副テーマ

Plat Box Simulator による オブジェクト指向概念の学習効果

北陸先端科学技術大学院大学
知識科学研究科知識社会システム学専攻

藤光　勢

2006 年 10 月

Copyright © 2006 by ChiKara Fujimitsu

副テーマ

Plat Box Simulator による
オブジェクト指向概念の
学習効果

指導教官　　橋本敬

2006.10

北陸先端科学技術大学院大学

知識科学研究科　知識社会システム学専攻

社会システム構築論講座

550060　藤光　勢(Chikara　Fujimitsu)

目次

1.　はじめに　……………………………………………………………1
1.1　目的　………………………………………………………………1
1.2　背景　………………………………………………………………1
1.2.1　社会シミュレーション　…………………………………………1
1.2.2　オブジェクト指向の概要と必要性　……………………………2
1.2.3　オブジェクト指向技術の全体像と発生過程　…………………3
1.2.4　Plat Box Simulator　………………………………………………4
1.2.5　PBS の利点　………………………………………………………4

2.　Plat Box Simulator および Eclipse の導入　………………………6
2.1　Plat Box Simulator のインストール　……………………………6
2.2　Java のインストール　……………………………………………6
2.2.1　Java のダウンロード　……………………………………………6
2.2.2　環境設定　…………………………………………………………6
2.3　Plat Box Simulator のインストール　……………………………8
2.3.1　Plat Box Simulator のダウンロード　……………………………8
2.3.2　Component Builder の設定　………………………………………8

3.　モデル作成　……………………………………………………………10
3.1　Plat Box Project の作成　…………………………………………10
3.1.1　新規プロジェクトの作成　………………………………………10
3.1.2　モデル作成　………………………………………………………12
3.2　Box Hospital の世界　………………………………………………13
3.2.1　分析　………………………………………………………………13
3.2.2　設計　………………………………………………………………14
3.2.3　実装　………………………………………………………………14
3.3　Model のデザイン　…………………………………………………15
3.3.1　Model の新規作成　………………………………………………15
3.3.2　オブジェクトの配置　……………………………………………16
3.3.3　Model ソースコードの生成①　…………………………………17
3.3.4　Relation のデザイン　……………………………………………17
3.3.5　Model ソースコードの生成②　…………………………………18
3.4　World のデザイン　…………………………………………………18
3.4.1　Agent の追加　……………………………………………………19
3.4.2　Relation の追加　…………………………………………………22

3.5　シミュレーション(一回目)　……………………………………………23
3.6　患者の Behavior デザイン　……………………………………………24
3.6.1　偏移条件の設定①　……………………………………………25
3.6.2　偏移条件の設定②　……………………………………………26
3.7　患者の Action デザイン　……………………………………………27
3.8　シミュレーション(二回目)　……………………………………………28
3.9　受付嬢の Behavior のデザイン　……………………………………………28
3.10　Information のデザイン　—受付処理のトリガー作成—　……………………30
3.10.1　ランダム要素の作成　……………………………………………30
3.10.2　診察要求格納リストの作成　(空の List Information をつくる)　…………31
3.10.3　診察要求格納リストにそれぞれの患者の供給を登録　……………………34
3.10.4　リスト内の Information をランダムに選ぶ　……………………………35
3.10.5　取得した Information を送る　……………………………………………35
3.10.6　ソースコードの作成　……………………………………………35
3.10.7　Agent 患者に Information 診察券を持たせる　……………………………36
3.10.8　Agent の再設定　……………………………………………37
3.10.9　シミュレーション(2 回目)　……………………………………………37
3.11　受付嬢の Action デザイン　……………………………………………38
3.11.1　診察券を受け取る　……………………………………………38
3.11.2　[受け取った診察券情報]を参照・比較可能な文字列に変換する　…………39
3.11.3　見比べる項目を作成する　……………………………………………39
3.11.4　移動案内のセリフ(文字列)を作成する　……………………………………40
3.11.5　診察券の情報を比較検討する　……………………………………………41
3.11.6　それぞれの症状に合った診療科を移動案内として伝える　…………………43
3.11.7　ソースコード作成する　……………………………………………44
3.11.8　受付嬢に Information 移動案内を持たせる　……………………………44
3.12　シミュレーション(3 回目)　……………………………………………45

4.　評価　……………………………………………46

5.　まとめ　……………………………………………46

6.　今後の課題　……………………………………………47
6.1　評価項目について　……………………………………………47
6.2　作業時間について　……………………………………………47
6.3　複数人による実施について　……………………………………………47

7. 謝辞 ··48

8. 参考文献・公式マニュアル ··48

9. 資料 ··i
9.1 Abstract 受付受理.java ··ii
9.2 Abstract 診察要求.java ··iii
9.3 BoxHospitalModel.java ··vi
9.4 boxhospitalworldWorld.java ··vii
9.5 受付処理.java ··x vii
9.6 診察要求.java ··x ix

図･表　目次

図 1.1　オブジェクト指向技術の全体像と発展過程･･3
表 1　 PBS の動作環境 ･･6
図 2.1　システムのプロパティ ･･6
図 2.2　環境変数 ･･7
図 2.3　システム変数の編集 ･･･7
図 2.4　Plat Box All ･･8
図 2.5　Eclipse For Plat Box ･･･8
図 2.6　ワークスペース･ランチャー ･･9

図 3.1　Eclipse プラットフォーム ･･･10
図 3.2　新規 Java パッケージ　 ･･･10
図 3.3　Plat Box プロジェクトの新規作成 ････････････････････････････････11
図 3.4　新規プロジェクト作成後 Eclipse ･･････････････････････････････････11
図 3.5　クラス図新規作成 ･･15
図 3.6　クラス図新規作成ダイアログ ･･････････････････････････････････････15
図 3.7　Box Hospital のクラス図 ･･･16
図 3.8　クラス図のソースコード生成 ･･････････････････････････････････････17
図 3.9　Relation ･･･17
図 3.10　World の新規作成 ･･･18
図 3.11　World の新規作成ダイアログ ･････････････････････････････････････18
図 3.12　モデルの読み込み ･･･19
図 3.13　Agent グループ ･･19
図 3.14　Agent グループの設定(患者、受付嬢) ･･････････････････････････････20
図 3.15　Agent 数の変更 ･･20
図 3.16　値ダイアログ ･･21
図 3.17　Relation グループ ･･21
図 3.18　Relation グループの設定 ･･22
図 3.19　World の生成 ･･22
図 3.20　Java アプリケーションの実行 ･････････････････････････････････････23
図 3.21　Communication Viewer ･･23
図 3.22　シミュレーション(1 回目) ･･･24
図 3.23　Behavior Designer で開く ･･･24
図 3.24　偏移のプロパティ ･･･25
図 3.25　内部アクションの作成 ･･･26
図 3.26　偏移線 ･･･26
図 3.27　Action Designer で開く ･･･27

図 3.28　Action 診察要求のソースコード生成 ……………………………………27
図 3.29　Action グループの設定　Behavior Type の付加 ……………………………28
図 3.30　Behavior 受付処理 ……………………………………………………28
図 3.31　偏移のプロパティ(Channel Event) ……………………………………29
図 3.32　乱数生成 ………………………………………………………………30
図 3.33　乱数設定 ………………………………………………………………31
図 3.34　List Information 生成 …………………………………………………31
図 3.35　文字列生成 ……………………………………………………………32
図 3.36　3 つの文字列 …………………………………………………………32
図 3.37　String Information 生成 ………………………………………………33
図 3.38　List へ追加 ……………………………………………………………34
表 2　要求項目 …………………………………………………………………34
図 3.39　リストからの参照 ……………………………………………………35
図 3.40　Information の送信 ……………………………………………………35
図 3.41　Agent グループの設定　Information の付加 …………………………36
図 3.42　シミュレーション(2 回目) ……………………………………………37
図 3.43　受付処理の Action Designer …………………………………………38
図 3.44　Information の取得 ……………………………………………………38
図 3.45　Information を文字列に変換 …………………………………………39
図 3.46　条件に使う文字列を生成 ……………………………………………39
図 3.47　移動案内になる文字列生成 …………………………………………40
図 3.48　論理式記述 ……………………………………………………………41
図 3.49　論理式記述 ……………………………………………………………42
図 3.50　条件となる論理式の完成 ……………………………………………42
図 3.51　条件分岐作成 …………………………………………………………43
図 3.52　条件分岐作成 …………………………………………………………43
図 3.53　Agent グループの設定　Information Type の付加 ……………………44
図 3.54　シミュレーション(3 回目) ……………………………………………45

表 3　作業時間 …………………………………………………………………47

1. はじめに

1.1　目的

Plat Box Simulator(以下PBS)を用いて行うオブジェクト指向教育の評価を行う。オブジェクト指向およびオブジェクト指向プログラミング(OOP : Object Oriented Programming)は習得が困難といわれている。オブジェクト指向として一つの括りにされる技術には、Javaなどのプログラミング言語、要求仕様や設計内容の図式表現、再利用可能なソフトウェア部品群、優れた設計のノウハウ、業務分析や要求定義の効果的な決定方法、システム開発を柔軟に進める開発手法と多岐にわたる。こういった総合的な技術であるオブジェクト指向は個々の要素技術を習得するだけでなく、全体的あるいは抽象的に物事を考えることが必要であり、その習得に何年も費やすと言われている。本稿はオブジェクト指向言語によるプログラミング、UML[1]を用いたシステムの分析および設計を行ったことがない初学者を対象とし、オブジェクト指向概念の学習にPBSを用いて行った場合の評価を行うことを目的としている。

1.2　背景

1.2.1　社会シミュレーション

現在、様々な研究分野においてシミュレーションという手法が注目を集めている。シミュレーションとは設定したモデルと初期条件からそのモデルを時間的に展開させることをさす。その展開結果を通じて実験者(研究者)はモデルについての知見を得ることができる。また、シミュレーションでは設定条件を容易に変更できるため、大規模で複雑なモデルを扱うことも可能である。

