適な構成を見つけるために OptQuest を活用しなさい。
10. Model 9-2 は、受注やピザ作成、そしてピザの箱詰めにおいて、プールされた従業員戦略を用いている。ここでは、専門職の従業員戦略を用いるモデルを開発しなさい（すなわち、各従業員は 1 つのタスクに専念する）。そして、Model 9-2 と結果を比較しなさい。どのような条件下で、どちらの戦略がよりふさわしいといえるか？実験により答えを裏付けなさい。

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や関連する Simio の機能を解説することである。本章の各節は、おおよそ独立している。最初から最後まで一貫する「中心的なテーマ」は存在せず、モデリングの必要性に応じて、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あり、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。コレクションはモデル内のオブジェクトの集合である。コレクションの例としては、Entity Populations や Queue States、Table Rows、Object Lists、Transporter Lists、Node Lists などがある（コレクションのすべての種類についてはヘルプを参照）。本節では、Search ステップの活用を例示するため、2つの例題モデルを解説する。Search に関する他の例は SimBit にある（Simio の Support リボンにある Sample SimBit Solutions をクリックし、キーワード Search で検索してほしい）。

10.1.1 Model 10-1：ステーションのエンティティを検索・取り出す

このモデルでは、到着エンティティを「待機ステーション」に待たせ、定期的に取り出し、処理を行いたい。これにより、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。

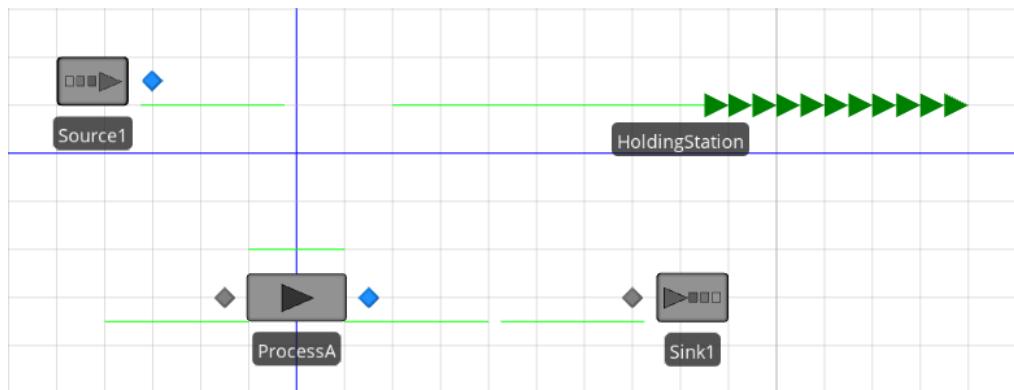


図 10.1 Model 10-1

Holding Station のラベルがついた待ち行列状態は、モデルで定義された Station 要素に対応している（図 10.2 参照）。Facility View の待ち行列は、Animation リボンの Detached Queue ボタンにより設置されている（待ち行列の状態名は HoldingStation.Contents）。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用いて、ステーションに搬送される（図 10.3 参照）。

Process1 では End Transfer ステップを用いて、エンティティの搬送を終了させる。このプロセスは、エンティティがステーションに侵入した際に自動的に発火されるイベント HoldingStation.Entered で実行される点に留意してほしい。

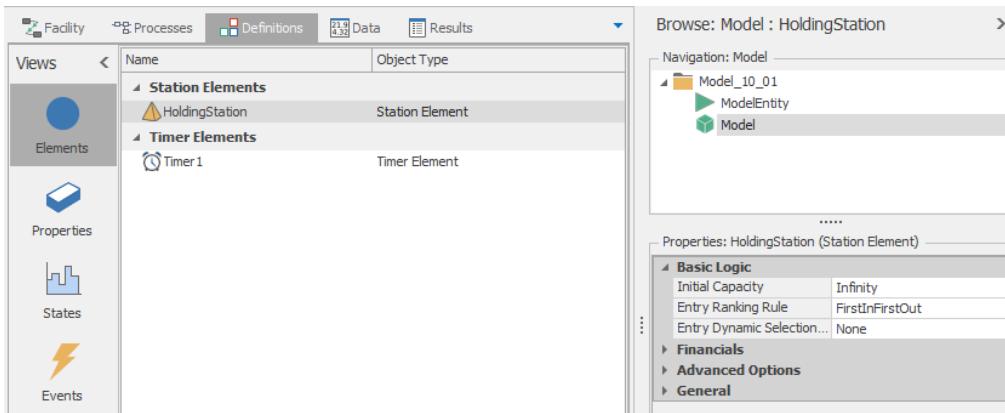


図 10.2 Model 10-1 の定義

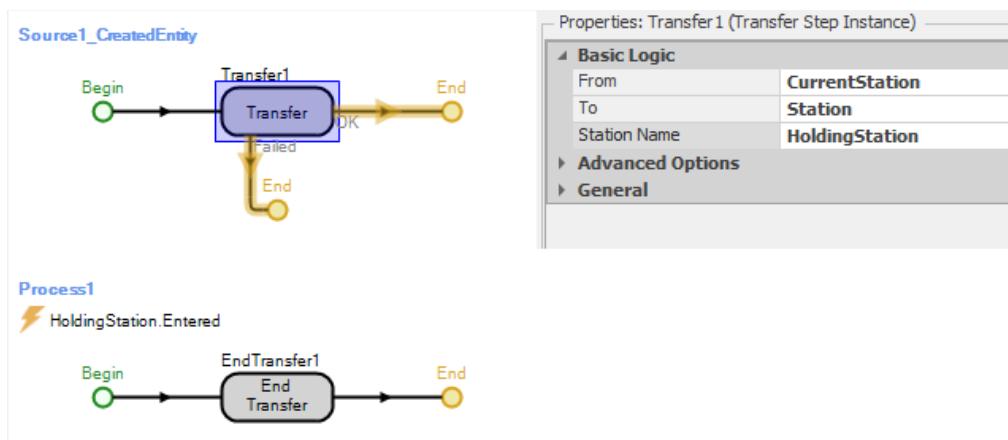


図 10.3 Source1 から HoldingStation への搬送に関する 2 つのプロセス

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（シミュレーション実行の最後まで）無期限に留まる待機ステーションに直接搬送される。エンティティを定期的に待機ステーションの外に取り出し、処理したいので、タイマーによって実行されるプロセスを用いて、待機ステーションを検索し、ProcessA の入力ノードに処理されるエンティティを搬送する。図 10.4 にこのプロセス (PullEntities) を示す。Search ステップでは、Collection Type に QueueState、Queue State Name に HoldingStation.Contents、Limit に Infinity を設定する。このプロパティ設定により、Search ステップは待ち行列 HoldingStation のすべてのエンティティを検索するようになる。見つかったエンティティの各々は、それから、オブジェクト ProcessA の入力ノードに搬送される。ここで、タイマーイベント (Timer1.Event) によってプロセスが実行されることに留意してほしい（タイマーの定義については図 10.2 を参照）。この簡単な例では、ステーションからエンティティを定期的に取り出すためにタイマーを活用しているが、他の多くのプルおよびプッシュメカニズムにも同じ Search ロジックを用いることができる。

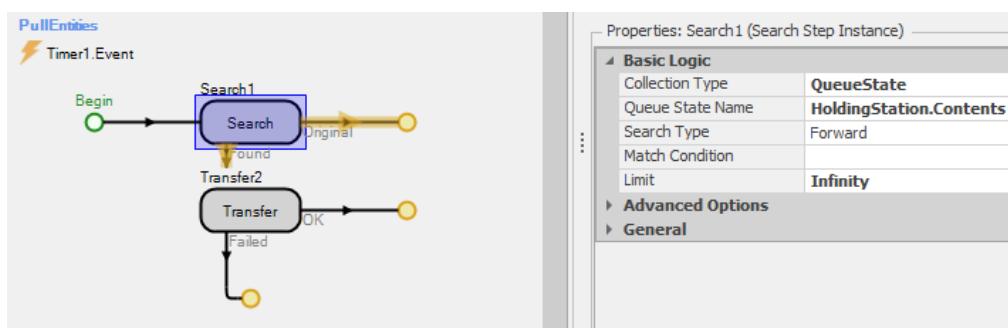


図 10.4 待機行列を検索し、ProcessA にエンティティを搬送するプロセス

10.1.2 Model 10-2：バッチ内の総処理時間の合計

Search ステップを用いたこの 2 つ目の例では、エンティティをバッチにまとめ、バッチを構成するエンティティの個々の処理時間を合計することによって、バッチの処理時間を決定したい。図 10.5 に完成したモデルを示す。Source1 がタイプ A のエンティティ（バッチの親エンティティ）を生成し、Source2 がタイプ B のエンティティ（バッチのメンバエンティティ）を生成する。タイプ B エンティティの個々の処理時間と、バッチの総処理時間（タイプ A エンティティ）、つまり累積値を記憶するために、ユーザ定義エンティティ状態 ProcessTime が用いられる。図 10.5 に示されているように、タイプ B エンティティの処理時間の値は、パラメータ (1, 2, 3) の三角分布を用いてランダムに割り当てられる。

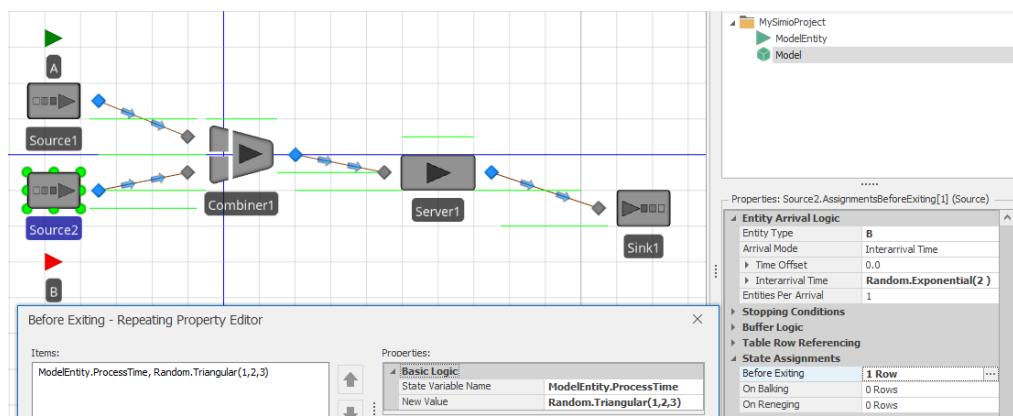


図 10.5 Model 10-2

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。ここで、Server1 でバッチの処理時間として利用するため、バッチにまとめられる前に、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たい。そのため、バッチが形成される Combiner1 で関連するプロセスが実行される前に、Processing アドオンプロセストリガーを用いる。図 10.6 にこのプロセスを示す。ここで Search ステップを用いて、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) に値を合計する。これにより、指定した式の値を合計し、トーカンの戻り値に合計を格納する。それから、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) を代入し、この状態変数を Server1 のプロパティ Processing Time で利用する。結果として、タイプ B の個々のバッチ構成要素は、それぞれサンプリングされた処理時間を保持し、バッチは処理時間として、これらの値の合計を持つことになる。

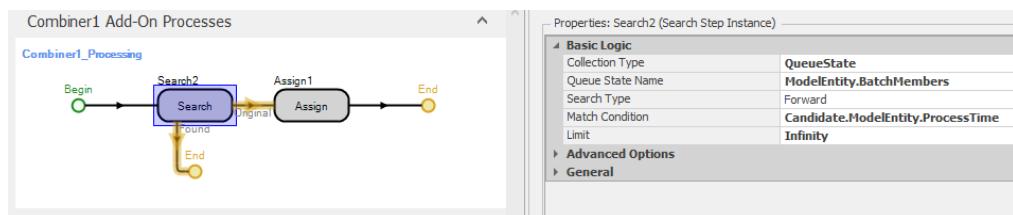


図 10.6 Model 10-2 における処理時間を合計するアドオンプロセス

Search ステップの議論を終える前に、留意すべき点を指摘しておく。Search ステップは非常に強力で、多くのことを実現できるが、乱用による潜在的な欠点がある。たとえば、10,000 行のテーブル、あるいは 10,000 エンティティのコレクションがあり、次の適切な行動を決定するため

に、事前にすべてのエンティティを検索したいとしよう。Simio ではこれを簡単に実現できるが、実行速度が顕著に遅くなることに気づくに違いない。検索の過剰利用は、実行速度の問題に対する一般的な原因となる。

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、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退去することを選択する際に発生する（たとえば、しばらく列に並んで待っていると、列があまりにもゆっくりと進むため、去ることを決めるような場合）。ジョッキーイング (Jockeying)、あるいは待ち行列乗換は、顧客がある待ち行列を放棄し、別の待ち行列に加わる際に発生する待ち行列のふるまいである。幸い、Simio は Standard Library オブジェクトのプロパティ Buffer Logic を利用して、退去と放棄の両方を包括的にサポートしている。Model 10-3 は、4.4 節の ATM モデルに、ATM 顧客について基本的な退去と放棄のふるまいを組み込んだ、修正版である。修正モデルでは、到着した客は待ち行列に 4 人の客がいれば退去し、待ち行列での滞留時間が 3 分経つと確率的に放棄する（これらの 2 つの値は恣意的に選択した。実際に活用する際の値は状況に依存するだろう）。

図 10.7 に、オブジェクト ATM1 のプロパティを表示した Model 10-3 を示す。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が送られる Sink モジュール Balks と、放棄するエンティティが向かう Sink モジュール Reneges である。プロパティウィンドウにおいて、オブジェクト ATM1 の Input Buffer Capacity が 4 に設定され、プロパティ Balk Decision Type に Blocked が選ばれていることに注目してほしい。これは、エンティティが満杯のバッファに到着したとき、ネットワークにブロックされる（既定の動作）のではなく、到着エンティティが Input@Balks に転送されることを意味する。

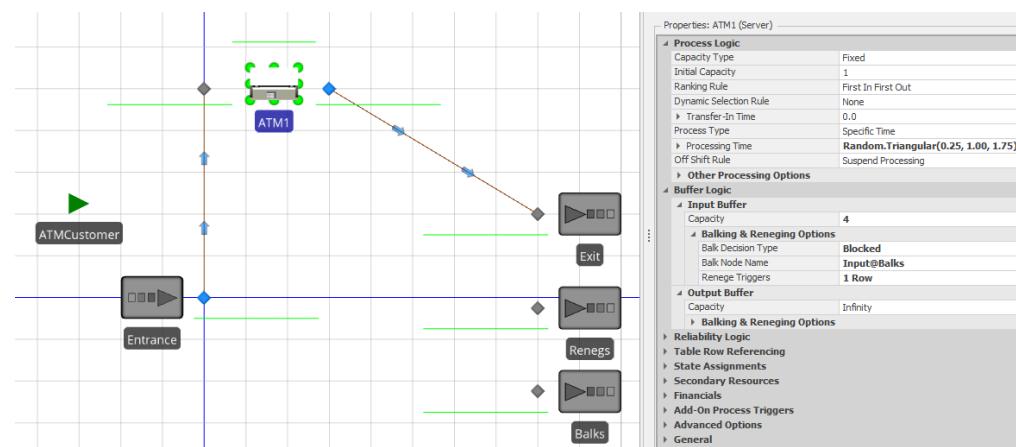


図 10.7 Model 10-3 とオブジェクト ATM1 のプロパティ

放棄のふるまいは、プロパティグループ Reneg Triggers で指定される（図 10.8 参照）。そこで、3 分間の滞留時間の後で、エンティティが確率 0.5 で放棄することを設定する。放棄するエンティティは、Input@Reneges（その Sink の入力ノード）に送られる。確率検定で「偽」となったエンティティは、単に待ち行列に留まる。

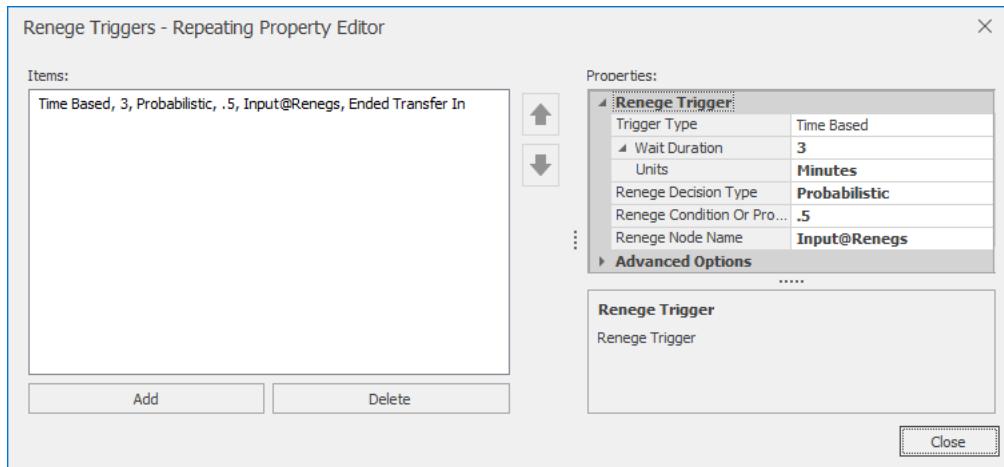


図 10.8 Model 10-3 のプロパティ Reneg Triggers

Model 10-3 の退去および放棄のふるまいは極めて基本的なものであるが、より複雑な（そして包括的な）退去、放棄および待ち行列乗換のふるまいを構築するために、同様の方法を利用することができる。他のほとんどの Simio トピックスと同じく、退去や放棄の活用を例示する、次のような SimBit Solutions がある：

- `BalkingOnSourceBlockingOnServer`：バッファ容量が存在しない窓口におけるシンプルな退去。
- `ChangingQueuesWhenServerFails`：窓口が故障したとき、別の窓口に移って待つエンティティ。
- `ServerQueueWithBalkingAndReneging`：エンティティが退去する際の固有の許容待機数と、放棄する際の固有の許容滞留時間を持つ。
- `MultiServerSystemWithJockeying`：任意のエンティティが待機している待ち行列よりも、別の待ち行列が短くなった場合に、常に短い待ち行列に移動する（乗り換える）。

10.3 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から、プロパティ Process Type を Specific Time のままにしてきた。これは、単一の処理時間を持つ、単一工程の処理を意味している。たとえば複数の工程が連続で行われたり、並列で実行されたり、あるいは条件によって変わらるような、より複雑な場合はどのようにしたらよいだろうか。これについて、Simio はタスクシーケンスというソリューションを提供している。

タスクシーケンス (Task Sequences) は、タスクを処理する構造的な順序を定義し、実行する。これを利用するもっとも簡単な方法は、Server の Process Type を Task Sequence に指定することである。これにより、処理する Tasks を設定する Processing Tasks リピートグループが展開される。なお、この機能は Combiner および Separator オブジェクトにもあり、また Task Sequences 要素を直接利用することもできる。Task は、処理時間のような簡単なことから、多くのオプションを持たせることもできる。単一のタスクを持つこともできるし、多くのタスクを設定することもできる。複数のタスクを持つ場合、直列、並列、条件付き、確率的に、そしてそれらを組み合わせて、実行することができる。

それでは、非常にシンプルなシナリオで解説しよう。2 工程の塗装作業を考える。第 1 工程は Prepare であり、0.5 ないし 1.5 分かかる（一様分布）。第 2 工程は Paint であり、最小 0.5、モード 1.0、最大 2.0 分かかる（三角分布）。部品は、平均 2.5 分の指数分布に従う時間間隔でシステムに到着する。このシンプルなシステムは 2 つのサーバで簡単にモデル化できるが、すべての作業を一か所で行うこととして、1 つのサーバで 2 工程を表現してみよう。

- Source、Server、Sink を追加し、Source のプロパティを適切に設定する。
- Server では、Process Type に Task Sequence を指定する必要がある。
- Processing Tasks をクリックし、Tasks リピートグループを開く。
- 1つ目のタスクを追加する。Name を Prepare に、Processing Time を Random.Uniform(0.5, 1.0) に変更する。その他のフィールドは既定値のままでする。
- 2つ目のタスクを追加する。Sequence Number を 20 に変更する。これは、1つ目のタスクである「10」に後続することを意味する。Name を Paint に、Processing Time を Random.Triangular(0.5, 1.0, 2.0) に設定する。ここでも、その他のフィールドは既定値のままでする。
- 図 10.9 の通りになっているか確認する。

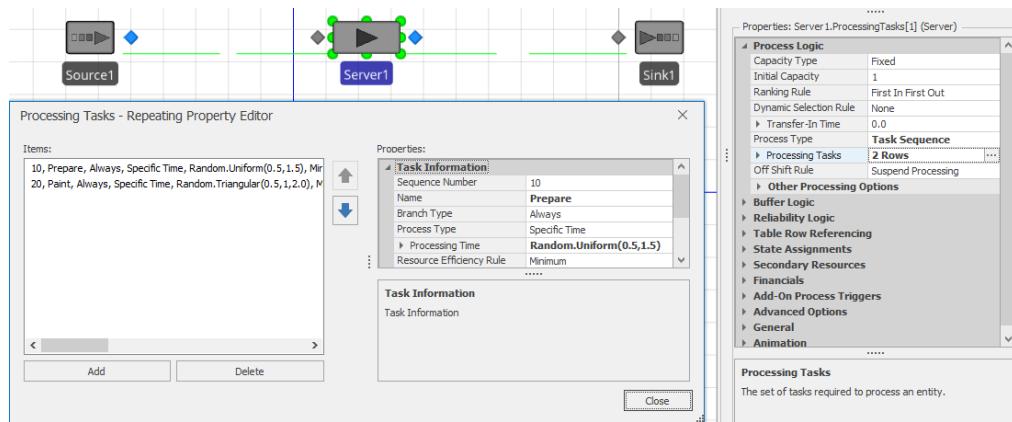


図 10.9 Model 10-4 の Task Sequences リピートグループ（初期）

現時点でもモデルを実行して作業を確認すべきであるが、正確に動作しているかを判断するのは難しい。Trace ウィンドウで Step を用いて、システムを移動する特定のエンティティの挙動を調査し、動作の様子を検討することを勧める。

それでは、モデルを拡張し、多くの機能を学び、モデルの挙動をより明確に説明できるようにしよう。このモデルでは、Source の近くに位置するホームノード BasicNode1 に、Prepper という名前の作業者がいるものとする。また、Sink 近くのホームノード BasicNode2 に、Painter という名前の 2 番目の作業者もいる。遊休中、両者は各自のホームノードに戻る。さらに、エンティティが Server で処理されている間に待機する場所の真上に、3 つ目の BasicNode である WorkPlace もある。

Server で Task Sequences リピートグループを再び開き、以前に説明したリソース定義によく似た Resource Requirements セクションに注目してほしい。適切な作業者を要求し、作業を行う WorkPlace ノードに来るようにするため、2 つのタスクを変更する（図 10.10 に 1 つ目のタスクを示す）。ここでモデルを実行すると、Prepper が Prepare タスクを行うために左から接近し、それから Paint タスクを行うために右から Painter が近づく様子がわかるだろう。

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 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。

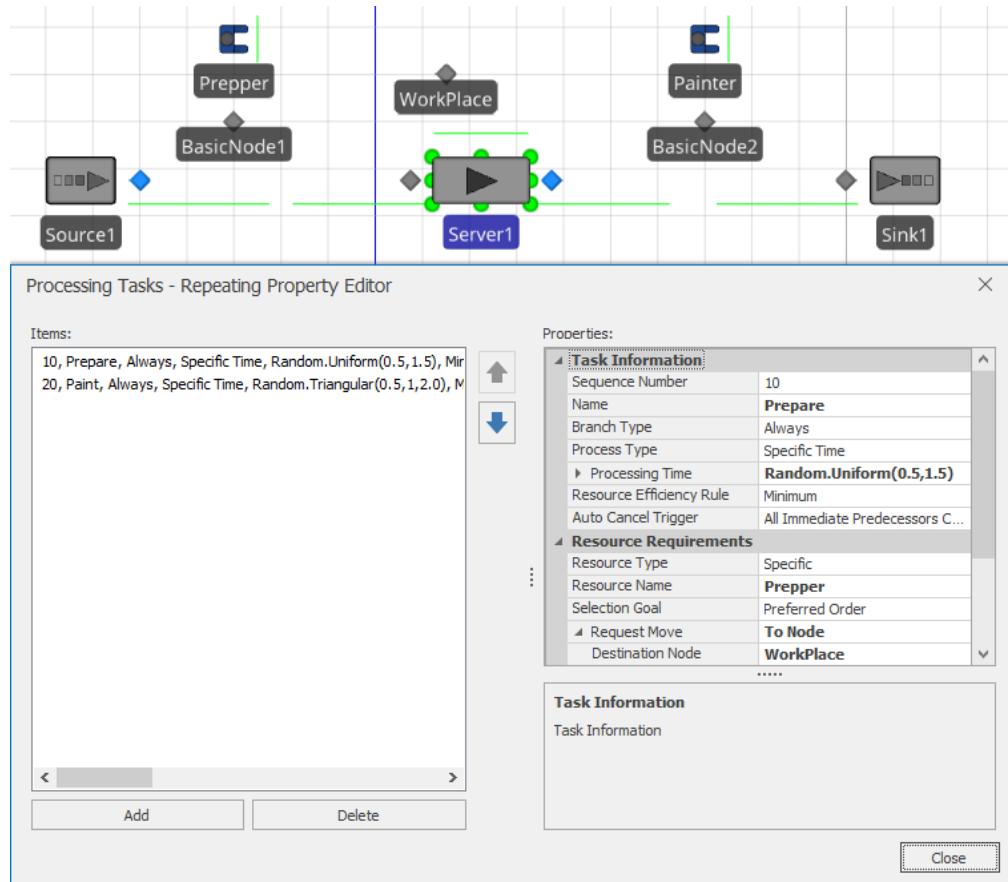


図 10.10 Model 10-4 の Task Sequences リピートグループ（完成）

すべてのタスクは、タスクがどのような状況で実行されるか決定するのを助けるために、Branch Type を持つこともできる。上記の例では、既定値の *Always* を用いた。これは、名称が意味するところ、常に両方のタスクを実行する。しかし、他にも Conditional (条件)、Probabilistic (確率)、Independent Probabilistic (独立確率) のオプションがある。たとえば、時間の 45%だけタスクの実行が必要とされるように、Preparation に分岐タイプ Probabilistic を指定することもできる。本段落の冒頭、タスクが実行されるか「決定するのを助ける」という表現を用いたことに留意してほしい。どのタスクが実行されるかに関して、別の考慮事項として、先行タスクが取り消されたかどうかも基準にして考えるべきである（たとえば、Preparation が必要ない場合、Paint を行うべきだろうか）。プロパティ Auto Cancel Trigger は、既定ではすべての直前の先行タスクが取り消された場合 (All Immediate Predecessors Cancelled)、タスクを取り消すが、アクションに None を選択することで、タスクを続行することもできる。

複数のタスクが存在する場合、処理される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) を指定した。比較的簡単な順序の場合、これがおそらくもっとも簡単な方法である。しかし、Server に戻ると、プロパティ Task Precedence Method を含む Other Task Sequence Options がある。すでに利用した Sequence Number Method の他に、Immediate Successor および Immediate Predecessor から選ぶことができる。後者 2 つのオプションは、図 10.11 のような複雑なシーケンス図をより簡単に作成できる。

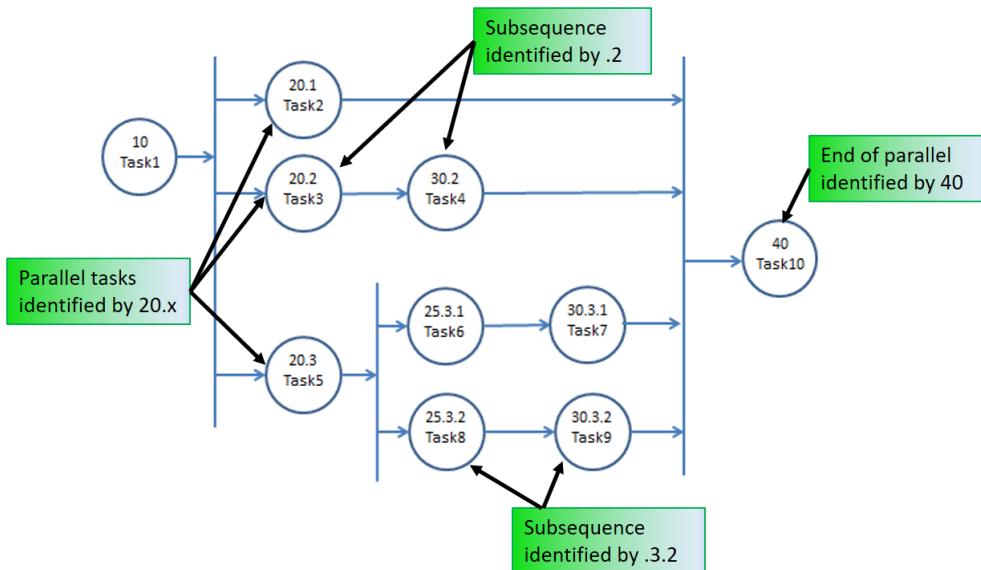


図 10.11 順序番号を用いた複雑なタスクシーケンス

Server を解説したとき、プロパティ Process Type の説明を省略し、Specific Time だけを利用したことを思い出してほしい。これは、Task Sequences について学んでからでないと、理解が難しいからであった。本節で学習したため、そのプロパティを考えてみよう。順序内の各タスクは、これまで省略してきた Process Type を持っている。ここでも詳細は述べないが、タスクが単純な処理時間に限定されるものではないことに言及するだけでも価値がある。Process Name を選択すると、任意の長さのプロセスを実行できる。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）に送り、完了するのを待つ。Sequence Dependent Setup では、Changeover Logic 要素で指定されたデータに基づいて、処理時間を自動的に計算する。

タスクシーケンスの最後の注意点は、1つのステーションで多くの、数百に及ぶようなタスクが存在する場合についてである。たとえば、航空機の組立において、航空機は地点 1 で数日にわたって数十人の作業員により数百のタスクが行われ、それから地点 2 に移動して同様の作業が続くだろう。上記のようにタスクシーケンスを手入力するのは、非常に面倒である。このようなモデルは 7.7 節で述べたようなデータ駆動型である。Server プロパティの Other Task Sequence Options に、データ駆動型のタスクシーケンスをサポートするオプションがある。

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、複数の作業者が働く直列 4 タスクを解説する。
- SimBit : TaskSequenceAndWorker は、動き回る作業者に同期するタスクを解説する。
- SimBit : ServerUsingTaskSequenceAlternativeMethodsForDefiningTaskPrecedence は、タスクの先行関係を定義する別の方法を解説する。
- SimBit : ServersUsingTaskSequenceWithDataTablesJobShop は、データテーブルの使用方法を解説する。
- 例題 : HealthCareClinic は、すべてのタスクに対するデータを関係データテーブルで指定する方法を解説する。

10.4 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連絡のために発火するメッセージである。イベントは、重要な事柄が発生したことを他のオブジェクトに知らせたり、カスタムロジックを組み込むために有用である。それでは、より深く、このトピックを見ていこう。

Simioでイベントを生成するには複数の方法があるが、実際のところ、多くのイベントは自動的に定義され、遂行される。たとえば、エンティティがノードに入ったり、出たりするたびに、イベントが生成される。ほとんどのモデルは大部分のオブジェクトについて入口および出口のノードを持っているため、非常に簡単に、重要な状態変化を検知して、その発生を他に伝えることができる。同様に、Standard Library オブジェクトの多くは、選択したイベントに応答することができる。たとえば、ジャストインタイム方式によるシステム内へのパレットの到着を表現したい場合、Source の Arrival Mode を用いて、パレットが指定したノードを離れるときにだけ、オブジェクトを生成することができる。詳しくは、SimBit の RegeneratingCombiner を参照してほしい。

Standard Library オブジェクトによって行われる自動イベントは多くある：

- エンティティが移動を開始あるいは終了するとき。破棄されたとき。
- リソースが割り当て、あるいは解放されたとき。容量が変化したとき。故障あるいは修理完了したとき。
- サーバが容量を変化したとき。故障あるいは修理完了したとき。
- 作業者あるいは車両がノードに進入あるいは離れるとき。搬送を開始あるいは終了するとき。割り当てあるいは解放されたとき。容量が変化したとき。故障あるいは修理完了したとき。

同様に、Standard Library オブジェクトのアクションの多くは、イベントによって遂行される：

- ソースは、エンティティの生成を遂行する（上述のとおり）。
- ソースは、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、車両は、故障を開始する。
- 車両あるいは作業者は、休止時間を終了する。

これらを組み合わせることにより、Standard Library オブジェクトのプロパティや、それらが生成する自動イベントを用いることによって、多くの動的な意思決定を行えることに気づかれるだろう。しかし、これを紹介する前に、もう少し深く学んでおこう。これらすべての自動イベントに加え、独自のイベントを定義し、要素やステップを用いた非常に柔軟な方法で、それらを遂行することができる。要素やステップは、Standard Library と同様の方法でイベントを遂行し、応答するが、より大きな柔軟性と制御様式を有している。そして、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。それでは、独自のイベントを遂行する、つまり「発火する」ためのいくつかの方法を見ていこう：

- Station 要素：離散的な品目を表すエンティティを保持するため、容量制限のある場所を定義するために使用される。Entered、Exited、CapacityChanged イベントを生成する。
- Timer 要素：指定された IntervalType に従って、イベント系列を発火する。タイマーが満たされたたびに Event という名称のイベントを生成する。タイマーは、今後のタイマーイベントを発火する、あるいはタイマーをリセットするために、外部のイベントに応答することもできる。
- Monitor 要素：離散的な値の変化、あるいは指定した状態変数または状態変数の集合における閾値の交差を検知するために用いられる。モニタが監視対象の状態変数の変化を検知するたびに 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用いられる。

上述のとおり、Standard Library オブジェクトの多くは、イベントを受け取ったときに自らを動作させるプロパティを持っている。ただし、定義済みの応答に限定されず、独自の反応を定義す

ることもできる。これを行うには、プロセス内で Wait ステップを用いるか、Process 自身を発火させるか、2 つの方法がある。

これまで利用してきたすべてのプロセスは、Simio により自動的に開始、つまり発火されるか(たとえば、OnRunInitialized)、あるいはプロセス内で定義されたオブジェクト(たとえば、アドオンプロセス)によって開始されていた。しかし、プロセス開始を発火する別の非常に有用な方法として、特定のイベントを予約する Triggering Event Name プロパティを用いるものがある。名称が示すとおり、指定したイベントを受信すると、このプロセスが開始され、発火されたプロセスの遂行を始めるためにトークンが生成される(Triggering Event Condition を指定すると、プロセスを開始する前に条件が True(真)と評価されなければならない)。Subscribe ステップによって、同様の関係を(典型的には特定のエンティティと)動的に結ぶことができる。さらに、Unsubscribe ステップを用いて、その関係を動的に破棄することもできる。

イベントに応答するためのより詳細な方法は、Wait ステップを用いることである。この場合、トークンが Wait ステップを実行するまで、ステップは不活性状態になる(つまり、すべてのイベントを無視する)。そのトークンは、1つ以上の指定されたイベント⁶が発生するまで、Wait ステップで「待機」する。これは、一般に、ある条件が発生するまでモデル内の特定の地点にトークン(および関連するオブジェクト)を滞留させるために用いられる。たとえば、あるノード(おそらくコンベア)上に、別のコンベアで処理されている複数の優先順位1の部品が完成するまで、複数の優先順位2の部品が待機しているとしよう。最後の優先順位1の部品が、優先順位2の部品が待機しているイベント「AllDone」を発火する。

Wait および Fire には多くの簡単な用途があるが、興味深く一般的な用途として、エンティティが連携地点に到着した際に、2つのエンティティの連携つまり同期を行うことがある。たとえば、患者が画像診断(たとえば、MRI)を受けようと待っているが、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としよう。しかし、患者が MRI に進む準備を整えるまで、保険の事務処理は続行できないとする。この場合、それぞれのエンティティが他方を待つための PatientReady および PaperworkReady イベントを設ければよいと考えるのは早計である。しかし、どちらのエンティティが先に到着するかわからぬため、他方のエンティティを待っていないイベントを早まって発火してしまう問題が発生する。これには、お互いのオブジェクトの存在、つまり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、それから他方のエンティティを発火あるいは待機するという解決策を用いるとよいだろう。このソリューションについて、その詳細の検討は読者自身にお任せする。

本節の最後の注意点は、イベントによるロジックの代替案を考察することである。イベント基準のアプローチは一般に非常に効率的であるが、ときにさらなる柔軟性を要求する:

- Scan ステップは、指定した条件が真になるまで、処理トークンをステップで保持するために用いられる。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、トークンは遅滞なくステップを離脱することが許可される。そうでなければ、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することは大きな柔軟性をもたらすが、2つの注意事項がある。条件は時間進行時にのみ判定されるため、瞬時(ゼロ時間)の条件を見逃す可能性があり、実際には、条件自身は時間進行時まで認識できない。2つ目の注意事項は、式の監視はイベント基準アプローチよりも遅くなる点である。この点は、大部分のモデルでは目立たないが、顕著な実行速度の遅延を引き起こすモデルもある。

⁶ Wait では単一のイベントのみを指定することが通例であるが、複数のイベントを指定することもでき、トークンが解放される前に、1つあるいはすべてのイベントが発生していることを選択するオプションがある。

- Search ステップは、本章のはじめに扱ったが、イベントによるロジックの仲間として用いられる。たとえば、潜在的に関心のあるイベントが発火されたとき、適切なアクションを決定するために、候補の集合を検索することができる。

10.5 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。ここで、さらなる可能性を追求するため、Simio がサポートしている Flow ライブラリと Extras ライブラリを扱いたい。そして、ユーザフォーラムの Shared Items に、他のユーザが提供する多くの興味深い項目があることを紹介する。

10.5.1 Flow ライブラリ

Standard Library のすべては主に、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、液体や液体のような動作は、連続的な、つまりフロー (Flow) 概念を用いることでより正確にモデル化できる。代表的な応用例は、医薬品、食品、飲料品、鉱業、石油化学製品、化学製品などである。実際には、パイプラインのネットワーク（あるいはコンベヤ）や、ミキサーとタンク（あるいは貯蔵パイル）の間、コンテナの充填など、定常的、断続的に原材料が流れれるような場合に有用である。

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 は包括的な機能を有している。また、多くの製品とは異なり、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してモデル化できる。このコンセプトは、エンティティの物理的な特性を見るとわかる。3 次元の物理的な大きさ（たとえば、**体積**）を持つことに加え、離散的なエンティティは**密度**も有する。離散的なものとしての表現（樽や山など）と、液体の量やその他の質量としての表現（100 リットルのオイル、100 トンの石炭など）の 2 つのエンティティの表現は、離散的なエンティティを、1 つのフロー要素として到着させたり、液体へ／からその体積を基に変換させたりできる。また、リンク上あるいはタンク内の原材料の流れを表現するために、エンティティの属性を割り付け、追跡することもできる。フローの各構成要素について、製品種類や目的地、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：液体もしくは指定したエンティティタイプの集団について、無限の流れの元を表現する。
- FlowSink：処理を完了した流れを破棄する。
- Tank：流れを保持するための体積あるいは重量による容量制限のある場所を表す。
- ContainerEntity：液体あるいは集団の量を表すフローエンティティを搬送する、移動可能な容器（たとえば、樽）。
- Filler：コンテナエンティティを入れる装置。
- Emptier：コンテナエンティティを取り出す装置。
- ItemToFlowConverter：離散的なものをフローエンティティに変換する。
- FlowToItemConverter：フローエンティティを離散的なものに変換する。
- FlowNode：別のオブジェクト（たとえば、Tank オブジェクト）あるいはリンクネットワークのフロー制御地点において、フローの流入・流出を調節する。
- FlowConnector：あるフローノード地点から別の地点への、直接のゼロ搬送距離の接続。
- Pipe：あるフローノード地点から別の地点への接続。フローの移動時間とパイプラインの体積に関心がある。

Flow ライブラリの詳細については、Simio ヘルプを参照してほしい。特に、このライブラリのさ

さまざまな側面を解説する SimBit の FlowConcepts にある 8 つのモデルを検討してほしい。

10.5.2 Extras ライブラリ

Extras ライブラリは、一般によく見られる構造物を追加するための補助として設計されている。このオブジェクトの多くは、他のオブジェクト群から構成される複合オブジェクトである。これにより、応用例のニーズを満たすように、複合オブジェクトのグラフィックスや挙動、プロパティを必要に応じて編集できる。

Extras ライブラリには、次のオブジェクトが含まれている：

- **Bay**：1 つ以上の橋形クレーン（天井クレーン）が移動できる四角形の領域を示す（たとえば、フロアの空間）。各ベイは、ブリッジの衝突を防ぐために、複数のゾーンに分割できる。
- **Crane**：橋形クレーン、あるいは天井クレーンを表す。これは、ブリッジ自身、ブリッジに沿って移動する運転室、垂直方向の移動を制御するリフト、「クレーン」エンドエフェクタ（フックや掴み装置）で構成される複合オブジェクトである。クレーンは車両に類似した特性（積載・積降、待機、初期ノード、遊休状態）と、全方向への加速度および速度を持つ。
- **Robot**：あるノードでエンティティをピックアップし、別のノードでドロップする関節ロボットを表す。これは、回転可能な固定土台、土台に連動して動く下腕、下腕に連動して動く上腕、上腕の先に取り付けられた手あるいはエンドエフェクタから構成される複合オブジェクトである。ロボットは車両に類似した特性（積載・積降、待機、初期ノード、遊休状態）と、全軸方向へのピッチおよび変化率を持つ。
- **Rack**：ラックのフレームを表す固定オブジェクトであるメインラックと、1 つ以上の対応する棚で構成される複合オブジェクトである。ラックの棚の数と各棚の容量（つまり、棚に保持できる最大エンティティ数）が、ラックの重要なプロパティである。
- **LiftTruck**：車両と対応する昇降装置からなる複合オブジェクトである。リフトトラックは、エンティティを棚に置いたり、棚から取り出したりするために、適切な棚の高さまで、対応する昇降装置を上下移動する。通常の車両のプロパティに加え、垂直方向の速度と昇降移動の距離を指定する 2 つのプロパティを持つ。
- **Elevator**：垂直方向に上下移動し、対応する Elevator Node でエンティティを乗降するランスポータである。
- **ElevatorNode**：各 Elevator Node は、指定したエレベーターの行き先を表し、対応するエレベータを参照する。