学術的な研究においてシミュレーションの利用目的には大まかに三つある。まず、第一の目的は、モデルの振舞いを理解するということだ。初期条件の初期値の組み合わせによって、モデルがどのような振舞いをするのかを観察できる。第二に、対象の内部メカニズムの模索するために用いられる。メカニズムの仮説的なモデルを作成し、振舞いと対象を比較する。比較の結果をフィードバックしながら徐々にモデルを対象へと近づけていく。第三に、過去のデータを用いて将来を予測することに利用される。前もって妥当なモデルが構築されていることが前提である。モデルが時間経過によってどういった結果になるのかを分析するというものである。

[1] Unified Modeling Language統一モデリング言語。オブジェクト指向のシステム開発でモデルを開発者間で共有するために用いる表記法。[1]

1.2.2　オブジェクト指向の概要と必要性

オブジェクト指向とは「もの」を第一に考える考え方である。「もの」とは現実世界に実体を持って存在している「もの」である。しかし、オブジェクト指向では現実世界からの抽出が必要な作業となるわけだが、抽出されたオブジェクトだけではソフトウェアを構築することは困難である。オブジェクト指向とは現実世界の「もの」を中心に考え、ソフトウェアの世界でオブジェクトとなる「もの」を抽出してソフトウェアを構成することである。ソフトウェアの世界で必要なオブジェクトを追加して、実際に動作するシステムの構築が可能となる。

オブジェクト指向がなぜ必要なのか。近年、ソフトウェア開発において最も重大な問題はシステムの大規模化である。企業の根幹システムとなると数十人、数百人が一年以上の歳月を掛けてソフトウェア開発を行うのが一般的となった。そういった現状からソフトウェアの開発では従来のようなスクラッチ開発[2]ではなく、様々な背景を持つソフトウェア部分を自分のプログラムと組み合わせて作成するようになった[2]。

[2] パッケージなどを使用せずに独自に一から行う開発手法。また、完全に一 から開発するわけではなく、既存システムを利用して開発を行う場合もスクラッチ 開発という。

1.2.3　オブジェクト指向技術の全体像と発生過程

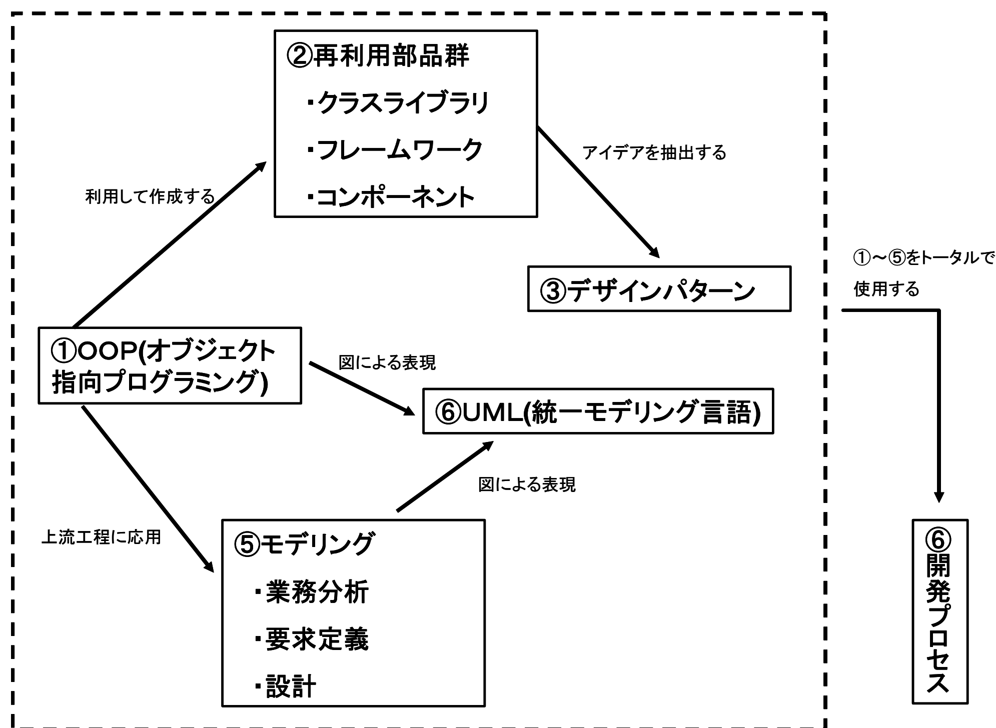

図 1.1　オブジェクト指向技術の全体像と発展過程

[出展]　平澤章，2004，「オブジェクト指向でなぜつくるのか」日経 BP 社．p.23

オブジェクト指向では大規模化で困難になったソフトウェア開発において、開発を円滑に進める総合的な技術の総体になっている[3]。平澤[平澤.2004]を要約するとオブジェクト指向の全体像と発展過程は次のようなものとなる。

オブジェクト指向技術の起源はプログラミング言語から始まっている(図 1.1①)。1967 年にノルウェーで考案されたSimula67 はオブジェクト指向のクラス[3]、ポリフォーリズム[4]、継承[5]という優れた仕組みを備えており、後に最初のOPP(Object Oriented Programming language:オブジェクト指向プログラミング言語)と呼ばれるようになった。Simula67 の有効性が認められるにつれ同様の仕組みを備えたプログラミング言語が数多く考案された。

---

[3] クラス(Class)とはオブジェクト指向においてデータとメッセージを記述した型を定義したもの、またインスタンス(Instance)はクラスにおける具体的なオブジェクトを指す。集合論の集合と要素に相当する。クラスを定義することによって同種類のオブジェクトをまとめて扱うことができるようになる。

[4] ポリモルフィズムとは、「poly＝多様の」と「morphism＝形態をもっている状態」の意味が合成された言葉。多態性と訳すことが多い。類似したクラスに対するメッセージの送信方法を共通にする仕組み[3]。

[5] クラスの共通点と相違点を体系的に整理する仕組みで、スーパークラスの性質はすべてサブクラスに継承されサブクラスではスーパークラスとの違いを定義するだけでよい[3]。

OPP の仕組みにより、クラスライブラリやフレームワークといった大規模ソフトウェアの再利用部品群を作成することが可能になった(図 1.1②)。また、そうした部品群を作成する際に一般的な設計のアイデアがデザインパターンとして抽出された(図 1.1③)。OPP によるソフトウェア開発を図式表現する技術が考案され、最終的には UML としてまとめられた(図 1.1④)。これに加えて OPP の概念を上流工程にも応用したモデリング(図 1.1⑤)やシステム開発全体を円滑に進行させるための開発プロセスも考案された(図 1.1⑥)。これらの技術群は扱う領域こそ異なるが開発を円滑に進めるという目的は一致している。

1.2.4 Plat Box Simulator

Plat Box Simulator(以下PBSと略)とは慶應義塾大学総合政策学部井庭崇研究室[6]によって開発された社会シミュレーション支援ツールである。シミュレーションの構築において、モデルにどういった要素取り入れるのかを決定する作業と、シミュレーションを作成するためのプログラミング作業が不可欠となる。前者後者にかかわらず、初心者が白紙の状態からモデルを構築していくことは大変な負担となる。この負担に対する支援として開発されたのがPlat Box Simulatorである。

井庭崇研究室ではモデル化を支援するモデル･フレームワーク「Plat Box 基礎モデル」や、シミュレーション作成を支援する「Component Builder」を提供している。

1.2.5 PBS の利点

PBS を用いる四つの利点を挙げる。

① 特定のプログラミング言語に依存しないモデル作成

これまでのシミュレーションのモデル作成には支援ツールを有無にかかわらず、最終的にはC言語やJava言語などのプログラミング言語による実装が必要であった。そのため、作成されたモデルが特定の言語に大きく依存する。依存の結果、用いた言語に対する知識が無い人はモデルを理解できず、仮に知識がある場合も理解することが困難となる。他方、UML[7]やアクション記述言語を用いると言語の文法に関係なくモデル作成が可能である。言語の知識が無い人でもモデル作成を行い、また他人のモデルも理解できる利点がある。

② 作業量の軽減

図で記述されたモデルからソースコードを自動生成することにより、本質的でない作業を削減することが出来る。

---

[6]慶應義塾大学総合政策学部 井庭崇研究室 Plat Box Project http://platbox.sfc.keio.ac.jp/からダウンロードが出来る。

③　反復的なモデル作成

シミュレーションのモデル作成はシミュレーションの正当性、妥当性を検証するために反復的に行われる。シミュレーションが実働するまで、作成者は概念モデリング、シミュレーションデザインを経て、設計をシミュレーションのソースコードに変換する作業を行う。しかし、最初から意図どおりに動くとは限らない。製作過程の様々な要因からバグが発生し、何度かの修正作業を重ねて完成する。

図に記述された設計モデルからソースコードを直接、自動生成することにより修正に伴うコストを取り除くことができる。

④　信頼性の向上

シミュレーションにおける信頼性は大変重要な問題である。通常のプログラミングと異なり、シミュレーションの場合は事前に結果を予測することが出来ない。そのため、バグの発見が通常より困難であり、シミュレーションのモデル作成には高い信頼性が求められる。

人の手で設計モデルをソースコードに変換する場合、たとえ設計が正しくともバグが混入する可能性がある。こうしたリスクを自動生成により回避し、高い信頼性をもったプログラミングが可能である。

2.　Plat Box Simulator および Eclipse の導入

この章ではモデル構築後、シミュレーションを実行という一連の流れを説明する。ここでは動作環境を整える導入部分から説明していく。

2.1　Plat Box Simulator のインストール

PBSの動作環境は以下の表1にまとめた。

表1.　PBSの動作環境

必須環境	
OS	Windows 98/98SE/Me/2000/XP
CPU	300 MHz以上
メインメモリ	128MB以上
推奨環境	
OS	Windows Me/2000/XP
CPU	700MHz以上
メインメモリ	256MB以上

2.2　Javaのインストール

2.2.1　Javaのダウンロード

PlatBoxのソースコードはJavaで記述される。そのため事前にPCをJavaが動作する環境にしておく必要がある。今回はJava2SDKをPCにインストールした。Java2SDKは以下のアドレスのサン･マイクロシステムズ[8]からダウンロードが出来る。

2.2.2　環境設定

ダウンロード後、インストールを実行する。その後、環境変数を設定する。設定方法は[Windowsスタート]⇒[コントロールパネル]⇒[システム]⇒[詳細設定]。図2.1を参照に[詳細設定]のタブから[環境変数]を選択する。