Support リボンを開き、Sample SimBitSolutions をクリックし、検索ボックスに Extras Library と入力すると、Extras Library オブジェクトの使用方法を説明する SimBits がいくつか見つかる。また、Extras Library の使用方法を説明したサンプルモデル（EngineRepairUsing ExtrasLibrary）もある。

10.5.3 Shared Items フォーラム

Shared Items ユーザフォーラムでは、オブジェクトやモデル、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、ユーザが投稿し交換できる場所である。このフォーラムへのリンクは、Support リボンにある。多くの便利なツールを見つけることができ、コンテンツは常に変化している。本項では、簡単に概要を述べる。

まず、いくつかのライブラリがある：

- **Cranes** ライブラリは、一般的な橋形クレーン（Extras ライブラリの Cranes オブジェクトに類似）だけでなく、クレーン運転室がクレーン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モデル化できる。

- Transportation ライブラリには、複数の車両のある鉄道や、液体の積荷を輸送する Tanker オブジェクトなどのオブジェクトがある。

また、このフォーラムには、次のような便利なオブジェクトもある：

- Vehicle with Driver：エンティティを輸送する移動可能なオブジェクトを運ぶ移動可能オブジェクト。
- Conveyor Transfer Device：エンティティの行き先を変更するため、3つのコンベヤを接続する。
- Combiner without Parent：「親」概念を利用せずに、エンティティを組み合わせる。
- その他、多数。

さらに、このフォーラムには、アドインやマクロなどの便利な機能も含まれている。以下に、その一部を挙げる：

- Import Objects from a Spreadsheet：スプレッドシートからモデル（オブジェクトやリンクなど）をインポートするアドイン。
- Import Experiment from CSV File
- Save Step：毎実行の直前に、モデルを自動的に保存する。
- Simio Quick Reference Card：新しいモデルを構築する際に、ユーザが必要とする重要なポイントのまとめ。

こここのコンテンツを活用し、また読者自身のコンテンツを投稿して、ユーザコミュニティに貢献してほしい。

10.6 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、統計的に正しい答えを得るために複数の反復実行を行い、最善の解を得るために複数のシナリオを実行する価値を十分に理解されているだろう。そして9章では、多くのシナリオから最善の代替案を見つけるために、OptQuest を活用する方法を述べた。残念ながら、これらすべてに共通している側面として、多くの反復実行を行う必要性がある。1回の反復実行の実行速度が1秒で、100回の反復実行が求められる場合、2分未満となり、それほど悪くはない。しかし、それぞれ9つの実験からなる500シナリオについて、少なくとも5回ずつ反復実行を行わなければならないとしたらどうだろう。1回の反復実行に2.5分かかるとすれば、900時間を超える時間が必要となる。実行を順次処理すると1か月以上も待つ必要のある実験計画は、おそらく受け入れられないだろう。この例は非常に極端であるが、大規模な実際のプロジェクトでは、多くの場合、同様の状況に遭遇するだろう。本節では、このような高い処理需要に対処する方法を解説する。

10.6.1 並列処理

多くのコンピュータは、少なくとも限られた並列処理の能力を持っており、Simio のすべてのバージョンはこの能力を活用できる。たとえば、デュアルスレッドのクアッドプロセッサを持っている場合、標準的な構成であれば、8つの並列「スレッド」を処理できる。Simio の用語でいえば、8つの反復実行を並列で実行できる。この機能を用いることにより、先の例では900時間を113時間まで削減できるが、それでも待つにはまだ長い。

もし16以上のプロセッサを持っていれば（本書執筆時点では普及していないが）、Simio は16

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 および RPS) では、ローカルネットワークの他のコンピュータとの分散実行 (Distribute Runs) をサポートしている (詳細は、Experiment Properties→Advanced Options→Distribute Runs を参照)。いずれかの能力を活用できる環境にある場合、16 の反復実行を並列処理すれば、先の例では約 60 時間まで短縮できる。まだ長い時間がかかるが、いくぶん妥当な時間になるだろう。

サーバファーム、つまりローカルネットワーク上の多数のコンピュータへのアクセス権を持っている場合、Simio は多数の反復実行を並列処理するためのライセンスを提供している。たとえば、さらに 100 個のプロセッサを実行するためのライセンスを持っていれば (プロセッサ数は合計 116 となり)、900 時間かかる問題を 8 時間未満で結果を得ることができる。

10.6.2 クラウド処理

前項を読まれて、「クアッドプロセッサのコンピュータを持っているけど、 $900 \div 4 = 225$ 時間も待てない」と思われているかもしれない。「クラウド」という用語を耳にされたことがあるだろう。私たちの多くは、意識しようとしていまいと、クラウド上のアプリケーションを利用している。Simio には、Simio Portal Edition という製品がある。残念ながら、商用に販売されている製品で、本書執筆時点ではアカデミックバージョンは存在しない。Simio Portal によって、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利用して、実験を設定し実行することができる。実用上の制限はあるが理論的には、前述の問題において必要な反復実行数を、1 回の反復実行に要する時間 (つまり 2.5 分) よりも短い時間で完了できる。ただし、現実世界の実際上の問題により、すべての結果を生成するには、おそらく 1 時間程度の時間がかかるかもしれない。しかし、900 時間の処理時間に比べれば、十分に速い実行速度だろう。

10.7 AI とニューラルネットワーク

人工知能 (Artificial Intelligence ; AI) とは、コンピュータ技術を用いて、パターン認識、問題解決、学習などの領域で人間のような思考を可能にすることを指す広いテーマである。AI は、視覚認識、音声認識、意思決定、言語間の翻訳など、通常は人間の知性を必要とするタスクの実行によく使用される。AI は、データから学習し、再プログラムすることなく時間の経過とともに精度を向上させるアプリケーションの構築に重点を置いた**機械学習 (Machine Learning)** が一般的な分野である。本節では、AI とシミュレーションをどのように連携させることができるか、特に、モデルからより良い判断をするための学習を支援するために、AI をどのように使用することができるかを説明する。

ご存知のように、Simio はインテリジェントオブジェクトを用いるというコンセプトに基づいており、オブジェクトの知能は決定ロジックを使って作成される。この機能をエンティティオブジェクト (つまり、エージェント) に適用すると、エージェントベースモデリング (ABM) の基礎が形成される。エージェントがシステムおよび互いにインテリジェントに相互作用するようにすることで、システム自体に期待される振る舞いに関してハードコーディングすることを避けることができる。そして、システムの挙動は、自律的な知的オブジェクト (スタジアムの人々、マーケットの企業、あるいは伝染病の感染者など) の相互作用から生まれることになる。

モデル作成者は、Server や Node など、他の Simio オブジェクトに決定ロジックを埋め込むことができる。これらは、インテリジェントな動作を提供し、将来のパフォーマンスを予測するのに役立つジョブやリソースなどを選択するためによく用いられる。Simio には、多くの場所に洗練された組込みルールがある。たとえば、Transfer ノードにある標準ディスピッチルールのリストがある (図 10.12 参照)。組込みルールに加え、ユーザはプロセスロジックによるカスタムルールを作成することができる。また、上級ユーザはアドイン構造を使用して.NET で新しいルールを作成し、Simio に追加することもできる。このようなルールベースモデルは、ルールベース AI の一種である。

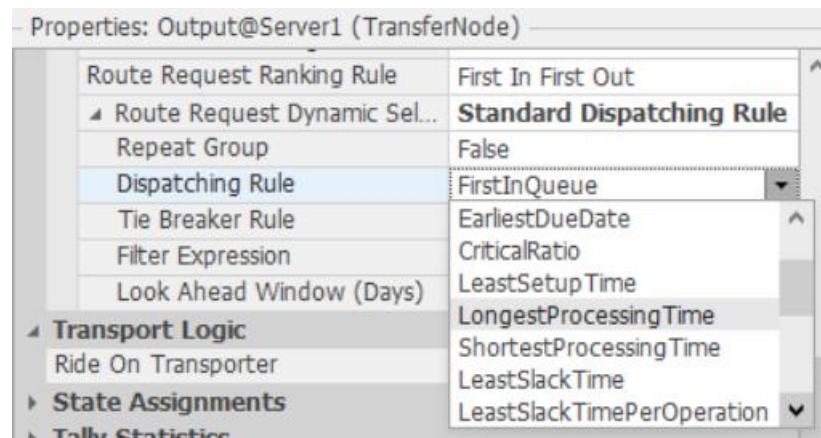


図 10.12 Transfer ノードにおけるディスパッチングルールの例

ルールのカスタムロジックを定義するのは難しい。たとえば、ある工場で 2 つのラインを選択する場合、各ラインは十数台のワークステーションから構成されており、最も早く仕事を完了できる（すなわち、生産スパンが最も小さい）ラインを選択したいと考える。生産スパンに影響を与える要因としては、下流のワークステーションの状況があり、各ワークステーションの作業量や作業の種類が完成時間に影響を与える。各ラインの下流のワークステーションには、段取り時間や仕掛け作業があり、どのラインが最も早くジョブを完了できるかの判断に影響を与える。これらすべての要因を考慮し、最適なラインを選択するルールを構築することは困難である。そこで、この状況を支援する AI の構成要素を次に議論することにしよう。

10.7.1 ニューラルネットワーク

ニューラルネットワーク（Neural Network。または人工ニューラルネットワーク、ANN）は、入力と、入力から生成される出力を持つ AI のコンポーネントである。ニューラルネットワークを使用すると、複雑なロジックを構築するプロセスをすべて回避し、シミュレーションモデルによってロジックを自動的に作成することができる。モデルにニューラルネットワークを追加して判断してもらい、シミュレーションデータを使ってニューラルネットワークを訓練することで、時間の経過とともに正しい判断ができるようになる。そのため、複雑なロジックをモデルに組み込む必要がないだけでなく、自己学習によって時間とともに改善されるルールを持つことができる。ニューラルネットワークをシミュレーションに取り入れることは、複雑な意思決定ロジックを持つアプリケーションのゲームエンジンとなる可能性を秘めている。そこで、ニューラルネットワークと、それが複雑な意思決定のために Simio にどのように組み込まれているかを検証してみよう。

それでは、ニューラルネットワークをもう少し深く掘り下げてみよう。ここでは標準的なフィードフォワード・ニューラルネットワークを説明するが、他にも特殊な用途に対応したバリエーションがある。簡単に言うと、ニューラルネットワークは一連の数値入力を 1 つ以上の出力にマッピングする。最も基本的な形はパーセプトロン（Perceptron。ニューロンともいう）と呼ばれる 1 つのノードと、入力の集合 X 、各入力に割り当てられた重み w 、ノードに割り当てられたバイアス（Bias）、ノード上の活性化関数 $F()$ 、そして入力、重み、バイアス、活性化関数から計算される单一の出力 Y がある。出力は、パーセプトロンが現在の入力から予測した値である。図 10.13 は 3 つの入力を持つパーセプトロンである。

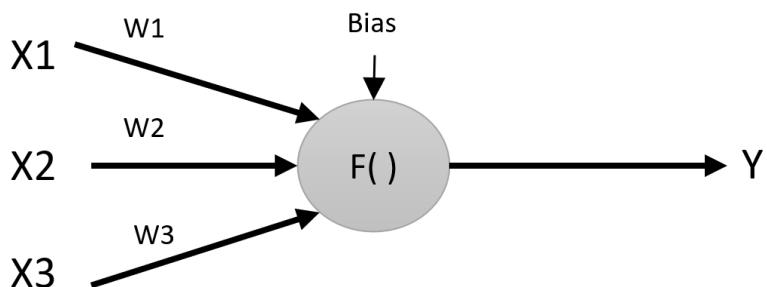


図 10.13 3つの入力を持つパーセプトロン

予測値 Y は、 $F(w_1 * X_1 + w_2 * X_2 + w_3 * x_3 + Bias)$ として計算される。ここで F は任意の関数（たとえば平方根）をとれる。つまり、重み付けされた入力を合計し、バイアスを加え、活性化関数を適用して、予測値を得るのである。

このモデルを適用するには、パーセプトロンが正しい予測値を生成するように重みとバイアスの値を決定する必要がある。たとえば、パーセプトロンを使って、土地の広さ、部屋の数、建築年に基づいて家の販売価格を予測する場合、実際の販売価格に近い販売価格の予測値を生成するように、重みとバイアスを選択することになる。これはモデルの学習と呼ばれるプロセスで、指定された入力と販売価格の履歴データを取り込み、予測販売価格と実際の販売価格の誤差を最小化するような重みとバイアスを計算する学習アルゴリズムを適用する。したがって、予測エンジンの品質は、学習データの品質に大きく影響される。

パーセプトロンの学習は、線形回帰モデルを適合させる際の係数の計算と似たような概念であることに留意してほしい。実際、ニューラルネットワークのモデルを学習することをモデルの適合と呼び、値を予測することを回帰と呼ぶことがある。しかし、ニューラルネットワークは非線形性を捉えることができ、多層推論が可能なため、単純な線形回帰モデルよりもはるかに強力な予測エンジンとなる。ニューラルネットワークは、パーセプトロンの基本概念を基に、複数のニューロンを組み合わせて、図 10.14 に示すような入力層 (Input Layer)、1 つ以上の隠れ層 (Hidden Layer。中間層)、出力層 (Output Layer) からなる層状のネットワークを形成する。

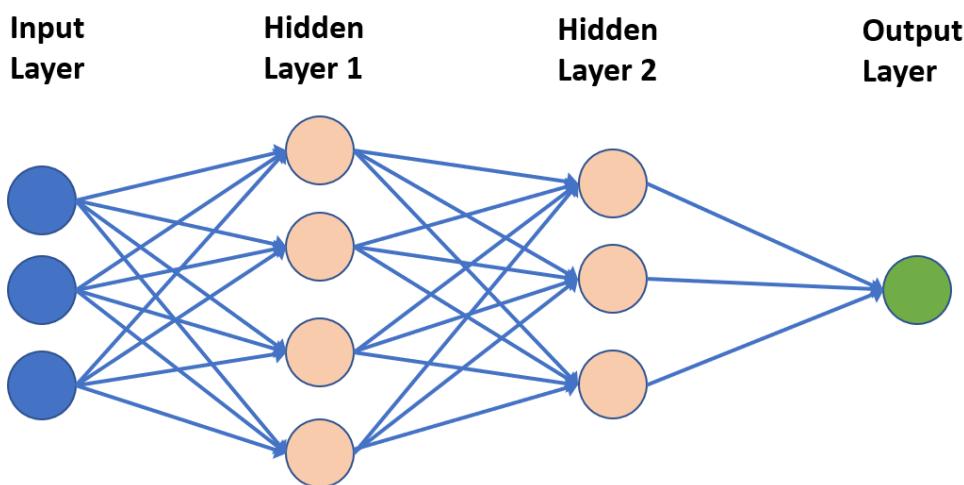


図 10.14 ニューラルネットワークの例

入力層のニューロンは（その名が示すように）予測エンジンの入力データを定義する。隠れ層は通常、ネットワーク内のほとんどのニューロンで構成され、目的の予測値を得るために核となる非線形データ操作を行う。少なくとも 2 つの隠れ層を持つネットワークは、ディープニューラルネットワーク (Deep Neural Network ; DNN) と呼ばれる。

出力層は最終的な予測値を提供する。ニューラルネットワークを回帰に使う（つまり、単一の値を予測する）場合、出力ノードは 1 つだけである。しかし、多くの場合、出力ニューロンはそれぞれオブジェクト（たとえば、犬、猫、馬、牛に分類される画像）を表し、各ニューロンの予測値は、それがそのオブジェクトとなる確率となる。

DNN の応用として、最近は多くの成功例がある。たとえば、Apple 社の Siri による音声認識・応答や、Tesla 社の自動運転車における画像処理などが挙げられる。しかし、DNN の応用において大きな課題の 1 つは、DNN を学習させるためのじゅうぶんな高品質の学習データを確保することである。そこで、シミュレーションと AI の融合により、DNN のための合成学習データ (synthetic training data) を無制限に生成することが可能になる。

ここで前の例に戻り、2 つの生産ラインの候補を選択するために DNN を使用し、シミュレーションで生成されたデータに基づいて DNN を訓練する問題について説明する。この例では、DNN を使って各生産ライン候補の生産スパンを予測し、生産スパンが最も小さいラインを選択する。この選択を行う際に、生産スパンの予測に使用した DNN の入力値を記録する。このモデルでは、エンティティは選択された生産ラインを下り続け、ラインの終点で実際の生産スパンを集計する。DNN の入力値とモデルに基づく実際の値は、DNN の学習データとして保存される。

モデルによって生成された合成学習データは、シミュレーションモデル内の意思決定ロジックのために自己学習した DNN の概念を解き明かす鍵になる。しかし、DNN の入力値を保存し、記録された実際の値と照合し、その値を学習データとして書き出すという作業は面倒だ。そのため、最近では Simio にニューラルネットワークを組み込む際、学習データの記録と Simio モデルの自己学習への活用を自動化することが注目されている。

この文章を書いている時点では、上記の手法を使って今すぐモデルに AI 構成要素を追加することはできるが、そのためにはかなりの知識が必要である。Simio では、2021 年後半に、これにより簡単に行えるような機能が追加されることが期待される。

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の項目から選択する強力な仕組みを解説した。それから、退去と放棄について扱い、エンティティを処理中のステーションから簡単にバイパスする、あるいは取り出す方法を述べた。次に、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において、並列あるいは直列のサブタスクを非常に柔軟に実行できることを示した。イベントによる決定ロジックでは、処理ロジックを制御するための、これまでとは異なる方法を提供した。

また、Flow および Extras ライブラリについて概観し、Standard Library だけでは簡単にモデル化できない多くの領域をカバーするために、Simio から提供されている機能を紹介した。そして、Shared Items フォーラムで、便利なオブジェクトやツールがユーザによって提供されていることを述べた。実験を行う際の注意点を述べた後、AI およびニューラルネットワークとシミュレーションを組み合わせるという新しい分野について簡単に説明した。11 章では、独自のカスタムルール、オブジェクトおよびライブラリを作成する方法について解説する。

10.9 問題

- 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。窓口の時間およびバッファ容量は既定値のまます。WIP を一定値の 8 に制限したい。8 つのエンティティがシステム内のあるかは問わず、常に合計を 8 に保ちたい。システム内のエンティティ数の平均および最大数を示し、期待されるふるまいを述べよ。
- 図 10.15 のシステムを考える。エンティティは、平均 2.5 分の指數分布に従う到着時間間隔で、各ソースから到着する。BasicNode1 への移動時間は 1 ないし 3 分の一様分布に従い、BasicNode2 への移動時間は平均 1 分の指數分布に従う。それぞれの BasicNode から

Combinerまでは、正確に1分の移動時間が必要である。エンティティは Combiner に完全に同期して到着し、Combiner で待つことはない（Combiner では、それぞれの親と1つのメンバーが組み合わされる）。先行するエンティティが後続のエンティティを待つように、イベントを用いて BasicNode にプロセスロジックを追加しなさい。

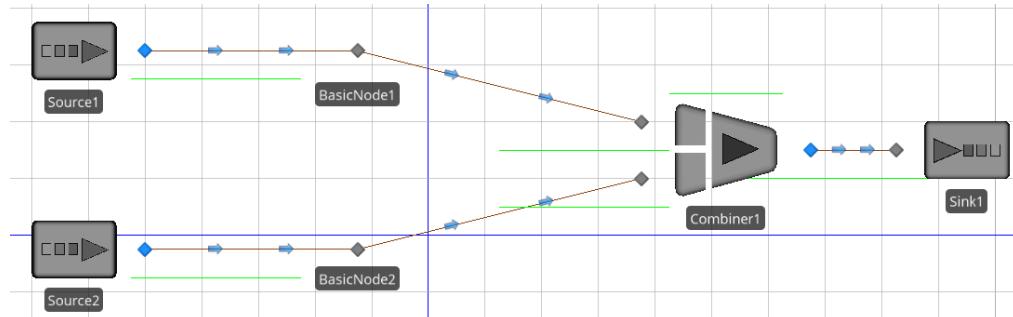


図 10.15 問題 2－同期システム

3. 問題 2 を拡張し、すべてのエンティティがオーダ番号を持ち、各ソースは 1 から始まるオーダ番号の順番でエンティティを生成する。この問題では、オーダ番号に基づいて組み合わせたいため、BasicNode で同一のオーダ番号のエンティティを同期させ、一緒に一組として解放する必要がある。
4. 図 10.16 の生産システムを考える。部品は Receiving における 8 分間隔で到着し、天井クレーンによって Grinding に搬送され、そこでの研磨工程に 3 ないし 7 分かかる。Grinding の後、80%はクレーンで Shipping に搬送され、20%はクレーンで Rework に送られる。Rework を終えた部品は、常にクレーンによって Grinding に戻される。通常、1 台の天井クレーンが遊休状態で Receiving の近くに待機している。クレーンの積載・積降時間はそれぞれ 1 分で、横方向の移動速度は秒速 1m である。1 日 24 時間のモデルを 5 日間実行し、クレーンおよび機械の期待利用率を求めなさい。

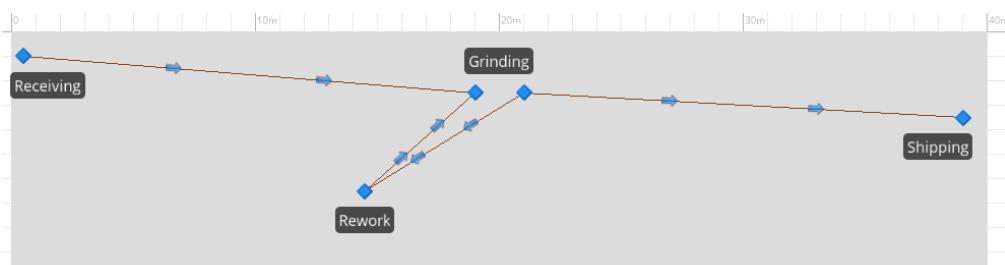


図 10.16 問題 4：クレーンシステム

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るとすると、合理的な注文の受入率はいくらになるか？その際のスループットの向上に関する変化にコメントしなさい。
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さな診療所をモデル化する。患者は、1日8時間を通して、1時間当たり約30名の割合で到着する。それぞれの定員が4名である2基のエレベータにより、患者はフロア間を移動する。患者は医師のうちの1名から3ないし7分の診察を受け、1階の出口に戻る。Extras ライブラリの Elevator オブジェクトを用いて、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

本書をここまで読まれた読者なら、Simio Standardライブラリだけを利用できるSimio Personal Editionを活用すれば、様々な課題を解決可能になると気づかれているだろう。また、上述のバージョンとSimio Design Editionで利用可能なアドオンプロセスを活用することで、ほとんどの問題を解決できるだろう。しかし、場合によっては、提供されている機能以上の能力が必要とされることもある。たとえば：

- ・ アドオンプロセスを用いるよりも、カスタマイズされたオブジェクトを必要とする場合。
- ・ 同じオブジェクトの拡張機能を繰り返し利用したり、独自の再利用可能なオブジェクトが必要な場合。
- ・ 状況に応じてシミュレーションの活用を支援できる経験のあるモデル作成者や、独自のカスタムツールの恩恵を受ける初心者が、同じチームとして働く場合。
- ・ 販売を目的として、商用ライブラリを構築したい場合。

Simioのカスタムオブジェクトおよびライブラリを作成することにより、上記のすべてを達成できる。これまでの9つの章を読み終え、かつ例題も理解できたのであれば、カスタムオブジェクトを定義するために、知っておくべきことの大部分を実際には身につけていることになる。本章の最初の部分では、これまで理解された概念と、未解説の概念を組合せて述べることにする。

カスタムオブジェクトはかなり柔軟かつ強力であるが、意欲的なモデル作成者は、現有オブジェクトを超えた活用を目指そうとするだろう。より経験のあるユーザに対する付加機能についても、Simioは用意している。プログラミングに関する知識（たとえば、Visual Basic、C++、C#、あるいは.NET言語類）を有しているのであれば、下記のことができる：

- ・ 設計時に活用する独自アドオンを作成する。たとえば、外部データからモデルを自動的に作成するなど。
- ・ Simio 内ですでに利用されている選択ルールに、新しい動的選択ルールを追加する。たとえば、Vehicle、Worker、あるいはServerに対して、包括的な作業選択ルールを作成するなど。
- ・ テーブルのインポートおよびエクスポートに対するカスタムルーチンを追加する。
- ・ 実験の設計（たとえば、シナリオの設定・分析）あるいは実行（たとえば、最適化）に関するカスタムアルゴリズムを追加する。
- ・ Simio エンジンの機能を拡張するために、独自の Step および Element を追加する。
- ・ 特殊な応用に適応し、Simio のリボンをカスタマイズする。

本章を読み終われば、これらの機能を概観できるだろう。本章の内容の多くは、「Introduction to Simio」(Pegden and Sturrock 2013) から許諾を得て引用している（この電子書籍は、Simio ソフトに付属しているものである）。

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、モデルを構築するために利用できるライブラリの一種に過ぎない。目的に合うような自分なりのオブジェクトライブラリを構築することもできるし、企業内で共有するためのライブラリの構築もできる。作成されたすべてのモデルをオブジェクト定義として扱えることは、Simioの基本的な考え方の一つである。モデルは、オブジェクトの基本的なふるまいを定義するためのものである。作られたSimioモデルをパッケージ化し、他のモデルの

一部として利用してもらうことは、容易にできる。また、1つのプロジェクト内でも、複数のサブモデルを構築できる。さらに、構築されたサブモデルを、自分のモデルの一部として利用することもできる。

ここで、議論したい事柄は、4章、5章、および9章で言及したSimioプロセスと関連しており、プロセスを拡張した内容である。それらの章では、オブジェクト定義の核心部分を変更せずに、いかにプロセスを利用し、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かについて学んだ。次節では、(たとえば、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ように)オブジェクト定義の核心部分の設定を変更したり、拡張させたり、あるいはゼロから新規オブジェクトを構築したりすることに重点を置くことにする。

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を参照)。プロパティ、状態変数、イベント、外部ビューのすべては、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およびProcessesウィンドウを組み合わせて定義されることになる。

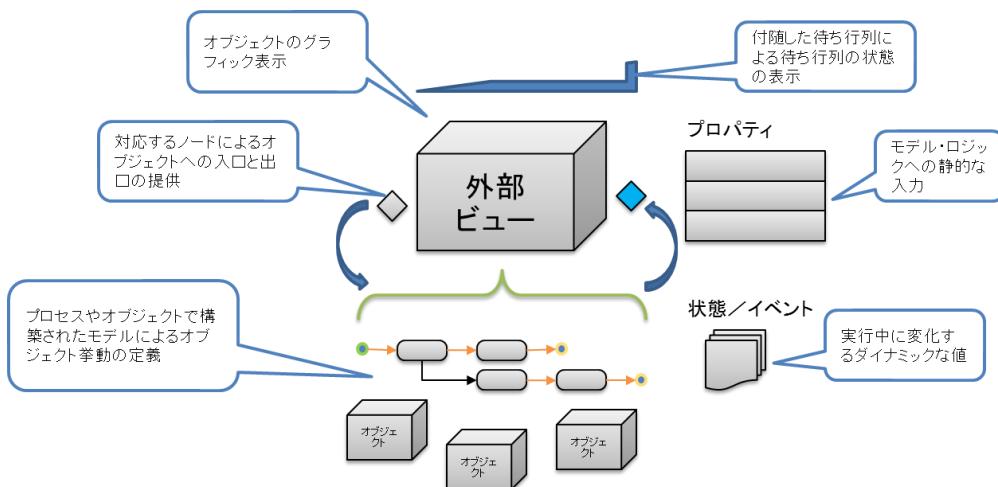


図11.1 オブジェクトの解剖図

ここで、まずモデルプロパティを利用し、モデルで参照されたり、実験で用いら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に対する入力を定義するためにも利用される。モデルプロパティは、モデルロジックへの入力を提供し、モデルの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ために用いられる。シミュレーション実行中に変更可能なあらゆる項目は、状態変数としてオブジェクト内で表現される。イベントは、エンティティのステーションへの進入や、ノードからの離脱など、瞬間に起こる論理事象である。オブジェクトは、自身に属するイベントを定義できる。また、自身に属するイベントを始動させると同時に、他のオブジェクトへイベントが起動した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す。外部ビューは、モデルにオブジェクトを設置する際に目にするものである。オブジェクトのロジックは、各イベントに対して、オブジェクトがどのように反応するかを定義するモデルである。ロジックはオブジェクトにふるまいを与える。ロジックは、既存のオブジェクトを利用することで、階層的に構築することができるし、またグラフィカルプロセスのフローから定義することもできる。

11.1.1 モデルのロジック

固定オブジェクトに関するロジックは、Facilityおよび/またはProcessesモデルによって定義され、ノードシンボルのInput Logic Typeによって決められる。特に、エンティティやトランスポータ、ノード、およびリンクの場合には、ノードシンボルが存在しないため、主にProcessesロジ

ックによって構築される。

システムに新しいモデルを構築する場合、ある意味では、新しいオブジェクト定義のロジック部分を表現したことに等しいだろう。モデルに入力するためのプロパティ、オブジェクトをグラフィカルに表現する外部ビュー、およびエンティティがモデルに入り出するするために利用するノードを追加すれば、どんなモデルでも、簡単にオブジェクト定義として利用できるものに変更できる。

オブジェクトに関するモデルロジックの定義には、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、階層的なモデルを構築することである。 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、アドオンプロセスの活用と組み合わせることもできる。このアプローチは、たとえば機械 2 台、作業員 1 人、および治具からなるワークセンターのような、高レベルの施設要素を構築する際に利用される。
- ・ 2 つ目のアプローチは、もっとも柔軟性を持っており、Processes モデルを利用し、ゼロからオブジェクト定義のふるまいを構築することである。このアプローチは、Standard ライブラリのオブジェクトを構築する際に用いられている。
- ・ 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あるいは拡張することである。希望するオブジェクトと似たようなふるまいを持つ既存オブジェクトが存在し、かつプロパティ名や概要、およびふるまいを微調整すれば、新しいオブジェクトの必要性を満たすことができる場合に、このアプローチが用いられることが多い。

11.1.2 外部ビュー

外部ビューは、固定あるいは動的なオブジェクトのインスタンスに関するグラフィカル表現である。また、ノードとリンクには、外部ビューを使わない。図11.2に示すように、外部ビューは、モデルのDefinitionsウィンドウにあるExternalパネルをクリックすることによって定義される。

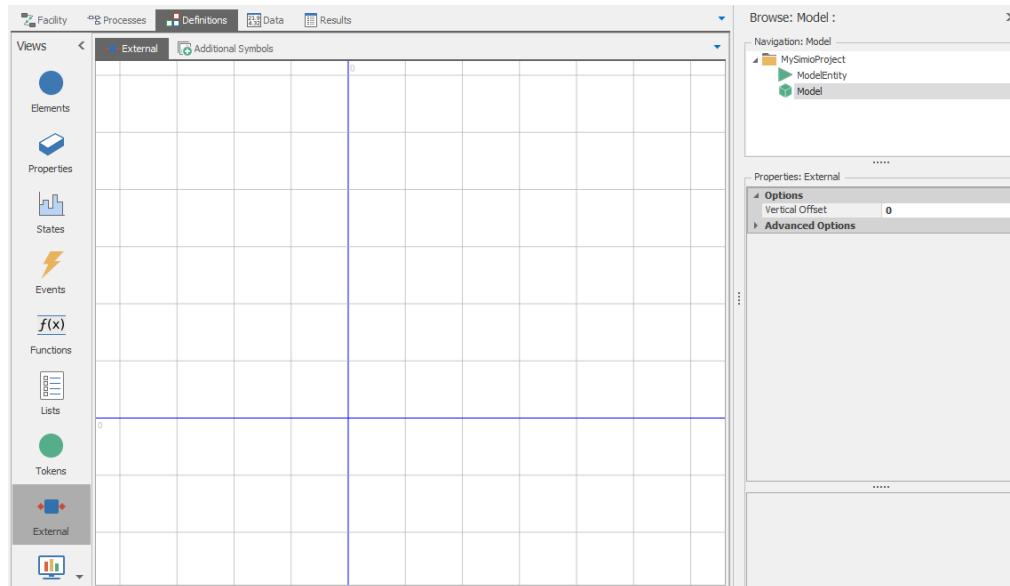


図11.2 外部ビュー

外部ビューは、オブジェクトのインスタンスをモデルに追加する際に目にするものである。外部ビューのグラフィックスは、シンボルライブラリから、あるいはTrimble 3D Warehouseからダウンロードされるシンボルから構成される。また、静的なグラフィックスについては、Drawing リボンの描画ツールを使用して描くことができる。さらに、外部ビューには、Animation ウィンドウからビューに設置される、アニメーション要素を付属させることもできる。これらのアニメーション

要素には、待ち行列、状態ラベル、プロット、円グラフ、四角あるいは円形のゲージ、およびボタンが含まれる。エンティティのような動的オブジェクトの場合、そのアニメートされるオブジェクトは、システムの内でそれぞれの動的オブジェクトに伴って移動することに留意してほしい。たとえば、エンティティは自身の状態変数の1つの値を表示する状態ラベルを携行でき、またユーザがクリックすると、何らかのアクションを起こすボタンを携行できる（Model 9-2に詳しい）。

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、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、入力および出力用のノードオブジェクトが必ず関連づけられている。このような関連するオブジェクトの特性に関しても、Drawingリボンを用いて、外部ビューにノードシンボルを設置することにより、そのオブジェクトの外部ビューとして定義する。これらのノードシンボルはノードオブジェクトではなく、元になるオブジェクトが設置される際に作成される、関連づけられたノードオブジェクトの位置を示すシンボルであることに留意してほしい。

外部ビューでノードシンボルを配置する場合、そのプロパティも必ず定義される。また、ノードクラスから、作成されるノードのタイプが指定される。そこでは、ドロップリストより、利用可能なノードの定義を一覧できる。この一覧には、Node（空のモデル）、BasicNode（基本的な接点）、TransferNode（目的地を設定し、エンティティが利用するトランスポータを選択することを支援する接点）や、ライブラリから読み込まれたり、プロジェクトで定義された他のノード定義が含まれる。Standardライブラリの場合には、BasicNodeクラスは一般的に入力ノードとして、またTransferNodeは出力ノードとして利用されることが多い。Input Logic Typeは、このオブジェクトに入ろうとする到着エンティティが、オブジェクトによってどのように処理されるかについて指定する。Noneオプションは、エンティティがオブジェクトに入る際に許可を求める指定する。ProcessStationオプションは、到着エンティティが指定されたエンティティに進入する際に、プロセスロジックを開始するために用いられるステーション進入イベントを始動することを設定する。この選択肢は、プロセスモデルを利用してオブジェクトロジックを定義するときに用いる。また、FacilityNodeオプションは、到着するエンティティを、オブジェクトのFacilityモデル内部のノードへと送ることを指定する。この選択肢は、Facilityモデルによってオブジェクトロジックを定義するときに用いられる。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を生成するために用いられる。NodeSymbolName@ObjectNameという書式を用いて、このオブジェクトがインスタンス化される際に、このシンボルの場所で生成される。たとえば、シンボル名がInput、オブジェクト名がLatheの場合、自動的に生成される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と命名され、出力ノードシンボルはOutputと命名されている。

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とは、構築する新しいオブジェクトは、そのプロパティ、状態変数、イベント、およびロジックを、既存のオブジェクト定義から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、最初に元のオブジェクトと同じプロパティとふるまいを持ち、元のオブジェクト定義を更新すれば、サブクラス化されたオブジェクトは新しいふるまいを継承することになる。しかし、サブクラス化を行った後、継承されたプロパティを再命名したり、非表示させたりすることが可能であり、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、プロセスを追加しふるまいを拡張させたりすることが可能である。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考えられる。この場合、通常のServerプロパティ（たとえば、Capacity Type）を非表示させたり、プロパティを再命名（たとえば、Process TimeをTreatment Timeに変更）することができる。また、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の故障パターンを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をクリックしてから、Modelsパネルを選択する。Editリボン（図11.3）によって、新しい空のモデルを追加したり、選択されたモデルのサブクラスを作成したり、ライブラリのオブジェクト定義からサブクラスを作成したりすることができる。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することができる。

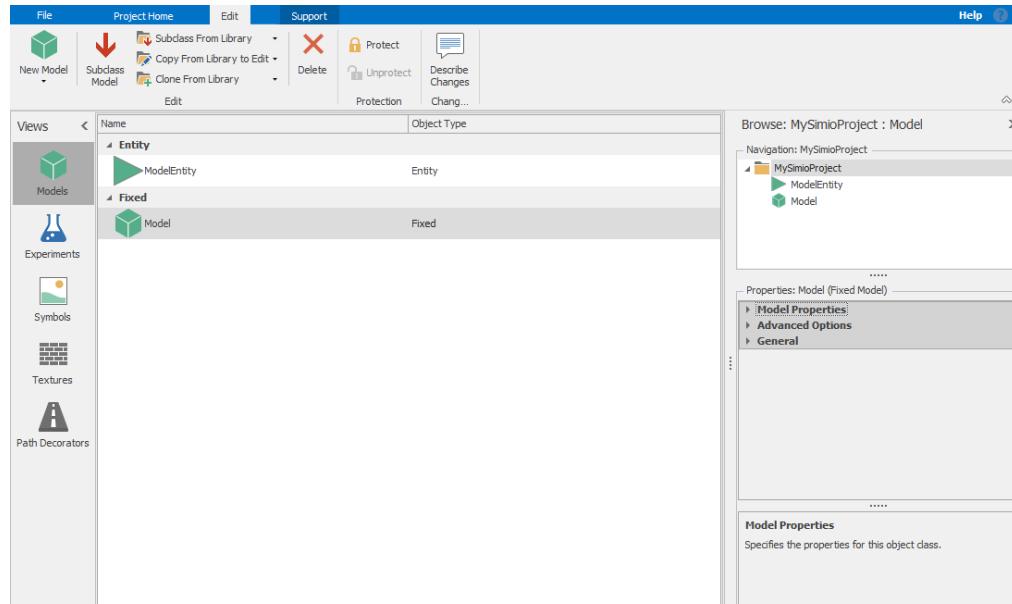


図11.3 Editリボン

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には、次の3つの方法がある：

- **SubClass From Library**：上述のように、ライブラリオブジェクトから、オブジェクトをサブクラス化（派生）する。この場合、既存のオブジェクト定義からプロセスロジックを継承しているため、元のオブジェクトを変更すれば、サブクラス化されたオブジェクトのふるまいも変更されることになる。
- **Copy From Library for Edit**：サブクラス化せずに、オブジェクトをコピーする。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コピーであり、元のオブジェクトとの継承関係を持たない。したがって、元のオブジェクト定義が変更された場合でも、新規作成されたオブジェクトは何も影響を受けないことになる。
- **Clone From Library**：元のオブジェクトと同一のオブジェクトを作成する（Simioからは、新規オブジェクトは元のオブジェクトと区別がつかない）。これは、モデルが異なるライブラリから編成されている場合の整理によく利用される。

また、オブジェクト定義を保護するために、パスワードを設定することもできる。この場合、オブジェクトの内部モデルを表示・変更する際に、パスワードの確認が要求される。

モデルプロパティは、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、つまりこのオブジェクトがサブクラス化された由来のオブジェクト）、モデルを表示するためのIcon、Author、Version NumberとDescriptionを含む。また、プロパティには、オブジェクトがリソースとして占有・解放される場合にはResource Objectが含まれ、オブジェクトがモデルとして実行されたり、他のモデル内のサブモデルとしてのみ利用される場合にはRunnable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れば、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ており、新しいメンバーを追加することも可能である。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ウィンドウから確認できる。継承されたメンバーおよび新規作成されたメンバーは、それぞれのカテゴリに配置され、別々に展開したり折りたたんだりできる。図11.4に示されたPropertiesパネルでは、ReworkTimeという名称の新しいプロパティだけが、Fixed（固定）であることが確認できる。このプロパティは、モデル内のオブジェクトおよびプロセスの両方から参照できることに留意してほしい。

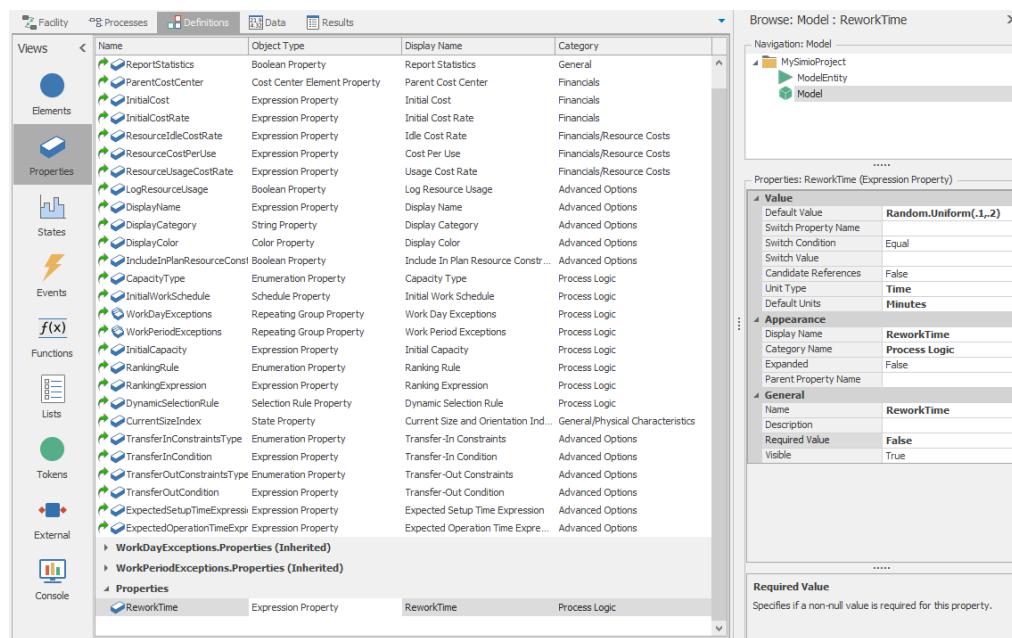


図11.4 DefinitionsウィンドウのPropertiesパネル

ReworkTimeの特性は、Propertiesウィンドウに示されている。Default Value、Unit Type/Default Units、Display Name、Description、Category Nameが含まれている。また、これらに関して、必須の設定項目であるのかや、ユーザに開示するものなのかを設定することもできる（それぞれRequired ValueとVisible）。さらに、Switch Property Name、Switch Condition、Switch Valueの設定も含まれる。これらは、ユーザが指定する他のプロパティの値に基づいて、プロパティの表示・非表示に用いることができる。たとえば、プロパティXの値が0より大きい場合だけ、プロパティYの値を表示させることができる。これにより、ユーザの入力に基づいて、Propertyウィンドウを動的に配置することが可能である。