図2.1　システムのプロパティ

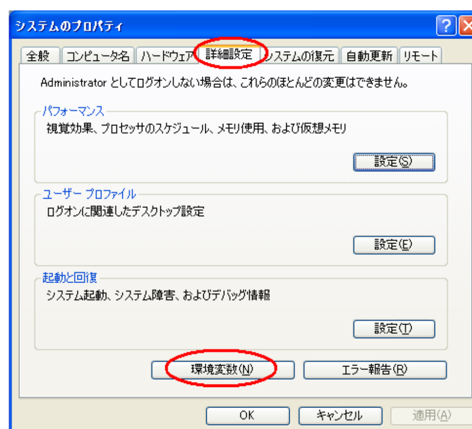

図2.2を参照にシステム環境変数の一覧から[Path]をクリックして[編集]を押す。

---

[8] http://www.atmarkit.co.jp/fjava/rensai2/javaent01/javaent01.html

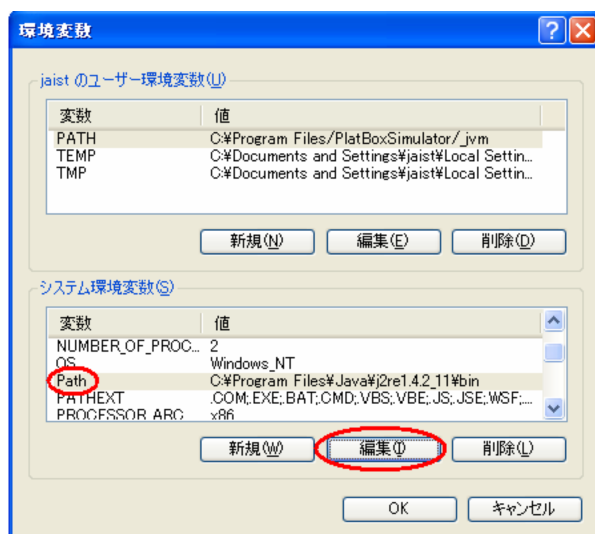

図2.2　環境変数

図2.3で示すように編集画面で[変数値]を先ほどインストールしたJavaのbinファイルまでのパスを記入し[OK]を押して設定完了。

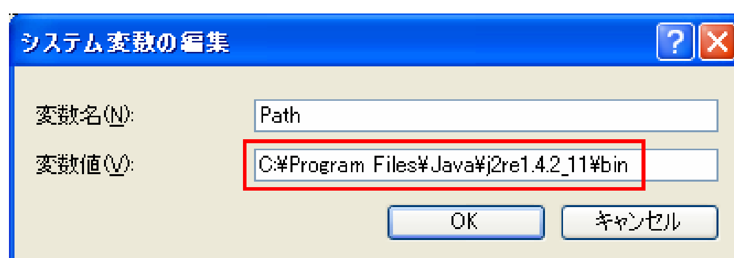

図2.3　システム変数の編集

2.3　Plat Box Simulatorのインストール

2.3.1　Plat Box Simulatorのダウンロード

http://platbox.sfc.keio.ac.jp/(慶應義塾大学総合政策学部 井庭崇研究室　Plat Box Project)からPlat Box SimulatorおよびComponent Builder(以下CB)をダウンロードする。(ダウンロードには登録が必要)。詳細は[4]を参照。

ダウンロード後、[PlatBoxSimulatorInstaller]の解凍を行う。図2.4で示した[Setupwin32.exe]と書かれたファイルを実行しインストールを行う。

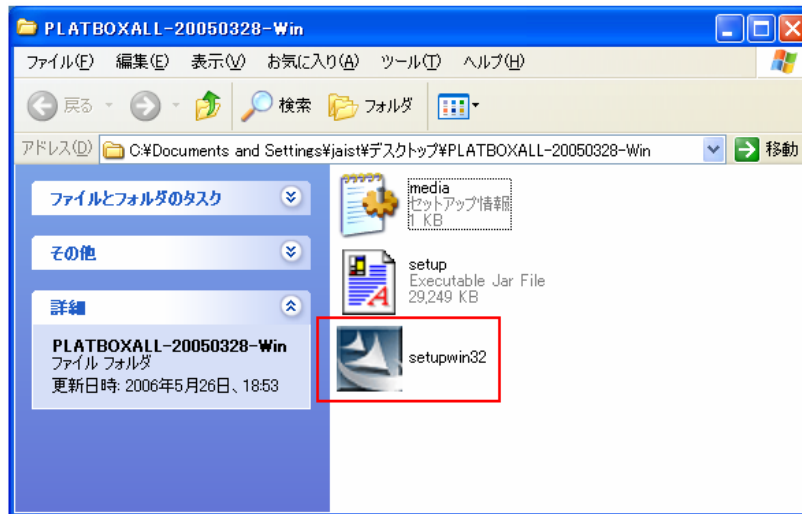

図2.4　Plat Box All

2.3.2　Component Builderの設定

次にComponent Builder(以下CB)の設定を行う。CBはPBSを構築、編集などを行うソフトウェアである。具体的にはEclipseというソフトウェア開発環境(IDE)の一つでJava開発者を中心に急速に普及している。そして、このEclipseをPBS用にカスタマイズしたものがCBである。圧縮されたファイルを解凍後、フォルダごと[C:¥Program Files]に移動する。次に図2.5で示した[eclipse.exe]を実行する。

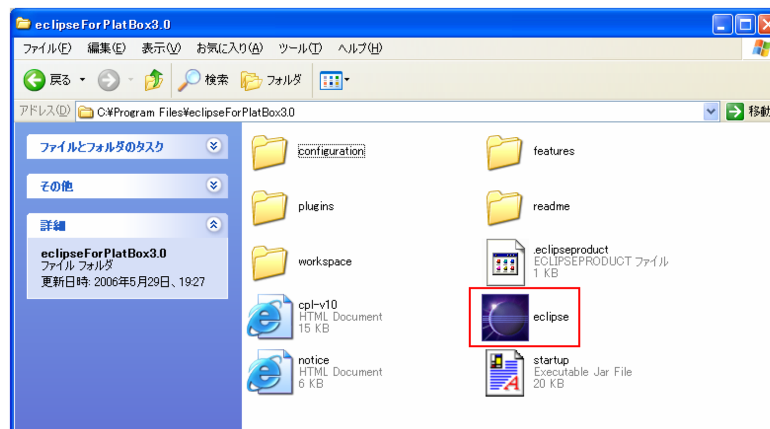

図2.5　Eclipse For Plat Box

Eclipse起動後、ワークスペース(作成したプロジェクトを保存する場所)を指定する。ディフォルト設定では図2.6のように[C:¥Program Files¥eclipseForPlatBox3.0¥workspace]になっているのでそのまま[OK]を押す。

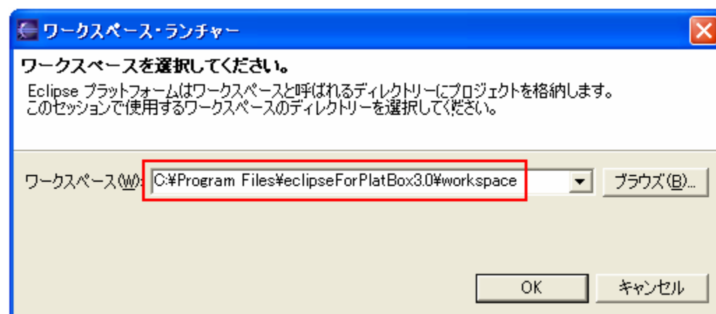

図2.6　ワークスペース･ランチャー

ワークスペースを設定後、CBの更新を行う。

3. モデル作成

環境が整ったのでシミュレーションの作成を行う。

3.1 Plat Box Project の作成

具体的なシミュレーション作成を行う前に、作成するモデルの情報を一括管理するための｢プロジェクト｣を作成する必要がある。手順は以下の通りである。

3.1.1 新規プロジェクトの作成

Eclipse を起動する。起動後は図 3.1 のダイアログが表示される。

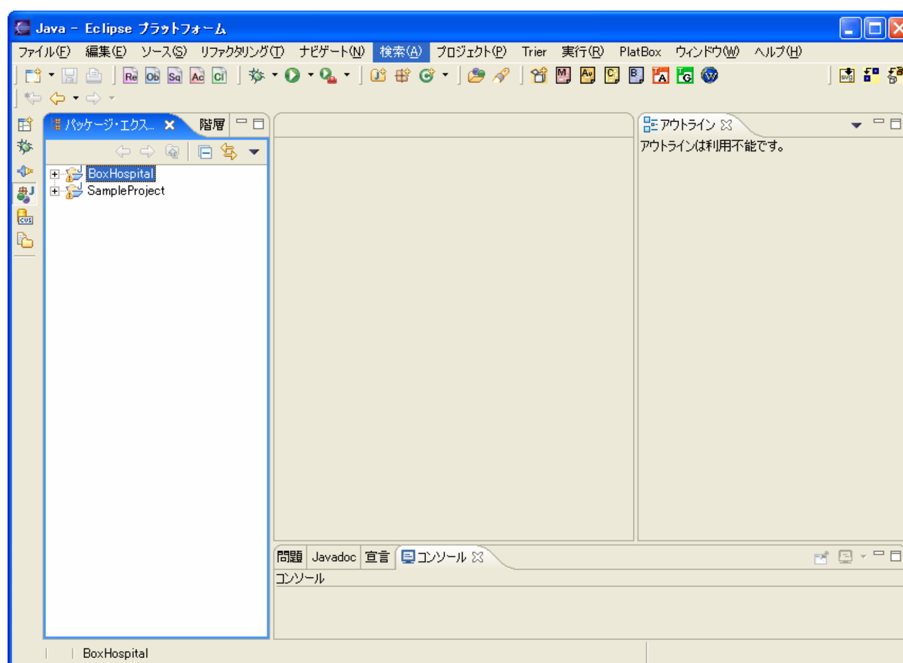

図 3.1　Eclipse プラットフォーム

[Create PlatBox Project]を押し、新規プロジェクトを作成する。

プロジェクト名は[BoxHospital]にして[終了]。自動でプロジェクトが生成される。

[パッケージ･エクスプローラ(左窓)]から新規作成した[BoxHospital]を選択する。

次に図 3.2 で示す[新規 Java パッケージ]をクリックする。

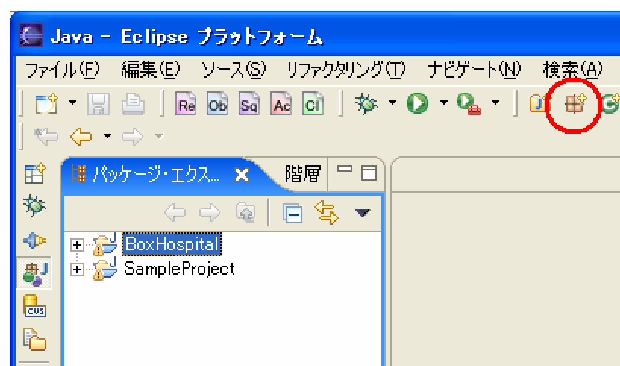

図 3.2　　新規 Java パッケージ

図 3.3 で示すように[新規 Java パッケージ]ウィザードで[名前]を[boxhospital]とする。

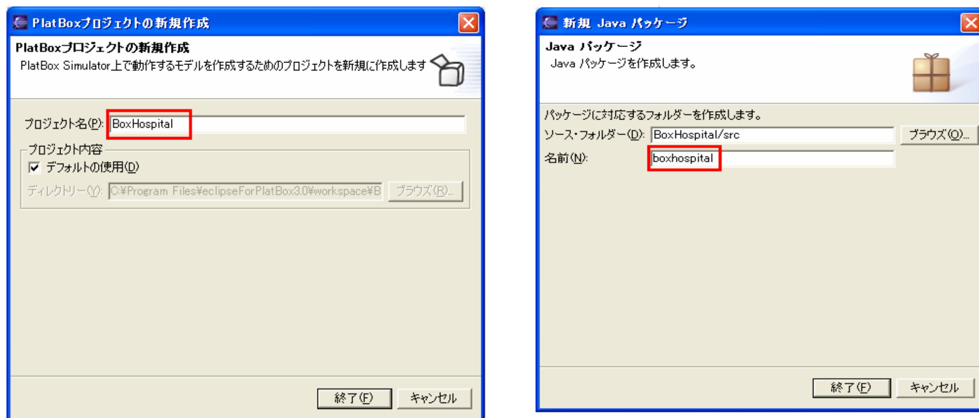

図 3.3　PlatBox プロジェクトの新規作成

以上の通りにプロジェクトと Java パッケージを作成するとエクスプローラ内は下図 3.4 のようになる。なお、[modeldb.xml]は次のモデリングを行うと自動生成される。パッケージを作成した段階では作成されていない。

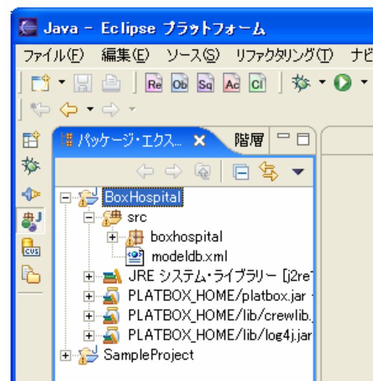

図 3.4　新規プロジェクト作成後 Eclipse

3.1.2　モデル作成準備

Box Hospital の世界にどのような人物が登場し、どのような行動をして、どういった情報や関係を持っているのかをデザインしていく。

全体的な流れとしては、

① 登場人物(必ずしも人ではない)[Agent]　情報[Information]　行動[Behavior]関係[Relation]の配置、Model のデザイン
② Behavior のデザイン
③ Action のデザイン
④ World 作成
⑤ Plat Box Simulator でシミュレートを行う。

である。ただ、これらの作業を一回で行うことは無理である。モデルを構築するごとに設計通り世界が構築されているかを確認しながら行わなければならない。特に初心者は確認を重要視する必要がある。なぜなら自分が今、行った操作がどうのように世界に繁栄されたのかを確かめながら行うとより一層の学習効果が得られるからである。この確認作業を上の流れに組み込むと、

① Model デザイン
② World 作成　&　Simulator 実行
③ Behavior デザイン
④ World 作成(更新)　&　Simulator 実行
⑤ Action デザイン
⑥ World 作成(更新)　&　Simulator 実行

である。もちろん Simulator の動作が意図したものではなかった場合、修正を加えていかなければならない。

3.2　Box Hospital の世界

Box Hospital についてシステム開発の過程(分析⇒設計⇒実装)を示す。

3.2.1　分析

大病院において長大な診察待ち時間は患者の大きな負担となる。この待ち時間を軽減できないか、軽減可能な診療システムはどのようなものかを模索する。その場合、現行のシステムはどのようになっているのかを知るのがこのレポートで取り上げた題材(Box Hospital)である。

病院は複数の診療科を備え、総合診療を行っている公共性の高い施設である。通常の患者は診察券を提示し、自らの症状に関係する診療科に訪れる。また、通常とは逆に、救急車で搬送されてくる緊急患者を受け入れている。また、患者は診察科が一つとは限らず、複数科にまたがる患者もいる。訪れる患者は一回の診察で終える一過性の類と入院や手術などの継続的な治療を必要とする二種類に分類できる。診療後は処方箋を貰う。これは院内の場合もあれば院外の場合もある。

では、病院のシステムはどのようなものが考えられるのか。患者が提示する診察券を受け取るのは人間とは限らず、近年は機械処理するところも増えている。もちろん、人間と機械混合もありえる。混合の場合は処理能力の差が設計に影響を及ぼすだろう。病院のシステムや患者の具合によっては診療前にレントゲン、MRI、CT スキャンや採血、アンケートによる問診などの検査が事前に行われることもあるし、診療中に併せて検査することもある。

以上の全てを考慮したうえで病院における診療の流れをシミュレートすることは初学者によって大変困難である。

よって今回は以下のように簡略にまとめておくにとどめる。

病院に訪れた患者は受付嬢(人間に限定)に向かって自分の症状(頭痛や腹痛、けがの三種類に限定)など診察券で伝える。その情報を得た受付嬢は症状の分類(限定されて多三種類の症状にそれぞれ対応した診療科のみを参照)に従って処理を行い、適切な診療科への移動案内を伝える。診療科の移動案内を出すだけで患者は実際には移動しない。病院に訪れる患者は設定上、変更可能だが世界にすでに実体化している。つまり、自動発声は行わず、規定数の患者の処理のみを行う。

3.2.2　設計

Box Hospital の世界では、以下のクラスを抽出してみた。

Agent 患者(病院に訪れた患者)

World に存在する患者。すでに実体として存在する。個体数は固定。

Agent 受付嬢(病院にて総合受付を行う)

World に存在する受付嬢。すでに実体として存在する。個体数は固定。

Information 診察券(患者の状態、具体的な症状を説明したもの)

患者の状態(頭痛、腹痛、膝のけが)の情報を付加したクラス

Information 移動案内(患者の要望を受けて受付処理したもの)

適切な診療科(外科、内科、脳神経外科)の案内情報を付加したクラス

Behavior 診察要求(自分の症状を受付嬢に伝える行動)

病院に到着し、患者自身の状態を診察券に付加し受付嬢に伝える行動

Behavior 受付処理(患者の要望を受けて適切な診療科を案内する行動)

患者の状態を分類表で参照し、適切な診療科の案内を伝える行動

Action 診察要求(Behavior 診察要求の具体的な行動)

頭痛、腹痛、膝のけがの 3 つからランダムに状態を付加し、その情報を受付嬢に伝える。

Action 受付処理(Behavior 受付処理の具体的な行動)

分類表を作成し患者の状態と比較を行う。項目が等しい診療科への案内を患者に伝える。

Relation 病院関係(患者と受付嬢の関係)

Information を伝えるときに用いる関係。

3.2.3　実装

PBS を用いたモデル作成では実質的な実装は[ソースコード作成]で自動生成されるため無い。本来ならば、ここで特定のプログラミング言語を用いて実装していく。つまり、PBS は設計と実装を同時進行的に行っている。(1.2.4PBS の利点②参照)

3.3　Model のデザイン

3.3.1　Model の新規作成

各パーツを配置していく。図3.5 を参照にパッケージ･エクスプローラの[Box Hospital]⇒[src](Source の意)⇒[box hospital]を選択。[Create Model for Plat Box Simulator]を押す。

図 3.6 の示すように[クラス図の新規作成]ウィザードでファイル名を[患者と受付嬢]として[終了]を押す。

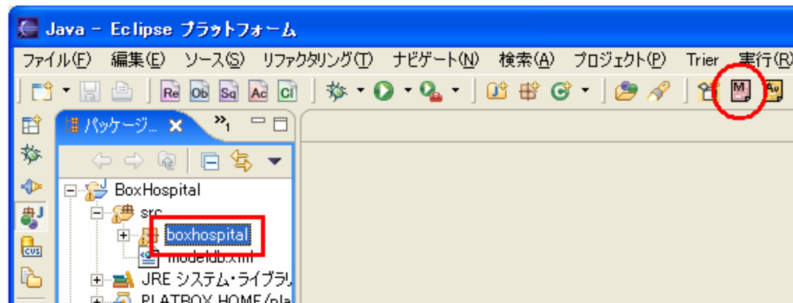

図 3.5　クラス図新規作成

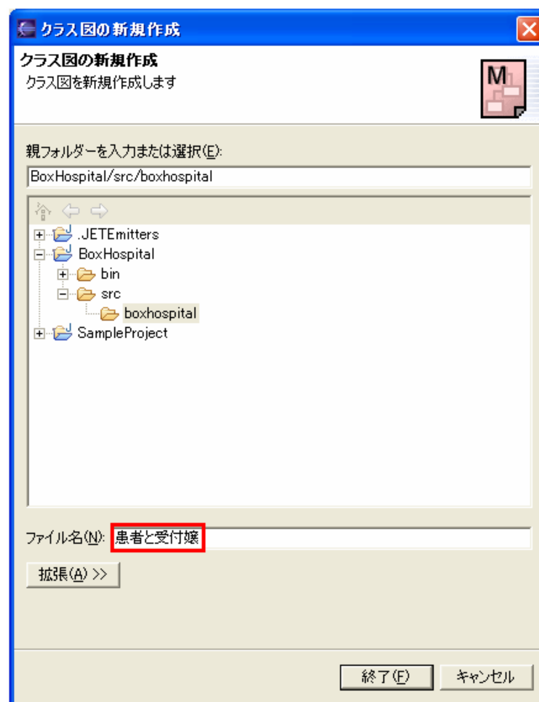

図 3.6　クラス図新規作成ダイアログ

3.2.2　オブジェクトの配置

図 3.7 を参照に[患者と受付嬢.model]に各パーツを配置していく。左窓の Agent を選び、フィールドでドラック&ドロップで描ける。描いたらタイトルをダブルクリックして名前を[患者]に変更する。同様にそれぞれのパーツをフィールドに描く。Relation 病院関係を除くすべてを配置したら[コネクション]の[関連]を選択。以下の図に従って線を引く。線は結びたい 2 つのパーツをシングルクリックすると自動的に 2 つを結ぶ線が描かれる。