継承されたプロパティの場合、Default Value、Display Name、Description、CategoryにVisibleフラグがあり、可視化するかどうかについて変更できる。したがって、プロパティを非表示せたり、それらの概観を変更したりすることができます。ただし、状態変数やイベントはモデルに追加することができるが、継承された状態変数やイベントの名称を変えたり、非表示にすることはできない。

11.2 Model 11-1：階層的オブジェクトの構築

この例では、2つのStandardライブラリのServerと1つのConnectorから成る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ことにする。1つ目のServerは、キャパシティが1で、アウトプットバッファを持たない。2つ目のServerは、1つ目のServerと同じでキャパシティは1であるが、インプットバッファを持たない。したがって、2つ目のServerが稼働中の場合、1つ目のServerをブロックすることになる。オブジェクトは、2つのServerのそれぞれの処理時間を指定する2つ

のプロパティを有する。また、オブジェクトは、各Serverのプロセス内にあるエンティティをアニメートすると同時に、各Serverのリソース状態に関する円グラフもアニメートする。

11.2.1 モデルロジック

階層的に（つまり、Facilityウィンドウの他のオブジェクトから）オブジェクトを構築するため、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ことから始める。

- 新しいプロジェクト（New Project）を開始する。Project Home リボンから新しい固定モデルを追加し、モデル名を TandemServer にする。
- TandemServer を選択し、アクティブモデルにする。そして、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、および Server2 の Input Buffer を 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。それぞれのプロパティ名を ProcessTimeOne および ProcessTimeTwo とする。また、Default Value、UnitType/Default Units、Display Name、Category Name の設定は、図 11.5 を参照すること。
- 継承されたプロパティを展開する。CapacityType、WorkSchedule、InitialCapacity という 3 つの継承プロパティは、TandemServer オブジェクトとの関連性が薄いので、図 11.6 で示すように、それぞれの General カテゴリの Visible 特性を False へ変更し、非表示させる。

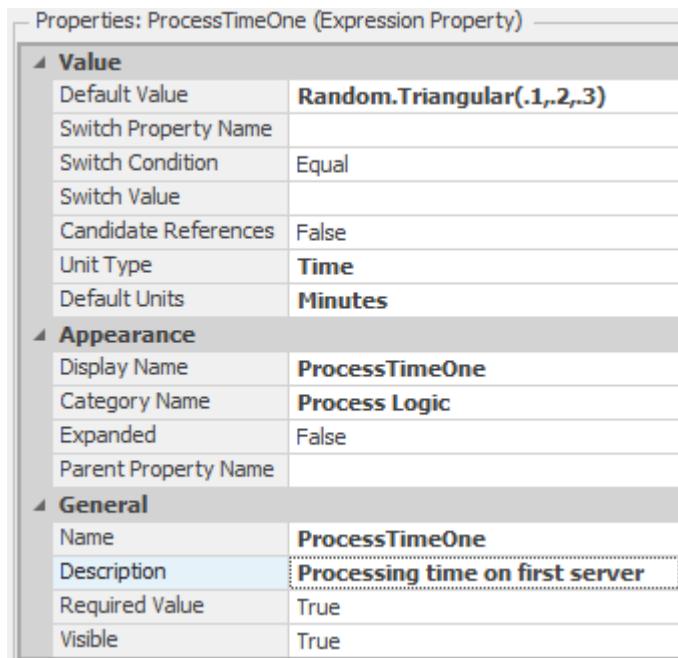


図 11.5 TandemServerのProcessTimeOneプロパティの設定

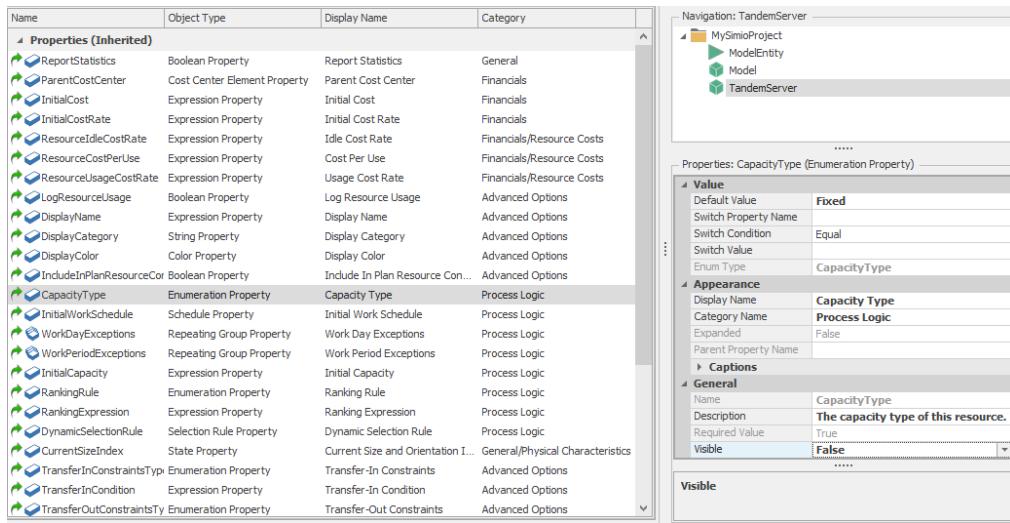


図11.6 繙承プロパティの非表示

11.2.2 外部ビュー

次に、外部ビュー（External View）を利用し、オブジェクトが配置されるときに、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ウィンドウにて定義されたオブジェクト、ノード、リンク、および状態変数アニメーションは、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見たいときに、特に非常に便利である。この例では、手動でユーザビューを作成して、細やかな制御を行いたい（そして、操作経験を積みたい）ので、デフォルトのユーザビューを隠すことから始める。

- Facility ウィンドウを選択する。外部ビューの自動表示を隠す場合、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。また、複数のアイテムがある場合、それぞれを別々に変更しなくとも、Ctrlキーとドラッグのマウス操作で、すべてのアイテムをハイライトさせてから、Externally Visible オプションを選べば、一度に解除することもできる。
- External パネルを選択し、リボンの Place Symbol ボタンを利用して、Server のシンボルが2つ続くように配置する。
- 1つ目の External Node をインプット側に置く。Node Class を BasicNode にし、Input Location Type を Node とする。さらに、Node を Input@Server1 とし、Name を Input と定義する。
- 2つ目の External Node をアウトプット側に置く。Node Class を TransferNode にし、Input Location Type を None (TandemServer のこちら側から進入しない) とし、Name を Output と定義する。
- Animation リボンにある Queue ボタンを利用して、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く。各待ち行列を配置したら、Appearance リボンの Alignment で None を選択する。それぞれの Queue States として、Server1.InputBuffer.Contents 、 Server1.Processing.Contents 、 Server2.Processing.Contents、Server2.OutputBuffer.Contents を指定する。
- Animation リボンにある Status Pie ボタンを使用し、縦5目盛×横6目盛の円グラフを2つ追加する。Data Type を ListState とし、それぞれの ListState は Server1.ResourceState と Server2.ResourceState にする。ここで画面は図11.7のようになっているだろう。

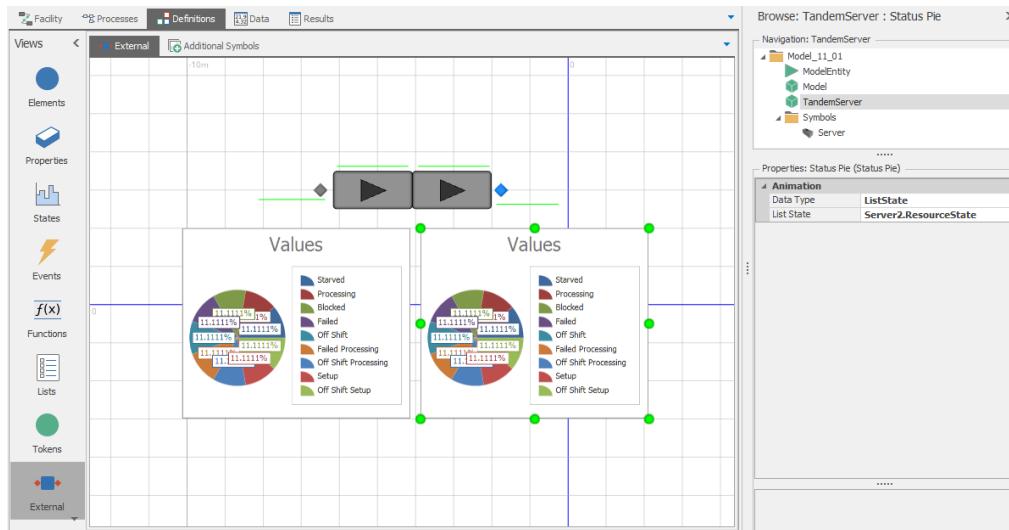


図11.7 TandemServerの外部ビューの定義

外部ビューにおいて、Inputノードシンボルによって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モデルでInput@Server1と命名されたノードへと送られることに注意してほしい。エンティティは、Server2の出力ノードに達するまで2つのServer内で処理されるため、そこでOutputと命名された外部ノードシンボルによって定義された関連ノードへと送り返したい。Simioに外部出力ノードであることを知らせるには、TandemServerのFacilityウィンドウに戻り、Server2の出力ノードを右クリックする。そして、プルダウンメニューからBind to New External Output Nodeを選び、External Node NameとしてOutputを指定する。これにより、TandemServerから離脱するエンティティが、Outputとして作成された関連出力ノードから送り返される。

次に、Processing Timeについて、Server1はProcessTimeOne、Server2はProcessTimeTwoとして設定する。それぞれのProcessing Timeプロパティを右クリックし、Set Referenced Propertyを選択し、以前に定義した関連プロパティの値を選択する。これで、これらの加工時間が、TandemServerに対して新たに作成されたプロパティとして定義された。

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。手順として、まずFacilityウィンドウでオブジェクトのロジックを定義した。次に、ユーザに見せたいプロパティとユーザから非表示にするプロパティを設定した。さらに、外部ビューを利用してことで、定義されたオブジェクトをFacilityウィンドウに配置するときの外観を定義した。

これで、新規のオブジェクト定義を利用する準備が整った。モデルをアクティブモデルにするため、メインモデル(Model)をクリックする。そして、Source、TandemServer(左下のProject Libraryから選択)、Sinkを1つずつ設置し、Pathでつなぐ。TandemServerをクリックすると、Propertyウィンドウにこれまで停止してきたカスタムプロパティが表示される。TandemServerオブジェクトを活用した簡単なモデルを図11.8に示す。

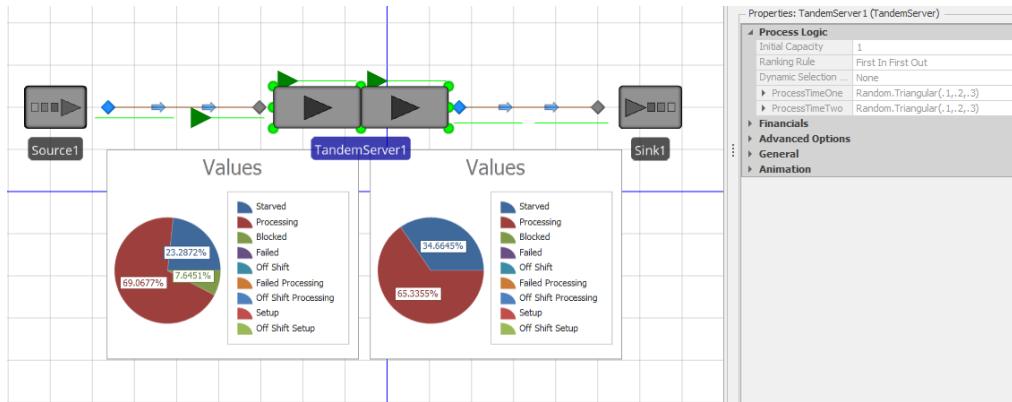


図11.8 実行中のTandemServer

11.3 Model 11-2：ベースオブジェクトの作成

Model 11-1では、Facilityモデルを利用して新しくオブジェクト定義を構築した。Model 11-2では、プロセスモデルを利用して新規のオブジェクト定義を構築する。新しいオブジェクトは、旋盤という機械の一種であり、一度に1つの部品を処理できる。この旋盤は、部品の加工を待機するインプットバッファと、次のロケーションに行くために待機するアウトプットバッファを有する。

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 Class⁷モデルを追加し、Latheと名付ける。Latheをアクティブモデルにしてから、DefinitionsウィンドウのPropertiesパネルをクリックする。新しいオブジェクトでは、まず新規に4つのプロパティを定義したうえで、ユーザから隠したいいくつかの継承プロパティを非表示にする。

- ExpressionタイプのStandard Propertyを新規に追加し、名前をTransferInTimeにする。DefaultValueを0.0、UnitTypeをTime、Default UnitsをMinutes、DisplayNameをTransfer In Time、Category NameをProcess Logicにする。
- ExpressionタイプのStandard Propertyをもう1つ新規に追加し、名前をProcessingTimeにする。DefaultValueをRandom.Triangular(0.1, 0.2, 0.3)、UnitTypeをTime、Default UnitsをMinutes、DisplayNameをProcessing Time、Category NameをProcess Logicにする。
- バッファサイズを定義するために、Integerタイプ（整数型）のStandard Propertyを新規に2つ追加する。名前をそれぞれInputBufferCapacity（表示名はInput Buffer）とOutputBufferCapacity（表示名はOutput Buffer）とする。DefaultValueをInfinity、Category NameをBuffer Capacityにする。
- 継承プロパティを展開する。継承プロパティのCapacityType、WorkSchedule、およびInitialCapacityは、Latheオブジェクトと関連が薄いので、Model 11-1と同じように、Generalカテゴリにて、これらのプロパティのVisible特性をFalseに変更する（図11.6参照）。

Processウィンドウのロジック構築を支援するために、利用するステーションを最初に追加する。Definitionsウィンドウから、Elementsパネルをクリックし、ステーションエレメントを3つ追加す

⁷ Fixed Classの下にProcessorオプションがあることに注意してほしい。ベースオブジェクトを構築する際にこの機能を用いることにより、自動的にいくつかのフレームワークが生成されるため、より容易に構築できる。この例題では、そのフレームワークがどのように作成されるかをよりよく理解するために、Processorは利用しない。

る。それぞれのステーションは、InputBuffer、Processing、OutputBufferと命名する。次に、InputBufferステーションのInitial CapacityとしてInputBufferCapacityという名前のプロパティを設定し（以前に行ったように、右クリックから、Set Referenced Propertyを選ぶ方法を用いる）、OutputBufferステーションのInitial CapacityとしてOutputBufferCapacityという名前のプロパティを設定する。そして、ProcessingステーションのInitial Capacityを1とする。

次に、(Latheオブジェクトのまま) Processウィンドウに移り、定義した3つのステーションを移動するプロセスフローを構築する。エンティティは、LatheオブジェクトおよびInputBufferに入るとき、以下でInputBufferTransferInと命名するプロセスが実行されることになる。

- Create Process ボタンをクリックし、最初のプロセスを作成する。InputBufferTransferInと名付け、Descriptionに「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度に自動生成される InputBuffer.Entered イベントによって始動される。したがって、プロセスの Triggering Event を InputBuffer.Entered と設定する。
- Delay、EndTransfer、および Transfer の各ステップを、この順番で設置する。
- DelayTime を TransferInTime に設定する。この時間は、ステーションに進入するまでに、エンティティに課される時間である。F2 キーを押し、Delay ステップの名前を TransferringIn に変更する。EndTransfer ステップは、搬送が完了したことを外部世界（コンベヤ、ロボット、人間など）へ発信する。搬送が完了すると同時に、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：変更するステップの「名前」は、そのプロセスの該当ステップに与えられたユニークラベルである。意味のある名称を用いることにより、プロセスの理解が容易になるだけではなく、モデルの追跡も楽になる。この機能は習慣的に用いるべきであろう。
- 上記の搬送が完了した後、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 に、To プロパティを Station に設定し、Station Name プロパティを Processing にする。

上述のInputBufferTransferInプロセスの完成図を、図11.9の最上段に示す。

Processingステーションは、搬入にかかる時間はゼロであるが、処理時間を要するため、ロジックが少し異なる：

- Create Process ボタンをクリックし、次のプロセスを作成する。ProcessingTransferIn と名付け、Description に「Delay for processing.」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度に自動生成される Processing.Entered イベントによって始動される。したがって、プロセスの Triggering Event を Processing.Entered と設定する。
- EndTransfer、Delay、および Transfer の各ステップを、この順番で設置する。
- EndTransfer ステップは、搬入の時間進行がゼロであるので、搬入を瞬時に終了させる。搬送が完了すると同時に、Processing ステーションに空きがあれば、新しい搬送が始まるかもしれない。このオブジェクトでは、能力は1に固定されているので、滞在中のエンティティは搬出されなければ、次の搬入は起こらない。

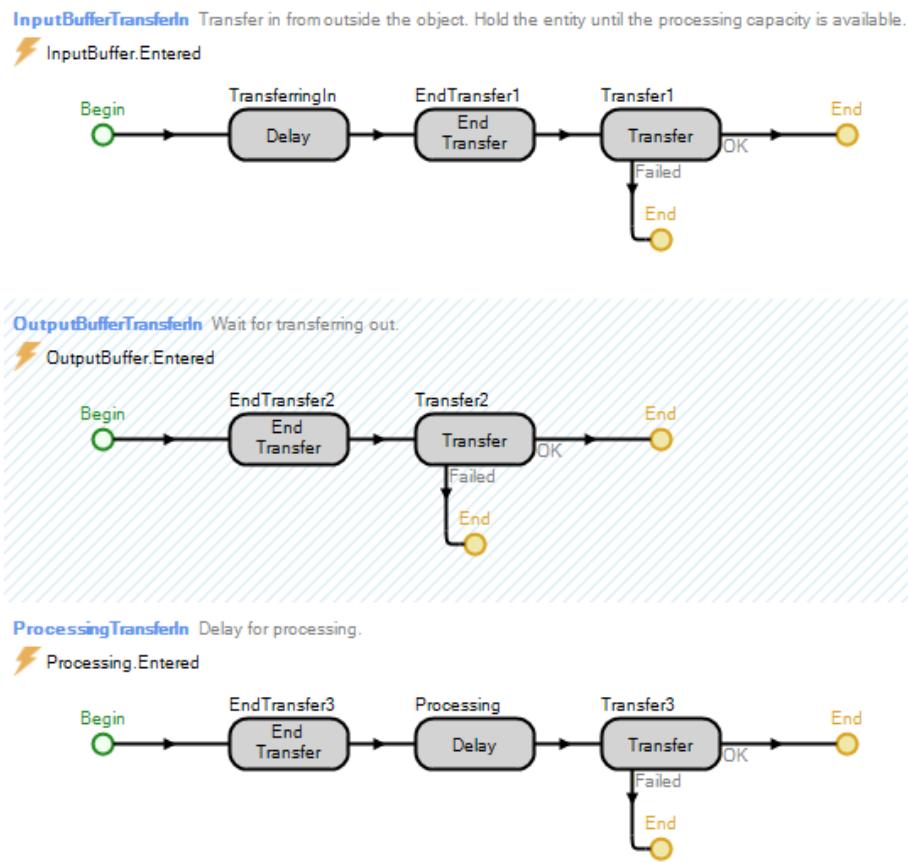


図11.9 Latheオブジェクトのふるまいを定義する3つのプロセス

- DelayTime を ProcessingTime に設定する。この時間は、処理時間としてユーザが指定するものである。F2 キーを押し、Delay ステップの名前を Processing に変更する。
- 処理の時間進行が完了した後、トークンは、Processingステーションから OutputBuffer 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 に、To プロパティを Station に設定し、Station Name プロパティを OutputBuffer にする。

ProcessingTransferInプロセスの完成図は、図11.9の最下段に示されている。

OutputBufferステーションのプロセスは、もっとも簡単であり、オブジェクトから退出するエンティティを通過させるだけである：

- Create Process ボタンをクリックし、最後のプロセスを作成する。OutputBufferTransferIn と名付け、Description に「Wait for transferring out.」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度に自動生成される OutputBuffer.Entered イベントによって始動される。したがって、プロセスの Triggering Event を OutputBuffer.Entered と設定する。
- EndTransfer および Transfer の各ステップを、この順に設置する。
- EndTransfer ステップは、搬入の時間進行がゼロであるので、搬入を瞬時に終了させる。
- トークンは、OutputBuffer から、Output という名称の Parent ExternalNode によって定義された関連ノードオブジェクト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 に、To プロパティを ParentExternalNode に設定する（注：External Node Name プロパティは、外部ビューにて定義した後に指定可能）。

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、下記のような設定を行う。

- ・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、入力だけに利用されるので、Node Class を BasicNode にし、Input と命名する。
- ・External Node を旋盤の右側に1つ追加する。出力側に能力を加えたいので、Node Class を TransferNode にし、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。到着するエンティティは、この外部ノードシンボルに対応する関連ノードオブジェクトに到着するとすぐに、InputBufferステーションに搬送されることに留意してほしい。
- ・InputBuffer ステーションをアニメートするため、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 ステーションをアニメートするため、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 ステーションをアニメートするため、旋盤シンボルに隣接するように Animated Queue を追加し、Queue State を Processing.Contents にする。また、8章で取り上げた Shiftキーを押しながらドラッグするテクニックを用いて、この待ち行列を上方に移動させ、旋盤の上部に配置する。

Latheの外部ビューを、図11.10に示す。

上記の設定によって外部ノードの定義が完了したので、Processウィンドウへ戻り、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

ここで、新しいオブジェクト定義を利用する準備が整った。NavigationウィンドウのModelをクリックし、モデルをアクティブモデルにする。そして、Source、Lathe（Project Libraryから選択）、およびSinkを1つずつドラッグして設置し、これらをPathでつなぐ。さらに、Latheをクリックしプロパティを設定する。モデルを実行すると、図11.11に示されるような様子が確認できるだろう。

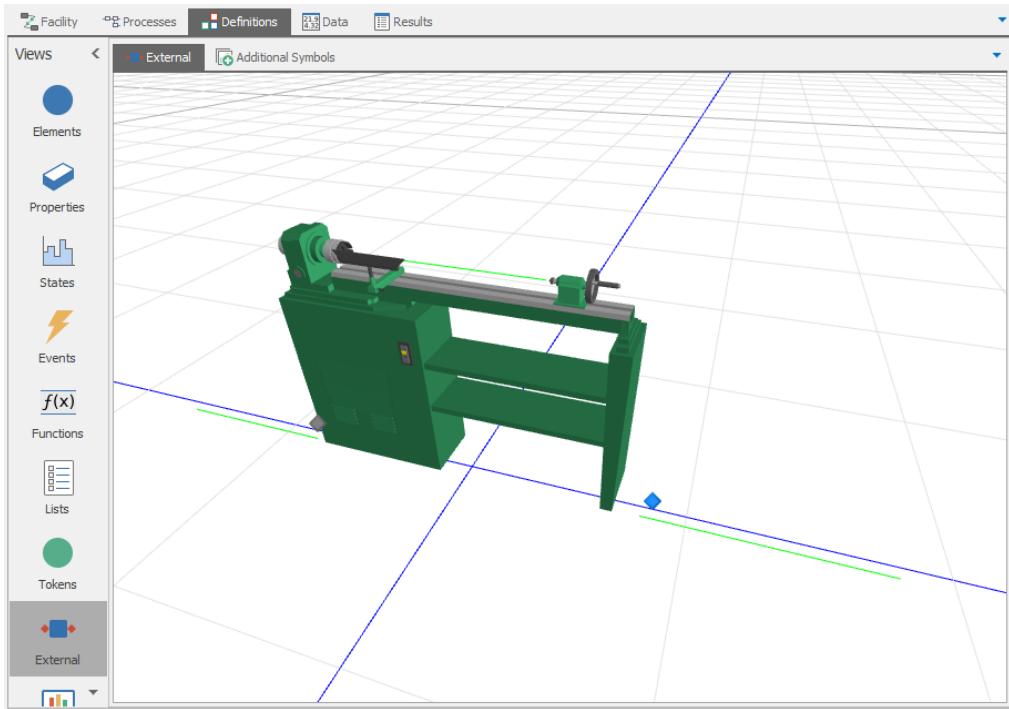


図11.10 Latheオブジェクトの外部ビュー

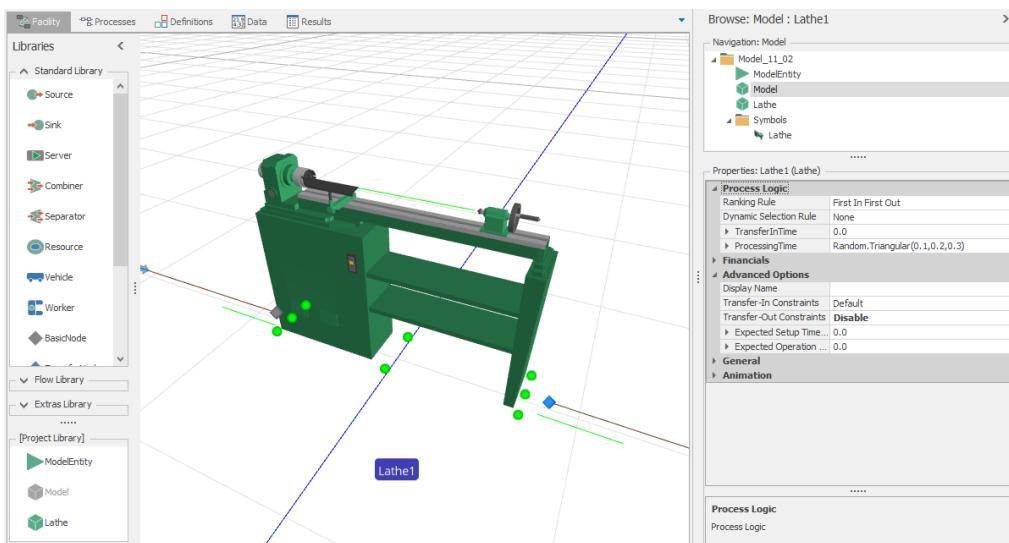


図11.11 Latheオブジェクトのモデルでの活用

11.4 Model 11-3：オブジェクトのサブクラス化

本節では、磁気共鳴撮像装置という医療機器、いわゆるMRIという新しいオブジェクト定義を例として構築する。この装置の動きは、StandardライブラリのServerと似ているので、Serverをサブクラス化することで、MRIを構築する。まず、一部のプロパティを非表示させたり、再命名する。また、MRIの修理を行うため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、この修理担当者を占有・解放する修理ロジックを構築する必要がある。

11.4.1 モデルロジック

新しいプロジェクトを開いて、FacilityウィンドウでServerを右クリックし、Subclassを選択する（Projectウィンドウでも同じ操作は可能）。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ので、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 をクリックしアクティブモデルして、Definitions ウィンドウの Properties パネルをクリックする。
- 新しいオブジェクト参照 (Object Reference) プロパティを追加し、RepairPerson と命名し、カテゴリを Reliability Logic にする。
- Switch Property Name を FailureType、Switch Condition を NotEqual、そして Switch Value を NoFailures に設定する。ユーザが NoFailures を選択する場合、このプロパティは非表示になる。
- このプロパティの 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に詳しく示す。

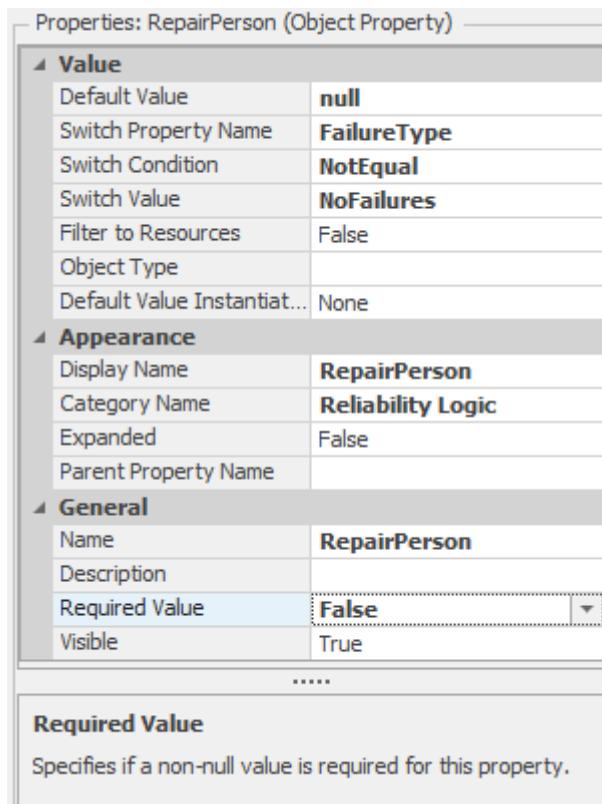


図11.12 RepairPersonのプロパティの定義

このオブジェクトで利用される用語を汎用のものから、医療専門用語に変更する。ProcessingTimeプロパティのDisplay NameをTreatment Timeに変更する。また、プロパティの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。

それから、Capacity Type、WorkSchedule、InitialCapacityという3つの継承プロパティを非表示にする。

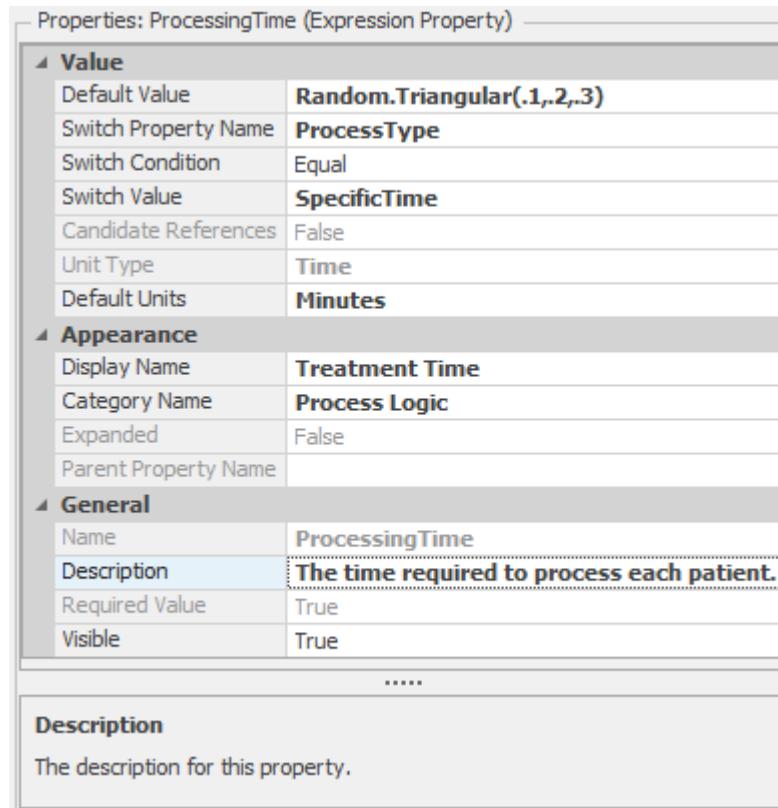


図11.13 ProcessingTimeをTreatment Timeに変更するプロパティの定義

次に、メンテナンス中に修理担当者を活用するために、継承されたプロセスロジックを修正する。Processesウィンドウをクリックし、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。この時点では、画面に表示されているプロセスは、Serverが所有しており、MRIが使用するよう継承されているため、プロセスを変更することができないことに留意してほしい。この継承は、プロセス名の左側にある緑色の矢印で確認できる。FailureOccurrenceLogicという名前の最初の継承プロセスを選択し、ProcessリボンのOverrideをクリックする。この操作により、Serverから継承されたプロセスがコピーされる。このプロセスは、Serverが所有する同じ名称のプロセスと同一であるが、コピーしたことで上書きされたプロセスとなり、編集することができるようになった。この上書きは、プロセス名左の矢印アイコンが下向きになったことで明示されている。元のプロセスに戻すには、ProcessesリボンにあるRestoreをクリックする。

それでは、このプロセスのロジックを変更していこう。SeizeステップとReleaseステップを、それぞれTimeToRepairという名称のDelayの前後に追加する。この2つのステップで、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を用いる）。図11.14に修正したプロセスの先頭付近を示す。

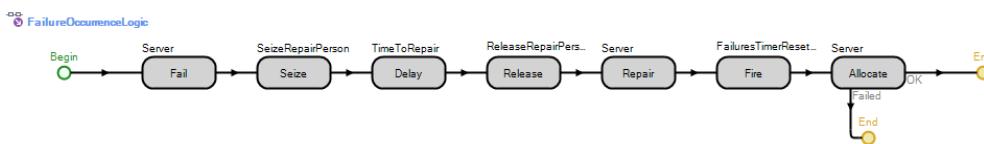


図11.14 修理担当者を追加するためのFailureOccurrenceLogicの上書き

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ので、ノードとフレームワークは気にしなくてもよい。

DefinitionsウィンドウのExternalパネルをクリックする。Place SymbolドロップリストからDownload Symbolを選択し、Trimble 3D WarehouseからMRIの画像を探してダウンロードする。InputBuffer.Contents、Processing.Contents、およびOutputBuffer.Contentsをアニメートするために、待ち行列を追加する。2つの外部ノードシンボルはServerから継承されているので、位置調整以外の編集を加えることができない。

11.4.2.1 本項のまとめ

ここで、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）、およびSinkを1つずつドラッグして設置し、Pathでそれらを接続する。MRIを選択してプロパティを設定し、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。

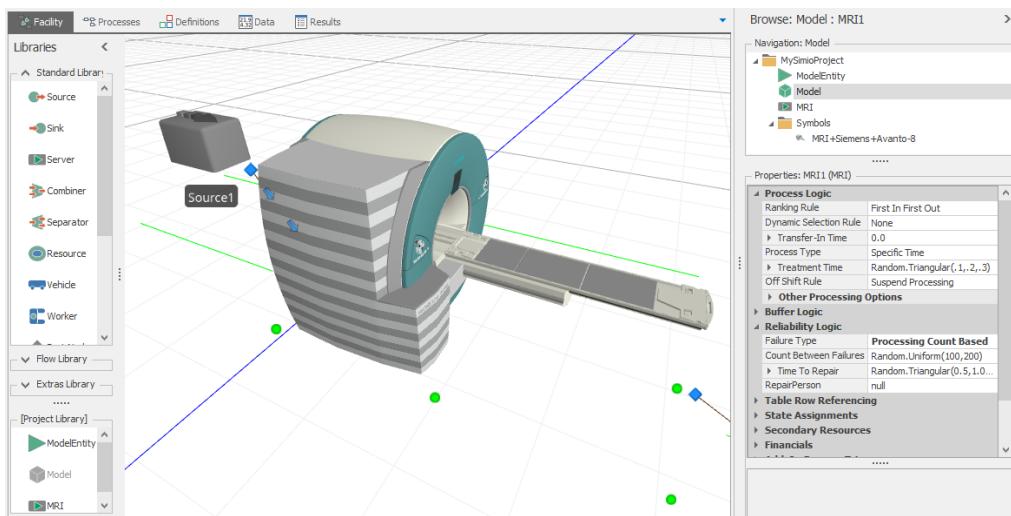


図11.15 新しいMRIオブジェクトの利用とカスタムプロパティ

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てモデルを構築すること、アドオンプロセスを使用してStandardライブラリを拡張すること、そして（本章のこれまでの部分で）カスタムオブジェクトの構築することが習得できただろう。そこで、中級程度のプログラミング能力を拥っていれば、Simioをさらに一歩進化させ、内蔵されているシミュレーションエンジン自体を拡張することができる。Simioアーキテクチャでは、Visual C#あるいはVisual Basic.Netのような.NET言語によって書かれた独自のカスタム機能を、Simioの多くの部分と統合することができる。

以下に、サポートされているユーザ拡張機能の種類を列挙する：

- ・ 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
- ・ 走行操作挙動
- ・ 設計時のアドイン
- ・ スケジューリング・アドイン
- ・ 表のインポートとバインディング
- ・ 実験の設計と実行

Simio拡張機能の接続点は、一連のインターフェースとして公開されている。そこに、ユーザによって拡張されたコンポーネントによって実装されるメソッドおよび呼び出し規則が記述されるこ

となる。この一連のインターフェースは、**Simio API** (Application Programming Interface) と呼ばれる。Simio APIに関する詳細な情報は、Simioのインストール先にあるAPI Reference Guide.chmファイルを参照してほしい（一般的に、C:\Program Files(x86)\Simioにある）。

いずれかの.NETプログラミング言語を利用して拡張機能を作成する際、SimioはC#ユーザに対して充実したサポートを提供している。Simioユーザ拡張機能を作成し始めるのを支援するため、Microsoft Visual Studioで作成された、多くの定義済みSimioプロジェクトやプロジェクト項目テンプレートが利用可能である。これらのテンプレートは、再利用可能かつカスタマイズ可能なプロジェクトやスタブを提供している。これらを利用すれば、ゼロから新しいプロジェクトやアイテムを作成する必要がなくなり、開発プロセスが加速するだろう。

さらに、Simioの標準インストールには、いくつかのユーザ拡張機能の例が含まれている。これらの例を参考にすることもできるし、自身の問題を解決するためにさらにカスタマイズすることも可能である。例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取りと書き込みを行う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と書きに利用するエレメントおよび4つのステップが含まれる。
- **ExcelGridDataProvider**：テーブルとExcelファイル間の読み込みと書き出しをサポートする。
- **CSVGridDataProvider**：表とテキストファイル間のインポートとエクスポートをサポートする。
- **ExcelGridDataProvider**：表とExcelファイル間のインポートとエクスポートをサポートする。
- **SelectBestScenario**：実験データ分析アドインの機能を説明する。
- **SimioSelectionRules**：Simio動的選択ルールのすべての実装を含む。
- **SourceServerSink**：コードから簡単なFacilityモデルを作成する設計時アドインを解説する。
- **SimioTravelSteeringBehaviors**：自由空間に移動できるエンティティのガイダンスをサポートする。
- **SimioScheduling**：スケジューリング・アプリケーションのためのリソース、リストとテーブルを配分する。

上記の例は、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、通常、PublicもしくはAll Usersフォルダ下にある。

11.5.1 ユーザ拡張機能の作成・展開の手順

ユーザ拡張機能の作成・展開に当たって、推奨される手順は次の通りである：

- Visual Studioで新規.NETプロジェクトを作成する。あるいは、Simio Visual Studioテンプレートを利用し、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、無料で利用可能なExpress版とCommunity版も提供している（www.microsoft.com/express/Windowsからフリーダウンロード可能）
- ユーザ拡張機能の実装を完了させ、.NETアセンブリファイル(.dll)を構築する。
- 拡張機能を展開するために、.dllファイルをコピーし、[MyDocuments]\SimioUserExtensions内に貼り付ける。もし、フォルダが存在しない場合、新規作成しなければなら

ない。別の選択肢として、ファイルを[Simio のインストールディレクトリ]¥UserExtensionsへコピーしてもよいだろう。ただし、ファイルをコピーする権限があるということを確認してから、操作すること。

正しく展開された拡張機能は、Simioの適切な場所で自動的に表示されることになる。拡張機能は、場合によって、ユーザアドインとして認識されることがある。次に、例を挙げる：

- ・ 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は、左側の User Defined パネルから利用可能である。
- ・ モデルの Definitions ウィンドウで Elements を定義する際、すべてのユーザ定義エレメントは、リボンの Elements タブの User Defined ボタンから利用可能である。

また、以下の様な別の場面では、Simioが新しい機能を持っているように表示される：

- ・ ユーザ定義選択ルールは、動的選択ルールとして、モデル内で利用可能である。
- ・ アプリケーションアドインは、Project Home タブにある Select Add-In ボタンからプロジェクトに利用できる。
- ・ 最初のテーブルが加えられた後、Table Imports and Binding アドインズが Bind To ボタンの下にある Table リボンに表示される。

Simioのヘルプから「extensions」（拡張機能）あるいは「API」を検索することによって、拡張機能に関する詳細な情報を確認することができる。これらの検索キーワードは、これらの機能の一般的な概要と導入部分の情報を提供している。より詳細な情報については、Program FilesのSimio フォルダにあるヘルプファイルSimioAPI Reference Guide.chmを参照してほしい。このファイルは、500ページを超える非常に詳細な技術情報を提供している。本書の執筆時点では、Simioのユーザ拡張機能の作成に関するトレーニングコースはないが、Learning Simioのスライドの付録として、ステップ・バイ・ステップで作成方法が解説されている。Simio Insiderに登録すれば（強く推奨。登録URL：www.simio.com/forums/）、別のサンプルや、Shared ItemsやAPIのトピックにおける議論を検索できる。

11.6 まとめ

本章では、Simioでオブジェクト定義を構築するために、基本的な概念を復習した。プログラマでなくとも、特定のモデルまたは応用領域のカスタムオブジェクトを構築できることは、Simioの重要な機能の一つといえるだろう。

本章の例題では、Project Libraryからオブジェクトを配置した。別の方として、独自のプロジェクトでオブジェクト定義を構築し、Project Homeリボンからライブラリとしてそのプロジェクトを読み込む方法もある。

Simioでは、アドオンプロセスを展開したStandardライブラリを用いるのか、あるいはカスタムロジックを有する新しいライブラリを作成するのか、選択を求められるケースがある。同じ応用事例に何度も遭遇するのであれば、カスタムオブジェクトを構築する方が便利であり、それによりアドオンプロセスを繰り返し読み込む必要がなくなる。もちろん、独自のオブジェクトにおいてもアドオンプロセスのサポートを受けることは可能である。これには、単純に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せばよい。