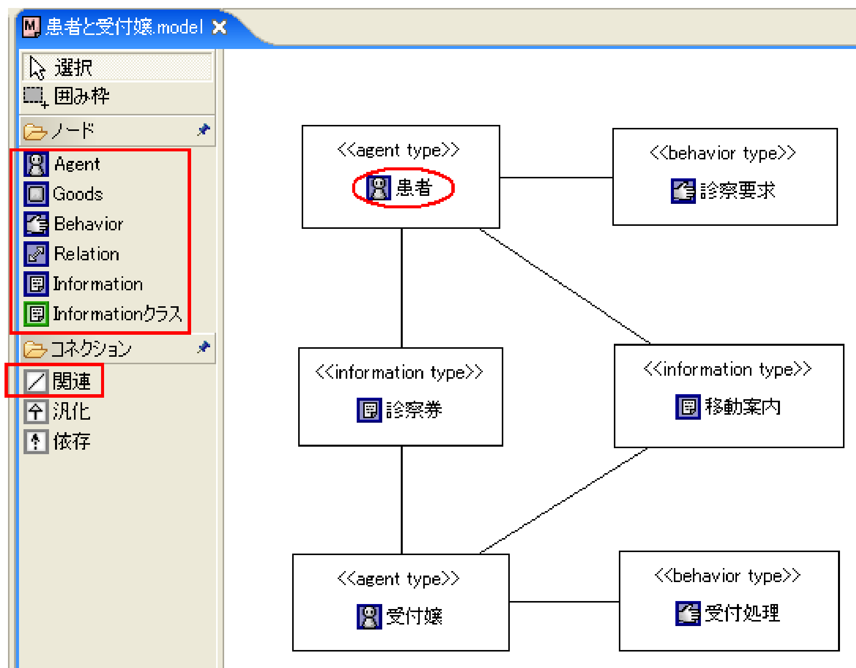

図 3.7　Box Hospital のクラス図

この図を描くには、言い換えると[関連]をどのように結ぶかが問題である。基本は分析(3.2.1)や設計(3.2.2)から関係性を求め、それぞれを結ぶ。例を挙げると、患者は診察券を受付嬢に提出する(患者→診察券→受付嬢)。患者は受付嬢から移動案内を受け取る(受付嬢→移動案内→患者)。この二つの行動それぞれが Behavior によって行われる(患者→診察要求、受付嬢→受付処理)。

3.3.3　Model ソースコードの生成①

図 3.8 のようにフィールドの何も記述してない場所で [右クリック]⇒[モデルのソースコード生成]を選択。パッケージ･エクスプローラに[BoxHospitalModel.java]が自動生成される。

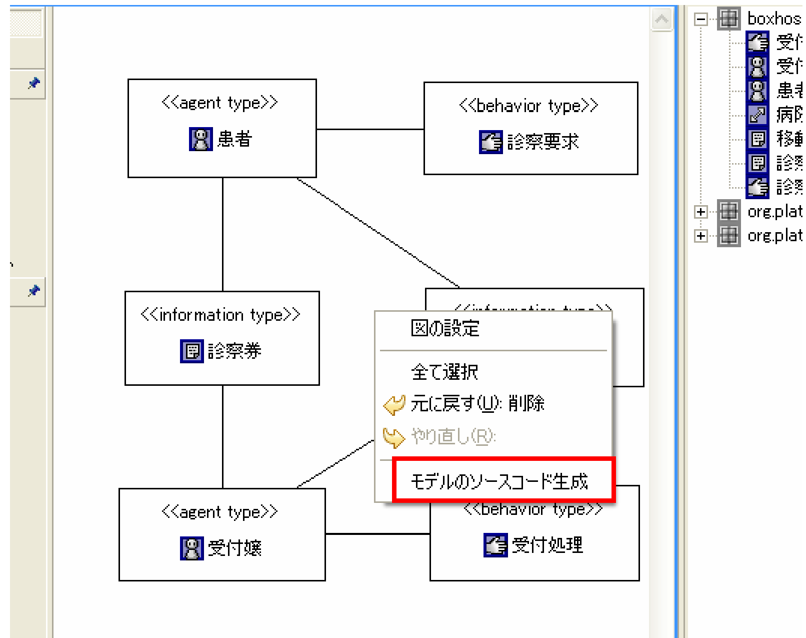

図 3.8　クラス図のソースコード生成

3.3.4　Relation のデザイン

次に患者と受付嬢の関係を表す Model を作る。パッケージ･エクスプローラの[BoxHospital]⇒[src]を選択後、[Create Model for PlatBox Simulator]を押す。ファイル名は[患者と受付嬢の関係]に変更する。Agent 患者、受付嬢と Relation を配置。先程の[患者と受付嬢.model]で作成した Agent 患者、受付嬢は作成しているので右窓のアウトラインの[boxhospital]からドラック＆ドロップでフィールドに配置する。Relation のタイトルを[病院関係]に変更する。図 3.9 を参照に[関連]の線で結ぶ。

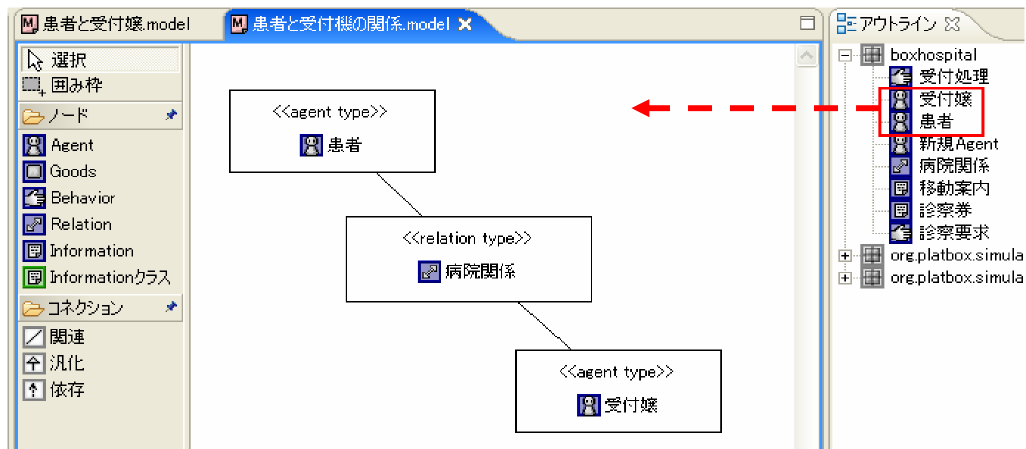

図 3.9　Relation

3.3.5　Model ソースコードの生成②

同様にフィールドで[右クリック]⇒[モデルのソースコード生成]を選択肢する。Model はひとつの Java ファイルに統合されるので新しい Java ファイルは生成されずに 2 回目以降は追加更新される。

3.4　World のデザイン

次に Model デザインしたものを反映させた世界[World]を作成する。図 3.10 で示した[box hospital]を選択して、[Create World for Plat Box Simulator]を押す。