少し経験を積めば、オブジェクトの定義は簡単で、モデリング作業を強力にサポートする機能であることがわかるだろう。

11.7 問題

1. モデルロジックを作成する3つの方法を比較しなさい。特定のオブジェクトを作成するとき、3つの選択肢のどれを選ぶかについて、根拠を示しなさい。
2. External ViewとConsoleの相違点について述べなさい。それぞれの制約は何であるか。
3. 塗装ラインのモデルを構築していると想定しよう。塗装ラインをより大きいモデル(たとえば、多くの類似の(同一ではない)塗装ラインのある自動車工場)のオブジェクトとして再利用する場合、それに適したものにするには一般にどのような変更を加える必要があるか。
4. Model 11-2を再度作成しなさい。ただし今度は、Fixed ClassのProcessorから作成しなさい。このアプローチの利点と欠点を論じなさい。
5. 時間基準の故障をModel 11-2に追加しなさい。Time Between Failures、Time To Repair、およびRepair Resourceの選択について、オプションを加えなさい。修理担当者が修理作業のために訪れるように、外部ノードを含めなさい。このオブジェクトの有効性を検証するために、定義されたオブジェクトを利用し、新しいモデルを構築しなさい。
6. 箱が入ってくる1つの入力ノード、およびパレットに積まれた箱が出て行く1つの出力ノードを持つPalletizerオブジェクトを作成しなさい。パレットはPalletizer内で必要に応じて生成される。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しなさい。
そして、2種類の箱があり、同じパレットに積み込まれないようなモデルで、作成したPalletizerオブジェクトを利用しなさい。満載になったパレットは、出荷エリアに搬送される。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. 実験ウィンドウで利用されるアドインを独自に作成しなさい。作成されるアドインは、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したりする機能を有する。
8. 釀造所の抽出プロセスを模擬するために、離散バッチ処理をモデルする必要がある。このシステムには、長時間加工・処理するための発酵タンクが多くある。全体のモデルを構築しやすくするために、発酵タンク・オブジェクトを作成することにする。
実験しやすいため、システムを構成する小さなサブセットを考案する。この抽出プロセスを表すサブセットは、図11.16で示すように、スケジュールに従って麦酒のバッチを生成する。検証するシステムの各バッチの発酵時間は約2時間である(注:現実には発酵に何日もかかる)。同じ麦酒種類、かつ2番目のバッチが最初のバッチの到着時点から60分以内に到着する場合、発酵タンクが、最大2つのバッチを受け入れることができる。受け入れた2つのバッチは、最初のバッチが単独で終えた時間に終わられる。しかし、麦酒種類が異なると、決してまとめて発酵できないことになる。上記の特徴を反映させるFermentation Tankオブジェクトを作成しなさい。また、図11.16に示す到着スケジュールでコントロールするシンプルなモデル実験を行いなさい。このモデルには、シングルな発酵タンクだけでよいだろう。テーブルコメントに指定されている結果が生成されたことを確かめなさい。

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 酿造所のバッチに関する到着スケジュール（テストデータ）

第12章 インダストリー4.0における シミュレーションによるスケジューリング

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を向上させる（例：代替案との比較、パラメータの最適化、パフォーマンスの予測）ために用いられてきた。確かにシステム設計を良くすることは価値があるものの、効率的なシステムの運営にも様々な問題がある。

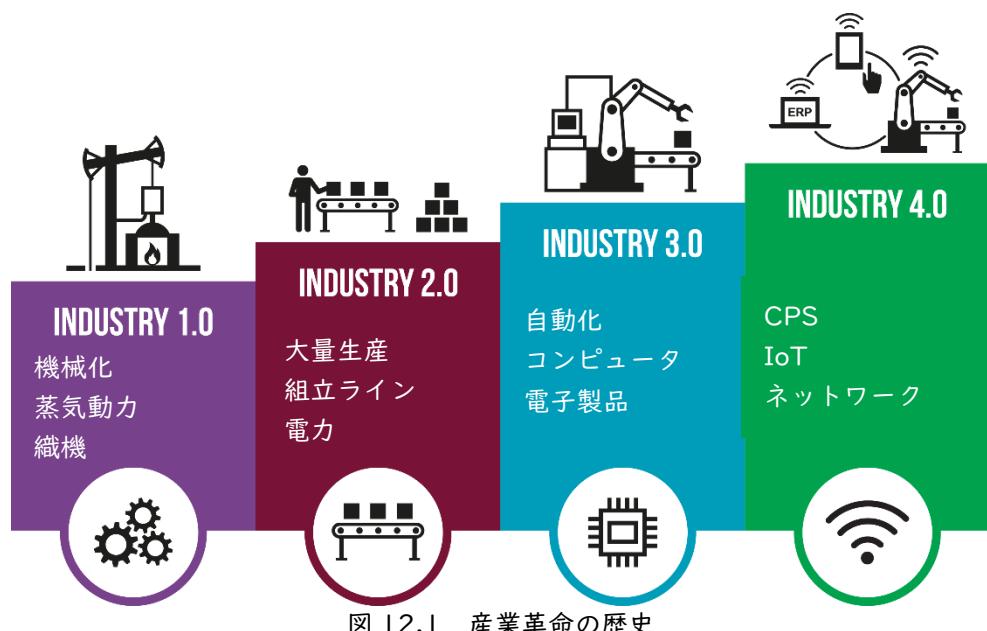
スマートファクトリーの構想は、過去数年で必要な技術が劇的に改善し、世界中から急速に注目を集めている。現在、このアプローチで工場を運営するとき、残されたいくつかの課題に対し、シミュレーション技術を進化させ、新しい領域にシミュレーションを適用することにより解決できる。

本章では、まずインダストリー4.0の進化と現状、およびその課題と機会について説明し、シミュレーションの従来の役割としてスマートファクトリーでの用法と、デジタルツイン分野での使用方法を検討する。また、計画とスケジューリングに使用されているテクノロジーの一部について説明し、リスクに基づく計画とスケジューリング（RPS; Risk-based Planning and Scheduling）の概要を解説する。本章の一部の資料は、SimioのWebサイトでも閲覧でき、本書は許諾を得て再録する（Simio LLC 2018）。

12.1 時代を超えた産業革命

インダストリー4.0は、「完全に接続された自動製造システム」への現在のトレンドを表すために使用される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、彼の著書 *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年の初めになされたが、実は最初の産業革命が文明を再定義する前、つまり200年前にも言われたことがある。

では、インダストリー4.0について説明する前に、私たちが今、どのような時代にいるのかを見てみよう（図12.1）。



12.1.1 第1次産業革命：機械生産

1760年代以前、世界中の人々は完全に隔絶されており、限られた知識や技能と食糧供給のある閉鎖的な地域社会で、家族の伝統に従い、変化を考えずに暮らしていた。英国を起点として、蒸気機関と蒸気動力の発明と改良により、農業の急発展と食糧供給が人口増加に拍車をかけ、新たな市場の発展につながった。

工場が水車の周りだけでなく、どこにでも設立できることにより、工業化が拡大した。それに伴って、都市が作られ、多くの消費者や鉄道の伸張によって、都市化がさらに一層拡張していった。織物、鉄鋼、食料などの財の生産が選好され、農業は廃止されていた。また、雇用が創出され、メディア、輸送システム、医療サービスなどの分野が一新され、人々の日常生活が変わった。

この入れ替わりのすべては、1800年台の半ばまで約80年の期間にわたって行われ、その影響はヨーロッパと南北アメリカに広がっていた。

12.1.2 第2次産業革命：大量生産

20世紀が始まり、人口と都市化は成長を続け、最新の発明や電力の普及が大いに後押しした。第2次産業革命は、組立ラインによる食品、衣料、物資の大量生産は発展の引き金となり、輸送、医療、軍需産業も躍進を遂げた。この時代はヨーロッパが優位を続けていたが、他の国々も独自に発展する工業力と技術を持っていた。

12.1.3 第3次産業革命：デジタル時代

二度の世界大戦後、1960年代までは、文明は再び落ち着きを取り戻した。戦時中の技術の新たな応用がイノベーションを促し、平和な時代に再び前向きな考え方をもたらした。シリコンウェハーの有用性が発見され、半導体やメインフレームコンピュータの開発につながった。そして、1970年代から1980年代にかけて、パーソナルコンピュータへと発展していった。1990年代のインターネットの普及に伴い、コンピュータ、すなわちデジタル革命が国際的にも順調に進んだ。

12.2 第4次産業革命：スマートファクトリー

第4次産業革命は、人類史上最大の飛躍的な成長であり、今後20年間で世界のGDPに10～15兆ドルという途方もない金額を加えるだろうと言われている(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)としても知られている。そのような名称である理由は、企業が自社製品を設計、製造、およびサポートする方法に大きな影響を与えるためである。仮想世界と実世界の融合により、製造業にとって、この新たな現象は再び成長の原動力となっている。人件費を削減し、生産性を向上させるためのアイディアを実行した後の次のステップは、製造業のデジタルトランスフォーメーションに関する技術の応用になる。この完全に接続され、自動化された製造システムでは、完全に統合してリンクされた一連の機械と人員から得られるリアルタイム情報に基づいて、生産に関する意思決定の最適化が行われる。

インダストリー4.0に含まれる既存のデジタルテクノロジーは以下のとおりである：

- ・ モノのインターネット(Internet of Things, IoT)と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/サービスとしてのソフトウェア(Software as a Service, SaaS)
- ・ ビッグデータ/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
- ・ 拡張現実／バーチャルリアリティ (AR/VR)
- ・ シミュレーション
- ・ IT／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により、大規模なプロセス改善と生産性の最適化を促進する。結果として得られるイノベーションは、最終的には情報とサービス基盤とする革新的なビジネスモデルの開発につながる。ほとんどの製造業者にとって現在、インダストリー4.0はまだ目指すべきビジョンにすぎないが、実は、先進な製造ロボットや3D プリンティングの活用により、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中であり、プラスチックおよび金属の分野に影響を与えている。生産の柔軟性、品質、安全性およびスピードにおいて、潜在的な利点と改善がすでに実現されている。しかし同時に、情報管理、生産スケジューリング、サイバーセキュリティ分野においては、新たな課題も生じている。

インダストリー4.0は、すでに元々の構想範囲を超えて、单一ないし数千のアイテムからなるバッチが同じリードタイムとコストで生産できるような新時代がやってきている。地域貿易の流れを促進するために、消費者の近くで、より小規模で機敏な工場を設立するという大きなマクロ経済のシフトが到来している。

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できるため、すべての利害関係者に影響を与える可能性がある。しかし、その成功は、システム全体の効果的な情報の流れに依存する。ここで、データの収集、計画およびスケジューリングにとっては、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の利点が非常に重要になる。

12.2.1 次は？

コラボレーションと知識共有によって新たな発見が続き、私たちの今いる時代にスピードと勢いが結集されている。デジタル革命を基盤として、現在では、広範囲に及ぶモバイルインターネット、安価な SSD 能力、そして大規模のコンピューティング資源が備えられている。より小型で強力なデバイスやセンサーを組み合わせることで、ロボティクス、ビッグデータ、シミュレーション、機械学習、人工知能などの技術を活用して、さらに先進的な情報を引き出すことができるようになる。

スマートファクトリーは、仮想システムと物理システム間の連携、また、予兆診断と生産性向上のために、デジタルツインとして表現することができる新しいオペレーティングモデルの作成を可能にする。このような機会を実現し、潜在的な解決策を効率的な解決策に変えるために、やるべきことはまだある。

それ以外にも、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飛躍的に前進しており、この最新革命の新たな分岐点は、物理ドメインとデジタルドメインを生物学的側面で結びつけることになる。昔の長くゆっくりと拡大した最初の産業革命と比べて、変化の速度は驚異的である。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変化し、世界をより小さくして、技術やアイディアを交換することで、分野間で新しい発明を広めることができる。

しかし、第1次産業革命の小さな第一歩がなければ、私達はまだ、旅行や探検がなく、限られた食糧供給源で、健康への影響を受けやすく、革新の見込みがないという、隔絶した生活を続けていたであろう。確かに現在、深刻な変化と重大な国際的発展が避けられないものとなっているが、私達はその発展を受け入れて、歴史の中で進むために自分の想像を開く必要がある。

クラウス・シュワブ氏が結論づけたように、『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、デバイスが協調して動作できるように安全なデータ通信が行なわれる。製造現場では、産業用モノのインターネット（IIoT）によって、製造プロセス内の機械と製品が相互に通信し、効率的な生産という最終目標を達成できる。インダストリー4.0に向けた動きが進むにつれて、デジタル化の進展が製造業に独自の課題や懸念をもたらしている。これらの課題を解決する1つの方法は、デジタルツインを活用することである。

デジタルツインは、製品、部品、システム、またはプロセスの仮想表現を提供する。デジタルツインを通して、それらが存在する前でもどのように機能するのかを確認できる。この用語はデバイスレベルにも適用される。デバイスのデジタルツインは、実際のデバイスが現実世界でどのように機能するかを、非常によく似た動きを仮想世界で確認できる。重要な応用として、工場全体が現実世界で実行されるのと非常によく似た動きを仮想世界で実行される、工場全体のデジタルツインがある。後者はデジタルツインの定義より広く、達成不可能と思われるかもしれないが、そうではない。すでに説明したデザイン指向モデルと同じように、私たちの目的は、「完璧なモデル」を作成することではなく、目標を達成するのに役立つ結果を生成するための「十分に近い」モデルを作成することである。その目標をどのように達成できるかを探ってみよう。

一部の実務家によると、デジタルツインを有効にするために必要なデータが含まれた他のすべてのシステムに完全に接続されている場合のみ、モデルをデジタルツインと呼ぶことができる。スタンダードアロンシミュレーションモデルはバーチャルファクトリモデルと呼ばれるが、それが完全に接続され、関連するERP、MESなどのシステムからのデータによって駆動されるリアルタイム（ほぼリアルタイム）モードで動作するまで、デジタルツインではない。したがって、デジタルツインとして効果的に機能するには、7.8節で紹介したようにデータからモデルを生成できることも重要である。これにより、最新のデータをインポートするだけで、リソース／マシンの追加や、モデルおよびスケジュールを自動的に作成するなど、データの変更に基づいてモデルが対応できるようになる。

インダストリー4.0時代には、技術革新の指指数的成長により、かつてないほどデータを収集、保存、操作することができる。より小型のセンサー、より安価なメモリーストレージ、より高速なプロセッサは、すべてワイヤレスでネットワークに接続されているため、物体をデジタル世界に投影するための動的シミュレーションとモデリングが容易になる。この仮想モデルは、運用、履歴、および環境データを受け取ることができる。

技術の進歩により、大量のデータの収集と共有がはるかに簡単になり、モデルへの適用が容易になって、結果を予測して推進するさまざまなシナリオを評価できるようになった。もちろん、重要なリソースのデジタルモデリングと同様に、データセキュリティはデータツインでも常に重要な考慮事項である。

実物資産のコンピュータ化されたバージョンとして、デジタルツインは、さまざまな価値のある目的のために使用できる。たとえば、残存耐用年数を決定したり、故障やプロジェクトのパフォーマンスを予測したり、財務収益を見積もることができる。このデータを用いて、潜在的な機会から利益を得るために、設計、製造、および運用を最適化することができる。

有用なデジタルツインを実装するためには、3段階のプロセスがある：

モデルを確立する：実物にデジタルデータを重ね合わせるだけでなく、3Dソフトウェアを使用して被写体をシミュレートする。その後、実物とモデルとの相互作用が行われ、関連するすべてのパラメータはモデルと通信する。データが与えられ、モデルは類似性を通じて、どのように動作するかを「学習」する。

モデルをアクティブにする：シミュレーションを実行することにより、モデルは既知データと与えられたデータの双方に従って継続的に更新される。履歴、他のモデル、接続されたデバイス、予

測、コストなど、他のソースから情報を取得することで、リスクと信頼水準に関連した知見を提供するために、オプション順にシナリオが実行される。

モデルから学習する：結果として得られた解決案を使用して、使用率やコスト、パフォーマンスの面で最適な結果を達成するため計画を実施し、潜在的な問題領域や不採算の状況を回避することができる。

デジタルツインは、単なるデバイスまたはシステムの設計図や概略ではなく、これらの要素が他の要素や環境どのように動的に相互作用するかといった振る舞いを含む、すべての要素の実際的な仮想表現である。これらの要素を監視し、診断と予後を改善し、問題の根本的な原因を調査して、性能と全体的な生産性を向上させることで、大きな利益が得られる。

正しく生成されたデジタルツインを使用すると、製品ライフサイクルのすべての段階（設計・構築・運用）に良い影響を与えるために、運用環境を動的に調整できる。このような応用では、デジタルツインモデルを作成する前に、目的と期待をよく理解しておく必要がある。そのとき初めて、目的を満たし、意図した利益を反映するような忠実したモデルを作成することができる。利益の例は次のとおりである：

- ・ 設備は自身の状態を監視し、メンテナンスのスケジュールを立て、必要に応じて交換部品を注文することもできる。
- ・ 混流生産を導入し、納期を遅らせることなく、機器の使用率を最大化するようなスケジュールを立てることができる。
- ・ リソースが変更された場合の迅速なリスクケジュールは、重要な納期に合わせて負荷を再最適化することで損失を削減できる。

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のために使用してきた。1.2節には、シミュレーションが日常的によく使われている多くの応用分野のいくつかがリストアップされている：

- ・ **サプライチェーンロジスティクス：**ジャストインタイム、リスク低減、発注点、生産配分、在庫水準、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：**原材料輸送、労働力輸送、配車、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、人員配置のレベルと割当て、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：**成長への投資、適切な投資決定、投資収益率の客観的評価
- ・ **生産性：**ラインの最適化、製品ミックスの変更、設備の割当て、人件費削減、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

シミュレーションがインダストリー4.0でどのように役立つかについての最初の答えは「上記のすべて」である。最も簡単に言うと、スマートファクトリーは単なる工場であり、他の工場と同じような問題を抱えている。シミュレーションは、これまで使用してきたのと同じ分野で、同じ利点を提供できる。一般に、シミュレーションを使用して、システムを客観的に評価し、最適な構成と運用についての知見を得ることができる。

もちろん、スマートファクトリーの実装は、「単なる工場」を超え、多くの重要な部分で異なっている。まず、スマートファクトリーは一般的に大きく、より多くの構成要素だけでなく、より洗練された構成要素を持っている。楽観主義者は「洗練された」を「問題がない」と読むかもしれない

いが、悲観主義者は「失敗する可能性が高い」と受け取るかもしれない。いずれにせよ、より多くの相互作用を持つ大規模なシステムは、分析が難しく、シミュレーションの伝統的な応用がさらに重要になる。**特定の高度な機能の影響を評価することは困難である。**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、連携して動作するシステムを設計し、そのシステムを調整して最適化できる唯一のツールになる可能性がある。

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、リアルタイムのデータをより多く利用できるようになる。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の強みではないが、最近の製品は、モデルにそのようなデータを組み込むことができる。この拡張されたデータアクセスにより、システムのパフォーマンスが向上する可能性もあるが、むしろさらに多くの故障ポイントの発見と、**実装前にリスク領域を特定するための十分な詳細さのモデルを作成できる機会にもなる。**

スマートファクトリーが異なるもう1つの特徴は、自動化と自律性のレベルである。スマートファクトリーの動的プロセスにより、システム故障にインテリジェントに対応し、自動的に修正措置を取るなど、運用上の柔軟性が可能になり、故障を直したり、適切なルーティング変更によって故障を回避したりできる。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によって、これらのアクションを評価するのに役立つ。**

通常の工場と同じように、スマートファクトリーは異なる構成や設定で何度も繰り返し実行することはできない。それに対し、シミュレーションはまさに繰り返して実験できるように設計されている。将来の運用を効果的に予測し、**数日をわずか数秒に短縮する。**さらに、システムのスケールアップまたはダウンの影響を検討する必要がある場合は、シミュレーションモデルを簡単に調整できる。結果として得られる情報は、プロセスとシステム全体に関する基本的な質問に答えるものとなる。たとえば、プロセスの所要時間、機器の使用頻度、リジェクトの発生頻度などである。その結果、遅延、使用率、ボトルネックなど、システム改善のパフォーマンス基準を予測できる。

この仮想ファクトリモデルを使用する大規模な組織にとっての大きな利点は、データ、システム、およびプロセスの標準化である。通常、大企業内の各工場では、システムの実装方法が異なる。そこで、各工場の設備を单一のERPインスタンスに移動すると、大きな問題が発生してしまう。ユーザは同じプロセスとワークフローを使用する必要があるが、どのプロセスが最適で、どのデータが正しいか、または望ましいかをどのように判断すれば良いのか分からぬ。仮想ファクトリモデルを用いて異なる運用ポリシーとデータをテストすることが、単一の最適なグローバルプロセスを決定し、それに応じてすべての工場を調整するための最良の方法である。データ生成アプローチでシミュレーションを使用することは、複数のグローバル工場を持つ大手企業にとって貴重で興味深いものである。

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コミュニケーションの支援は、特にスマートファクトリーに当てはまる。複雑なシステムがどのように機能するのかを説明するのは非常に難しく、それを理解するのはさらに難しいかもしれない。モデルを作成するには、各サブシステムがどのように機能するのかを理解してから、その知識をモデルに表現する必要がある。シミュレーションモデル自体は、その知識のリポジトリになる。つまり、構成要素に埋め込まれた直接の知識と、モデルの実行から生じる間接的な知識の両方である。同様に、2Dまたは3Dモデルのアニメーションはシステムを理解するための非常に貴重な方法であるため、利害関係者はシステムの仕組みをよりよく理解し、問題解決により効果的に参加できるため、結果への賛同が得られる。

また、時間はかかるが、モデリング段階では、プロセスに精通しているオペレータやスタッフの関与が必要である。この関与は直接的な当事者意識を与え、後の調査結果を実装する段階でも役立つ。そのために、現実的なシミュレーションは、システム全体の状況でのパフォーマンスの向上をテストし、理解するための、スプレッドシートのような代替手段よりもはるかに簡単かつ迅速なツールであることが証明されている。これは特に、ユーザや意思決定者にデモンストレーションを行

う場合に当てはまる。

このようにして、シミュレーションは以下のことを支援する：

- ・ 結果として生じるシステムパフォーマンスの予測。
- ・ システムの各部分がどのように相互作用するのかの発見。
- ・ パフォーマンスを測定および比較するための統計量の追跡。
- ・ システム構成と全体的なパフォーマンスのナレッジベースの提供。
- ・ 貴重なコミュニケーションツールとしてのサービス。

要約すると、従来の設計の役割でシミュレーションを使用すると、スマートファクトリーの開発、展開、実行において競争上の優位性が高まる。システムはより少ない時間、より少ない問題で展開でき、そしてより速く最適な収益をもたらすことができる。

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持つ複雑なシステムについて、毎日のスケジューリングのため、シミュレーションへの需要が促進された。この運用上のニーズにより、シミュレーションの価値は、システム設計を改善するという従来の役割を超えて、より早く、より効率的なプロセス管理、および向上した性能と生産性を提供するという領域にまで広がった。リスクに基づく計画とスケジューリング (RPS) などの最新のテクノロジーを使用して、システム設計の評価と生成のために構築されたものと同じモデルが、インダストリー4.0の環境では、毎日のオペレーションスケジューリングにおいて重要なビジネスツールになる。

デジタル化された製造では、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、生産プロセス中のプロセス分析と制御に役立つデータを送信することができる。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追加される。この技術により、インダストリー4.0の工場で製造された「スマートな」製品は、原材料から最終製品までの工程を通して、ステータスレポートを送信することができる。

製造プロセス全体でデータの可用性が向上することで、柔軟性と即応性が高まり、バッチサイズの縮小と受注生産への移行が可能になる。

この適応性を利用するため、インダストリー4.0のスケジューリングシステムは以下を行う必要がある：

- ・ すべての要素を正確にモデル化する。
- ・ スケジュールを素早く計算する。
- ・ 簡単に可視化できる。

フィジカルとデジタルの世界のギャップを埋めるために、インダストリー4.0の特徴としてのIoTデバイス、ビッグデータ、クラウドコンピューティングを使用するスケジューリングシステムが、今まで以上に必要とされている。

従来、スケジューリングには、手動、制約ベース、そしてシミュレーション、この3つの方法がある。

多くの場合、ホワイトボードやスプレッドシートのようなツールを使用した、労働集約型の手動スケジューリングは、小規模または複雑度の低いシステムでは効果的である。しかし、名前が示すように、手動によるアプローチは、すべての関連と実行可能な代替案を理解する人の能力に大きく依存している。このようなアプローチは、データの膨大な量と複雑さにより、大規模で動的な生産環境では非現実的である。

制約ベースのスケジューリングでは、すべてのシステム制約を表すために定式化された式を解く

必要がある。数学モデルは、スマートファクトリーのすべての要素で構築することができる。ただし、データを入力して解くことは非常に複雑であり、恐らく長い時間がかかる。また、解釈、可視化、および実装が困難な解決策が見つかった場合、その実行を可能にするために、重要な条件を無視または単純化する必要がある。

手動および制約ベースのスケジューリングでは、複雑さを減らすために、施設の単一の部門またはセクションに対してスケジューリングが行われることがよくある。これらのローカライズされたスケジュールは多くの場合、セクション間にプロセスバッファを必要とし、時間や在庫、キャパシティを無駄にしてしまう。

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最適なソリューションとして際立っている。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できる。リソース（設備、工具、作業者など）、ならびにプロセスで消費および生産される材料を表すことができる。このようにして、システムを介したジョブの流れをシミュレートし、各段階での正確なリソースと材料の使用状況を表示し、リアルタイムのステータス更新を提供することができる。

意思決定ロジックには、たとえば、最小の段取り替え時間の選択や、作業者の経験から追加されたカスタムルールなどをモデルに埋め込むことができる。これらのアルゴリズムを組み合わせることで、システム内の実際の材料と部品の流れを正確にモデル化する一連のルールが作成される。

このテクノロジーにより、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、生産プロセスのあらゆる側面で計算と並べ替えを実行できる。この機能は、デジタルワークステーションによって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、スケジューリングが迅速、詳細、そして正確であることを意味する。

スマートファクトリーにおけるスケジューリングの以下の3つの主要な要件は、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によって満たされる：

- ・ すべての要素の正確なモデリング：柔軟なモデルは、運用上の制約およびカスタムルールの完全な表現など、コンピュータ化された情報から生成される。
- ・ スケジュールの高速計算：スケジュールの計算と代替案のスケジューリング、比較および配布は迅速かつ正確に行われる。
- ・ 簡単に可視化：コンピュータシミュレーションにより、スケジュールをすべての組織レベルで明確かつ効果的に伝達することができる。

労働効率の向上は、シミュレーションによるスケジューリングのもう1つの利点である。生成された詳細情報は、労働力を有効にするための最も重要な方法の1つとも言える「スマートグラス」のような技術の使用を可能にする。スマートグラスは、タイムリーで詳細な指示を従業員に提供する。シミュレーションモデルでは、スケジュールを常に評価することにより、実際のデータと現在の情報を使用して、次のタスクを実行するための最も効率的な方法を各作業者に指示する。

このようなスケジュールは、スマートファクトリーの重要な部分だが、モデルは、次の節で説明するように、実際には単にスケジュールするよりも、さらに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コンポーネントシステムから収集したデータを使用し、離散事象型シミュレーションソフトウェアを用いて、生産ライン全体をシミュレートすることができる。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に関するリアルタイム情報をモデルに取り込むことができ、シミュレーションを通じてさまざまな計画やスケジュールを作成できる。潜在的な損失や中断を最小限に抑えながら、供給元または生産計画の代替案を相互に評価できる。

単純な品切れや機器の故障、大規模な自然災害など、変化が発生した場合、シミュレーションモ

モデルは生産への影響と下流のサービスへの影響を表現することができる。その後、修正された行動方針を手動または自動で評価し、解決策を実行することができる。

インダストリー4.0環境でシミュレーションを使用してスケジュールを設定し、リスクを減らすことの利点には、一貫した生産を保証することが含まれる。そこでは、コストが管理され、品質はあらゆる状況下で維持される。

スケジューリングを活用することで、高度なデータ駆動型シミュレーションモデルは、デジタルツインの役割を果たすことができる。図12.2は、シミュレーションモデルがスマートファクトリーの中核にどのように位置するかを示している。すべての重要なサブシステムと通信し、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的に作成し、さらなるアクションのためにそのスケジュールの構成要素と結果を各サブシステムに配布することができる。

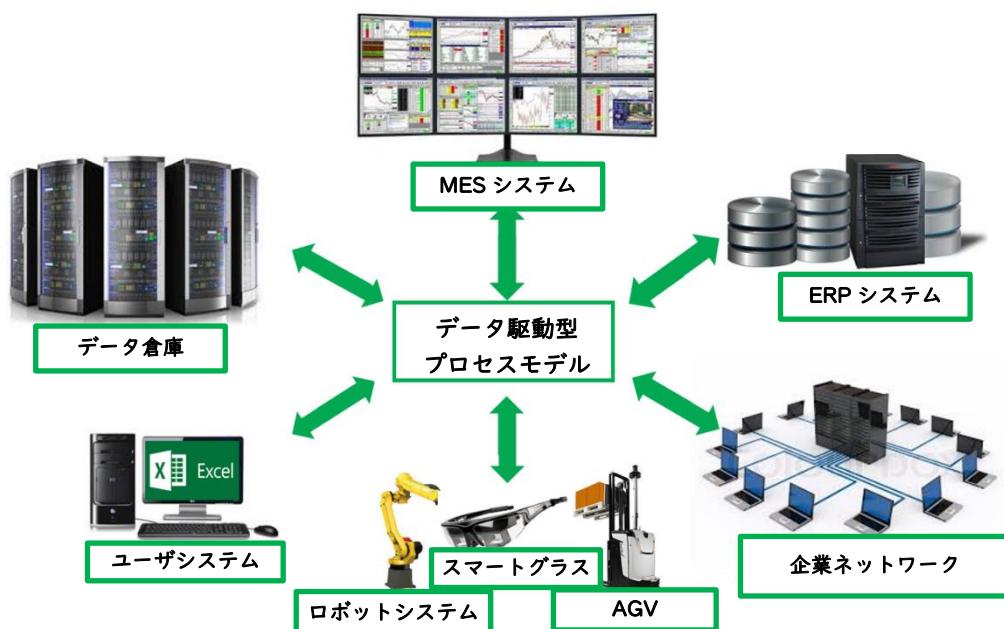


図12.2 デジタルツインによるスマートファクトリーの実現

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、以下の理由により、そのようなアプリケーションに非常に適している：

- ・ 任意のサブシステムとバッチまたはリアルタイムで通信する能力。
- ・ ファクトリーを表すために必要かつ複雑なふるまいのモデル化。
- ・ 洗練された技術を実行した適切な「最適」スケジュールの生成。
- ・ 実行のために利害関係者へのスケジュールの報告。
- ・ 繰り返し報告される計画からの逸脱の待機。

これは、ほとんどのスマートファクトリー計画に残された重要なギャップを埋める。

12.6 計画とスケジューリングにおける未解決の問題

計画とスケジューリングは関連した応用分野であるため、たいていは一緒に議論される。計画は「大きい絵」の分析である。つまり、どれだけ作ることができるか、どのように作るべきであるか、そして原材料とリソースがいつ、どこでどのように、それを作るために必要であるかについて考えることである。計画は通常、無限の原材料を想定した能力の集計ビューで行われる。スケジューリングは、運用上の詳細に関連している。現在の生産状況、実際のキャパシティ、リソースの可用性、および仕掛品（WIP）を所与とし、どのような優先順位や順序、および方策的な決定が、重要な目

標をもっともよく達成する結果となるのかを考えることである。計画が、実施の数日前、数週前、または数ヶ月前に立てられるのに対して、スケジューリングは、多くの場合、数分前、数時間前、または数日前になされる。多くの応用では、計画とスケジュールのタスクは別々になされる。実際、未実現の潜在的な性能を大幅に残したまま、片方が無視されている一方で、他方だけが完了していることは珍しくない。

簡単な種類の計画は、リードタイムに基づくものである。たとえば、ある種類の多くの部品が、「標準的に」発注の3週間後に出荷されることが、過去の平均からわかっているとしよう。他の要素を捨象すれば、それらを生産したい場合、3週間を見積もっておけばよいだろう。これは、多くの場合、リソース稼働率をじゅうぶんに考慮しているわけではない。もし「標準」よりも多くの仕掛品があれば、予測よりも時間がかかることがあるため、リードタイムは楽観的になるかもしれない。

別の簡単な種類の計画では、部品がシステムをどのように移動するか、そしてリソースがどのように利用されているかを示すために、マグネットボードやホワイトボード、スプレッドシートを用いて、手作業でガントチャートを作成する。これは非常に労働集約的なオペレーションであるだろう。そして、システムの複雑さと計画者の経験の程度によって、結果として得られる計画の質が非常に変動しやすいものになるだろう。

3番目の計画方法は、一般にプログラミング言語で表現されるカスタムアルゴリズムを用いて設計・開発された、**特注のシステム**である。これは特定のドメインと特定のシステムに対して、高度にカスタム設計されている。かなりうまく働く可能性はあるが、たいていは非常に高い費用や実行時間がかかり、さらにカスタマイズの程度が影響し、再利用の機会が少なくなる。

一般的にもっとも人気がある方法の1つは、**先進的計画スケジューリング**(Advanced Planning and Scheduling; APS)である。APSは、生産需要を満たすために、生産能力、リソース、および原材料を最適に配分する。統合基幹業務システム(ERP)ソリューション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多くのAPS市販製品があるが、これらのソリューションには、広く認識されているいくつかの短所がある。たいてい、ERPシステムと日々の生産は、それらの成功を妨害する2つの限界、すなわち複雑性と変動性が影響し、分断されたままとなっている：

複雑性：1つ目の限界は、あいまいで複雑なシステムを効果的に扱うことができないことである。特注のシステムはあらゆるシステムを表すことのできる可能性があるが、詳細な特注のシステムを作成するために必要な費用および時間は、たいてい、そのシステムが実践的なソリューションとして機能するのを妨げる。システムが標準的なベンチマークに非常に近いなら、上述したような技術はうまくいく傾向にあるが、システムがベンチマークと異なっている場合、そのツールは適切なソリューションを提供するには詳細さに欠けるだろう。取扱われていない重要な状況には、複雑なマテリアルハンドリング(例：クレーン、ロボット装置、トランスポータ、作業者)、特殊な作業とリソース割当て(例：段取り替え、順序依存の設定、作業者)、経験に基づく意思決定ロジックと運用規則(例：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：2つ目の限界は、システムの中で変動を効果的に扱うことができないことである。すべての処理時間を知っていないければならないし、他のすべての変動性を通常は無視する。たとえば、予測できない中断時間や機械の故障は、明確に考慮されない。また、作業者や原材料に関する問題は決して起こらないし、他の否定的なイベントは起こらない。結果として生じる計画は、ひどく楽観的である。図12.3は、ガントチャートの形で典型的なスケジューリングの出力を示している。緑色の破線は、完成予定日(黒)と納入日(グレイ)間の余裕時間を表している。残念ながら、計画された余裕時間がじゅうぶんであるかどうかを判断することは難しい。実行可能なスケジュールで開始したとしても、変動性や計画にない出来事がパフォーマンスを下げ、時間経過とともにスケジュールが実行不可能になっていくことはよくあることである。予測されたスケジュールと実績の間に大きな差異が生じることは、普通のことである。遅延を防ぐために、スケジューラは、余分の時間や在庫、あるいは容量を組み合わせて、バッファを持たせなければならない。これらすべては、システムに費用を加える事柄である。

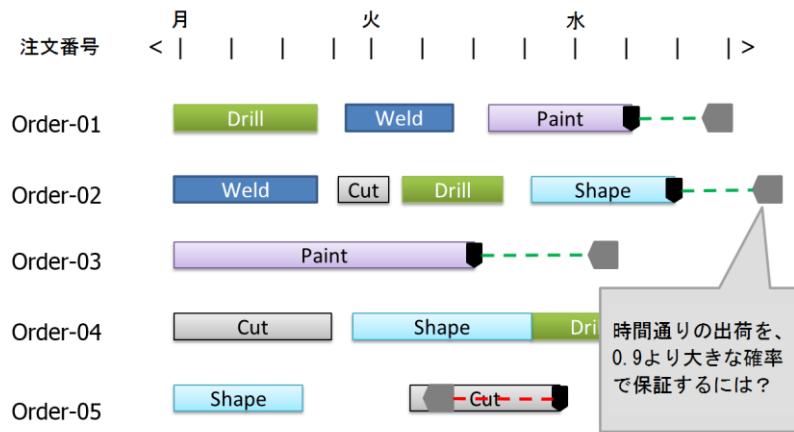


図 12.3 計画から得られる典型的なガントチャート

所与の能力制限のあるリソース（たとえば、作業者、機械、移動手段）を考慮して、実行可能なスケジュールを生成するという問題は、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、2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。

最初のアプローチは、制約条件として表現された数学上の関係式によって、システムを定義する**数学的最適化法**である。そして、遅延ジョブ数を最小化するといった目標を達成するようにしながら、他方で制約条件を満たすように数学モデルの解を探索するために、アルゴリズムに基づくソルバーを利用する。残念ながら、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わかっていない問題として分類される。したがって、実際には、スケジューリング問題の最適解を発見するというよりも、「良好な」解を見つけるという意図で、ヒューリスティックなソルバーを用いなければならない。このアプローチを用いた商品でよく知られたものには、IBM の ILOG 製品ファミリー (CPLEX) と SAP の APO-PP/DS の2つがある。

スケジューリングの数学的最適化手法には、いくつかのよく知られている欠点がある。数学的制約条件でシステムを表現することは、非常に複雑で費用のかかるプロセスである。そして、システムは変化するので、数学モデルは時間の経過とともに維持することが難しくなる。さらに、数学的制約条件を用いて正確にモデル化できず、無視せざるを得ない、実システムにおける重要な制約条件もあるだろう。その結果として得られるスケジュールは、数学モデルを満たすかもしれないが、実システムで実行可能なものではない。最後に、数学モデルの解を生成するために用いられるソルバーは、良好な候補スケジュールを作成するためにたいてい何時間もかかる。したがって、これらのスケジュールは夜通し、もしくは週末に、よく実行される。計画にない出来事が発生すると（たとえば、機械の故障や原材料の到着遅延、作業者の病欠など）、スケジュールはすぐに陳腐化するため、スケジュールの有効期限は一般に短い。

本節は、すべての項目の解説を目的としたものではなく、いくつかの概念と一般的な問題の概要を簡単に説明した。より詳細な内容は、優れたテキストである *Factory Physics* (Hopp and Spearman 2008) を参照されたい。

12.7 シミュレーションによるスケジューリング

有限能力スケジューリングの2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用いる方法である。計画とスケジューリングの支援としてシミュレーションツールを利用するコンセプトは、何十年間も前から存在する。著者の1人は、1980年代前半に製鋼スケジューリングシステムを開発するためにシミュレーションを用いたことがある。スケジューリングの応用では、現状のシステムでシミュレーションモデルを初期化し、モデル内で実

際の計画された仕事の流れをシミュレートする。スケジュールを生成するために、シミュレーションを実行する際に、すべての変動と計画にないイベントを排除しなければならない。

シミュレーションによる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。最適化手法で必要とされる時間よりも少ない時間で生成することができる。シミュレーションによ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よって決まる。たとえば、機械などのリソースが遊休状態になるとき、次に処理されるエンティティを選択するためにモデル内のルールが活用される。このルールは、もっとも高い優先順位の仕事のような簡単な静的ルールであったり、または段取時間を最小にするような順序を導くルールのような複雑な動的選択ルールであったり、あるいは納期までの残り時間を残りの作業時間で割った値（クリティカルレシオ）の最小値を持つ仕事を選ぶことで、緊急度に基づいて仕事を選択するルールであるだろう。

データ駆動型の伝統的な、または「懐古的な」のジョブショッピング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た。たとえば、システムはワークステーションの集合として見られ、各ワークステーションは前処理、加工、および分解フェーズに分かれている。そして、システム内を移動する各仕事は、あるワークステーションから別のワークステーションまで、決まったルートに従う。ソフトウェアは、ワークステーション、原材料、および仕事を記述するために、データを用いるように作られている。適用事例が懐古的なモデルにうまく合致していれば、よい解が得られるかもしれない。そうでなければ、ニーズにモデルをカスタマイズして合わせる機会は限られるだろう。実システムに存在する重要な制約条件を無視し、モデルに含めないようにするしかない。

また、有限能力スケジューリングに汎用の離散イベントシミュレーション（DES）製品を用いることもできる。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エンジンを用いるための典型的な構造を示している。

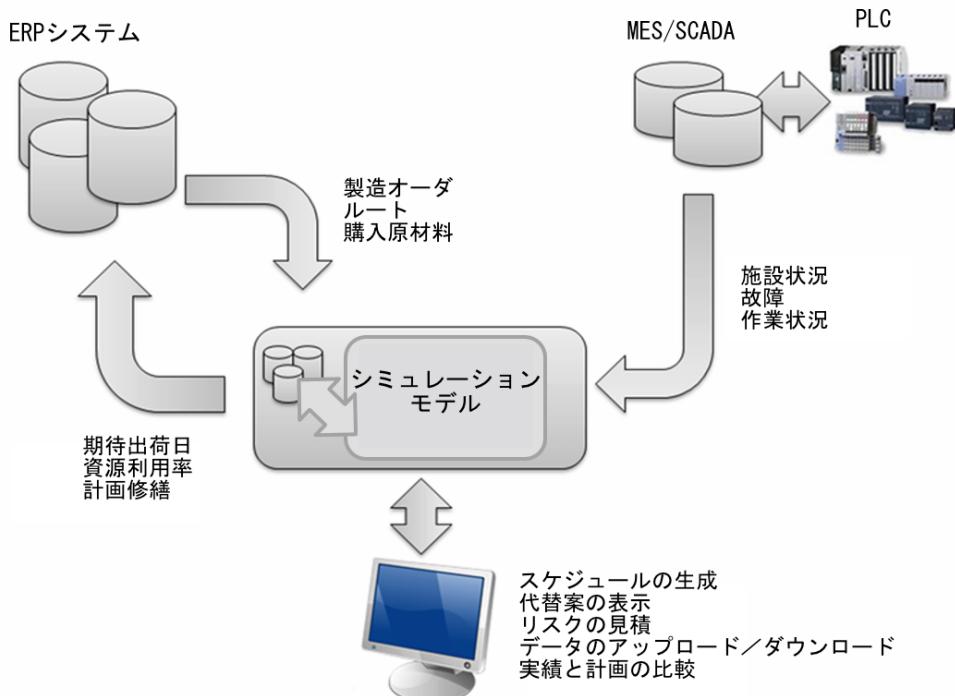


図12.4 典型的なシミュレーションによるスケジューリングシステムの構造

このアプローチの利点は以下の通りである：

- 柔軟性に富む：汎用ツールは、システムのどんな重要な側面もモデル化できる。システム設計のために作られたモデルようである。

- ・測定することができる：設計のためのシミュレーションと同様に、繰り返すことができる（すべきである）。問題の一部を解くことができ、その解を用いて始めることができる。繰り返すことと、希望するスケジュール精度をモデルが提供できるまで、必要に応じてモデルに幅と深さが加わる。
- ・先行事例を活用できる：スケジューリングに必要なシステムモデルは、設計を微調整するために必要な（そして、できればすでに活用された）ものと非常に似ているため、計画とスケジューリングのために、その設計モデルの利用を拡張することができる。
- ・確率的に運用できる：設計モデルがシステム構成を評価するために確率的な分析を用いるように、計画モデルは就業規則やスケジューリングシステムにおける他の運用上の特性を確率的に評価できる。これは、最初からよりよい意思決定を行える「より賢い」スケジューリングシステムとなる。
- ・確定的に運用できる：確定的な計画を生成するために、確率的な機能を無効にすることができる。上述のように、これは楽観的なスケジュールをもたらすが、**詳細さの水準**が比較的高いため、他のツールを用いたスケジュールよりも正確な傾向がある。そして、それがどれくらい楽観的であるかを評価することができる（次の項目を参照）。
- ・リスクを評価できる：確定的な計画が生成された後で、実行するために確率的な機能を用いることができる。再び変動性（起こりうる悪いことのすべて）を有効にすることによって、そしてその計画に対して複数の反復実行を行うことによって、重要なパフォーマンス目標をどの程度達成できるかを評価できる。もっとも費用対効果がよい方法でリスクを管理するようにスケジュールを客観的に調整するために、この情報を用いることができる。
- ・必要なパフォーマンス指標をすべて利用できる：モデル実行中のいつでも、モデルはパフォーマンス目標の主要な情報を集めることができるために、意味のある方法でスケジュールの実行可能性とリスクを測定できる。

しかしながら、スケジューリングに汎用の DES 製品を用いようとする場合、それらはその目的に対して専用に設計されてはいないため、いくつか特有の難点がある。起こりうる問題のいくつかを以下に述べる：

- ・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についてサマリ統計量を出力する。関連性はあるが、スケジューリングへの応用における主要な焦点は、個々の仕事（エンティティ）とリソースであり、たいていガントチャートや詳細な追跡ログの形で提供される。このレベルの詳細さは、汎用の DES 製品では一般に自動的に記録されない。
- ・モデル初期設定：シミュレーションのシステム設計への適用事例では、空かつ遊休のモデルで開始し、それからバイアスを排除するためにシミュレーションの初期部分を切り捨てる。スケジューリングへの応用では、処理中の仕事やシステム内のルート中の場所に存在する仕事を含めて、システムの現状をモデルの初期状態にできることが重要である。ほとんどの DES 製品では、これを行うのは容易ではない。
- ・ランダム性の制御：DES モデルは通常、ランダムな時間（たとえば、処理時間）とイベント（たとえば、機械故障）を含んでいる。計画を生成している間、時間の期待値を用いる一方で、すべての確率的イベントは無効にしたい。しかし、計画がいったん生成されれば、計画に伴うリスクを評価するために、モデルに変動性を加えたい。典型的な DES 製品は、両方の運用モードをサポートするように設計されていない。
- ・企業データとの連結：計画あるいはスケジューリングモデルを動かすために必要な情報は、一般に企業の ERP システムやデータベースに存在する。どちらの場合でも、通常、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んでいる。ほとんどの DES 製品は、

- リレーショナルデータソースと接続する、あるいは連動するように設計されていない。
- ・状況の更新：計画およびスケジューリングモデルは、たとえば機械故障のような、実際のシステムに生じる変化に絶え間なく適応しなければならない。これには、状況の変化を入力するための対話型インターフェースを必要とする。
 - ・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を支援するためのユーザインタフェースを持っている。スケジューリングおよび計画への適用において、計画を生成し、想定される業務的意思決定（たとえば、残業の追加や原材料発送の催促）のリスクを評価するために、（他の誰かによって開発された）既存のモデルを利用するスタッフに対して、専用のユーザインタフェースが必要とされる。

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、シミュレーションアプローチの重要な利点をすべて活用しながら、これらの短所を克服でき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への適用に対して、確定的および確率的シミュレーションを組み合わせて、伝統的な DES の全能力を引き出すツールである（Sturrock 2012a）。RPS の技術的背景については、Deliver On Your Promise - How Simulation-Based Scheduling Will Change Your Business (Pegden 2017) で詳しく説明されている。RPS は、ほぼすべての生産システムに存在する変動性を完全に考慮するために伝統的な 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前もって緩和するために必要な情報を提供する。RPS は、基盤となるシミュレーションモデルを 2 つの用途で利用する。シミュレーションモデルは、あらゆる水準の詳細さで構築することができ、実システムに存在するランダム変動のすべてを組み込むことができる。

RPS は、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することで、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、おおよそ APS ソリューションによって生成された確定的なスケジュールと等価であるが、必要に応じてより詳細な説明が可能である。しかし、その後 RPS は、ランダム性を有効にして（確率的な）同じシミュレーションモデルを活用し、（利用可能なら複数のプロセッサを利用し）複数回スケジュールの生成を繰り返し、反復実行間のスケジュールの結果に関する統計量を記録する。記録されるパフォーマンス指標は、（納期などの）目標達成の可能性（つまり尤度）や、完成期日の期待される節目の日程（システムの変動によって、計画期日よりも一般に遅れる）、楽観的および悲観的に見積もった完成時間（変動性に基づくパーセンタイルの推定）などがある。図 12.3 に示す RPS 分析と図 12.5 を比較してほしい。ここで、リスク分析によれば、Order-02 は適切な余裕を持っているように見えるが、特定の注文やそれが必要とするリソースおよび原材料に関するリスクを考慮すると、期限内に完成することを表す尤度は比較的低い（47%）。計画段階でリスクの客観的尺度を確認することにより、もっとも効果的な方法でリスクを緩和する機会が得られる。

RPS は、スケジューリングのために当該システムの専用シミュレーションモデルを構築するというシミュレーションベースのアプローチを用いる。このアプローチの主な利点は、システムの制約条件を完全に表現するために、シミュレーションソフトウェアのモデリング能力を存分に活用できることである。つまり、すべてのシミュレーションツールを用いて、システムをモデル化できる。また、（シミュレーションソフトウェアがその能力を提供している場合）複雑なシステムをモデル化するために、カスタムオブジェクトを用いることができる。さらには、クレーンやコンベアなどの複雑なマテリアルハンドリング装置や、フォークリフトや AGV などの搬送車の動きも（それらの移動経路で起こる混雑も含めて）表現することができる。オープンや治具を変更できるマシンニングセンターなど、複雑なワークステーションもモデル化することができる。

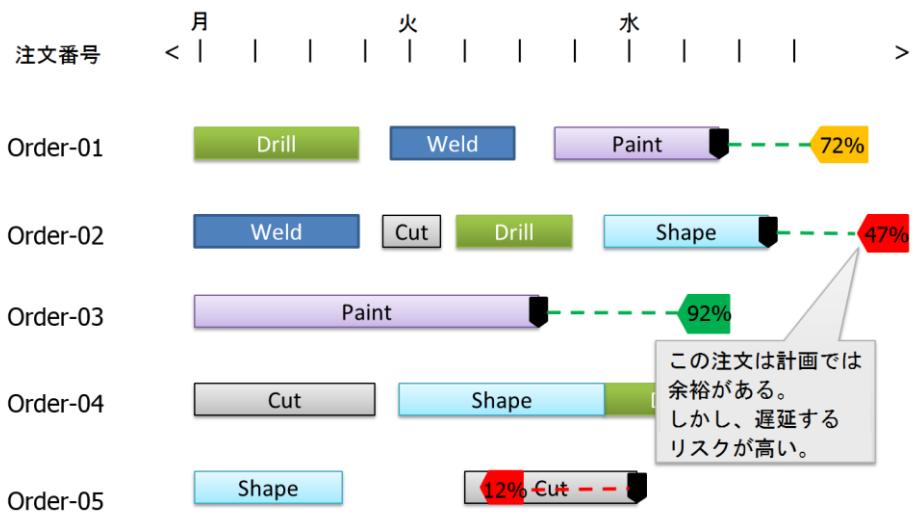


図 12.5 高リスクの注文を特定したガントチャート

RPS は、モデルに含める制約条件の種類や数を限定しない。したがって、もはや生産システムで重要な制約条件を無視する必要はない。複雑な生産やサプライチェーンの現実を完全に表現したモデルを用いることで、確定的な計画と対応するリスク分析の双方を生成することができる。また、施設の設計に対する変更を評価するために構築されたものと同じモデルを、RPS の導入に活用することができる。すなわち、これは、施設設計および日程計画の双方を改善するために、1つのモデルを使い回せることを意味している。

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を作成しながら、運用戦略を継続的に見直し、仮説分析を実行するための継続的な改善プラットフォームとして使用できる。これは、生産する新部品の導入や設置する新機械／ラインなどのテストに、オフラインで使用できる。更新されたシステムの「実運用」は、評価モデルを実運用モデルにすることで、簡単にできる。設計を証明するために使用されるモデルは、ソフトウェアを再実装したり、コストのかかる更新や変更を行うことなく、システムの変更に基づいてスケジュールに直ちに影響を与える。

サプライチェーンのパフォーマンスを向上させるために、モデルの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との間の整合性を改善することができる。同じモデルは、計画のために 3~6 週間、スケジューリングのために 1~3 週間、そして実行のための詳細な生産スケジュールのために、おそらく 1~2 日実行される。これにより、調達が正しい所要日数に基づいて行われるため、原材料の利用可能性が保証される。このより正確な情報を使用して、ERP システムの更新（たとえば、更新を SAP にフィードバック）を行うことができる。

RPS は、OptQuest のような最適化プログラムにもリンクできる。企業の KPI を設定し、自動実験を実行して、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つけ、工場を効果的に実行してから、適切にスケジュールすることができる。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) システムを構成・実装するための理想的なツールにもなる。シミュレーションベース DDMRP システムには、ほぼ瞬時の再スケジューリングと動的な設定更新の利点がある。

12.9 Simio RPS による計画とスケジューリング

Simio RPS Edition は、Simio の拡張版であり、リスクに基づく計画とスケジューリングへの応用のために、特別に設計された機能を備えている。この拡張機能はアカデミックパッケージには含まれないが、現在 RPS 機能を持っていなければ、教職員の方が academic@simio.com にリクエストすることで、機関ソフトウェアおよび学生ソフトウェアの双方に対するアップグレードを申し込むことができる。RPS 機能の有無を確認するもっとも簡単な方法は、Facility ビューのすぐ上の

Planning タブ（図12.6）を探すことである。RPSの機能の多くがPlanningタブからアクセスできる。アカデミック版のSimio RPSは、商用版と非常によく似ているが、最も一般的なバージョンでは、20のリソースのログと2つのターゲットのモニタリングが可能である。学習にはじゅうぶんだが、企業での活用には一般に不足だろう。

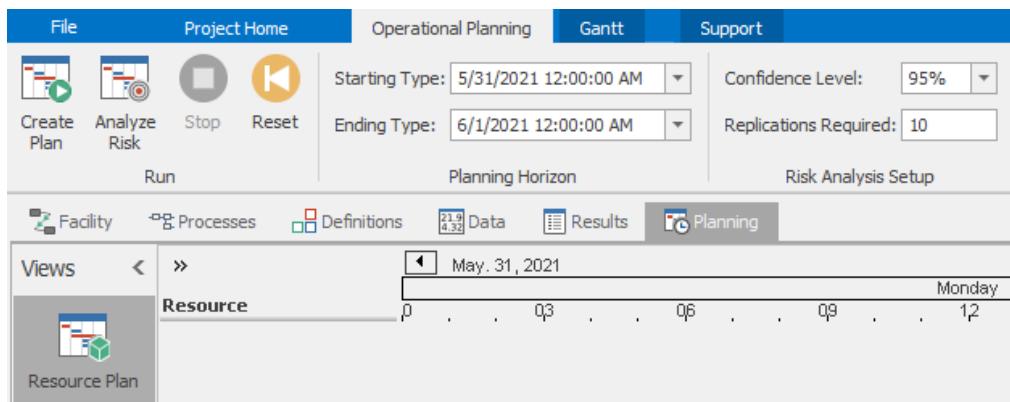


図12.6 Planning タブによるRPSスケジューリング機能へのアクセス

Simio RPSソリューションを動かすため用いられる基本モデルは、Simio製品群のどれでも構築することができるが、RPSで使うためにそのモデルを準備するにはSimio RPS機能が必要である。この準備には、アクティビティログの有効化、出力状態変数およびスケジューリング目標のテーブルへの追加や、スケジューリング担当者に対してユーザインターフェースをカスタマイズすることなどが含まれる。

伝統的なSimioの応用例では、データテーブルは、モデルに入力を供給するためだけに用いられる。しかしながら、RPSの適用例では、データテーブルも、モデル実行の間、値を記録するために用いられる。これは、標準プロパティの列に加えて、テーブル定義を状態変数列に追加することによって実装されている。たとえば、テーブル状態変数 Data-Time は、仕事の出荷日を記録するために用いられ、テーブル状態変数 Real は、仕事の累積費用を記録するのに用いられるだろう。テーブル状態変数は、Standard Library のオブジェクトにおいて、状態変数の割り当てを利用することにより記述される。あるいは、任意のプロセスロジックにおいて Assign ステップを用いることで、より柔軟に記述することもできる。出力の状態変数列は、Simio RPS Edition の Data タブにある States リボン、あるいは State ボタンを用いてテーブルに加えられる。

スケジューリングの目標は、スケジュールにおいて希望する結果に一致する値である。目標の典型的な例は、仕事の出荷日である（納期より前に仕事を完了したい）。しかしながら、目標は完成期日に限定されてはいない。目標として、シミュレーションにおいて測定できるすべての事柄（たとえば、材料の到着、熟練度、費用、生産量、品質、設備総合効率）およびあらゆる水準（たとえば、全社的なパフォーマンス、部門別、サブアセンブリ別、他の主要管理点）で把握される。

目標は、目標の値を指定する式で定義される。そして、その値の上限下限も定義され、これらの境界に対応する各範囲へと分類される。たとえば、納期の目標は、上限を超えた範囲を「Late (遅延)」と分類し、上限より下の範囲を「On Time (定刻)」と分類するだろう。Simioでは、Late と On Time の用語を用いて統計量を報告する。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 と比べる目標 (TargetShipDate) の定義を示している。そこでは、OnTime、Late あるいは Incomplete (未完成) の3つの結果が起こりうる。同様に、費用目標は、「Cost Overrun (予算超過)」と「On Budget (予算通り)」に範囲を分類するだろう。目標には、日付／時刻、あるいは総生産費用などの一般的な値を用いることもできる。また、納期や費用などの目標は、上限よりも下にしたい値である。他方、完了仕事数などの目標は、下限より上にしたい値である。Target列は、Simio RPSのDataタブにあるTargetsリボンまたはTargetボタンを用いて追加される。

The screenshot shows the Simio software interface. On the left, there's a sidebar with icons for Facility, Processes, Definitions, Data, Results, and Planning. Below these are sections for Views, Tables, Data Connectors, Lookup Tables, Rate Tables, Work Schedules, and a calendar icon showing the number 23. The main area displays a table titled "Manufacturing Orders" with columns: Release Date, Due Date, Order Status, Priority, Ship Date, ProductionCost (USD), Value, and Status. The table contains 16 rows of data from May 25 to 31, 2021. To the right of the table is a "Properties" panel for "TargetShipDate (Target)". It shows the value as "5/31/2021 12:00:00 AM" with various properties like Expression, Data Format, Unit Type, and Value Classification. There are also sections for Appearance, Operational Planning, and General.

図 12.7 テーブル状態変数とプロパティによる目標の定義

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においては、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。しかしながら、変動および計画にないイベントのあるリスク分析を実行する場合、Simio はモデルのすべての反復実行について各目標値のパフォーマンスを記録する。それから、この情報を用いて、定刻通りに出荷できる確率や、期待／楽観的／悲観的な出荷日などのリスク尺度を計算する。

標準の Simio ユーザインターフェースは、モデルの構築と実験を重視している。RPSへの応用においては、計画およびスケジューリング担当者のニーズに合うような異なるユーザインターフェースが必要である。彼らは一般にモデルを構築しないが、代わりに計画とスケジュールを生成するために、操作可能な設定でモデルを用いる。つまり、計画とスケジューリングを行うユーザには、別の専用ユーザインターフェースが必要である。Simio RPS は、計画とスケジューリング担当者に合わせたユーザインターフェースを、容易に作成することができる。また、スケジューリング担当者に対して表示され、彼らが編集できるデータ型をすべて構築することができるし、実装を特定の応用分野に合わせて完全にカスタマイズすることもできる。

12.10 スケジューリングのインターフェース

これまで、Simio RPS をデザインモードで使用してきた。RPS には、以前に開発されたモデルに基づいてスケジュールを生成および表示するためのスケジューラモードもある。スケジューラモードでは、計画とスケジューリングを考慮したカスタマイズされたユーザインターフェースが提供されている。このユーザインターフェース（図 12.8）は、新しいモデルの構築をサポートしていないため、標準の RPS ユーザインターフェースよりも小さく、シンプルになっている。スケジューラモードでは、標準の Simio 製品を用いて構築された Simio モデルを実行し、その後、RPS のデザインモードを用いて展開するために準備される。スケジューラモードは、File→Settings オプションで変更すると利用できる。



図 12.8 スケジューラモードで使用できる簡素化されたリボン

スケジューラモードの主な目的は、確定的なモードにおいて、モデル内で計画された仕事を実行することによって、計画／スケジュールを生成することである。そこでは、すべてのランダムな時間が期待値に置き換えられ、すべてのランダムイベントが抑制される。この確定的スケジュールは（すべての確定的スケジュールと同様に）楽観的である。しかしながら、この後にランダムイベン

トを分析するスケジューラモードの機能を用いることになる。

スケジューラモードは、専用レポートと共に、結果として生じる計画／スケジュールのたくさんの静的および動的の双方のグラフィカルな表示を提供する。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、エンティティおよびリソースの双方を中心としたガントチャートなどがある。エンティティのガントチャートは、チャートの行に各エンティティ（たとえば、仕事やオーダー）を表示する。そして、水平方向の日時のタイムスケールに沿って描かれた長方形は、各リソースがそのエンティティによって保持されるタイムスパンを表現している。リソースのガントチャートは、チャートの行に各リソース（たとえば、機械や作業者）を表示する。そして、水平方向の日時のタイムスケールに沿って描かれた長方形は、このリソースを用いるエンティティを表現している。これらのガントチャートは、後述するSimioの例の一部である。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。図では、Order-04の遅延に対する原因が示されており、Weld（溶接）装置を長時間待ち、機械を動かす作業者についても少しの間待たなければならないことがわかる。

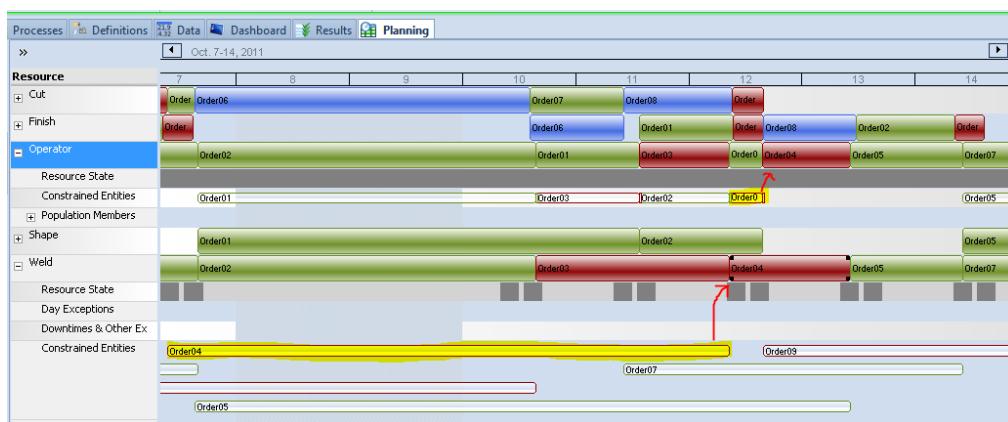


図12.9 Order-04の遅延に関する制約条件の検討

専用レポートには、各リソースの作業リストが含まれる。そのリストでは、計画期間中にリソースが実行する各作業の開始および終了時間が提示される。一般に、作業リストは、システム内の各ワークステーションの作業者に対して提供され、そのワークステーションで予想される作業量を確認できる。また、専用レポートには、計画期間中の時間に対するリソース状態を記したリソース利用率レポートや、計画期間中のシステム内に非付加価値をもたらす制約条件（たとえば、原材料不足や作業者利用不可）をまとめた制約条件レポートなどがある。図12.10は、作業者に対するリソースの作業手配（あるいは差立）リストの例を示している。これらのレポートのより多くの例は、次項で説明する。

スケジューラモードは、特定の仕事、機械、作業者、あるいは原材料の情報をまとめて提供するために、デザインモードを用いて設定された独自のダッシュボードも表示できる。これらのダッシュボードは、計画／スケジュールをさらに詳細に検討するため、任意の仕事あるいは機械に対する単一の要約表示に、いくつかの図および表形式のスケジュール情報を組み合わせている。

スケジューラモードを利用することで、計画およびスケジューリング担当者は、高いリスクのスケジュールを改善するために、代替案を検討することができる。異なる計画は、残業の影響や、原材料の発送、あるいは特定の目標を達成するために仕事の優先順位を変更するなどの影響を確認するために評価される。

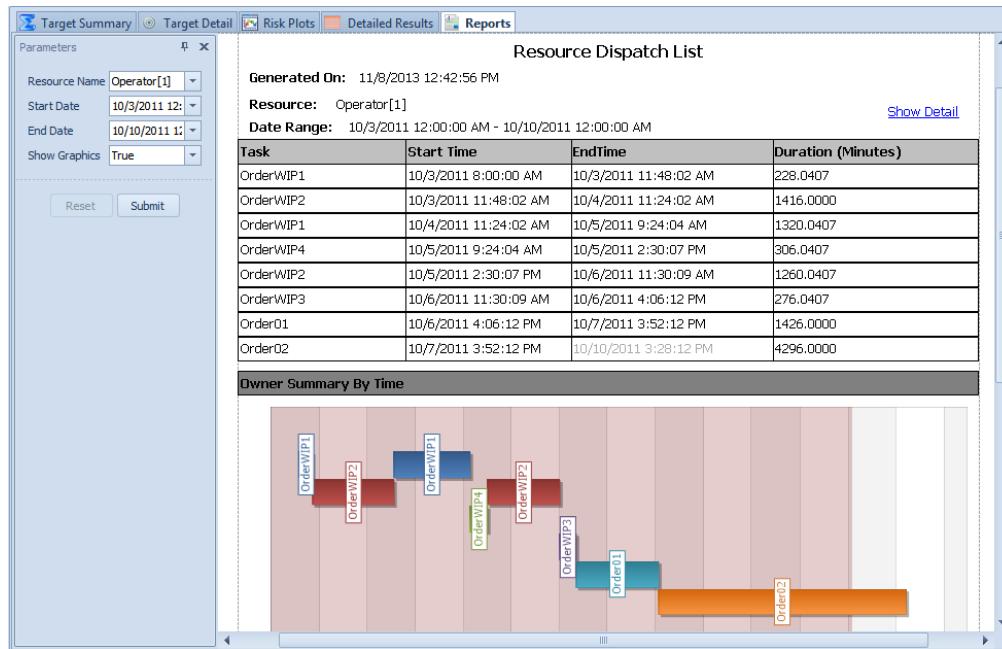


図 12.10 作業者に対するリソースの作業手配リスト

12.11 Model 12-1：スケジューリングへのモデルファーストアプローチ

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、新しい施設や、MES のようなサブシステムがまだ開発中で、モデル構成データがすぐに利用できない施設に最も適している。このアプローチはモデリングに時間がかかるが、外部データの要件が低いので、シミュレーションに基づく設計の改善を、早い段階で大きな損失なしに実行できる（12.4 節を参照）という利点がある。

まずは、簡単なモデルを構築し、計画機能を有効にして、生成された計画を確認することから始める。次に、未決注文のファイルをインポートし、そのデータファイルを使用できるようにモデルを変更する。そして、スケジュールリスクを評価できるように、モデルをもう少しカスタマイズして、出荷予定日と出荷目標日の追跡を追加する。最後に、スケジュールの評価と使用に役立つ組込み分析ツールについて簡単に説明する。

12.11.1 簡単なスケジューリングモデルの構築

New Project を立て、1 つの Source、3 つの Server、1 つの Sink を設置して、図 12.11 に示すように、Server に名前を付けて、Path で接続する。モデルを構築するときは、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する。そして、3 つの Server を同時に選択し、Advanced Options プロパティグループから Log Resource Usage の設定を True にする。さらに、Run リボンの Ending Type を Run Length : 1 hour に設定する。

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 (左上のボタンをクリック) を見ると、各リソースがリストに記載されており、右側には各リソースのアクティビティが見える。具体的には、各リソースでの各エンティティの処理開始時間と終了時間が分かる。Gantt リボンの Zoom In や Zoom Range 機能を使って、Gantt ビューの時間軸上にマウスをスクロールさせるだけで、図 12.12 のように、より詳細にアクティビティを確認することができる。

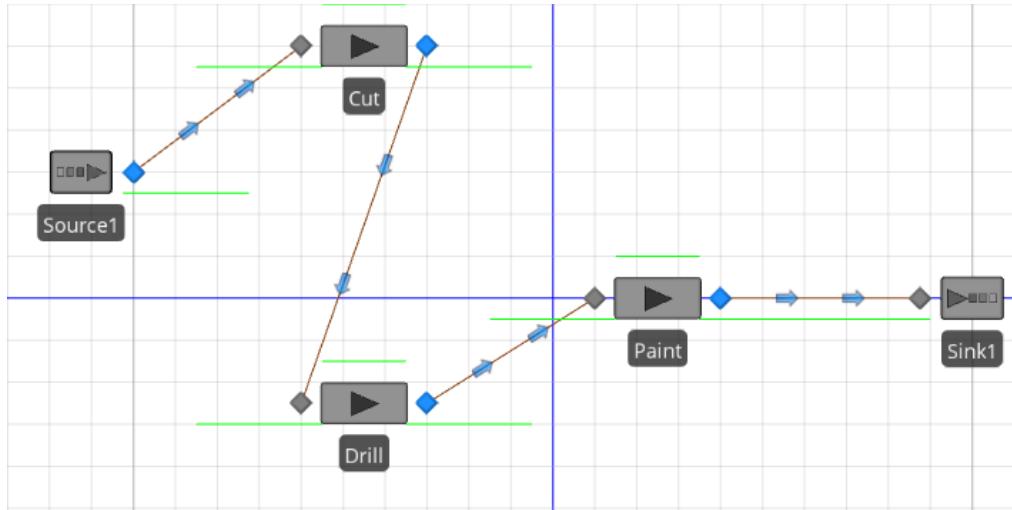


図 12.11 Model 12-1 の Facility ウィンドウ

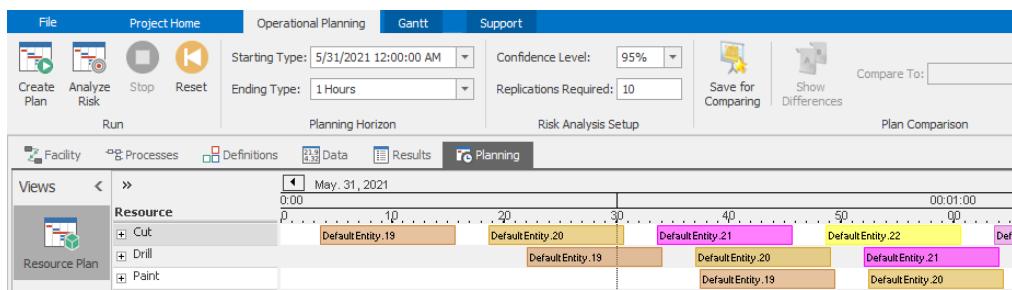


図 12.12 Model 12-1 の Resource Plan を拡大: エンティティの詳細を表示

左側の Entity Workflow ボタンをクリックすると、すべてのエンティティについて、それぞれの行に、どのリソースが使用され、各リソースでの処理開始時間と終了時間が Gantt ビューで表示される。ここでも Zoom 機能を使用して、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一覧表示できる（図 12.13）。エンティティ ID は（数値ではなく）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 の間になる（おそらく直観的ではない）。後で対処法について説明する。

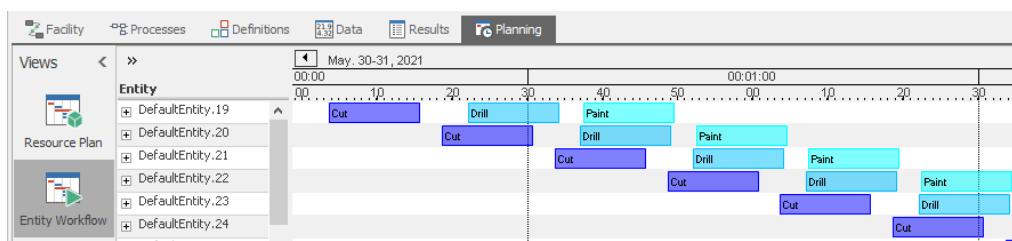


図 12.13 Model 12-1 の Entity Workflow を拡大: リソースの詳細を表示

12.11.2 より現実的なモデルの作成

生産したい注文のデータファイルとより適切な処理時間を使用して、モデルをより現実的にしてみよう。まず、Data タブの Tables ビューに戻り、Tables リボンに移動して、テーブルを追加する（Add Data Table ボタンを押す）。この新しいテーブルに ArrivalTable という名前を付ける。students ダウンロードファイルにある Model_12_01_DataFiles フォルダに ArrivalTableData.csv という名前の CSV ファイルがある。Tables リボンの Create Binding ボタンを用いて、この CSV ファイルを新しいテーブルに結びつける。これにより、テーブルとファイル間にリレーションシップを確立し、同時にデータを導入するためのスキーマ（列定義）を作成した。この操作によって、ファイルをテーブルに Import できる（図 12.14）。このテーブルに

は実際の日付が記載されているため、モデルの実行期間を実データの期間中に設定する。最初の注文到着に合わせて、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 ArrivalsTable

Facility ビューで Source1 オブジェクトを選び、新テーブルを用いてエンティティを生成するように設定する。Arrival Mode を ArrivalTable、Arrival Time プロパティを ArrivalTable.ArrivalTime にする。サーバの処理時間もさらに現実的にしてみよう。3つのサーバを一緒に選択して、Processing Time の Units を Minutes から Hours に変更する。最後に、エンティティをテーブルにある OrderID で区別するため、ModelEntity を Facility ビューにドラッグして、プロパティを編集する。Advanced Options カテゴリの Display Name を ArrivalTable.OrderId に、Animation カテゴリの Dynamic Label Text を ArrivalTable.OrderId にする。

上述した一連の拡張を終えた後、結果を見てみよう。Planning タブの Entity Workflow Gantt に戻して、前回の実行から変更のある部分を示した赤い色のバーが表示されている。Create Plan ボタンをクリックして、ガントチャートを更新しよう。新しい時刻に調整するためにもう一度ズームすると、ArrivalTable での識別されているように、より意味のある名前で一覧表示される（図 12.15）。また、Resource Plan では、各リソースは以前の表示のままだが、エンティティにはより意味のある名前が付けられる（図 12.16）。

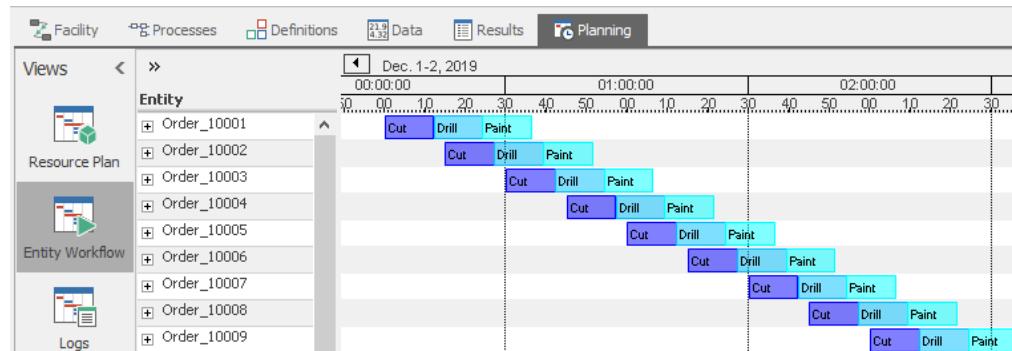


図 12.15 時刻とラベルを調整した Model 12-1 の Entity Workflow Gantt

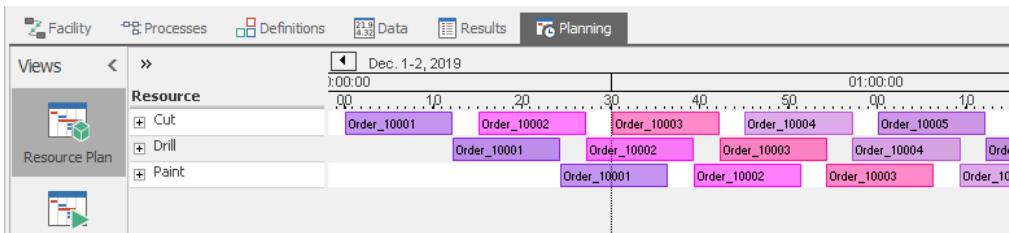


図 12.16 時刻とラベルを調整した Model 12-1 の Resource Plan Gantt

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、スケジュールがどの程度うまく運用されているのかを評価するのに役立ついくつかの機能を追加しよう。重要な項目の一つは、各注文の出荷予定日を記録することである。まずは State をテーブルに追加する。テーブルに戻り、States を選択する。状態変数 DateTime を選択してテーブルに追加し、この列に ScheduledShipDate という名前を付ける。この列は実行中に値が代入される出力列なので、最初はデータがないことを示すエラーが表示される。注文が完成したときに向かう Sink1 でその値を代入する必要がある。Sink1 の State Assignments セクションで、ArrivalTable.ScheduledShipDate に TimeNow を代入する。

もう一つの重要な項目は、注文が時間通りに出荷できるかを評価することである。これを行うには、テーブルにターゲット (Target) を追加する。Table に戻り、Targets リボンを開く。Add Targets ボタンをクリックすると、ターゲットの「Value (値)」と「Status (状態)」の2つの列がテーブルに追加される。このターゲットに TargetShipDate という名前を付ける。評価したい式は、Data Format が DateTime の ArrivalTable.ScheduledShipDate で、比較したい値は ArrivalTable.ExpectedShipDate である。この値を超えたくないので、上限 (Upper Bound) にする。わかりやすくするために、通常はデフォルトの用語を置き換える。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 になっていることがわかる。

システムのバランスを取るために、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、テーブルをもう一度見てみよう。今回、私たちの注文はほぼすべて On Time になっている。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。そのフラグが最後オペレーションの右側にあるとき、正の余裕時間を示す（つまり、注文が目標より早く完成できる）。しかし、その結果が確実か否かはまだわからない。Analyze Risk ボタンをクリックすると、モデルは変動性のある複数回の反復実行が行われ、フラグの色が変わり、注文が On Time になる尤度が表示される。

このモデルはまだ多くの変動性を含んでいないので、3つの部分に変動を追加しよう。まず、注文到着時刻に変動性を加える。注文が通常よりも最長15分早く、最長30分遅れて到着するため、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 が他の機械よりも少し処理時間長く、かつ時間予測が難しいと仮定し、Processing Time を Random.Exponential(0.2) に設定する。最後に、3台のサーバのすべてに信頼性の問題があるとしよう。Reliability でそれぞれに Calendar Time Based の故障を追加し、Uptime Between Failure と Time To Repair はデフォルトのままにする。Entity Workflow Gantt に戻り、Analyze Risk をクリックすると、確定的な計画ではほとんどの注文が予定通りに完成されたが、変動性を考慮すると、実際には注文は予定通りに完成できる可能性が低いと分かる（図 12.17）。

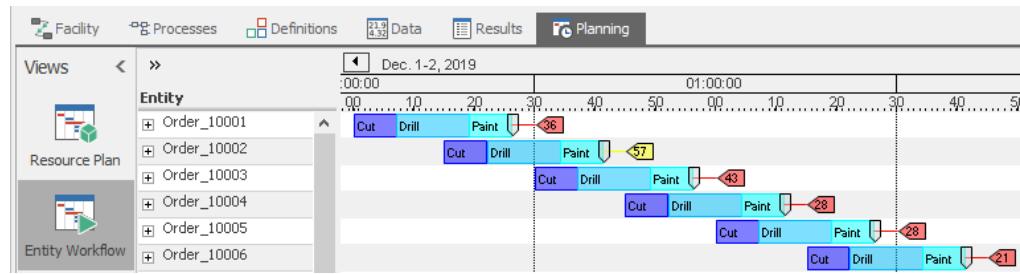


図 12.17 変動性を有する Model 12-1 のリスク分析

12.11.4 その他の評価ツール

シミュレーションによるスケジューリングの実行によって得られたすべての結果と出力データは、Planning タブに表示される。分析ツールを詳細に説明することは本書の範囲を超えており、読者自身で体験することをお勧めするいくつかのハイライトについて紹介する。

左側のパネルには、Scheduler モードで利用可能なツールが表示されている。2つのガントチャートについてはすでに簡単に説明した。しかし、計画されたスケジュールを評価し改善するのに役立つ、ガント関連の機能が他にも数多くある。たとえば、どちらかのガントチャートにあるアイテムをクリックすると、Properties ウィンドウにオーダーの関連情報が表示され、その上にマウスを移動すると、特定のタスクに関する詳細情報が記載されたポップアップが表示される(図 12.18)。

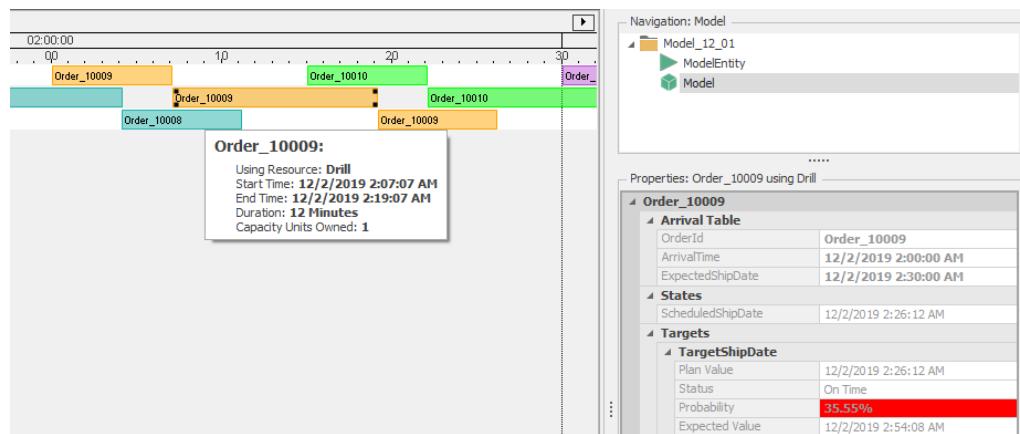


図 12.18 ガントチャートのポップアップとプロパティ

これまでの例のモデルでは必要としないが、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が(Constraints のように)表示される。また、Gantt リボンのボタンで、表示するオプションデータを選択できる。以上述べた 2 点は図 12.19 に示されている。

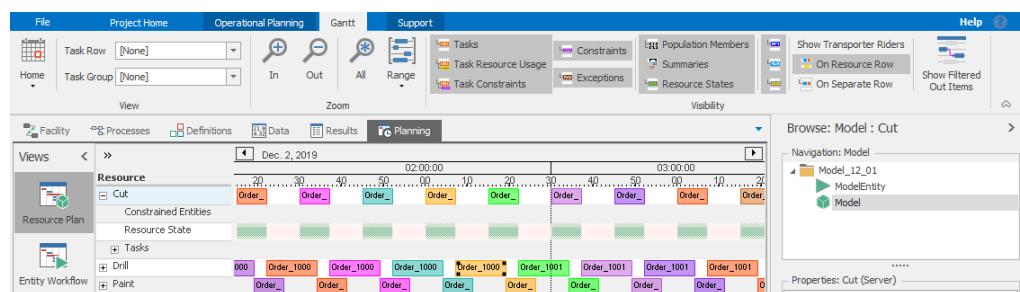


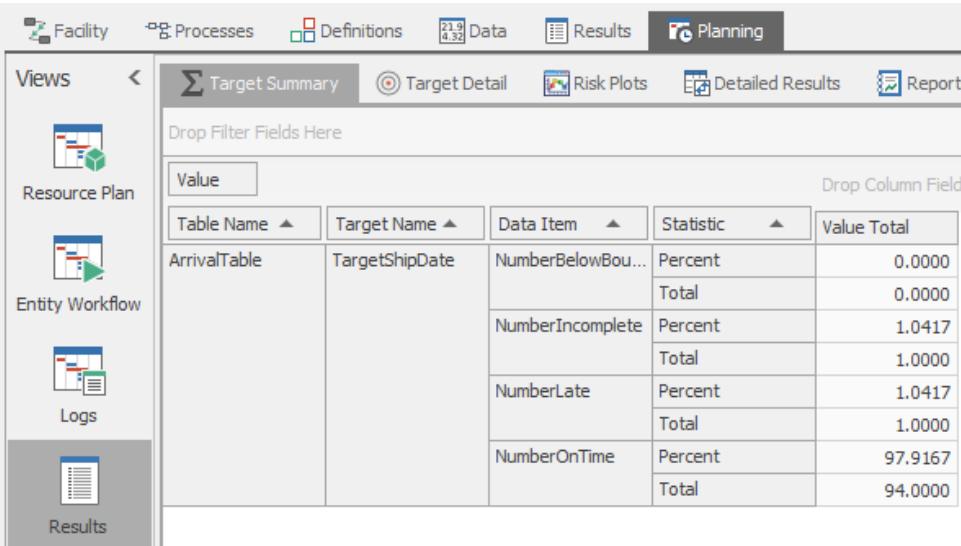
図 12.19 ガントチャートにおける追加的な行データの表示

左側の 3 番目のボタンは Logs (ログ) であり、リソース関連、制約、トランスポータ、原材料および統計など、10 種類のカテゴリのすべてのイベントを記録および表示する。ガントチャートや後述する分析機能の多くは、これらのログのデータに基づいている。ログをフィルタして読みや