図 3.11 の示すように[World の新規作成]で[ファイル名]を[boxhospitalworld]に変更後[終了]を押す。自動生成が開始される。

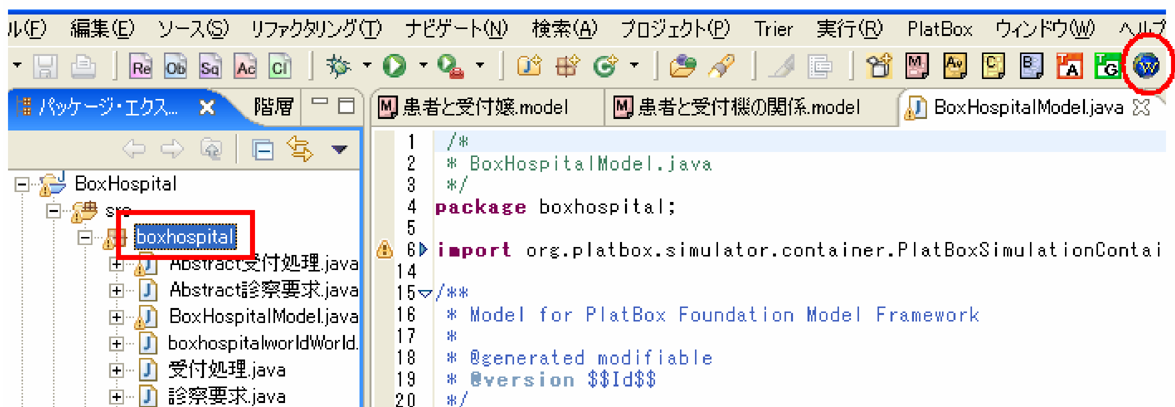

図 3.10　World の新規作成

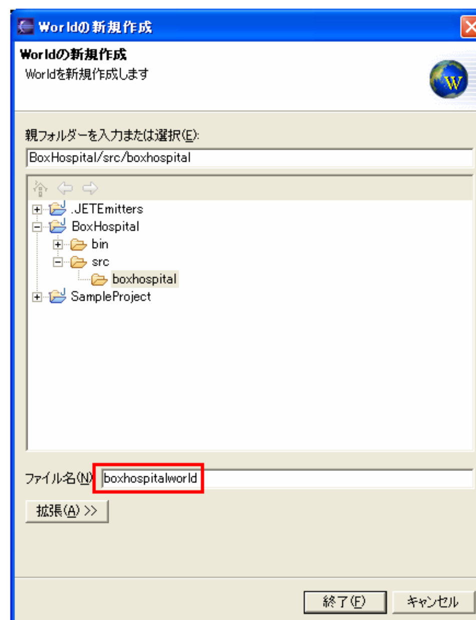

図 3.11　World の新規作成ダイアログ

生成されたファイルをダブルクリックする。図 3.12 で示す[モデルの読み込み]を押す。

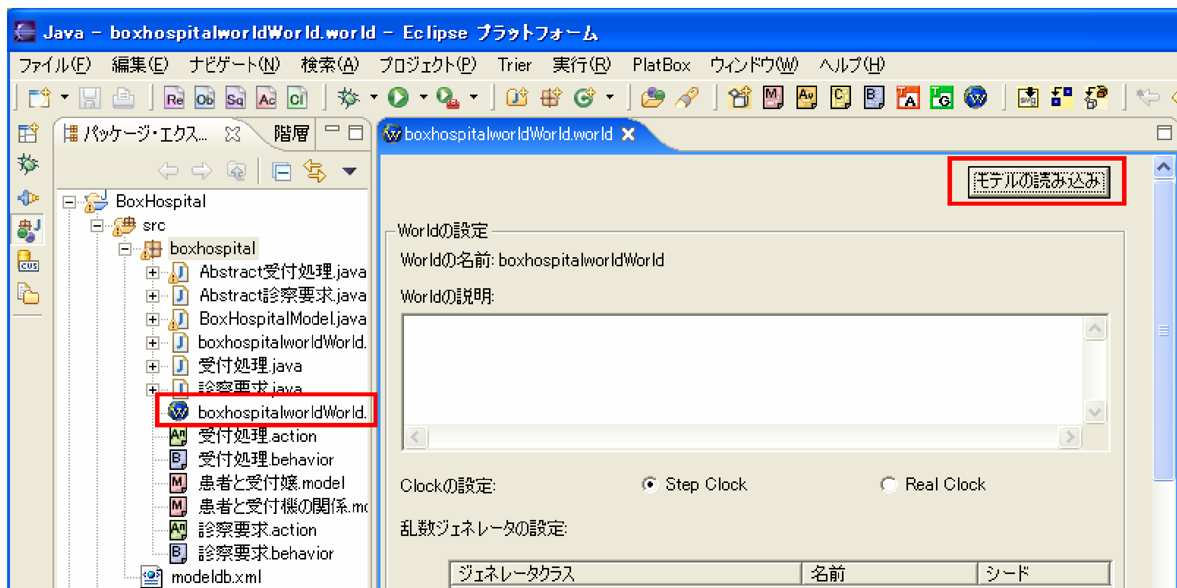

図 3.12　　モデルの読み込み

3.4.1　Agent の追加

続いて画面をスクロールさせて中段にある[Agent グループ]の部分を表示させる。図 3.13 で示した[追加]を押して設定ウィンドウを開く。

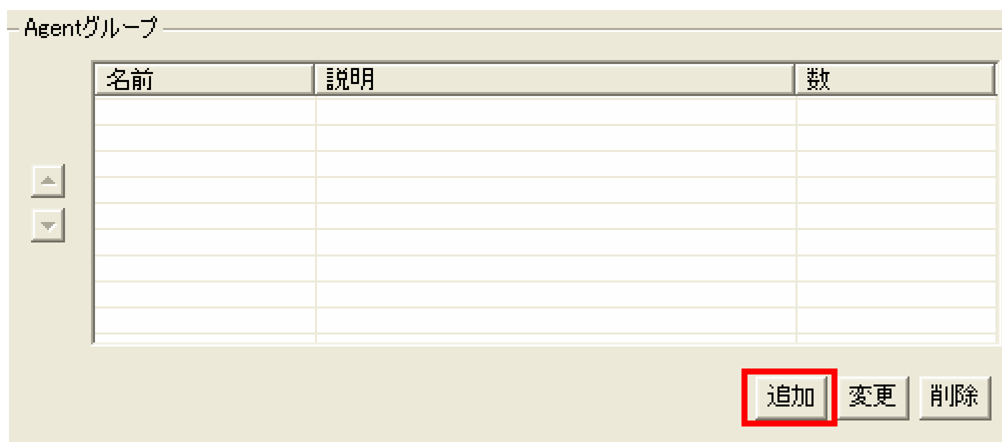

図 3.13　Agent グループ

[名前]を[患者集団]に変更。図 3.14 で示すように Agent Type はタブの中から[AGENTTYPE_患者]を選択し[OK]を押す。同様に受付嬢の Agent も World に登録する。[追加]を押し[名前]は[受付嬢集団]、 Agent Type は [AGENTTYPE_受付嬢]を選択して[OK]を押す。

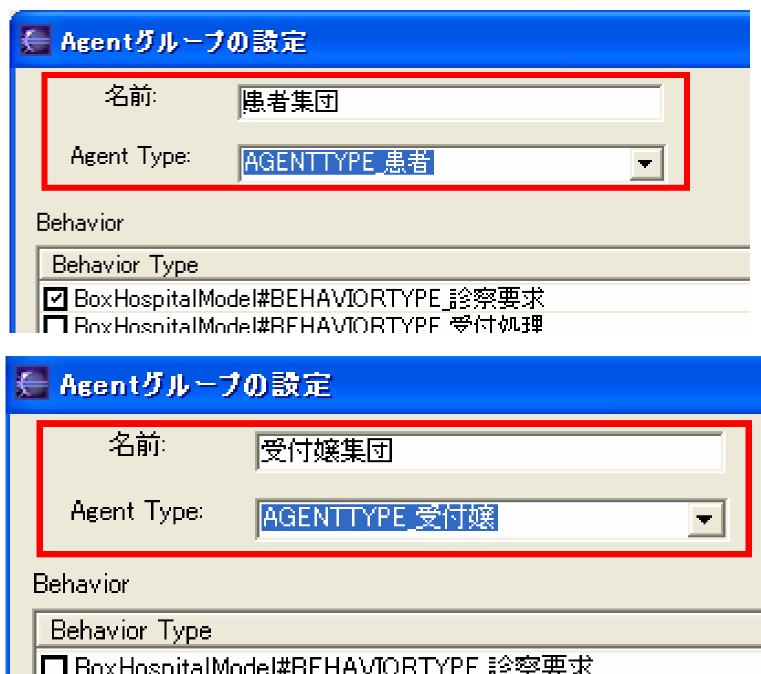

図 3.14　Agent グループの設定(患者、受付嬢)

次に Agent の個体数を変更する。図 3.15 示したように数を変更したい Agent の[値]のセルをクリックする。そうすると線で囲んだところにボタンが現れる。そのボタンをクリックしウィザードが立ち上げる。

Agentグループ

名前	説明	数
患者集団	(患者)1behaviors,0goods,1info	2
受付嬢集団	(受付嬢)1behaviors,0goods,1info	1

図 3.15　Agent 数の変更

図 3.16 を参照にして[型]を[Int]を選択、[値の直接入力]で数を入力する。

図 3.16　値ダイアログ

次に患者グループと受付嬢グループの関係を設定する。World の画面をスクロールして [Relation グループ]の部分を表示する。図 3.17 で示す[追加]ボタンを押す。

図 3.17　Relation グループ

3.4.2　Relation の追加

図 3.18 を参照に以下の操作を行う。[Relation グループ設定]ウィザードで[関係元の Agent グループ]では[患者集団]にチェック、[関係先の Agent グループ]では[受付嬢集団]にチェックを入れる。[名前]を[患者と受付嬢の病院関係]をする。二段目の[Relation Type]をコンボボックスの候補から選ぶ(Model で作った Relation が候補としてあるはず。無い場合は Model ソースコードの生成失敗か Model デザインの失敗である)。[方向]は[双方向]。Relation パターンは[Random]を選択、[本数]は[1]とする。なお Agent 同士の関係が 1 対 1 ならば[Random]は成立しない(警告が出る)。最後に[OK]を押す。

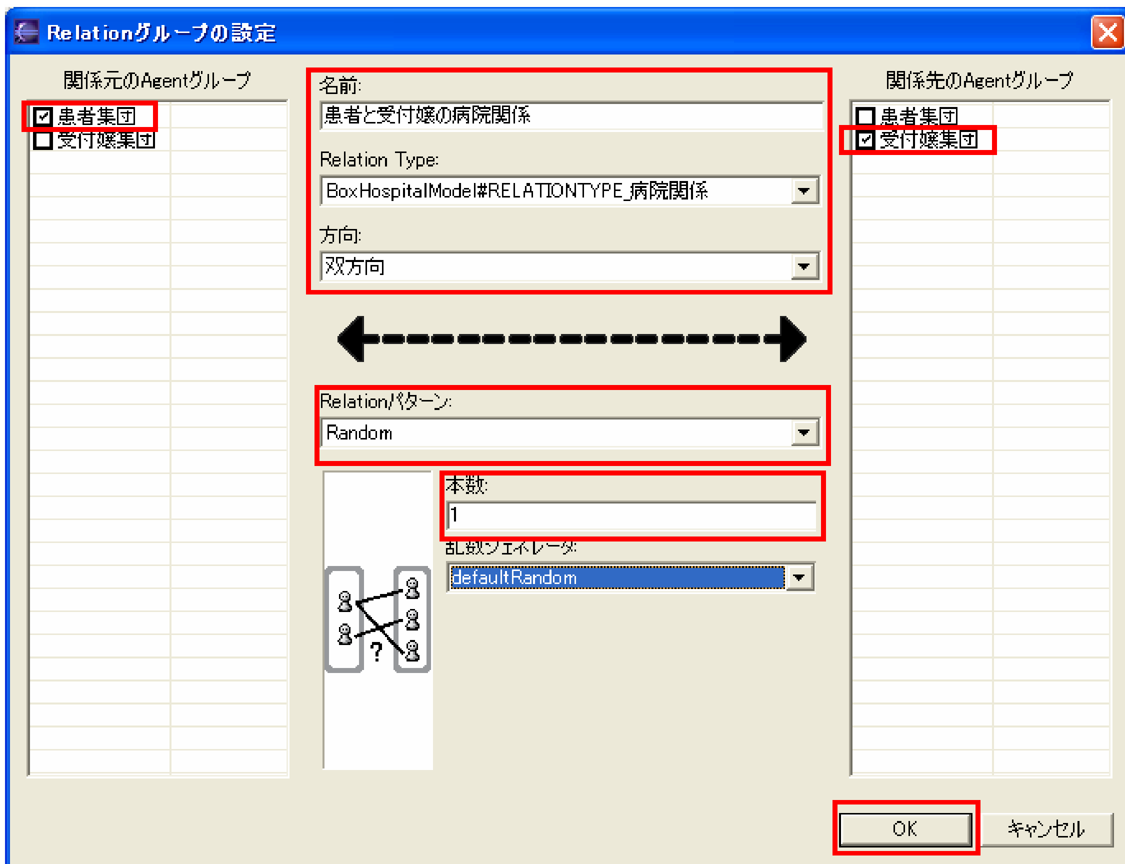

図 3.18　　Relation グループの設定

最後に、最下部にある図 4.15 に示す[World の生成]を押す。押すと World のコードが自動生成される。

図 3.19　World の生成

3.5　シミュレーション(1 回目)

実際に PBS を行い現在、世界がどのようになっているかを確認する。
PBS を起動させるには図 3.20 で示すように[実行]のコンボボックスの[次を実行]⇒[Java アプリケーション]を選択する。(次回以降は[実行]を押すだけで起動する)

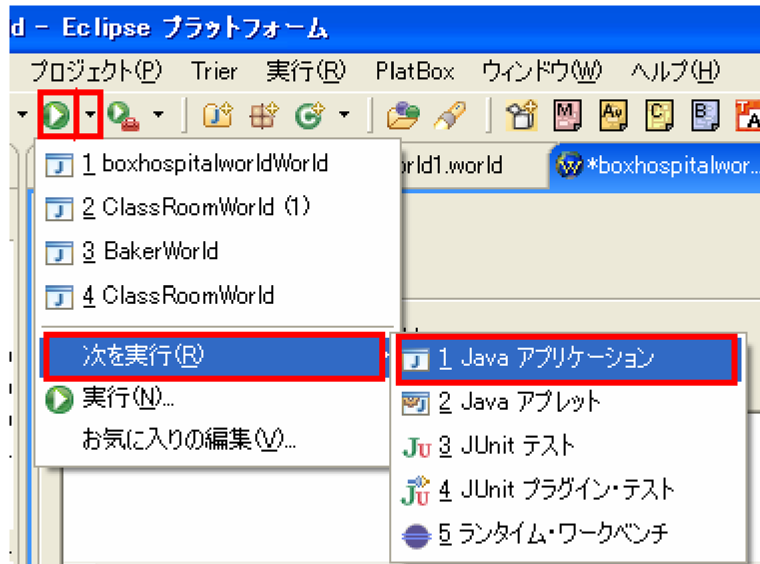

図 3.20　Java アプリケーションの実行

PBS を起動させ、作成した World を視覚化するために以下の操作をする。
図 3.21 を参照にして[ビューア]⇒[Communication Viewer]を選択する。

図 3.21　Communication Viewer

[Communication Viewer]を起動させると、作成した世界がイメージ化される。
図 3.22 では患者が 2 人､受付嬢 1 人がそれぞれ病院関係で結ばれているのが確認できる。

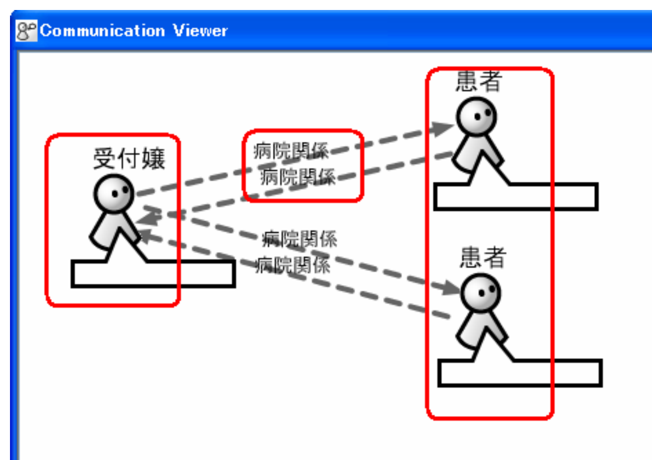

図 3.22　シミュレーション(1 回目)

3.6　患者の Behavior デザイン

意図したとおりに Agent 群が World に登録できたので、次は Agent の立ち振る舞い Behavior をデザインする。2-3-1 で作成した[患者と受付嬢.model]をダブルクリックして開く。図 3.23 で示すように Behavior Type[診察要求]を右クリックしてメニューを出し、[Behavior Designer で開く]を選択する。

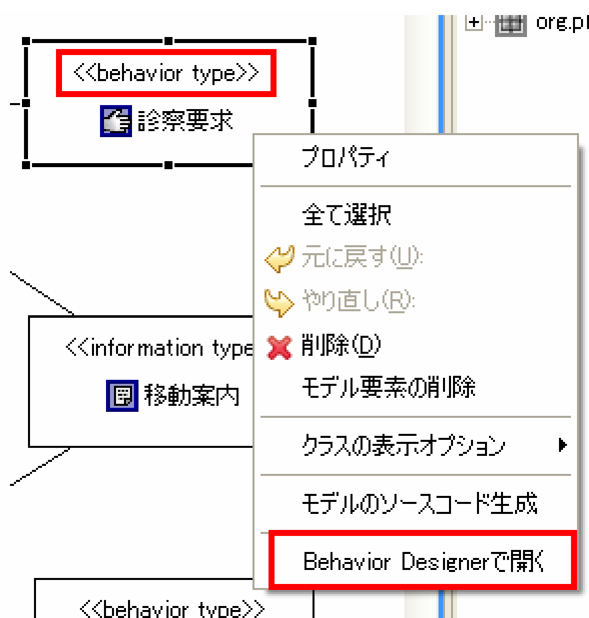

図 3.23　Behavior Designer で開く

右のパレットにある[初期状態]を選択した後キャンパスに配置する。同様に[状態]を選択、キャンパスに配置する。配置した[状態]にそれぞれタイトルをつける(Agent Design と同じ。ダブルクリックでタイトル変更可能)。そして、その[状態]を[遷移]で結ぶ。

3.6.1　遷移条件の設定①

次に遷移条件を設定する。遷移条件とは遷移がどのようなときに起こり、どのような条件を満たし、どのようなアクションを起こすのか設定することである[5]。実際の変更には[遷移のプロパティ]ダイアログで行う。先程、作成した遷移線をダブルクリックする。まず初めに[状況伝達行動]から[待機状態]の遷移条件を設定する。

図 3.24 で示すように[遷移のプロパティ]ダイアログを表示させ、イベントを[Time Event]に変更。次に[新しい内部アクション]のボタンを押す。

※ [Time Event]･･･一定時間後に自動的に遷移する。

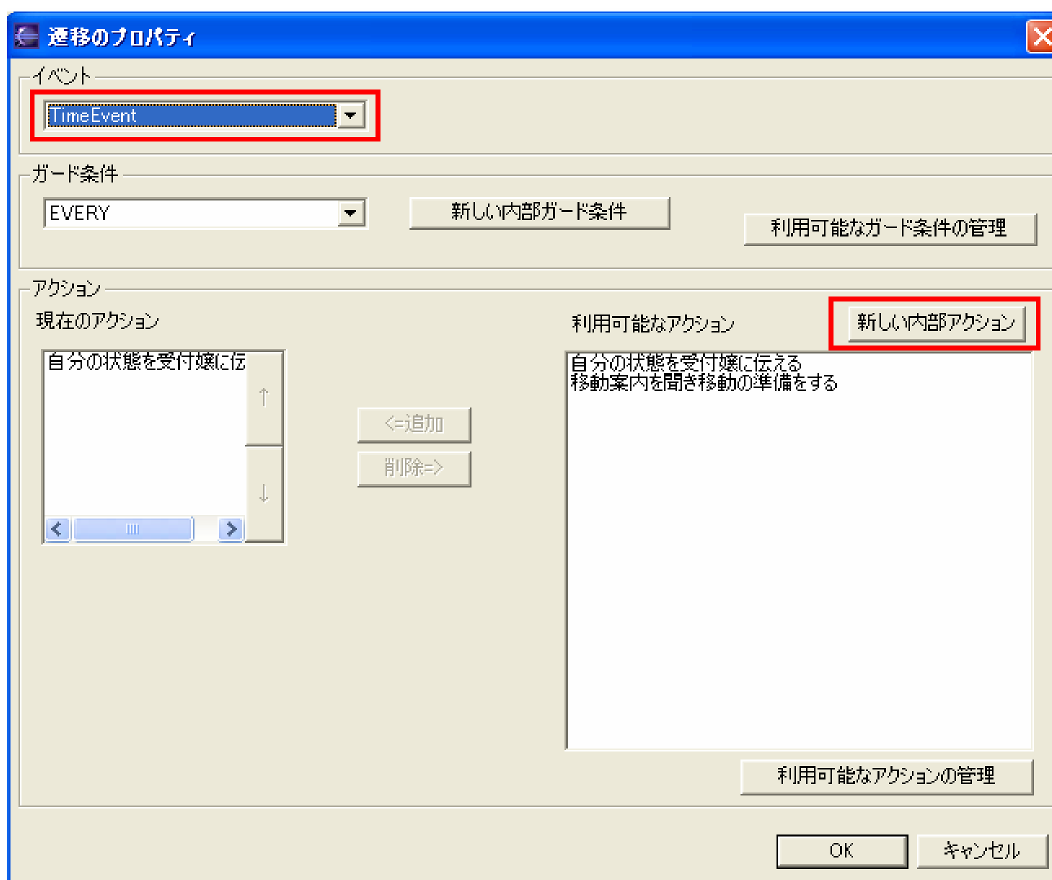

図 3.24　遷移のプロパティ

[内部アクションの作成]ダイアログを表示させ、名前を図 3.25 のように[自分の状態を受付嬢に伝える]として[OK]を押す。[現在のアクション]に登録されたのを確認後、[OK]を押す。