すくしたり、データを識別しやすくするためにログに列を追加することもできる。

左側の Tables ボタンをクリックすると、モデルの Data タブですでに利用したテーブルと類似した、タブ付き表示が生成されている。主な違いは、モデル作成者が、各テーブルのどの列を表示するか（逆に、計画立案者に対して非表示にするか）や、どの列を計画立案者が変更できるか（あるいは表示専用にするか）を選択できることである。これを用いてテーブルを簡素化し、計画立案者が許可されていない変更（たとえば、出荷予定日の変更など）をすることからシステムを保護することができる。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を示す列が追加される。

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 に示している Target Summary や、Target Detail および Risk Plots にも、さらに詳細なリスク分析データが表示される。Detailed Results タブには、インタラクティブな実行で使用可能なものとよく似たピボットテーブルが表示される。Reports、Dashboard Reports および Table Reports では、計画立案者や他の利害関係者のニーズに合わせた定義済みのカスタムレポートや、対話型のダッシュボードを表示する機能が提供される。



The screenshot shows the SAP Leonardo interface with the 'Planning' tab selected. On the left, there's a sidebar with 'Views' and four icons: 'Resource Plan', 'Entity Workflow', 'Logs', and 'Results' (which is highlighted). The main area has tabs at the top: 'Target Summary' (selected), 'Target Detail', 'Risk Plots', 'Detailed Results', and 'Report'. Below the tabs is a section titled 'Drop Filter Fields Here'. A table titled 'Value' is displayed with the following data:

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 タブにおける Target Summary の表示

まとめると、以上のツールを使用すると、計画立案者は自分の計画がなぜ現在のように実行されているのかを理解し、改善の可能性を分析し、結果を他者と共有できるだろう。

12.12 Model 12-2：スケジューリングへのデータファーストアプローチ

12.11 節では、モデルファーストアプローチで部分的にデータ駆動型の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議論した。後者のアプローチは、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適している。このアプローチの大きな利点は、ベースモデルをはるかに早く作成できることである。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があるので、B2MML データファイル (7.8 節) からモデルを構築し、そのモデルをどのように強化できるかを探求しよう。

図 12.21 に示すように、モデル化しているシステムには、4つの工程に対してそれぞれ機械 2 台があり、各製品には独自の工程順序がある。まず、組込みツールを用いてデータテーブルを設定し、インポートしたデータで使用される定義済みオブジェクトを用いてモデルを構成する。それから、テーブルを生成するために B2MML データファイルのセットをインポートする。データの分析に役立てるために、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。

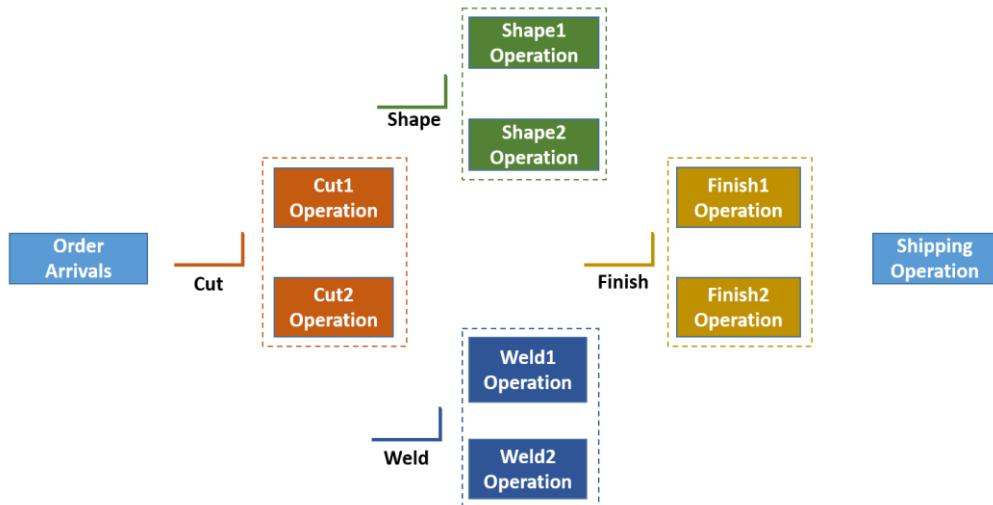


図 12.21 Model 12-2 のシステム概要

12.12.1 データインポートのためのモデル設定

Simio B2MML 準拠テーブルは以下のとおりである：Resources、Routing Destinations、Materials、Material Lots、Manufacturing Orders、Routings、Bill Of Materials、Work In Process、Manufacturing Orders Output。上述したテーブルを全部作成し、最後のテーブル以外のすべてをインポートするが、その前に、インポートのためのモデルを構成する。これを行うには、File ボタンを押し、New From Template を選ぶ。この操作で図 12.22 に示すテンプレート一覧が開く。

ISA95SchemaProductBasedRoutings テンプレートは B2MML に準拠したテーブルを生成するので、このモデルにはこのテンプレートを選択する。ISA95SchemaProductBasedRoutings 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、および Manufacturing Orders Output テーブルが表示される。

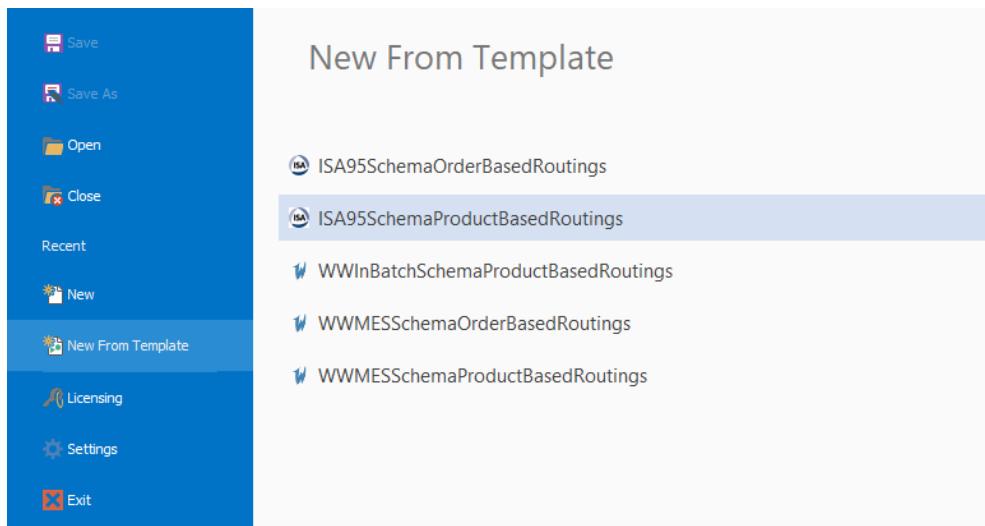


図 12.22 New From Template のテンプレート一覧

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 テーブルを選択して、Create Binding オプションで CSV を選び、student ダウンロードにある Model_12_02_DataFiles フォルダから

Resources.csvを選択する。次に、TableリボンのImportボタンを押す。Facilityビューに移動すると、リソースがモデルに追加されたことがわかる。

では、Dataタブに戻る。他の7つのテーブルについて上記のプロセスを繰り返し、関連するCSVファイルにバインドしてからインポートする。Facilityビューに戻ると、完成したモデルが表示される。図12.23のナビゲーションビューには、Configure Scheduling Resourcesボタンをクリックして、このモデルに追加されたオブジェクトが示されている。Shape1オブジェクトが選択されており、SchedServerオブジェクトであること、およびWork Schedule、Processing Tasks、Assignmentsなどの多くのプロパティがテーブルから直接データ抽出されるように事前設定されていることがPropertiesウィンドウに表示されている。

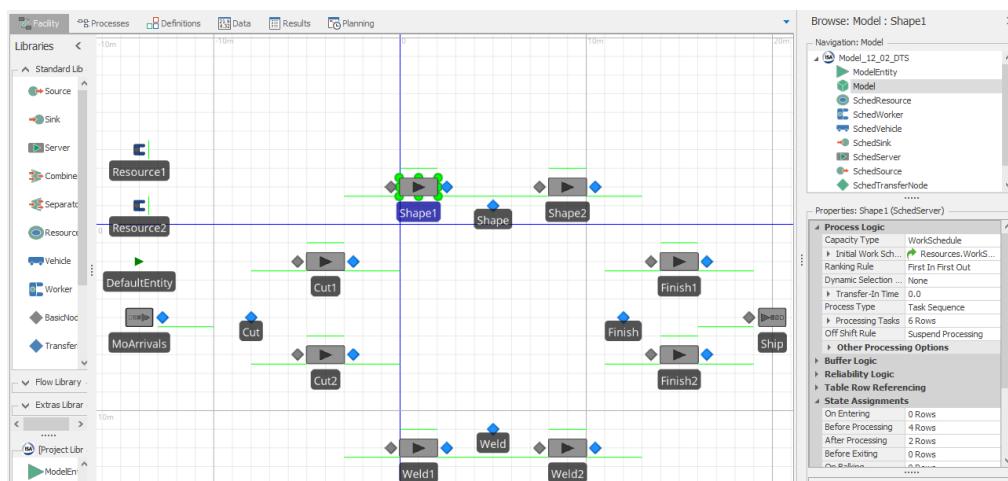


図12.23 データのインポートが完了したModel 12-2

12.12.3 モデルの実行と分析

私たちのモデルはデータファイルだけで構築されており、モデルをインタラクティブに実行してアニメーションを見ることができる。実行する前に、注意しなければならない重要な点がある。先ほど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。そのため、現在の日付でモデルを実行すると、すべての注文がかなり遅れることになる。一番簡単なのは、モデルの開始タイプ（開始時間）を12/2/2017 8:00に変更することである。

このモデルを使用してスケジューリングを行う前に、RunリボンのAdvanced Optionsに移動してEnable Interactive Loggingを選択する必要がある。使用した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オプションが設定されている。これで、Planningタブに移動して、Create Planボタンをクリックすると、ガントチャートや前述した他の分析が生成される。

このようなデータスキームで動作するように設計されたいくつかの定義済みダッシュボードをインポートしよう。これらのダッシュボードはXMLファイルとして保存され、CSVファイルと同じフォルダにある。3つのダッシュボードには、原材料の詳細、オーダーの詳細、およびオペレータが使用する差立リストがある。ダッシュボードをインポートするには、ResultsタブのDashboard Reportsウィンドウ（PlanningのResultsウィンドウではない）に移動して、Dashboardsリボンを選択する。Importボタンを押して、以前と同じフォルダからDispatch List.xmlファイルを選択する。Materials.xmlファイルとOrder Details.xmlファイルにもこの手順を繰り返す。Planningタブ→Resultsウィンドウ→Dashboard Reportsサブタブに戻ると、表示する3つのレポートのいずれかを選択できる。図12.24には、Order Detailsダッシュボードレポートを示す。

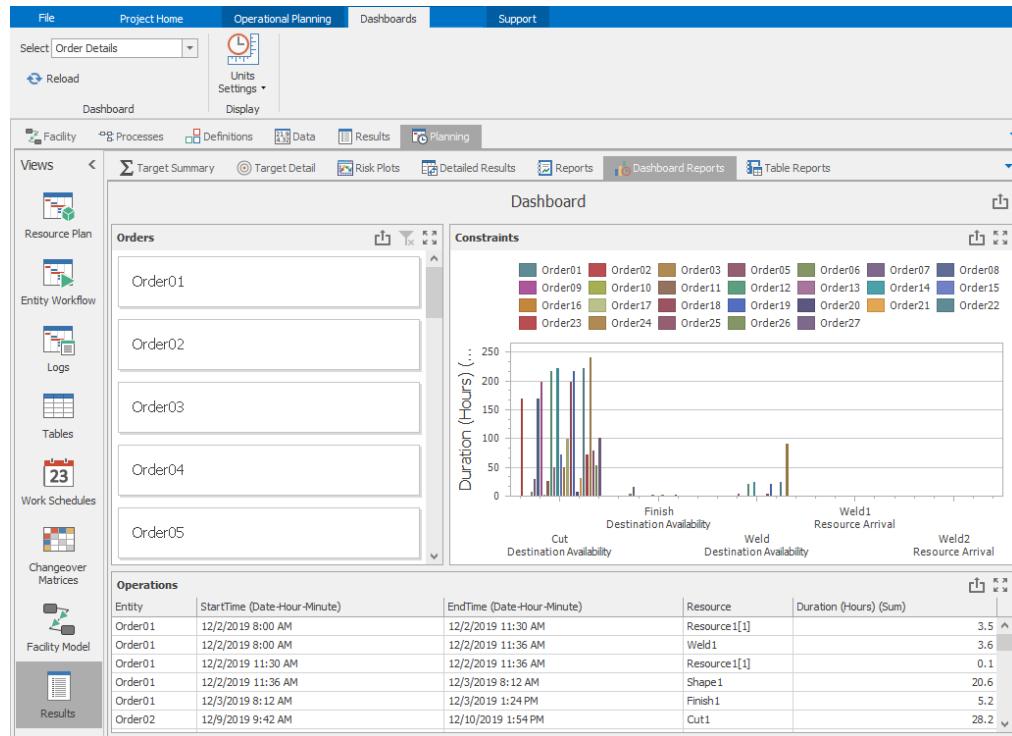


図 12.24 Model 12-2 の Order Details ダッシュボードレポート

最後に、いくつかの伝統的な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) に移動して Table Reports リボンを選択する。ManufacturingOrdersOutput の Import ボタンを押し、以前と同じフォルダから Dispatch List Report.repx ファイルを選択する。同じような手順で ManufacturingOrders に OrderDetails.repx ファイルをインポートする。この 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 サブタブに戻ると、2 つの新しいカスタムレポートのどちらかを表示用に選択できるようになっている。

この例は明らかに簡単な例ではあるが、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する可能性を示している。このアプローチにより、比較的少ない労力で初期の機能的なモデルを提供することができる。また、よりよいソリューションを提供するために、さらに詳細な設定とロジックでモデルを強化することもできる。これは非常にパワフルなアプローチである。

12.13 追加情報と例

12.13.1 Simio E-book による計画とスケジューリング

Simio の Web サイトには、電子書籍 Manufacturing Digital Twin, SolutionDescription がある。これは、Textbook Publication ページ (<http://textbook.simio.com/>) あるいは Support リボンの Books ボタンで確認できる。これは、シミュレーションベースのスケジューリングの学習を継続するのに最適である。その本では、標準的なデータスキーマと多くの一般的なスケジューリングの概念、およびそれぞれが Simio でどのように扱われているかについて説明している。

また、同じところに電子書籍 Deliver On Your Promise - How Simulation-Based Scheduling will Change Your Business もある。その本は、スケジューリングの複雑なプロセスについての理解を深めたい管理者に最適である。本章にあるいくつかのトピックについての詳細と、いくつかのケーススタディについて説明している。スケジューリング問題を解決しようとしているマネージャと、この pdf (またはオンラインで利用可能な印刷版) を共有することをお勧めする。

12.13.2 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含まれており、それぞれの付属の pdf ファイルに内容が記載されている。それらのファイルは、Support リボンの Examples ボタンで確認できる。

12.13.2.1 個別部品の生産計画

この例でスケジュールされるシステムは、個別部品製造工場である。工場は完成品を生産するジョブショップである。3つの完成品が生産され、それぞれは独自のルーティングとセットアップと処理時間、そのルーティング内の各機械における資材所要量を有している。限られたシステムリソースをじゅうぶんに考慮した30日間の生産スケジュールを作成したいと考えている。この工場は、スケジューリングルールに基づいて動的に移動する部品と、機能的にグループ化された機械で構成されている。一部の機械では、セカンダリリソースが必要な場合や、工程順序に依存するセットアップ時間を有する場合がある。スケジューリング概要で説明した B2MML ベースのデータスキーマが、この例の生産データを保持するために使用される。

12.13.2.2 自転車組立のスケジューリング

この例でスケジュールされるシステムは、自転車組立工場である。この工場は、最終的な組立ラインで完成品を製造するジョブショップであり、顧客の注文に基づいて最終組立のための部品も製造する。このスケジュールには、最終組立に必要な多数の購入部品も含まれている。システム内の限られた資源と部品をじゅうぶんに考慮して、この工場の7日間の生産計画を作成したい。

12.13.2.3 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。そして、販売オーダーから、システム内の初期在庫を基に生成される、**製造オーダー** (Manufacturing Orders, MO) と**購買オーダー** (Purchase Orders, PO) を決定する。この工場では、システム内の限られた資源および部品をじゅうぶんに考慮した14日間の生産スケジュールを作成したい。

12.13.2.4 飲料バッチ生産のスケジューリング

この例では、限られたリソースをじゅうぶんに考慮して、工場の30日間の生産スケジュールを生成したい。この例には、材料と完成品の両方のオーダーが含まれている。このシステムでは原材料も制約としてモデル化されている。3つの材料は、ミキサーで混合され、次いでタンク (Tank) に入れられる。完成品は、充填機 (Filler) から処理が始まり、次に包装機 (Packing) で梱包される。プロセスのこれらの各ステップで作業者が必要であり、完成品を処理するためには、適切な材料がタンク内で利用可能でなければならない。ボトルやラベルなどの原材料もこのシステムでモデル化され、完成品の充填プロセスの一部として必要とされる。

12.13.2.5 労働力の効率のよいスケジューリング

この例では、効率が異なる労働者が働く工場のオーダーをスケジューリングするために、データから生成されたモデルを構築する。このジョブショップは、4種類の完成品を生産している。このモデルでは、システム内の限られたリソースをじゅうぶんに考慮した6ヶ月の生産スケジュールを作成する。この工場は、Press、Anodize、Mill、Paint と名付けられた機能的にグループ化された4つのワークステーションから構成される。それぞれに待機場所があり、そこから数台の機械にオーダーがルーティングされるようになっている。このモデルの構築を開始するために使用したテンプレートは、Standard Library オブジェクトの Subclass 機能を使用して作成した独自のオブジェクト定義を提供する。これらは、SchedSource、SchedServer などと呼ばれ、特にスケジューリ

ングモデルで使用するために設計されている。この例は、オブジェクトデータをデータテーブルに格納することでモデルが自動的に作成される、**データ駆動型モデリング**の概念を示している。

12.13.2.6 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」と似ているが、1つの部品ルーティング内で発生し得るすべての並列アクティビティを処理するために使用される、Server としてモデル化された**並列ルート制御**の活用を示している。モデルの初期化には、`PopulateImmediateSuccessorSequences` プロセスが含まれ、所与の部品種類のルーティングにおける各ステップの後続順序番号を示す `Immediate Successor Sequences` 出力テーブルが生成される。`ParallelRouteController` サーバ内で使用される `ParallelRouteController_TaskProcessName` プロセスは、並列で処理されるアクティビティの完了を待つルーティング内のさまざまなステップに対してエンティティを生成する。各オーダー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 `ParallelRouteController` サーバに存在する。

12.14 まとめ

本章では、インダストリー4.0 とその起源、そして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 プロジェクトの分析におけるシミュレーションの従来の役割と、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シミュレーションの新たな役割についても説明した。計画とスケジューリングにおける一般的なアプローチや、その長所と短所について検討した。それから、リスクに基づく計画とスケジューリング (RPS) に対してシミュレーションを利用し、計画／スケジュールにおける問題を指摘し、スケジューラに追加した新たな機能を紹介した。また、RPS を活用するための Simio の機能を簡単に解説した。部分的にデータ駆動を活用する伝統的な方法で、1番目の例を作った。そして、データ駆動型であるだけでなく、モデル自体が完全に B2MML 準拠のデータファイルから構築された（データから生成された）2番目の例を構築し、自動的なモデル構築の力を示した。最後に、Simio ソフトウェアに含まれている3つの大きな例題で、さらに多くの概念が説明されることで締めくくった。ここでは、RPS についてごく簡単にしか説明できなかったが、読者の関心を喚起できたとすれば幸いである。RPS の活用については、他にも多くの利点があり、特にデジタルツインの設計、評価および実装に関して、新しい応用分野も広がっている。商用アプリケーションに加えて、RPS は学生プロジェクトや研究調査に対して、有望な領域であるだろう。

12.15 問題

1. インダストリー3.0 とインダストリー4.0 を比較しなさい。
2. デジタルツインとは何か？デジタルツインの構成要素とその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アプローチに関する共通の問題と、シミュレーションと RPS がこれらの問題に対処する方法について説明しなさい。
4. Simio ダッシュボードは、Simio テーブルレポートとどのように違うか？
5. Model 12-1を開いて（または再構築して）、Cut と Drill 工程間の部品移動に分速4フィートの AGV（車両）2台を追加する。リソースおよびエンティティのガントチャートから、AGV によって移動を制約された最初のエンティティを見つけ、スクリーンショットを撮りなさい。
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により、Cut 3 という名前の機械を Cut 工程に追加しなさい。スケジュールパフォーマンスの変化を判断するには、Planning タブの Operational Planning リボンにある Save for Compare と Show Differences 機能を使う。また、変更の影響を示す Target Detail レポートのスクリーンショットを撮りなさい。
7. 問題 6 に、任意の機械をもう1台追加できるとしたら、何を追加するか？それはなぜか？

付録 A Simio を活用したケーススタディ

本章には、システムを分析するための Simio モデルにおける開発と活用に関する 4 つの「導入」と 2 つの「発展」のケーススタディが含まれている。これらの問題はより範囲が広く、これまでの問題のように詳細に記述されていない。最初の 2 つのケースは、サンプルモデルに基づいていくつかの分析結果も示している。しかしながら、これまでの章のモデルと異なり、モデルそのものを詳しく説明しない。

最初のケース (A.1 節) では、機械加工、検査および清浄作業の生産システムであり、提案された 2 つの改良に伴う現在のシステム構成の分析を実施している。2 番目のケース (A.2 節) では、遊園地を分析しており、「ファストパス」チケットオプションと、それが待ち時間に与える影響について考察する。3 番目のケース (A.3 節) では、レストランをモデル化しており、スタッフ配置と容量における重要な課題に関して考える。4 番目のケース (A.4 節) は、ある銀行支店について考察しており、スケジューリングや設備選択が費用に与える影響についての問題を扱っている。

2 つの発展的なケーススタディは、大規模で現実的な問題を表すことを意図しており、アカデミックではない設定に遭遇するかもしれない。「本当の世界」における問題は、典型的な課題問題のようにうまく形成され、完全できちんとしていることはない。多くの場合は、欠落したり、あるいはあいまいな情報であったり、微妙な立場であったり、そして問題の解釈によって複数の解が存在することに直面する。これらのケースと同じような状況に出会うかもしれない。これらの状況に遭遇するときは、妥当な仮定を設定し、記録しておくべきである。

授業に最適なケーススタディの動画は次の URL にある：

<https://www.simio.com/resources/webinar/>

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 を参照）を評価して、2 つの可能性のある変更について調査するように依頼されている。それらは両方とも、検査ステーションおよび 1 人の検査員を、自動化された検査工程に交換することが含まれている。提案された変更の目的は、システムパフォーマンスを改善することである。

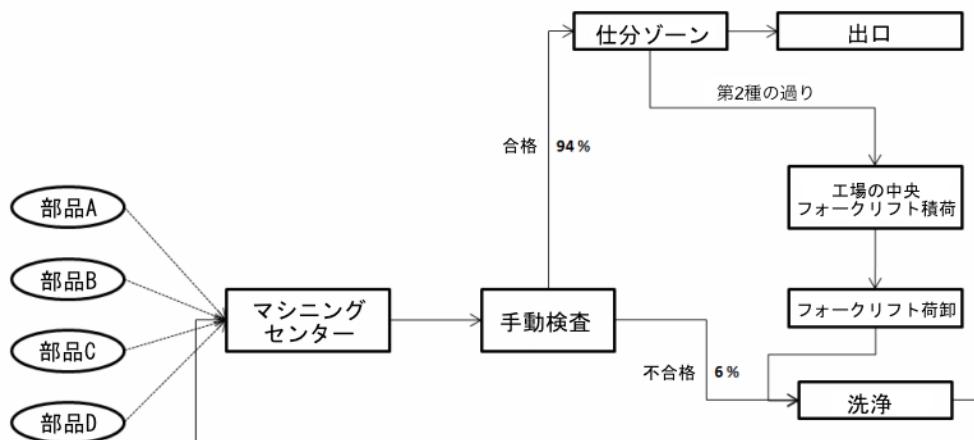


図 A.1 機械加工と検査工程レイアウト

現在のシステムでは、4 つの異なる部品タイプが自動化されたマシニングセンターに到着する。そこでは、一度に 1 つの部品が処理される。機械加工を完了した後に、部品は人間の検査員によっ

て一つひとつ検査される。検査工程では、次の誤りが生じやすい。すなわち、「合格 (good)」の部品を「不合格 (bad)」として分類してしまう「第 1 種の誤り」、あるいは「不合格 (bad)」の部品を「合格 (good)」として分類してしまう「第 2 種の誤り」である。検査後に「合格」として分類された部品は、工場の別のゾーンに送られて加工される。一方、「不合格」として分類された部品は、自動洗浄機 (1 個の容量) で洗浄され、それからマシニングセンターで再加工されなければならない (新しい部品と同じ処理パラメータを用いる)。

その他のシステムの詳細は以下の通りである：

- 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。
- すべての部品は、機械加工に続いて検査に進む。部品を検査する時間は、部品タイプから独立していて、平均 2 分の指數分布に従う。検査された部品の 6% が「不合格」となることがわかっており、それらは洗浄され、再加工されなければならない (これは、洗浄・再加工される回数から独立している)。洗浄された後に、検査に戻った部品は、機械加工待ち行列で「新しい」部品よりも優先権を持つ。各部品を洗浄するには 12 分かかる (洗浄時間も部品タイプから独立している)。
- 「合格」と判断されて、別のゾーンに送られるすべての部品のうち、3% は後に「不合格」と判断されることがわかっている (第 2 種の誤り)。これらの部品は、工場の中央にあるパレットにおかれる。そして、フォークリフトが定期的にそのパレットを回収し、洗浄機械まで届ける。これらの部品は自動洗浄機の待ち行列において優先される。フォークリフトのピックアップ時間の間隔は平均 3 時間の指數分布に従う。
- 「不合格」と判断されて自動洗浄機に送られるすべての部品のうち、7% は実際には合格であり、洗浄と再加工の必要がない (第 1 種の誤り)。表 A.2 で、不要な洗浄と処理に対する 1 部品当たりの費用を示している。
- マシニングセンターと洗浄機械は共にランダムな時間依存の故障を被りやすい。どちらの機械も、アップタイム (故障時間間隔) は平均 180 分の指數分布に従い、修理時間は平均 15 分の指數分布に従う。機械のどちらかが停止するときに部品が処理されているなら、処理は中断されるが、部品は破棄されない。
- システムは 1 日当たり 2 シフトで、1 週間当たり 6 日間操業する。それぞれのシフトは長さ 8 時間である。そして、作業は前のシフトの終わりに残されたものが引き継がれる (すなわち、シフトの始めに、始動効果がまったくない)。

表 A.1 到着と機械加工データ

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

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的洗浄	\$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

評価依頼された 2 つのシナリオは、共に検査工程の自動化に関係している：

1. それぞれの部品タイプで、4 台の「単純な」ロボットを設置し活用する。これらのロボットは、一度に 1 つの部品を検査する。第 1 種および第 2 種の誤りの確率は、それぞれ 0.5% と 0.1% である。それぞれの「単純な」ロボットが部品を検査する時間は、部品タイプから独立しており、パラメータ(5.5, 6.0, 6.5) 分の三角分布に従う。
2. 4 種類すべての部品を検査できる 1 台の「複雑な」ロボットを設置し活用する。しかし、一度に 1 つの部品しか検査できない。第 1 種および第 2 種の誤りの確率は、それぞれ 0.5% と 0.1% である。「複雑な」ロボットが部品を検査する時間は、部品タイプから独立していて、パラメータ(1.0, 2.0, 2.5) 分の三角分布に従う。

これらのロボットは、マシニングセンターおよび自動洗浄機と同じ会社によって製造されているので、このロボットも、機械加工と清浄機械でこれまでに与えられているのと同じ確率分布でランダムな故障を被りやすいと考えられる。分析で用いられる費用データは表 A.2 に示されている。すでにロボットを購入するための経済性分析をしており、表 A.2 には関連費用（購入費、設置費用、操業費、および維持費など）を 1 週当たりの費用として示している。

Simio シミュレーションモデルを構築し、どの検査プロセスを利用すべきか分析し決定する。結果には各シナリオで以下の推定値を含める：

- 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。
- 週当たりの処理を完了した（システムを離脱した）平均部品総数。

Simio モデルを用いて、以下の間に答える。答えはシミュレーション実験から得られる結果と分析に基づかなければならない。

1. ロボットの仕様を見た後、マネージャが「単純な」ロボットの検査の平均時間が、現在の工程のものよりはるかに高いのに気づいた。彼は、現在の検査工程を変える場合、実際に全体の生産（すなわち、1 週間当たりの完成部品数）を減少させるかもしれないと心配している。検査方法を変えると、各週で完成する部品数は変化するだろうか？ そうだとすれば、完成部品数が現在のシステムに対してどう変化するか（すなわち、増加／減少の量）を調べてほしい。
2. 3 つの検査手法の内、費用を最小にするためにどのプロセスを勧めるか？
3. 自動検査工程の 1 つまたは両方が、手動検査と同じくらい、あるいはより費用がかかることがわかった場合、どうすれば手動検査より週当たりの費用を低く操業することができるかを調べてほしい。すべての費用は固定であり、調整できないと仮定する。

A.1.2 サンプルモデルと結果

本項では、著者が構築したサンプルモデルの結果を示している。結果は仮定した数値に依存しているので、あなたの結果とは正確に合わないかもしれませんことに留意してほしい。サンプルモデルの Facility ビューを、図 A.2 に示している。

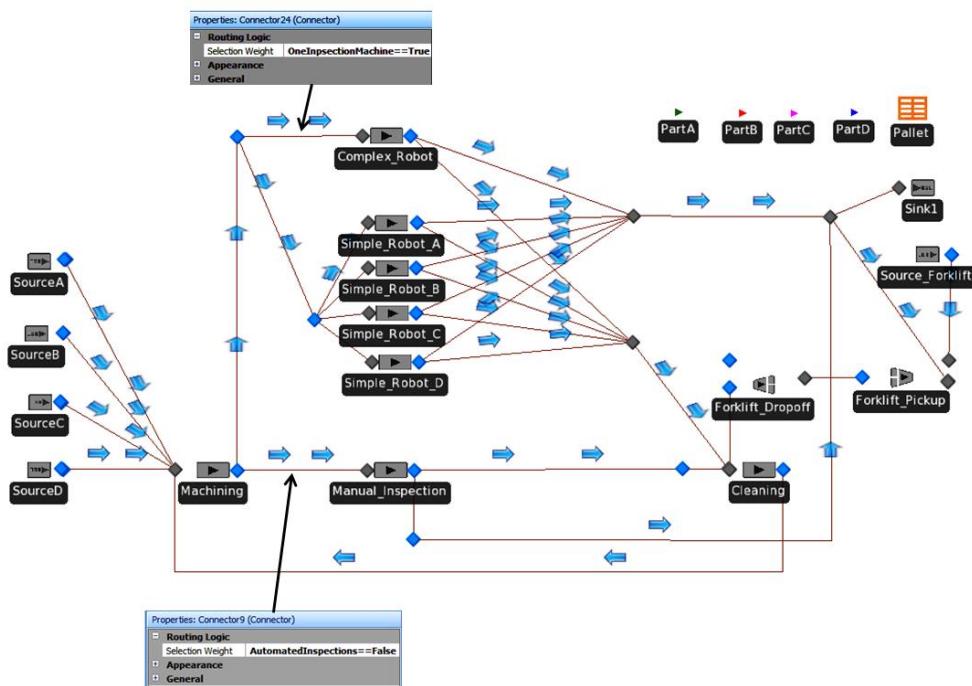


図 A.2 サンプル機械加工検査モデルの Facility ビュー

実験では、以下のコントロールとパラメータを用いた：

- ・ 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：
 - ・ Automatic Inspection（自動検査；論理演算子）
 - ・ One Inspection Machine（1 台の複雑な検査機械；論理演算子）
 - ・ Weekly Machine Cost（週当たりの機械加工費用）
 - ・ Inspection Worker Hours（検査作業者の時間）
 - ・ Warm-up（ウォームアップ）

図 A.3 は、異なるコントロールによって、特定のシナリオのアクティビティと総費用を決定するために、どのように Simio Experiment を用いたかを示している。分析において Simio のシナリオ比較機能を用いることができるよう、実質的に 3 つすべてのオプションを単一のモデルにプログラミングした。同じ実験において異なるシナリオを比較するために、論理演算子 Automatic Inspection と One Inspection Machine を用いた。コントロールに対応するボックスをチェックすると、True の値をプロパティに設定できる。異なる検査プロセスが処理される前に、コネクタの Selection Weight プロパティにおいて、これらの値をモデルが参照する。これについて、2 つの例が図 A.2 に示されている。

週当たりの総費用に用いる式は、以下の通りである：

```
TotalCost = WeeklyMachineCosts
+ WeeklyHoldingCost.Value
+ TotalTypeIErrors.Value * 5
+ InspectionWorkerHours * 50          (A.1)
```

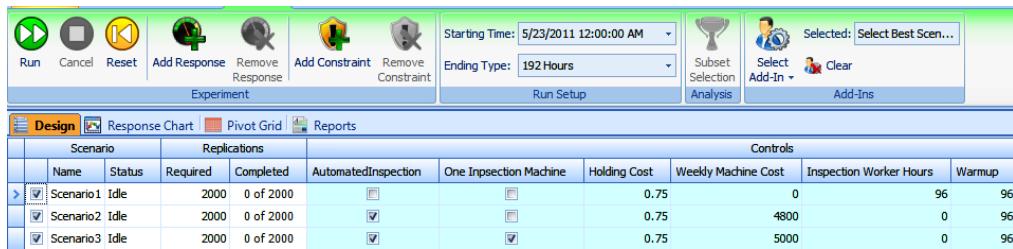


図 A.3 機械加工検査 Experiment Design のスナップ

式 A.1 の 4 つの項を計算するための方法は、以下の通りである：

1. **WeeklyMachineCosts** : A.I.I 項でこれらの費用を示した。プロパティ **WeeklyMachineCosts** 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** : これは、モデル状態変数 **HoldCost** を参照する出力統計の値である。式 A.2 が、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）ごとに状態変数 **HoldCost** を増加するのに用いられる：

$$\text{HoldCost} = (\text{TimeNow} - \text{ModelEntity.ArrToSys}) * 0.75 + \text{HoldCost} \quad (\text{A.2})$$

式 A.2 は、エンティティの総システム内時間を計算し、A.I.I 項で与えられた部品保持費用をその値に掛けている。そして、状態変数はこの値を用いて増加される。この手法を用いて、出力統計量 **WeeklyHoldingCost** はすべてのエンティティに関する保持費用の合計になる。

3. **TotalTypeIErrors.Value * 5** : 第 1 種の誤りに関連した費用として、A.I.I 項で \$5 の費用がかかるることを示した。**TotalTypeIErrors** は、モデル状態変数 **TypeIError** を参照する出力統計量である。第 1 種の誤りが検査工程で発生するときに、この状態変数が増加される。
4. **InspectionWorkerHours * 50** : A.I.I 項で毎時 \$50 の費用がかかるることを示した。プロパティ **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 とリストアップされた実験パラメータの式を用いて、各シナリオの週当たりの平均費用を計算した。これらの結果を表 A.3 に示している（すべての半幅は期待値の 95% の信頼区間にに対するものである）。表 A.4 に、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合計の平均を示す。週当たりに処理が完了した（システムを離脱する）平均総部品数が表 A.5 で示されている。

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種の誤りの平均数

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 95%信頼区間	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 95%信頼区間	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 95%信頼区間	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の平均部品処理数

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

I. 検査方法を変えると、各週で完成する部品数は変化するだろうか？そうだとすれば、完成部品数が現在のシステムに対してどう変化するか（すなわち、増加／減少の量）を調べてほしい。

簡単な待ち行列解析（定常システムであれば、流入と流出は等しいため、部品の期待数は到着率×処理時間で表される）でこの問の答えを得ることができるが、シミュレーション実験の結果を用いることもできる。図 A.4 における SMORE プロットからは、1 週当たりに処理された平均総部品数は、シナリオごとにあまり違いがないように見える。一方で、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が統計的に有意な差がないことを確かめるために、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）のそれぞれに対して、2 つの対応のある t 検定を実施した。これらのテスト結果が図 A.5 と図 A.6 に示されている。

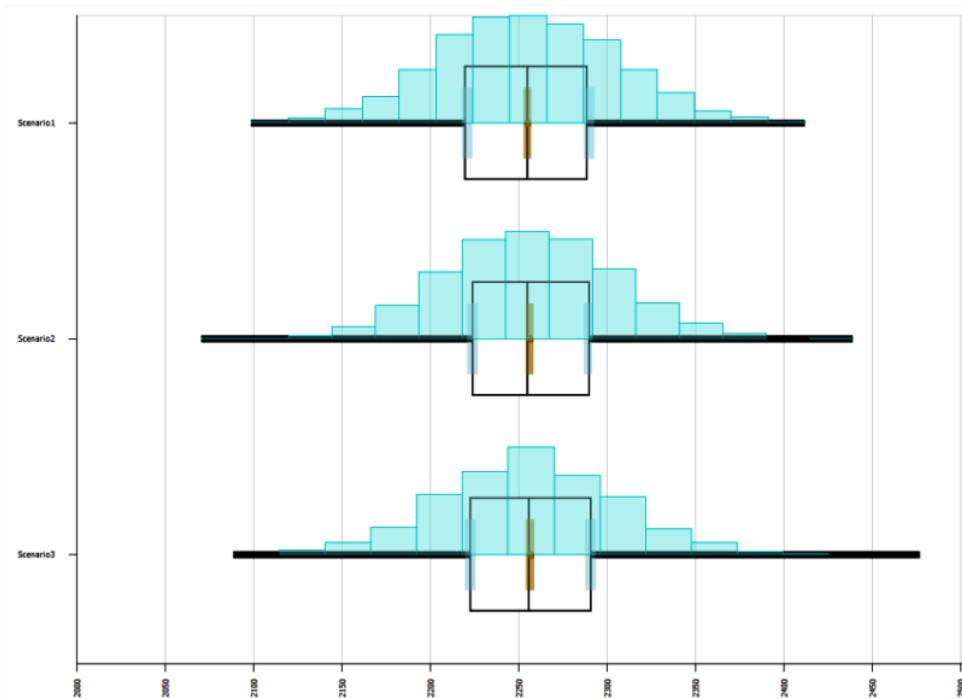


図 A.4 各シナリオにおける週当たり完成部品平均数の SMORE プロット

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)

平均の差に対する 95% の信頼区間 [-1.54, 4.38]において、ゼロを含んでいるため、シナリオ 1 と 2 の間には、週当たりに処理された部品の平均数に関して統計的に有意な差がない。シナリオ 1 と 3 も図 A.6 の比較から、同じ判断ができる。詳しくは、週当たりに処理された平均部品数に関して、平均の差に対する 95% の信頼区間が [-3.70, 2.38] でゼロを含んでいるため、シナリオ 1 と 3 の間にもまた統計的に有意な差がない。この統計的証拠によって、検査手法を変える場合、週当たりの完成部品数はあまり変化しないと結論を下すことができる。

これらのシナリオを比較する別のアプローチとして、Simio の **Scenario Subset Selection** 機能がある。表 A.6 に従ってレスポンスの目的関数を設定し、「Subset Analysis」を利用すると、Simio は順位付けおよび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)」と「Reject (棄却)」のサブグループに分類する。Experiment ウィンドウの Design ウィンドウで、「Reject」の場合は Response 列におけるセルが薄い茶色になり、「Possible Best」のシナリオは茶色のままとなる。

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

この分析法で得られた結果が図 A.7 である。Scenario Subset Selection を用いた Simio の分

析は、対応のある t 検定と同様の結果である。各シナリオのレスポンス TotalProcessed は薄い茶色になっていない。したがって、Simio の Subset Selection は、処理された部品の週当たりの平均数を最大にするための「Possible Best」として、3 つのシナリオのすべてを挙げた。この分析は毎週処理される部品数には著しい違いがないという予測を裏付けている。

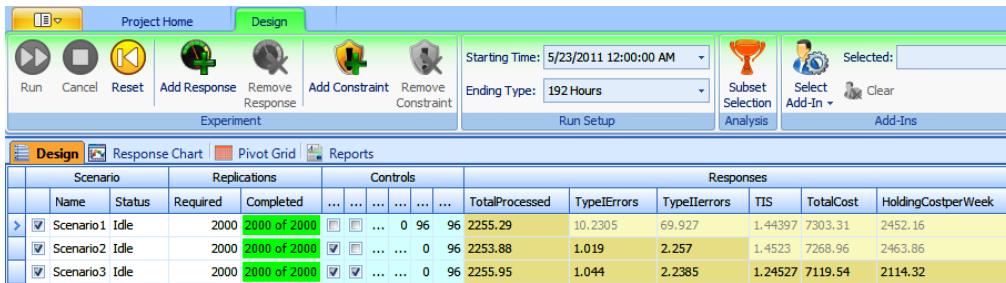


図 A.7 Simio の Subset Selection を用いたレスポンス分析の結果

2. 3つの検査手法の内、費用を最小にするためにどのプロセスを勧めるか？

図 A.8 に示された SMORE プロットからは、どの検査方法が最も安価であるかは明確でない。シナリオ間の週当たりの費用について、なんらかの統計的に有意な差があるかを判断するために、再び Simio の Scenario Subset Selection を用いる。図 A.7 では、シナリオ 1 と 2 の総費用に対するレスポンス値が薄茶色で「Reject」を示しており、シナリオ 3 のそれは「Possible Best」である。この情報および全体の生産が減少しないという事実に基づき、1 台の複雑なロボットを用いて検査工程を変更すべきであると勧めることができる。

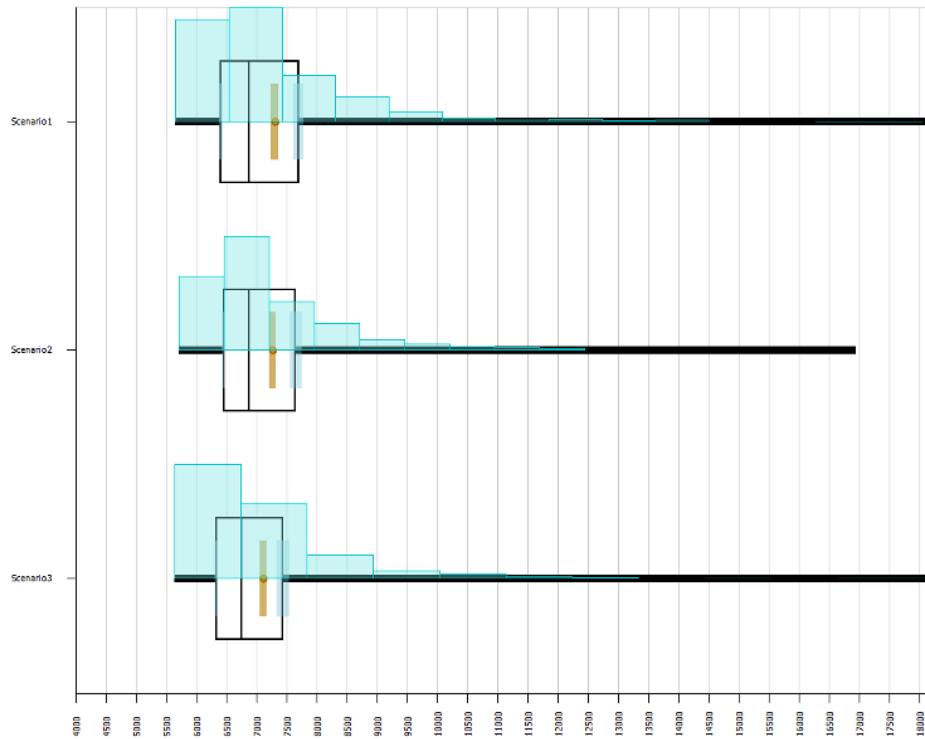


図 A.8 各シナリオにおける総費用の SMORE プロット

3. 自動検査工程の 1 つまたは両方が、手動検査と同じくらい、あるいはより費用がかからることがわかった場合、どうすれば手動検査より週当たりの費用を低く操業することができるかを調べてほしい。すべての費用は固定であり、調整できないと仮定する。

すでに説明したように、シナリオ 1 と 2 の間に週当たりの費用にはまったく統計的に有意な差

がない。どうすればこの検査工程の費用を下げることができるかを調べるために、式 A.1において総費用を構成する要素を考える。シナリオ 2 の総費用となっている 3 つの費用は、機械費用、第 1 種の誤りに関する費用および部品保持費用である。固定の機械費用を下げるることは選択肢ではなく、また第 1 種の誤りに対する費用は週当たり約 \$5 しかかからないので、シナリオ 2 の総費用を下げるためには部品保持費用を下げなければならないと考えられる。保持費用は、部品がシステムで費やす時間に直接関係しているので、4 個の単純機械における処理時間を短縮することが妥当な解決策ではないか。この仮説は、もっとも低い総費用のシナリオが、もっとも低いエンティティの平均システム内時間を記録し、もっとも低い週当たりの保持費用でもあるという、図 A.7 の結果によっても支持される。この仮説を検証するために、単純ロボットの処理時間をシステムチックに短縮した実験を設定する（表 A.7 を参照）。

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)

この実験結果を表す図 A.9 から、単純なロボットの平均処理時間が減少するにつれ、総費用は下がっている（表 A.7 で示されているように、シナリオ 4 ないし 8 では単純なロボット検査時間が 0.1 分ずつ減少している）。

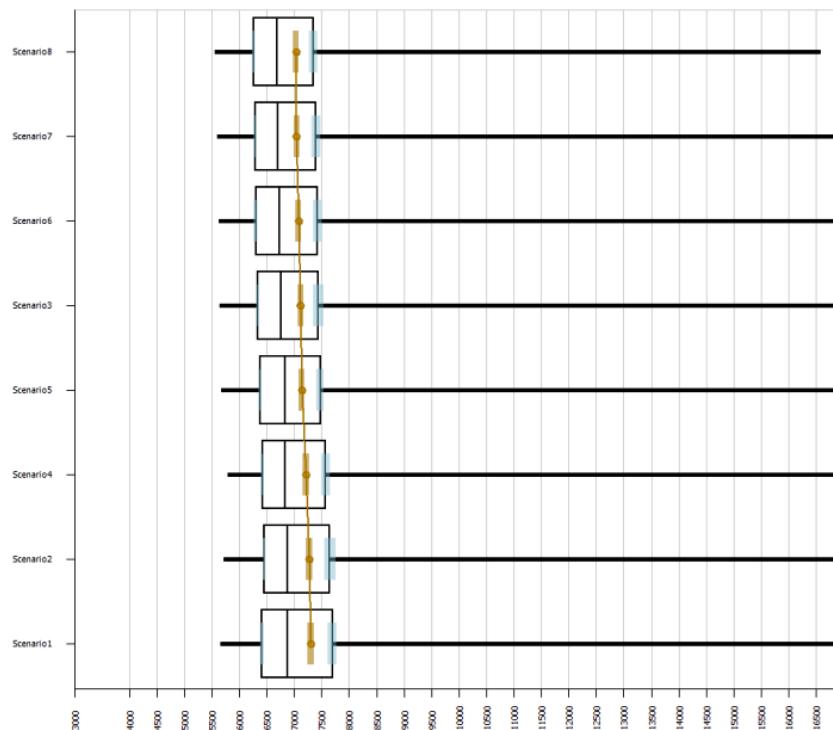


図 A.9 8 シナリオの総費用に対する SMORE プロット

Simio Scenario Subset Selection を用いて、さらに単純なロボットの処理時間を変更させた効果を分析する。図 A.10 は、表 A.6 と同様にレスポンスの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 ないし 8 のものと統計的にかなり差がある。したがって、ロボットのメーカーが平均処理時間を 0.3 分短縮できると（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もたらす。また図 A.10 には、4 つの「Possible Best」シナリオ、つまりシナリオ 3、6、7 および 8 が示されている。これらのシナリオで反復回数を 2,000 から 2,500 まで増加させて、Scenario Subset Selection を用いると、「Possible Best」からシナリオ 3 が除かれる（図 A.11 を参照）。この情報に基づいて、もし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 分減少する（つまり、三角分布(5.2, 5.7, 6.2)の処理時間にする）ことができれば、部品検査のために 4 台の単純なロボットの活用を明確に推薦できる。

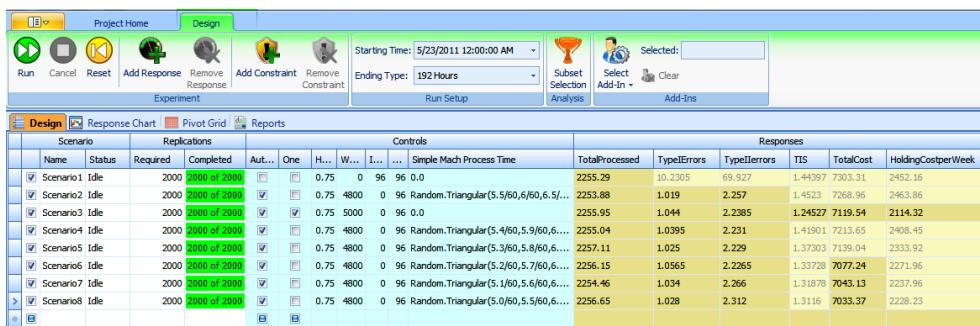


図 A.10 8 シナリオに Scenario Subset Selection を用いたレスポンス分析の結果

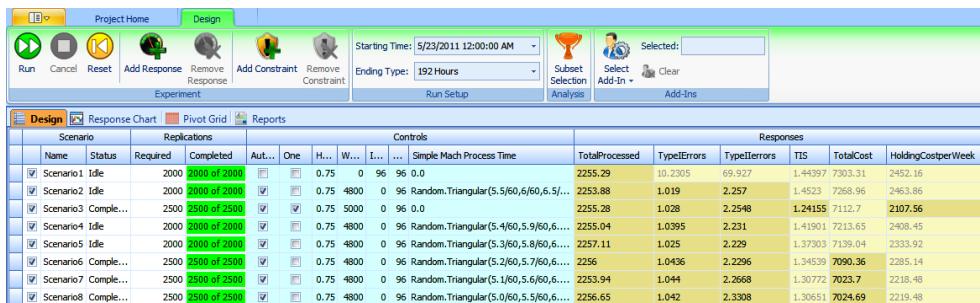


図 A.11 8 シナリオに Scenario Subset Selection を用いたレスポンス分析の結果
(シナリオ 3、6、7 および 8 で実行反復回数を 2,500 に増加)

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)、およびトイレエリア (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）。遊園地は午前 10 時から午後 10 時まで開園している。顧客は、遊園地が午前 10 時で開くと到着し始め、午後 6 時に到着が終わる。非定常ポアソン過程により、表 A.8 の割合で顧客が到着する。

顧客は遊園地に入ると、ジェットコースターと観覧車で優先権を持つことができる「ファストパス」を購入する機会がある。遊園地のマネージャは、ファストパスの追加による待ち時間について顧客からの苦情があり、ファストパスプロセスを管理するために支援を要請した。マネージャの目的は、ファストパス顧客と一般客の待ち時間を制約条件として、ファストパスの総合的な利益を最大にすることである。

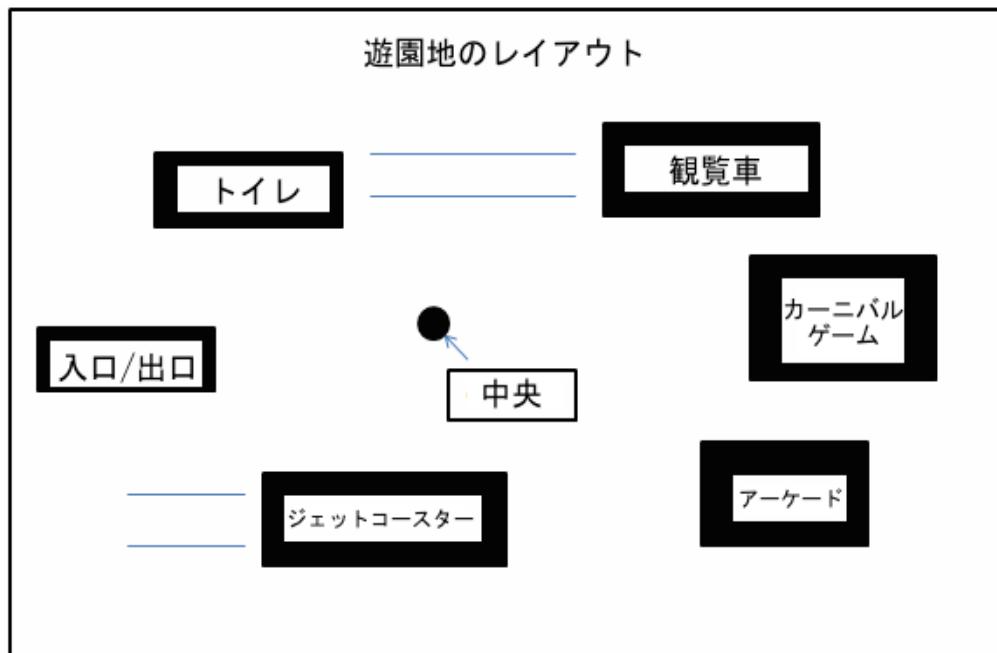


図 A.12 遊園地のレイアウト

表 A.8 遊園地到着率スケジュール

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

- 1日に配布されるファストパスがまったくなければ、顧客は乗るために通常約20分間待つ（これは現在のシステムにおける過去の実績である）。顧客は遊園地で待つことに慣れている一方で、60分より長く待つと、非常に高い不満を抱くようになる。
- 「長過ぎる」列に対して、一般客はその待ち行列に入るのを思いとどまるが、ファストパス顧客は、待ち行列が区別されているため、躊躇なくそれらの列に入っていく。さらに、観覧車の列が80人以上、ジェットコースターの列が100人以上になると、一般客は並ぶのをやめ、他のことに興味が移る。
- ファストパス顧客は、乗るときに5~10分間待っても構わないと思っているが、10分以上待つと不満を抱くようになる。
- 50%の顧客は、アーケードゲームに行くことまったく関心がない。
- 50%の顧客は1回トイレに行き、残りの50%の顧客は2回行く。2つのトイレ（各5の能力）しかなく、現在1日を通して平均約9分の待ち時間がある。
- 顧客は2回以上売店（フードコートも含まれる）に行かない。
- 遊園地で過ごす時間の期待値は顧客によって異なる。表A.9は、遊園地で過ごす時間の確率質量関数(PMF)を示している。それは、0.5、1、2、4、または5(時間)をとる離散確率変数である。

表 A.9 遊園地における顧客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 顧客移動時間とエリア選択

顧客は利用可能性と期待待ち時間に基づいて、次の目的地を選ぶ。トイレと売店においては、上述の通り、顧客は一度か二度の訪問に制限される。遊園地には、中央に設置された「大きなボード」があり、それにはアトラクションの現在の待ち時間の期待値が表示されている。すべての顧客は次の目的地決定に役立つ大きなボードをみるために、それぞれのエリアを訪問する前に遊園地の中央エリアを経由する。ある顧客がその時点で利用できるエリアから、いずれかを選択する可能性は等しい。たとえば、ジェットコースターには非常に長い列があり、すでに食事を終えており、そしてアーケードゲームにまったく興味を持っていないなら、顧客が次にカーニバルエリア、トイレ、または観覧車に行こうとする確率は同じである。顧客は、遊園地で滞在する時間を超えたときに、出口への経路も利用可能な選択肢になる。中央エリアからその他エリアへの顧客歩行時間を、表 A.10 に示す。表中の最小時間は、顧客が目的地に着くのにかかる最小時間を表す。変動は最小時間以上にかかる歩行時間の追加時間であり、指數分布の平均値として表される。

表 A.10 顧客歩行時間（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。

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

車両は、最初のジェットコースターの車両が運行中の際に、乗り込み可能な 2 台目の車両があることを示している。観覧車のゴンドラには、一度に 2 人が乗車できる。

A.2.1.4 その他のエリア

他の 4 つのエリアには、表 A.12 に示されているように、固有のサービス時間と容量がある。バ

ツッファ容量は、サービスを受けていない間にそのエリアで待つ顧客の最大数である（たとえば、すべてのアーケードゲームが利用中の場合、最大5人が待つことができる）。

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の両方の待ち行列における平均と最大時間。
- ・ 遊園地におけるファストパスと一般客の平均数。
- ・ アトラクションで待ち行列で待つファストパスと一般客の平均時間。
- ・ トイレにおける顧客の平均待ち人数と平均時間。

マネージャはファストパスプロセスに関心を持っている。ここでは、遊園地がファストパスの販売数を制限できることを仮定する（すなわち、来園する $x\%$ の顧客がファストパスを購入できる。マネージャは、ファストパスの価格設定で制限ができると確信している）。顧客の満足度が減少し、不満を持つ前の限界として、ファストパスの割合をどの程度にすべきであるとマネージャに提言すればよいだろうか？（すなわち、 x はいくつにすべきであるか）

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のように、特定の結果はいくつかの前提に依存するので、あなたの結果とこの結果は正確には合わないかもしれない。ここに、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。遊園地におけるファストパス顧客数を増加させるに従って、一般客およびファストパス顧客の待ち時間が増加すると見込まれる。これを検証するために、ファストパス顧客の割合を制御するプロパティを加え、このプロパティを1%の増分で0%から25%に変更する実験を作成した。図 A.14 はレスポンスの SMORE プロットであり、ファストパス顧客の割合ごとの、一般客の待ち時間を示している。予想したように、ファストパス購入者の割合が増加するのに従って、待ち時間は増加する。

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顧客の待ち時間を SMORE プロットに示している。こちらも、ファストパス購入者の割合が増加するのに従って、待ち時間は増加する。

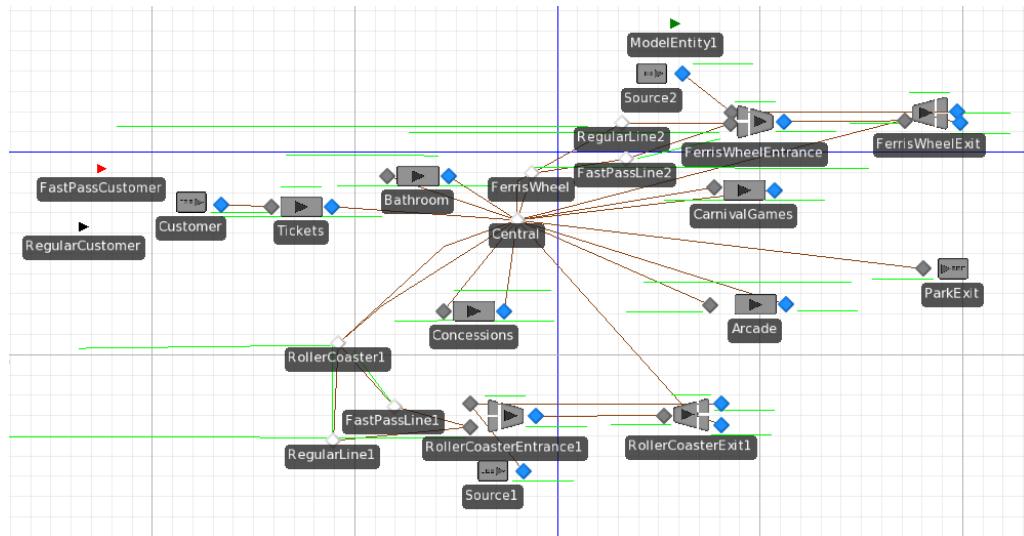


図 A.13 遊園地のサンプル Simio モデル

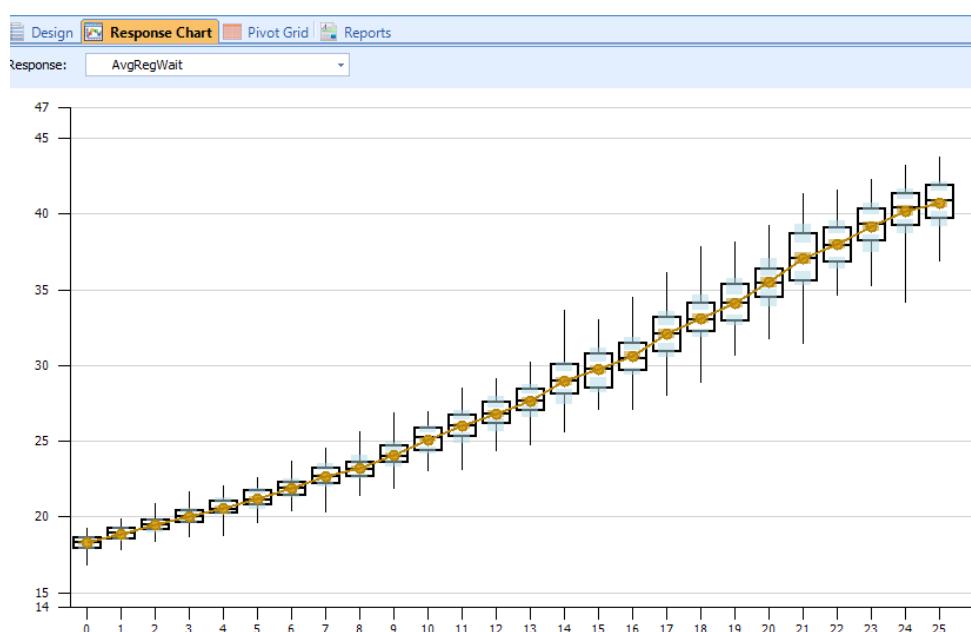


図 A.14 ファストパス顧客の割合ごとの一般客の平均待ち時間の SMORE プロット

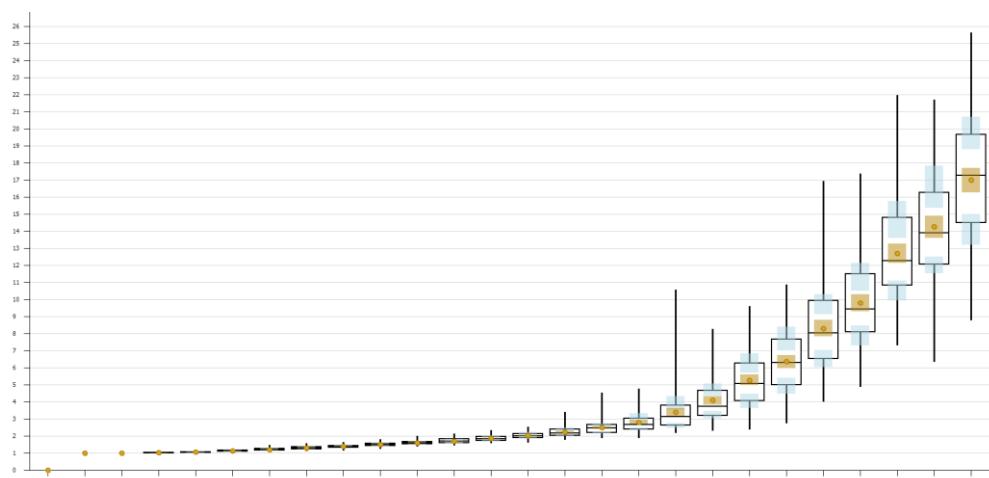


図 A.15 ファストパス顧客の割合ごとのファストパス顧客の平均待ち時間の SMORE プロット

ファストパス顧客の「もっともよい」割合を決定するために、顧客の待ち時間に対する追加的な費用情報が必要だろう。ただし、SMORE プロットはマネージャが決定をする際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、15%の顧客にファストパスオプションがあるなら、一般客が 30~35 分間待ち、ファストパス顧客は約 5 分間待つことが読み取れる。もちろん、これらの推定値は図 A.14 および図 A.15 に挙げた SMORE プロットの目視検査に基づいており、より正確な結果を得るためにより詳細な追加実験が必要だろう。

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。レストランは、午前 8 時に開店し、午前 11 時まで朝食を提供しており、午後 8 時まで昼食および夕食を提供している。午後 8 時にドアは閉めるが、顧客は食事を終えるまで滞在できる。顧客には、店内での食事、または持ち帰り (To Go) のオプションがある。過去の実績から、10%の顧客は To Go を注文する。そして、キッチンで作られる際には、To Go の注文が優先される。

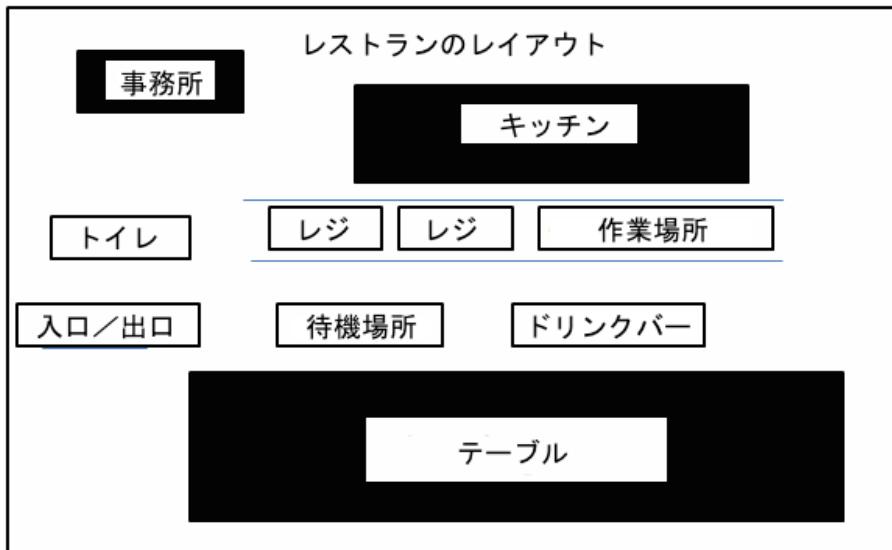


図 A.16 レストランのレイアウト

グループが入店すると、推定の待ち時間が伝えられる。グループが店内で食事をする場合、滞在するかまたは立ち去る（ボークする）かを決める。グループが滞在することを選択すると、注文をして、そしてキッチンでは食事を準備する。レストランの従業員が待ちグループに注文の品を渡すと、（グループが To Go を注文したなら）そのグループは立ち去り、あるいは 1 つのテーブルに座る。グループが食べ終えた後に、他の人が座ることができるよう、レストランの従業員はテーブルをきれいに清掃する。

表 A.13 に示されている非定常ポアソン過程の割合に従って、顧客はグループのサイズが 1、2、3 または 4 人で到着し、それぞれの確率は 0.10、0.30、0.40 および 0.20 である。注文に必要な時間はグループのサイズに依存する（たとえば、3 人のグループの注文時間は、1 人の注文時間の 3 倍になる）。サービス時間の分布とそれぞれの作業を担当する従業員の人数については、表 A.14 を参照してほしい。キッチンに送られた注文は、グループごとに分類されて、一緒に届けられる。

もしテーブルを待つ待ち時間の期待値が 60 分以上であるか、または 5 つ以上のグループがテーブルを待っていると、次のグループは待つことをためらう。現在、25 のテーブルがレストランにある。グループが食事にかかる時間は三角分布(25, 45, 60)分である。

表 A.13 レストランへの平均的なグループ到着率のスケジュール

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

現在、3種類の従業員がいる。調理人、一般従業員、およびマネージャである。調理人は、キッチンに送られた注文の品を準備して作る。一般従業員は、受付（最大2つのレジスター）で働き、完成した注文を配膳し、テーブルを掃除する。マネージャは、調理したり、注文の品を配膳することを補助するが、テーブルを掃除したり、受付では働くない。マネージャの優先作業は、キッチンで注文の品を作ることである。すべての時間に少なくとも1人のマネージャがいる。10人以上の従業員が同時にキッチンにいれば、キッチンは混雑する。

（注文直後から）30分より長い間待つと、顧客は不満を抱くようになる。そして、料理は作られて8分後には冷たくなる。人件費は、マネージャは時給\$10.00、調理人は時給\$7.50、一般従業員は時給\$6.00である。個々の顧客は平均的に\$9.00を費やす。

忙しい日において、現在のスタッフ配置は、2人のマネージャと、8人の調理人、5人の一般従業員である。オーナーは、同様の新しいレストランをオープンする計画を温めており、現在のシステムに関して以下のことを知りたがっている：

- 長い待ち行列により、いくつのグループがボーグ（退店）するか？テーブルが空くのを待つ平均待ち時間はどれだけか？
- 人件費を最小にして、（待ち時間と熱いまでの料理提供に関して）顧客満足を維持するために、何人のマネージャ、調理人、および一般従業員を、1週間の各曜日に配置すべきか？選択したシナリオの総利益額（売上－人件費）はいくらか？
- テーブルが空くのを長く待つことを避け、ボーグする数を減少させるために、いくつのテーブルが必要か？
- テーブルの空きを長く待つことでボーグが頻発するので、レストランは本当の需要を経験していない。前の問題で提案したテーブル数でシナリオを作成する。それぞれのタイプで何人の従業員が必要か、そして、このシステムでテーブルを待つ時間の平均を決定する。

- 総利益額はいくらか？
5. 現在の到着率が 20%増加したら、ボーク数を減少させるために、いくつのテーブルを提案すればよいか？
 6. 到着率が 30%減少した場合、それぞれのタイプで何人の従業員を提案するか？

A.4 小さな銀行支店

小さな銀行支店で、以下のような設定とオペレーションを考えている。銀行支店には、ATM や窓口があり、マネージャがいる。銀行支店は午前 9 時から午後 5 時、1 週間当たり 5 日間営業している。午後 5 時以降到着する顧客は入ることができないが、午後 5 時に閉店するとき、銀行にいるすべての顧客について対応できるくらいの間、開いた状態である。図 A.17 は、銀行の模式図である。

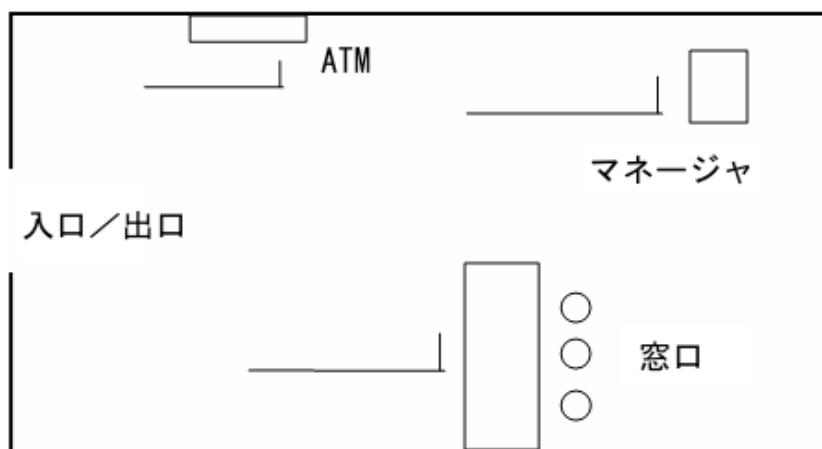


図 A.17 銀行のレイアウト

顧客は 4 つの独立する到着の流れに従って到達する。それぞれ表 A.15 で与えるように、区分ごとに一定の割合の非定常ポアソン過程である。到着した顧客は、サービスを受けるため、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）のいずれかに行く。窓口または ATM のどちらでもよい顧客（窓口／ATM 顧客）は、いつもどの列がもっとも短いかに基づいて決定をする。両方の列が等しい長さであれば、窓口／ATM 顧客は常に ATM を利用する。もっとも短い列に 4 人以上の顧客がいる場合、70% の確率で顧客は立ち去る（ボークする）。ボークしたすべての顧客は、「不満足な顧客」であると考えられる。

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 顧客が到着するとき、もし ATM の列が 3 人以上であり、窓口の列が 3 人未満であれば、その顧客は 85% の確率で ATM 顧客から窓口顧客に切り替わる。そして、窓口の待ち行列に入るが、もし窓口の待ち行列が 4 人以上になると、80% の確率でこの ATM 顧客はボークする。窓口顧客が到着するとき、もし窓口の列に 4 人以上いれば、顧客は 90% の割合で帰る。最近では、このような「不満足な顧客」が発生する度に、銀行はおよそ 100 ドルの費用を負っているというよう

な経済性分析もある。

ほとんどの顧客はそれぞれの取引の後に銀行を出していく一方、ATM 顧客の 2%と窓口顧客の 10%は、ATM か窓口でサービスを終了した後に、マネージャに会う必要がある。さらに、顧客が窓口の待ち行列で 5 分以上を過ごすと腹を立てて、窓口のサービスを終了した後に 50%の確率でマネージャに会いたくなる。もし顧客がマネージャの待ち行列で 15 分以上並んだ場合、70%の確率で腹を立てる。待ち時間の結果に腹を立てたすべての顧客も「不満足な顧客」として考慮する。銀行のエリアの間の顧客移動時間は、表 A.16 で与えられていてモデルに含まれる。それらすべては確定的であると仮定する。

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	—

ATM は、一度に 1 人の顧客にサービスすることができ、顧客の取引に遅れを生じさせる詰まりがランダムに発生することがわかっている。詰まりは機械のハード的なリセットで修理しなければならない。最近、マネージャの 1 人がインターンに、ATM がどれくらいの頻度で詰まるのか、そしてハード的なリセットを完了するにはどれくらい時間がかかるかを決定するためにデータを集めさせた。本書のウェブサイトの students にある ATMdata_bank.xls ファイルにこれらのデータがある（どのようにこのウェブサイトにアクセスするかは「はじめに」を参照してほしい）。アップタイム（ATM がハードリセットの終わりから次に詰まるまでの時間）とリセットに必要とされる時間のデータについて、2 つのヒストグラムが図 A.18 と図 A.19 にそれぞれ示されている。

マネージャは一度に 1 人の顧客に対応することができ、営業中はいつも 1 人のマネージャがいる。窓口には「ベテラン (experienced)」と「新人 (inexperienced)」の 2 つのタイプの担当者がいる。窓口の列には単一の待ち行列があり、各窓口は一度に 1 人の顧客しか対応することができない。ベテランと新人の窓口に関して 1 時間ごとの費用はそれぞれ \$65 と \$45 である。この費用には間接費（保険、福利厚生、401K プラン、休暇手当など）と担当者の時給が含まれる。顧客サービス時間は表 A.17 で与えられた分布に従う。マネージャにとって、顧客がマネージャを訪問する前に、ATM または窓口を経由したがどうかと、サービス時間は独立している。

慣例によって、その銀行では、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、銀行の運営費を最小にするために、それぞれの時間帯にスケジュールすべき窓口数を決定する際に、あなたのアドバイスを必要としている（分析の目的として、提供される費用だけに集中するべきである）。窓口エリアには 3 つのステーションしかないのに、どの時間帯でも稼働している窓口は最大 3 つである。各窓口は、スケジュールされた時間帯では休憩なしで働く：

- ・ 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

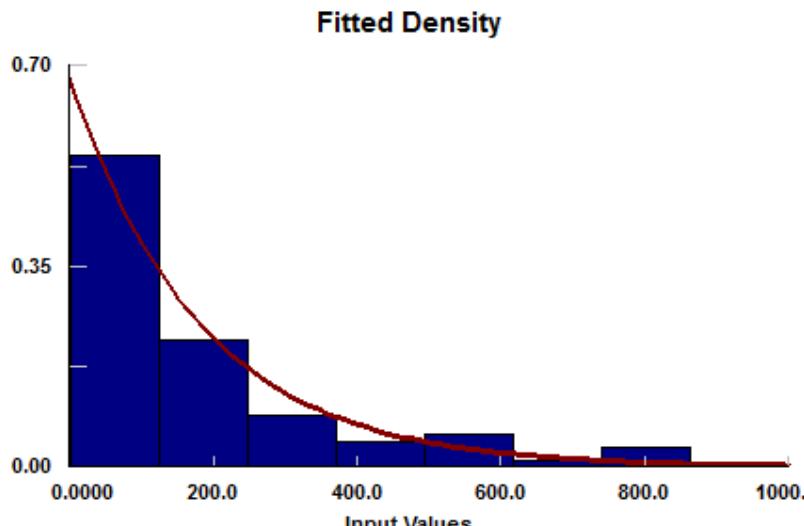


図 A.18 ATM のアップタイムとして収集されたデータのヒストグラム

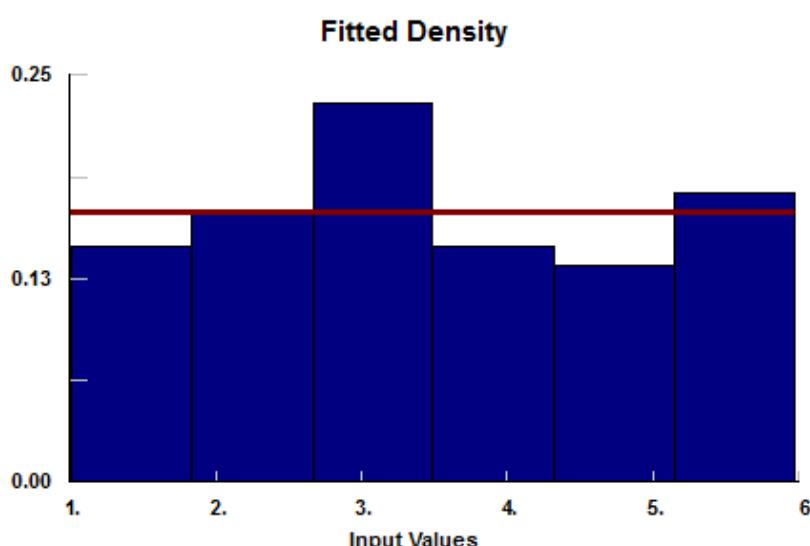


図 A.19 詰まった ATM のハードリセットの時間として収集されたデータのヒストグラム

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	tri(0.50, 0.83, 1.33)
ベテラン窓口	tri(2, 3, 4)
新人窓口	tri(4, 5, 6)
マネージャ	expo(5)

また、銀行のマネージャは追加の ATM を設置することを考えている。ATM 業者は、ATM を購入するために 2 つのオプションを銀行に提供している。第 1 の選択は、新しい機械である。そして、2 番目のオプションは、以前に大手の銀行で使われていた中古 ATM である。業者は、1 日当たりの費用を新しい機械は 500 ドル、中古機は 350 ドルと見積もっている。中古 ATM は、現在銀行にある ATM と同様の割合で詰まり、ハード的なリセットが必要になる。新しい ATM 機は、現在のものとほぼ同じくらいの頻度で詰まるが、20 秒で自動リセットを実行できる機能を持っている。この情報から、銀行マネージャは追加の ATM 機を購入するかどうかに関してあなたの提言を必要としている。そして、購入するとすれば、どちらがよいのか？

上述の銀行支店をモデル化するのに Simio を用いる。モデルに加えて、銀行マネージャによって要求された以下の項目にも回答してほしい。あなたの提言は、本節の始めに提供された費用測定基準（「不満足な顧客」に対する費用、追加的な ATM を導入する費用、窓口費用）によって、銀行の運営費を最小にすることに基準を置くべきである。あなたの回答は実験で得た結果の分析によって、裏付けられなければならない。

1. 銀行は追加の ATM 機を購入するべきか？そうだとすれば、新しいものか中古機か？
2. 銀行が窓口の 1 日のスケジュールで与えられた 4 つの各時間帯について、それぞれの時間帯で配置すべき窓口数を求めなさい。
3. 費用測定基準によると、銀行の期待される 1 日当たりの運営費はいくらか？
4. 「不満足な顧客」に対する 1 日当たり期待費用はいくらか？
5. ATM 機はどれくらいの頻度で詰まるか？修理にはどれくらい時間がかかるか？（これら両方の質問的回答は、これまでに習得した手法で行うこと）
6. 平均的に、銀行では 1 日当たり何人の顧客に対応するか？この値は、銀行に入り、何らかの取引を行ったすべての顧客が含まれる（列があまりに長すぎて、銀行に入った直後に去ってしまう顧客は含まれない）。
7. 「不満足な顧客」は到着した顧客すべてのうち、どのくらいの割合であると考えられるか？

A.5 Vacation 市の空港

Vacation 市の空港 (VCA) は、人気のある観光地の近くに位置するかなり大きな国際空港である。そこは、季節によって到着率が大きく変動する。VCA 経営者は、顧客サービスレベル (CSL) に関するエアライン会社を調査し、ターミナル 2 の総システム内時間に満足していないことに気づいた。経営者は費用を意識しながら、CSL を増加させたがっている。特に、彼らは適切なスタッフ配置の決定と、新しいゲートを加えるか、それとも新しい滑走路を加えるかの決定に役立つ客観的な分析を期待している。主要パラメータは、大小航空機のシステム内時間、ゲート稼働率、およびそれぞれのオペレーションにおける作業者の利用率である。

VCA には 2 つの国際ターミナルがある（図 A.20 を参照）。ターミナル 1 には、12 の駐機ゲートと 16 のレーンがある。ターミナル 2 には、12 のゲートと 42 のレーンがある。両方のターミナルで共通の 3 つの滑走路があり、それぞれが異なる到着率である。航空機が滑走路の 1 つに着陸すると、すぐにターミナルに到着するよう誘導路に向かう。それぞれの時間分布に従って、駐機レーンとゲートに航空機を誘導する。駐機後に、荷物や乗客の降車（同時に実行）、掃除、燃料補給、ケータリング（順番に実行）や、荷物や乗客の搭乗（同時に実行）などのいくつかのサービスが必要となる。それぞれのサービスチームでは、異なるリソースが利用可能である。すべてのサービスが完了すると、飛行機は離陸のために適切な誘導路を通って、外国行きの滑走路の 1 つに向かう。そのプロセスを図 A.21 に示す。表 A.18 は既存のシステムで利用可能なチームのサイズとチーム数である。

飛行機の到着は時刻に依存している（表 A.19 を参照）。到着率は昼ごろ最大に達する。空港経営者は航空機を 2 つのグループに分類した。大型航空機（少なくとも 120 席）と小型機（120 未満の席）である。すべてのゲートは、小型および大型の双方の航空機を扱う大きさがある。問題の計画期間において、ターミナル 2 で扱われる到着の割合は、滑走路 1 は 20%、滑走路 2 は 20%、そして滑走路 3 は 30% である。フライトの調整はターミナル 1 で行われる。

11,000 フィートの新しい滑走路の費用は、1 リニアフィート当たり \$2,200 である。21,000 平方フィートの新しいゲートの費用は、1 平方フィート当たり \$28 である。各チームの（関連設備を含み、各費用を按分した）1 時間ごとの費用は表 A.20 で示される。

VCA の経営者は、上述したそれぞれのオペレーションにおける典型的な処理時間（分）の 100

個のサンプルを提供した。教科書サイトの students にある AirportCaseStudyInputData.zip ファイルでこれらのデータをみることができる。



図 A.20 Vacation 市の空港の航空写真

表 A.18 VCA ターミナル 2 の既存作業要員

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

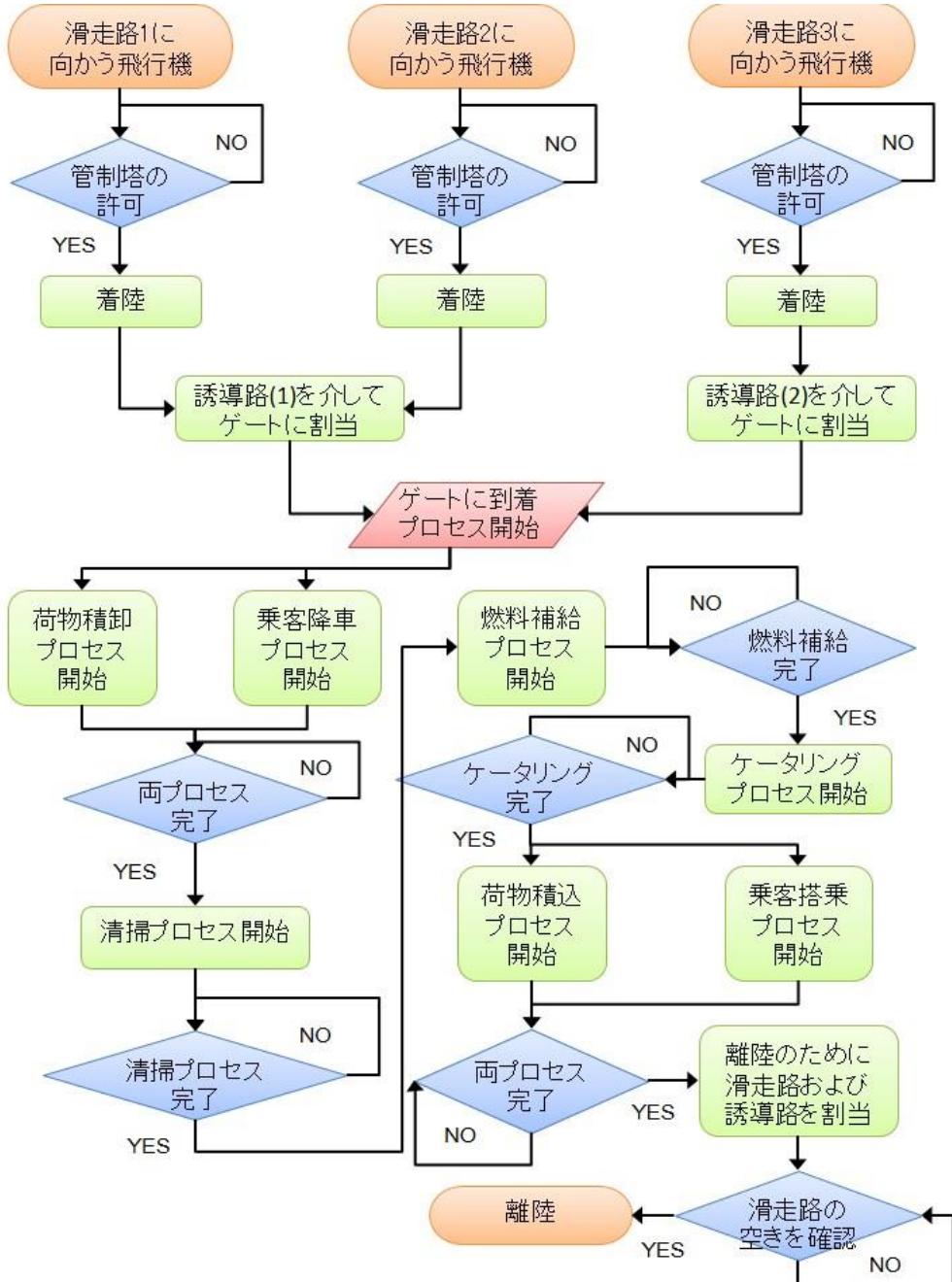


図 A.21 既存システムの VCA のオペレーションのフローチャート

表 A.20 VCA ターミナル 2 で必要な作業者チームの 1 時間ごとの費用

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

謝辞：Vacation 市の空港は、実際の空港の問題を解決するための Hazal Karaman と Cagatay Mekiker によるプロジェクトに基づいている。このプロジェクトは大学院レベルの初級シミュレーション講義用にピツツバーグ大学で開発された。

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) にポッドを採用しようとしている、かなり大規模な病院である。ポッドとは 7 つの部屋に囲まれたエリアのことであり、スタッフがそのポッドに割り当てられている。この病院では 6 個のポッドと、それぞれに 1 人もしくは 2 人の看護師を配置することを計画している。STBH の経営者は、次の 2 つの評価を必要としている：

- ・ 部屋に入る前に患者が長期間待つことがないようにするために、各ポッドに必要な看護師の人数はどれくらいか？ただし、看護師の利用率を低くしないようにしたい。
- ・ 血液検査を指示してから検査室に到着するまでに、どのくらいの時間がかかるか？血液検査を行う時間は 15 分未満を保たなければならない (STBH のポリシー)。

STBH の ED には、2 つの入口がある（図 A.22 を参照）。1 つの入口は、ED に歩いてくる患者のためのものである。そして、もう片方の入口は救急車によって運ばれる患者のためのものである。歩いてくる患者は、最初に受付に行き、書類に記入する。そして、トリアージエリア（重症度判定エリア）に行かなければならない。そこでは、看護師によって診断され、処置の優先度を割り当てる。処置の優先度は症状の重症度を示す（1～5、もっとも重大は 5）。処置の優先度が割り当てられると、トリアージの看護師はコンピュータシステムに患者情報を入力し、ポッドの看護師の利用可能度（ポッドが空いていて、さらに看護師が別の患者のために働いていない場合）に基づいて看護師に患者を割り当てる。そして、トリアージの看護師は、ポッドの看護師が空きのある部屋のポッドの入口にその患者を連れて行くため、廊下の患者から離れる。ポッドの看護師は入口に来て、部屋に患者を連れて行く。ここで、看護師は患者が部屋で落ち着くように補助し、必要な設備をセットアップする。患者がポッドに到着するとすぐに、医師は部屋に入り、患者の血液検査を指示する可能性がある。血液検査は 75% の患者に行われる。看護師の時間が空いているとき、看護師は患者から血を採取し、患者の名前のラベルをつけて、病院事務職員 (HUC) に渡す。HUC は血液を受け取り、必要な検査のラベルをし、シートを経由して検査室に送る。HUC のデスクでの処理時間は、三角分布(3, 4, 5) 分である。



図 A.22 STBH の設備レイアウト

教科書サイト（「はじめに」を参照）の students のファイル STBH_ArrivalData.xls には、処置の優先度のレベルごとの到着データが含まれている。過去のデータでは、処置の優先度 4 の患者の 80%、処置の優先度 5 の患者の 95% は救急車で到着する。他のすべての患者は、予約なしで到着する。処理時間データ（最小、最頻、最大の三角分布と仮定）は、表 A.21 に示されている。

トリアージでは、患者は血圧と体温を測定され、一般的な病状が記録される。重大な病気の病状を訴える患者は、トリアージの時間を速め、できるだけ早く処置のために部屋に連れて行かれる。患者の部屋の配分は以下の方針を用いる。最初に来た患者はポッド 1 へ送り、そこがうまっている（7 つのすべての部屋が占有されている）と、次の患者はいっぱいになるまでポッド 2 へ送られ

る。そうしている間に、もし患者がポッド 1 から解放されると、システムに入る次の患者はポッド 1 に入る。小さい番号のポッドが空いているときはいつでも、このようにされる。これは、病院の運営費やスタッフ配置にも役立つ。ED における必要な処置が終わった患者は、システムから離れたと考える。

表 A.21 患者の処置の優先度による STBH の三角分布の処理時間（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

STBH の経営者が期待している主要な測定基準は、以下の通りである：

- ATWA：待合所における平均時間（システムに歩いてくる患者のすべてのタイプの平均）。
- NBT：実施された血液検査数。
- ATB：血液検査の平均時間。
- AITS, …, A5TS：5 つそれぞれの処置の優先度レベルごとのシステム内時間。

STBH の経営者は、特に次の 6 つのシナリオに興味を持っている（他のシナリオも評価されるかもしれない）：

1. 病院のスタッフ配置が普通のレベルより低いとき（12 人ではなく、10 人の看護師）、たまたま列車衝突や飛行機墜落のような大規模事故があり、処置の優先度が 4 と 5 の約 54 人の患者が ED に運ばれたときの、1 日をモデル化してほしい。
2. すべての看護師が勤務しているが、処置の優先度が 1、2 および 3 のおよそ 72 人の患者がシステムに入ってくる 2 日半の期間をモデル化してほしい。このシナリオは、軽い伝染病が発生し、多くの人々が咽喉炎、頭痛、または胃の不調にかかっている状況を想定している。
3. 通常の週と、スタッフの配置が少なく（1 ポッド当たり 1 人の看護師）、そして病院に入る患者の数も標準より少ない週（1 週間に 200 人以下の患者）を比較してほしい。
4. すべての患者のサービス時間が 10% 増加する状況と、通常の状況を 1 週間比較してほしい。
5. 通常の平日の後に、1 つのポッドに 1 人の看護師を配置する週末をモデル化してほしい。
6. ウィルスが蔓延し、ED に処置の優先度が 1 および 2 の 120 人の患者が 3 日間にわたって来る状況をモデル化してほしい。スタッフの配置レベルは標準より低いと仮定する。たとえば、それぞれ 2 人の看護師がいる 3 つのポッドがあり、残りの 3 つのポッドには、それぞれ 1 人の看護師しかいない。これはウィルスにかかるので、患者の 90% には血液検査が必要であると仮定する。

謝辞：STBH は、Karun Alaganan と Daniel Marquez による、実際の病院の問題を解決するためのプロジェクトに基づいている。このプロジェクトは大学院レベルの初級シミュレーション講義用にピツツバーグ大学で開発された。

付録 B Simio 学生コンペティションの問題

Simio の顧客が直面する課題から着想を得た、現実に近い問題に世界中の学生がチャレンジするシミュレーションコンテストが、毎年 2 回開催されている。毎年 1000 以上のチームから 4000 人を超える学生が参加し、急速に大きな大会に成長している。問題の解決には、学生のモデル構築スキルだけでなく、創造力やプロジェクトマネジメント能力、動画制作やプレゼンテーション能力が必要になる。参加学生は大会自体に面白さや価値を見出し、さらに受賞者は魅力的なキャリアの選択肢を見つけることができる。

コンテスト参加の詳細は <http://www.simio.com/academics/StudentCompetition> を参照してほしい（図 B.1）。この Simio の問題は、領域も課題も多岐にわたる。興味深いプロジェクトや実践的な問題に関心があれば、この付録にコンテスト問題の一部を要約して述べるため、参考にしてほしい。さらに詳しく知りたい場合は、<http://www.simio.com/academics/StudentCompetition/problem-archive.php> で問題の全文を閲覧できる。

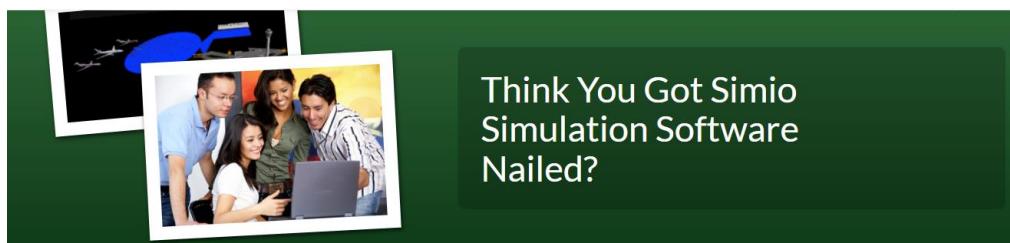


図 B.1 Simio 学生コンペティション

B.1 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。旅客が空港の駐車場に自家用車を停め、旅行中にその車をレンタルすれば、駐車料金を無料にするほか、レンタル収入も得られる仕組みである。彼らの車は、空港に降り立つ旅客にレンタカーとして提供される。基本的なビジネスコンセプトは以下のとおりである：

- ・ このサービスを事前登録し、車で空港に到着する出発旅客は、その車をレンタカーとして提供すれば、3 週間まで駐車場を無料で利用できる。
- ・ レンタカーとして提供された車は、まず、車の所有者の負担で清掃される。借りられた車は、レンタル期間の終わりに無償で再洗浄される。したがって、貸借者と所有者の双方は、きれいな車で空港を離れることになる。
- ・ 到着旅客は、スマートフォンアプリを使って、利用可能な車を選択することができる。
- ・ レンタカーは、車の型式と年に基づいて、4 種類に分類される。レンタル価格は車の種類によって設定され、車の所有者にはレンタル収入の 50%が支払われる。

多くの車がレンタルされるため、この会社が使える駐車スペースの台数よりも多くの顧客に駐車サービスを提供できる。

まず、中規模の空港で試験運用し、このコンセプトの収益性を確認してから、他の空港へと展開したい。この試験運用システムを成功の確率が最大になるように適切に設計し、潜在的な投資家たちにこのビジネスモデルを認めてもらうことが非常に重要である。このような理由から、システムのレイアウトも含め、重要な設計の意思決定をするために、シミュレーションを活用したい。

B.2 Simio 掘削ロジスティクス社

Simio 掘削ロジスティクス社 (SDL) は、さまざまなオフショアの掘削拠点間で原材料を輸送するため、複数のオフショア船をチャーターしている。「現状」のシステムでは、少数の船でオフショア拠点に対するサービスに従事している。原材料の不足によりオフショア拠点が遊休となるコストが極めて高いため、SDL は掘削効率を最大化するもっとも高い期待値に合わせて、船を調達する傾向にある。しかし、港とオフショア拠点の双方において、相当の待機時間が発生していることをデータが示している。SDL は船のスケジューリングを改善し、業務改善を通じてオフショア船のサイズを削減したいと考えている。

この問題ではまず、特定の掘削拠点専用の船のサイズで「現状」のシステムをモデル化する。このモデルでは、異なるサイズの船を考慮し、オフショア船による数種類の貨物品の輸送をモデル化する。システムの複雑さは、船への貨物の積載作業や、船による輸送や積降に際して天候や波の高さを含める程度である。この「現状」のモデルは、システムの改善戦略を評価するベースラインを提供する。

このプロジェクトにおける次のもっとも重要な点は、代替案となる複数の戦略を構築し、評価することである。たとえば、同水準の高いサービスレベルを維持しながら総コストを削減する「改善」システムを構築するために、すべての拠点にサービスを提供する共通船団として船をプールすることを検討したい。プロジェクトの結果は、シミュレーションモデルの質と、提案された「改善」システムの全体的な効率およびコストの両面から評価される。

B.3 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ており、通常の診療時間だけでなく、夜間や週末も外来患者の治療を行っている。UCCS は、感染症やインフルエンザ、軽度のケガや骨折など、一次医療の現場で見られる症状を治療する。また、UCCS は、看護師、内科医、助手、整形外科専門医などの専門家を配属している。

施設には、受付、待合室、治療のためのエリアと、さまざまな検査や処置のための部屋がある。患者の需要は時間帯や曜日で変動するだけでなく、センターによっても異なる。UCCS は、一般的なレイアウト、スタッフ配置、部屋の配置、その他の運営方法について、最高のものを決定したいと考えている。また、検査や軽度の症状に関して、多くの患者が予約を取るためにインセンティブを増やすことによって、必要なスタッフ数を削減できるかを評価したいとも考えている。

異なる場所にある複数のセンターを最適化するために同じモデルを利用したいので、患者の動態や、施設やスタッフの構成および最適化のための実験について簡単に修正できるように、シミュレーションモデルはデータ駆動型とする必要がある。提案は UCCS の経営陣に対して行われるため、3D アニメーションと結果の透明性が特に重視される。

B.4 航空機生産の問題

ある航空機メーカーが、最終組立システムに対する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、それぞれ 10ないし 20 のタスクからなる作業を担当する。各タスクは、作業者、ツールおよびマテリアルハンドリングについて、独自の要件を持つ。システムはスケジュールによって稼働しており、ライン休止時に完了していないワークは、後工程に送られない。このワークの搬送に関する生産方針は、このプロジェクトで評価される項目の一つである。生産プロセスは急勾配の学習曲線を持つ労働集約型であり、仕掛品は高価である。したがって、労働生産性と仕掛品の削減に関するプロセスの改善が重要である。

生産プロセスには現在、2つの既存製品が存在するが、同社は新しい製品の追加を計画している。第3の生産ラインの追加が、生産計画を評価する必要性を増加させている。

同社は、2つの生産ラインを持つ現状と、2つまたは3つの生産ラインとなる将来の状態について最適な生産計画と、新しいスループット率について評価したいと考えている。したがって、異なる需要のプロファイル、ライン割当戦略、リソース配分戦略、およびワークの搬送に関する生産方針について、評価する必要がある。

B.5 ラテンアメリカのサプライチェーン

ラテンアメリカに拠点を置いている小売店が、カリブ海地域と南米への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。需要は地域によって異なり、市場は気まぐれなので、市場への柔軟性とスピードが不可欠である。同社は、1つ以上の物流センターの建設を評価したいと考えている。

地域別の物流センターに投資すべきか、あるいはアジアの工場から直接送るべきか。直接送らない場合、将来の拡大戦略を念頭に、物流センターの場所を決める必要がある。考慮すべき項目には、各国の市場成長率だけでなく、その国の港湾からのアクセスも含まれる。

同社はまた、物流センターのSKU（最小管理単位）ごとに異なる発注方針および戦略を評価したいと考えている。工場の生産リードタイムや生産能力の制約だけでなく、港湾あるいは船の制約も考慮する必要がある。輸送や在庫保管、機会損失、発注処理、その他にかかる費用が、意思決定の重要な要素である。

B.6 紙パルプ製造業の供給

紙パルプ製造業のアメリカの業界団体は、原材料としての木材の獲得に関する非効率性を認識している。各社は独立して操業しているため、材木運搬トラックは競争相手の製材所に向かうために、途中の製材所を頻繁に通り過ぎる。これはシステムに、実際には避けられる輸送コストを加えている。彼らは、欧州のコンソーシアムを意識して、物流コストを最小化するために、木材の運搬を管理したいと考えている。

彼らは、製材所に対する材木の輸送を管理するための、新しいコンソーシアムの創設を評価したい。目標は、輸送コストの最小化である。ある地域に3つの連続的に操業している製材所があり、それぞれは1日当たり4ないし6キロトンの木材を消費する。各製材所には独立した需要と最大在庫量があり、いくつかの製材所は離れた場所に追加の在庫を保有できる「ドロップヤード」を有している。木材は、小規模の伐採業者により、その地域で伐採される。独立した各伐採業者は、季節によって変動する独自の伐採および輸送能力を持つ。

コンソーシアムは、必要な投資額や最善の操業パラメータ、および期待される節約額を評価できるように、現在のシステムと提案されるシステムの両方の結果を期待している。

B.7 グローバルな通貨取引

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。両替はそれぞれの取引において顧客のわずかな手数料から利益を得ている。毎日何千ものランダムで予測不可能な取引があり、銀行は各種の現金を保有する必要がある。各日の終わりに、銀行はCLS銀行（多通貨同時決済銀行）との間で決済を実施し、各通貨の数量をリセットする。CLSはこの両替に少しの手数料を請求する。これにより、銀行のリスクが一日に限定される（日中流動性リスクとして知られている）。いずれか種類の通貨がなくなった場合、銀行は取引先とスワップを実施することができる。これは本質的にCLSによって提供されるサービスと同じであるが、一日中いつでも生じる可能性がある。スワップはリスクを軽減するのに役立つが、CLSと比較して高価である。

銀行は、法的にリスク管理する義務がある。もし銀行が何らかの理由で顧客取引を支援できなければ、世界の金融市場に多大な混乱を引き起こす可能性がある。したがって、銀行は政府に対し、

多様なストレスシナリオに関して、このような状況を回避するだけの十分な現金を保有していることを証明しなければならない。通貨ごとに手持ちの残高を決めるることは難しい問題である。受け入れ可能なリスクの条件下で利益を最大化するために、システムの活動（取引、スワップ、および決済からのキャッシュフローなど）をシミュレートしてほしい。これには、収益、コスト、およびリスクの分析が含まれる。これらの分析は、さまざまなシナリオにわたって行うこと。

B.8 Sunrun ソーラーパネルの設置

Sunrun は、米国最大の住宅用太陽光発電会社である。彼らは住宅所有者のニーズを満たし、設置に関するあらゆる面を担当し、ソーラーシステムを調整する。Sunrun は 21 の州で事業を展開し、毎年成長を続けている。

このプロジェクトでは、ソーラーパネル設置の「ラストワンマイル」の複雑さを考えてほしい。このプロセスは、実際の機器の移動、購入者およびその家族（職場）への説明と、屋根へのソーラーパネルの設置が含まれる。現時点では、現在の標準プロセスから、設置を遅らせるたり長くする可能性がある予期しないイベントをより適切に処理できる新しい配送プロセスに変更するという提案がある。シニアマネージャは、現在のプロセスの評価、2つのシナリオの比較、および設置プロセスを改善するための推奨事項を要求している。

B.9 種子生産

種子生産プロセスは、特定の交配種の畠への植え付け、シーズン中の作業、収穫、そして生産施設での種子処理など、さまざまな活動にわたる。北米では、収穫期は 8 月中旬から 10 月上旬まで、毎年 8~10 週間ほど続く。収穫が完了した後に、種子処理および包装のためにある領域が生産用地として割り当てられる。種子生産施設は 9 月中旬から翌年の 4 月まで稼働している。

異なる製品ミックスで複数の種子生産施設を運営しているため、計画機能はいくつかの制約を考慮して、各施設の生産計画を決定する必要がある。それは、適切な種子が適切な時期に適切な顧客に出荷されるようにするために、製品ポートフォリオ、複雑な部品表、施設内の製品のルーティング、物理的スペースやマテリアルハンドリングシステム、設備構成と効率性、そしてその他の項目間の労働力の制約がある。

B.10 メガストア流通センター

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、最先端のテクノロジーと高度な分析を駆使して、コストを最小限に抑えている。これは、彼らの物流センター運営の洗練されたレベルに反映されている。リアルタイムの情報が、意思決定の指針として積極的に活用されている。検証、最適化、自動化は日常業務の一部となっている。オペレーションが複雑なため、追加投資が導入後にどのように機能するかを予測するのは困難である。具体的には、人件費を最小限に抑えるために、どのようにオーダーの優先順位をつけ、ディスパッチすればよいのだろうか。システム性能に関する質問に答える最良の方法は、システムの詳細なデジタルツインを作成し、活動をシミュレートし、結果を観察・定量化することである。

労働力は主要なコストドライバーであり、制約のある資源である。システムは、以下の作業分野に分割できる：収納（車両制約あり）、選択、および補充。この問題の焦点は「収納」である。主な目標は、時間当たりのパレット数、利用率、コスト、移動距離などの指標を改善するために、さまざまな配車方法を評価することである。私たちは、より多くのインバウンド需要を処理するための追加ハードウェアを評価するのに役立つ、指定された目標を達成する方法を探している。このモデルは、パラメータを簡単に変更できるように、データ駆動型の手法を用いて実装する必要がある。

B.11 地方空港の計画

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。しかし、この利用者数の増加により、空港は頻繁に混雑に悩まされ、地域の経済成長の妨げになる可能性がある。このため、空港では、混雑を緩和し、乗客の利便性を向上させるための改善プロジェクトを評価している。

空港内の移動時間（到着、チェックイン、手荷物預入れ、セキュリティ、ゲート到着）に着目し、混雑を緩和する方法を検討する。空港での滞在時間を短縮するためのオプションはいくつかあるが、コストや滞在時間のばらつきを考慮すると、どれが最も効果的な方法かはわからない。現在検討されている改善策には、チェックインとセキュリティの改善と能力向上、ターミナル内の乗客運搬車の追加などがある。空港は、これらの選択肢を評価し、最も費用対効果の高い方法で、旅客の滞留時間を短縮する方法を決定する必要がある。

空港内で発生している過密状態に加え、ターミナルと周辺道路では交通渋滞が問題になっている。交通局は、この混雑を緩和するために、空港に別のバス路線を追加することを提案した。また、バスの利用を促進するため、バス利用者専用の保安検査場の設置も検討されている。しかし、空港はこの提案を実行した場合にどのような影響があるのか不明である。

改善案の影響を比較するためのベースラインとなる、現在の空港運営のモデルを開発しなさい。そして、最も費用対効果の高い運用のための包括的な推奨を行うために、提案を評価しなさい（および他の提案をしなさい）。

B.12 ミーバック配送センター

ミーバック配送センター（MDC）は、顧客に配送されるフットウェアやアパレルの商品を保管する倉庫である。これらの注文は、消費者が一人で購入する「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、注文の増加と同時に、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に、ピッキング作業の戦略を改善したいと考えている。計画担当者と管理者は、ピッキングの下流で働くチームを考慮に入れた、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ります。このプロジェクトの範囲は、棚に置かれた倉庫からユニットをピッキングすることから始まり、注文品を梱包して出荷することまでである。

注文が増加しているため、MDCは保管場所からユニットを回収するウェーブピッキング手法（ウェービング）に移行している。MDCの倉庫をウェービングするには、オーダをグループ化し、グループ化されたオーダのユニットを、対応する倉庫のゾーン保管場所ごとに分割する。そのゾーンを作業員がピッキングし、回収したユニットをステージングエリアに移動させる。ウェーブの全ユニットがステージングエリアに到着すると、ウェーブは梱包と出荷に回される。ウェーブは再びオーダに編成され、出荷される。

MDCは、労働力の利用を最適化し、顧客納品目標の未達を最小限に抑える倉庫管理のための最適なウェービングおよびピッキング戦略についての推奨事項を探している。ウェーブピッキング戦略には、オーダの属性、ユニットの位置、許容できるピッキング容器に基づくウェーブの理想的なサイズを含める必要がある。また、同じウェーブにオーダを配置する基準も調査する必要がある。さらに、MDCはウェービングプロセスに有利な倉庫の再配置を確認したいと考えている。MDCは、将来のウェーブの策定に役立つ一連のルールを期待している。

B.13 小規模レストランの運営

Simio BBQ Smoke Pitは、地元の新進気鋭のレストランである。BBQスマートミートと定番のサイドメニューが特徴だ。賑やかなダウンタウンにある古い建物を改築した、急ごしらえの料理店である。サイズの制約から、この店は持ち帰りのみとなっている。しかし、Simio BBQ Smoke Pitの人気が高まるにつれ、商品の欠品や労働力の制約により、顧客の待ち時間が長くなっている。こ

408 付録B Simio 学生コンペティションの問題

のレストランでは、需要に対応するため、サービスのボトルネックを解消するための新しいポリシーを決定する必要がある。

バーベキュー肉には長い燻製時間が必要で、サイドメニューにも長い調理時間が必要なため、レストランは調理済みの食品を適切なレベルに保つのに苦労している。在庫が少なすぎると、顧客満足度に影響する。食材の不足が頻繁に起こると、待ち時間が長くなり、顧客が離れていってしまい、レストランの潜在的なビジネスを失う可能性がある。一方、過剰な在庫を作ると、余分なコストが発生する可能性がある。肉は高価であり、余分な食べ残しはその日のうちに処分しなければならない。

課題は、利益と顧客満足度を最大化するために、スタッフの配置と料理の調理量のバランスをとることである。このレストランでは、顧客の到着と注文のパターン、必要なリソース、そして料理の調理率を調査したいと考えている。Simio BBQ Smoke Pit は、メニューにある様々な食品をそれぞれ補充するための最適な戦略を望んでいる。この補充戦略は、いつ、どれだけの料理を調理するかだけでなく、調理場ステーションと保管キャビネットの間に調理途中の食材をどのように割り当てるかにも影響する。さらに、Simio BBQ は、スタッフレベルの変更や設備の追加による改善の可能性が、投資に見合うものであるかどうかも知りたいと考えている。

B.14 DDMRP を用いたサプライチェーン計画

Simio Shelving Shop は、デザインにバリエーションを持たせた数種類のシェルビングユニットを専門に扱うシェルビング製造・卸売業者である。このショップでは、現在、注文の充足率が低く、ショップの収益に影響を与えていている。経営陣は最近、外部のコンサルタントを雇い、在庫とバッファリングシステムの見直しに関する提案を求めた。コンサルタントは、在庫切れの問題に対処するために、新しいバッファリングと在庫管理の概念である需要主導型資材所要量計画 (DDMRP) を提案した。

この工場では現在、静的なバッファレベルを使用しており、すべてのステーションに一定量の原材料がバッファされており、工場は出荷可能な各最終製品を一定数維持している。現在のバッファリング方式は、工場とそのサプライヤ全体のリードタイムおよび品質のばらつきと相まって、完成品と原材料の在庫切れを引き起こしている。経営陣は、DDMRP バッファリングソリューションを工場に導入するというコンサルタントの提案を受け入れた。ダイナミックバッファリングソリューションによって、選択した主要業績評価指標 (KPI)、特に充足率と平均在庫コストが改善されることが確信しているからである。また、副次的な目的として、経営陣はバッファレベルのリスクを監視する優れた方法を求めており、DDMRP バッファプロファイルがそれを実現する。

課題は、工場に動的な DDMRP バッファを設定することである。これには、営業、品質、機械、サプライヤからの既存データを活用して、まずバッファの位置を決定し、次にバッファのパラメータを微調整して、選択した KPI を向上させることが含まれる。バッファを統合した後、経営者は工場の KPI の変化を予測したい。さらに、経営者は、リードタイムと製品品質にばらつきのある潜在的なサプライヤを再評価したいと考えている。この評価には、潜在的なサプライヤに最適なバッファを再作成し、そのサプライヤが会社の充足率と平均在庫コストに与える影響を判断することが含まれる。

付録 C 本書の使い方

この付録では、フォーマットの意味の理解、ニーズに合った最適な Simio ソフトウェアの入手、本書で参照するファイルやその他のリソースの場所について、いくつかのヒントを紹介する。

C.1 フォーマットガイドライン

本書では、用語の意味が伝わりやすいように、書式を統一している：

- 新しい用語を紹介するときは、**太字**の書式を使用する。たとえば、「SimBits は小さくてわかりやすく編集されたモデルである」というように、その定義に注意を促す。
- 太字**を使って用語や単語を強調し、見落としがないように注意を喚起する。たとえば、「カスタムロジックを指定するために 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 ウィンドウから定義・編集する」など。
- 入力または定義済みリストから選択するように指示されたものは、別のフォントで識別する。たとえば、「Server Name プロパティには、Server1 と入力する」「Ranking Rule には、First in First Out を選択する」など。これと同じ形式がファイル名の識別に使用される。
- 要素名、ステップ名、ロケーション、ボタン、プロパティ名など、その他の用語はすべて大文字と小文字が混在し、特別なフォントはない（例：Material、Process Type、Project Library、Facility Window）。

C.2 Simio ソフトウェア

本書は、Simio 14 以降を対象に書かれている。より新しいバージョンのソフトウェアでは、細かい違いや、より簡単なモデリング方法を見つけることができるかもしれないが、コンセプトは同一なはずである。

Simio Personal Edition のソフトウェアは、<https://www.simio.com/evaluate> から無料でダウンロードすることができる。これは、Simio を自分で調べたり学習するにはよいソフトウェアだが、一般には、初期のコースワークや、非常に小さな課題しか提出されないコースにのみ適している。Personal Edition は、一般に、学位取得のためにモデルの提出を必要とするコースには適していない。

Simio Academic Version は、本書で使用する推奨ソフトウェアである。モデルサイズや機能面で異なるいくつかのエディションがある。アカデミック版ソフトウェアは、機能的には Simio の商用ソフトウェアと同様だが、非商用利用のみのライセンスとなっている。また、すべてのアカデミックソフトウェアには、最適化ソフトウェア OptQuest が含まれている。

Simio Academic Version を無償で使用するための助成金を希望する教員は、<https://www.simio.com/academics/order-academic-software/> から申請できる。無償の助成金は、教員と TA の利用、および学生が利用する情報処理室へのインストールのためにライセンスされている。1 つの助成金で、教員や学部に必要なライセンス数を提供することができる。Simio 助成金で提供されるソフトウェアからの無料アップグレードを要求する正当な理由がある場合、有資格の教員は、<https://www.simio.com/academics/faculty-upgrade-request.php> でアップグレードを要求することができる。

Simio Student Edition ソフトウェアは、上記の Simio Academic Version と似ているが、コースに登録されている学生個人が使用するためにライセンスされている。各ライセンスは、わずかな料金 (25US ドル) で販売されており、暦年間有効である。デフォルトのライセンスは、RPS の機能はなく、中程度の大きさのモデルを構築することが可能である。正当な理由がある学生は、

<https://www.simio.com/academics/student-upgrade-request.php> で、サイズ無制限のモデル構築と RPS 機能の追加を無償でリクエストすることができる。

Simio のアカデミック製品のより詳細な説明は、<https://www.simio.com/academics/simio-academic-simulation-products.php> にある。

C.2.1 Mac で Simio を使うには

Simio は Windows のアプリケーションだが、多くの Simio ユーザは Mac 上で Simio を実行することを選択している。Windows 用に設計された他の一般的なアプリケーションと同様に、最初のステップは Mac 上で Windows インスタンスを提供することである。Parallels、Bootcamp、Oracle VM VirtualBox、VMware Fusion など、いくつかの一般的な方法がある。どの方法も、Windows のライセンスと、Mac にインストールするソフトウェアを入手する必要がある。学校によっては、Microsoft Windows の無料版や割引版を提供している場合がある。詳細は、学校の管理者に問い合わせてほしい。そうでない場合は、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することができる。

上記のプログラムには、それぞれ個別の長所と短所がある。その他のオプションの中には、学校を通じて、または学割で利用できるものもあるかもしれない。すべての選択肢を調査して、自分に最適な解決策を見つけることをお勧めする。

これらの方法を用いて Simio をインストールし、操作するための追加ガイダンスは、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、1つの zip ファイルにまとめられている：

https://textbook.simio.com/SASMAA/files/SASMAA_StudentFiles_6e.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、https://textbook.simio.com/SASMAA/files/MMC_calculator.zip からダウンロードできる。

mmc.exe を起動するには、Microsoft Windows のコマンドプロンプトウィンドウ（通常は Windows の[スタート]→[プログラムまたはすべてのプログラム]→[アクセサリ]フォルダ内）を実行する必要がある。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は C:¥>と表示されるはずである。そこで、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

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である。商用バージョンは <https://www.geerms.com/> で入手できる。50 データポイントに制限された無料の学生版も、同じ Web サイトで入手可能である。6 章では、少し古いがより高機能な（100 データポイントの）教科書版 Stat::Fit が使われている。これは、https://textbook.simio.com/SASMAA/files/StatFit_TextbookVersion.zip からダウンロードできる。

教科書版の Stat::Fit をシステム上で動作させるには、上記の zip ファイルをダウンロードし、アクセス権を持っている分かりやすいフォルダに解凍する。解凍した statfit.exe をダブルクリックすると Stat::Fit が起動するが、デスクトップにショートカットを作成すると、より簡単に実行することができる：デスクトップ上で右クリックして[新規作成]→[ショートカット]を選択し、

statfit.exe ファイル（たとえば、C:\StatFit\statfit.exe）の場所を指定するか、参照する。なお、Stat::Fit の教科書版には、他のソフトウェアでおなじみの正式なインストール手順はない。

C.3.4 @Risk ソフトウェア

@Risk ソフトウェアは本書の中核をなすものではないが、3.2.3 項で簡単に取り上げている。本書執筆時点では、Palisade 社は完全な商用ソフトウェアである@Risk Industrial の 15 日間試用版を提供している (<https://go.palisade.com/RISKDownload.html>)。これは学生のみに適用されるが、もちろん 15 日間しか使えない。それ以上の期間使用する場合は、85 ドルで学生用ライセンスを取得することも可能である。教員は、一括購入オプションや、（より高価な）アカデミックライセンスを利用することができる。

早く使い始めるには、<https://go.palisade.com/OnDemand-IntrotoRisk.html> にある無料のウェビナー「Introduction to Risk Analysis using @RISK」を視聴するとよい。また、@Risk に関する多くの動画を YouTube のチャンネルで見ることができる：

<https://www.youtube.com/user/PalisadeCorp>

C.3.5 解答とその他のファイル（教員用）

PowerPoint スライド、厳選された問題の解答、すぐに教えられる LMS オンラインコース、学生用ソフトウェアのアクセスコード、その他多くの教員用リソースは、登録したインストラクタのみが利用可能である。詳細なリストとアクセス方法は、登録した教員に送られる Grant Award メールに記載されている。授業に適したケーススタディウェビナーは、次の URL で視聴できる：

<https://www.simio.com/resources/webinar/>

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. Elektrotteknikeren, 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] D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris. Fundamentals of Queueing Theory. Wiley, Hoboken, New Jersey, fourth edition, 2008.
- [19] 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.
- [20] W.J. Hopp and M.L. Spearman. Factory Physics. McGraw Hill, New York, New York, 2008.
- [21] ISA-95.com. Business to manufacturing markup language (B2MML). <https://isa95.com/b2mml/>, 2016.
- [22] J.R. Jackson. Networks of waiting lines. Operations Research, 5:518–521, 1957.
- [23] N.L. Johnson, A.W. Kemp, and S. Kotz. Univariate Discrete Distributions. Wiley, New York,

- third edition, 2005.
- [24] N.L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 1. Wiley, New York, second edition, 1994.
 - [25] N.L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume 2. Wiley, New York, second edition, 1995.
 - [26] 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.
 - [27] W.D. Kelton. Representing and generating uncertainty effectively. *Proceedings of the 2009 Winter Simulation Conference*, 40–44, 2009.
 - [28] W.D. Kelton, R.P. Sadowski, and N.B. Zupick. *Simulation With Arena*. McGraw-Hill, New York, sixth edition, 2015. (高桑宗右エ門監訳、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)
 - [29] 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.
 - [30] P.J. Kiviat, R. Villanueva, and H.M. Markowitz. *The SIMSCRIPT II Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, 1969.
 - [31] L. Kleinrock. *Queueing Systems: Volume I – Theory*. Wiley, New York, 1975.
 - [32] 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.
 - [33] A.M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, New York, fifth edition, 2015.
 - [34] 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.
 - [35] P. L'Ecuyer and R. Simard. Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33: Article 22, 2007.
 - [36] L. Leemis. Nonparametric estimation of the intensity function for a nonhomogeneous poisson process. *Management Science*, 37:886–900, 1991.
 - [37] D.H. Lehmer. Mathematical methods in large-scale computing units. *Annals of the Computing Laboratory of Harvard University*, 26:141–146, 1951.
 - [38] D.V. Lindley. The theory of queues with a single server. *Proceedings of the Cambridge Philosophical Society*, 48:277–289, 1952.
 - [39] J.D.C. Little. A proof for the queuing formula $I = \lambda w$. *Operations Research*, 9:383–387, 1961.
 - [40] J.D.C. Little. Little's law as viewed on its 50th anniversary. *Operations Research*, 59:536–549, 2011.
 - [41] 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.
 - [42] MESA International. Business to manufacturing markup language (B2MML). <http://www.mesa.org/en/B2MML.asp>, 2018.
 - [43] L. Mueller. Norwood re-department simulation models: Present and future. Master's thesis, Department of Quantitative Analysis and Operations Management, University of Cincinnati, 2009.
 - [44] R.E. Nance and R.G. Sargent. Perspectives on the evolution of simulation. *Operations Research*, 50:161–172, 2002.
 - [45] B.L. Nelson. *Stochastic Modeling, Analysis and Simulation*. McGraw-Hill, New York, first edition, 1995.

- [46] B.L. Nelson. The more plot: Displaying measures of risk & error from simulation output. Proceedings of the 2008 Winter Simulation Conference, 413-416, 2008.
- [47] B.L. Nelson. Personal communication. 2013.
- [48] B.L. Nelson. Personal communication. 2016.
- [49] C.D. Pegden. Deliver On Your Promise How Simulation-Based Scheduling Will Change Your Business. Simio LLC, Sewickley, PA, 2017.
- [50] C.D. Pegden, R.E. Shannon, and R.P. Sadowski. Introduction to Simulation Using SIMAN. McGraw-Hill, New York, second edition, 1995. (高桑宗右エ門訳 (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)
- [51] C.D. Pegden and D.T. Sturrock. Rapid Modeling Solutions: Introduction to Simulation and Simio. Simio LLC, Pittsburgh, Pennsylvania, 2013.
- [52] A.A.B. Pritsker. The GASP IV Simulation Language. Wiley, New York, 1974.
- [53] A.A.B. Pritsker. Introduction to Simulation and SLAM II. Wiley, New York, fourth edition, 1995.
- [54] D. Robb. Gartner taps predictive analytics as next big business intelligence trend. Enterprise Apps Today, 2012.
- [55] S.M. Ross. Introduction to Probability Models. Academic Press, Burlington, Massachusetts, tenth edition, 2010.
- [56] S.L. Savage. The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty. Wiley, Hoboken, New Jersey, 2009.
- [57] T.J. Schriber. Simulation Using GPSS. Wiley, New York, 1974.
- [58] T.J. Schriber. An Introduction to Simulation Using GPSS/H. Wiley, New York, 1991.
- [59] 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)
- [60] A.F. Seila, V. Ceric, and P. Tadikamalla. Applied Simulation Modeling. Thomson Brooks/Cole, Belmont, California, 2003.
- [61] Simio LLC. Simio web site. <https://www.simio.com>, 2018.
- [62] E. Song and B.L. Nelson. Quickly assessing contributions to input uncertainty. IIE Transactions, 47:1-17, 2015.
- [63] 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.
- [64] W.S. Stidham. A last word on $I = \lambda w$. Operations Research, 22:417-421, 1974.
- [65] D.T. Sturrock. New solutions for production dilemmas. Industrial Engineer Magazine, 44:47-52, 2012a.
- [66] D.T. Sturrock. Simulationist bill of rights. Success in Simulation Blog, 2012b.
- [67] J.J. Swain. Simulated worlds (simulation software survey). OR/MS Today, 42(5), 2015.
- [68] J.W. Tukey. Exploratory Data Analysis. Addison-Wesley, Reading, Massachusetts, 1977.
- [69] Visual Components. Factory simulation and the internet of things. <https://www.visualcomponents.com/insights/articles/factory-simulation-internet-things/>, 2015.
- [70] H.M. Wagner. Principles of Operations Research, With Applications to Managerial Decisions. Prentice Hall, Englewood Cliffs, New Jersey, 1969.