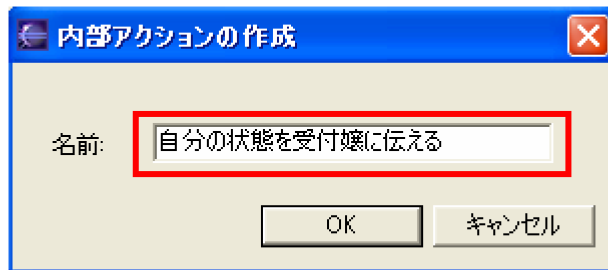

図 3.25　内部アクションの作成

3.6.2　遷移条件の設定②

次に[待機状態]から[移動状態]への遷移条件を設定する。先と同様に遷移線をダブルクリック。イベントは[Time Event]、内部アクションは図 3.26 で示すように[移動案内を聞き移動の準備をする]とする。<u>(後でTime Eventを変更することに留意)</u>

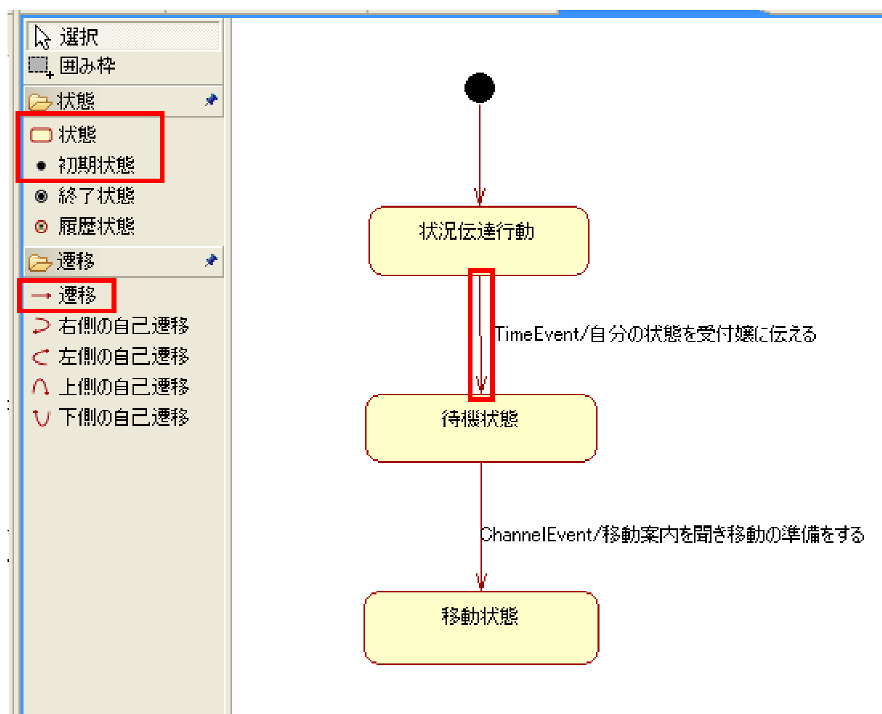

図 3.26　遷移線

3.7　患者の Action デザイン

3.6 で作成した Behavior の具体的な中身、Action を設定する。

まず始めに、既存の Behavior ファイルを開く(先程作成した[診察要求.behavior]を開く)。図 3.27 で示すようにキャンパスを右クリックして[Action Designer で開く]を選択。選択後、自動生成が開始される。パッケージ･エクスプローラ上に[Abstract 診察要求.java]と[診察要求.action]ファイルが生起する。

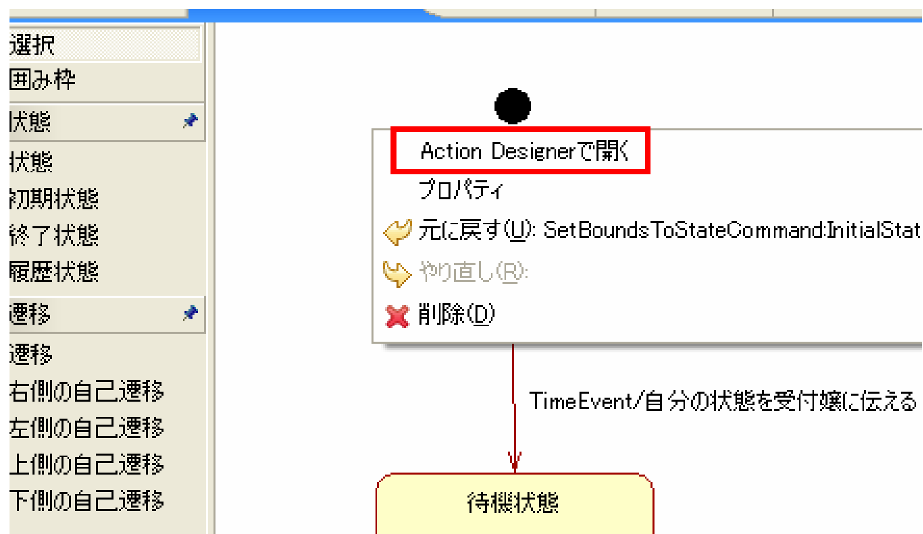

図 3.27　Action Designer で開く

図 3.28 で示すように[Action Designer]が起動したら、キャンパスを右クリックして[ソースコード生成]を選択。自動生成で[診察要求.java]が生起する。

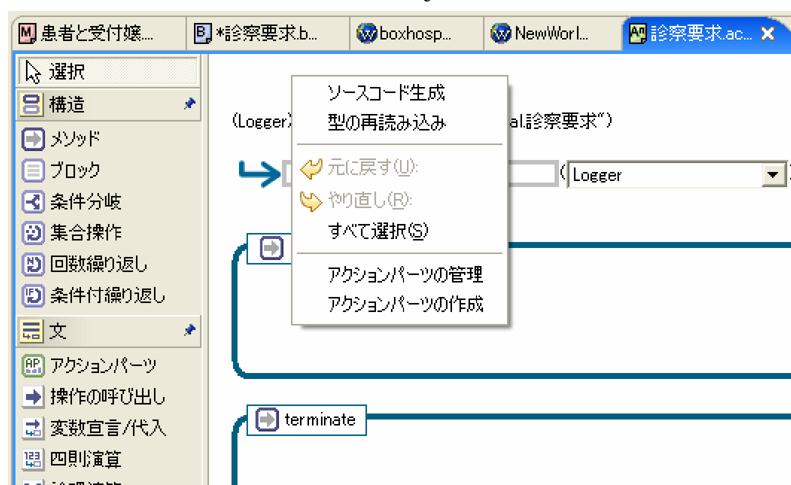

図 3.28　Action 診察要求のソースコード生成

3.8　シミュレーション(二回目)

[boxhospitalworld.world]を開く。最上段にある[モデルの読み込み]ボタンを押す。(Behavior と Action を変更した場合は必ず[モデルの読み込み]を実行させる)[Agent 患者]に作成した[診察要求.behavior]を付加する。[Agent グループ]から[患者集団]を選択して、[変更]を押して[Agent グループの設定]ダイアログを表示。図 3.29 で示すように[Behavior Type]の[診察要求]にチェックを入れて[OK]を押す。

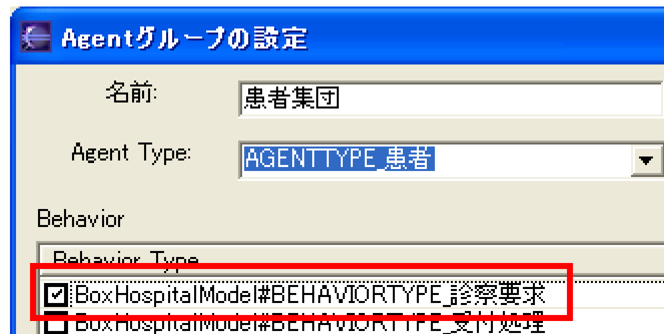

図 3.29　Action グループの設定　Behavior Type の付加

最下段にある[World の生起]を押す。PBS を起動させ、患者の状態が[状態報告行動]⇒[待機状態]⇒[移動状態]に変化することを確認する。

3.9　受付嬢の Behavior のデザイン

[患者と受付嬢.model]の Behavior Type[受付処理]を右クリックして[Behavior Designer で開く]を選択。

[初期状態]と[状態]をそれぞれ一つずつ配置する。[状態]のタイトルを[受付処理]とする。遷移線で結ぶ。受付嬢は常に一定の受付処理を繰り返すので図 3.30 で示すように[左側の自己遷移]でループするように遷移線を描く。※自己遷移は上下左右、同じ役割を持つ。4 つの自己遷移をそれぞれに異なる条件を持たせることでシミュレーションの幅が広がる。

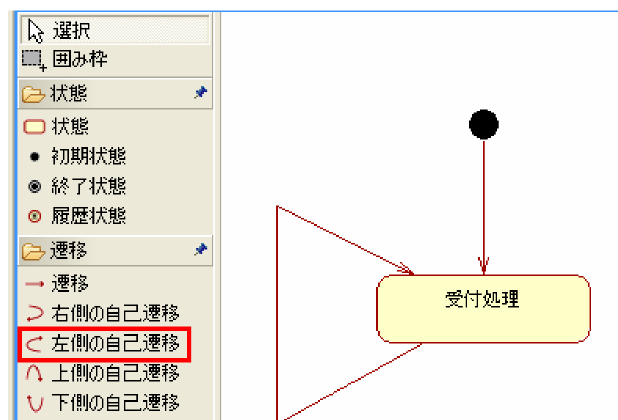

図 3.30　Behavior 受付処理

[左側の自己遷移]をダブルクリックして遷移条件を設定する。図 3.30 を参照にイベントは[Channel Event]。続いて[新しい内部アクション]ダイアログにて[要求を受け移動案内を出す]のアクションを作成。

※Channel Event･･･他の Agent からの Action によって遷移するトリガー[6]

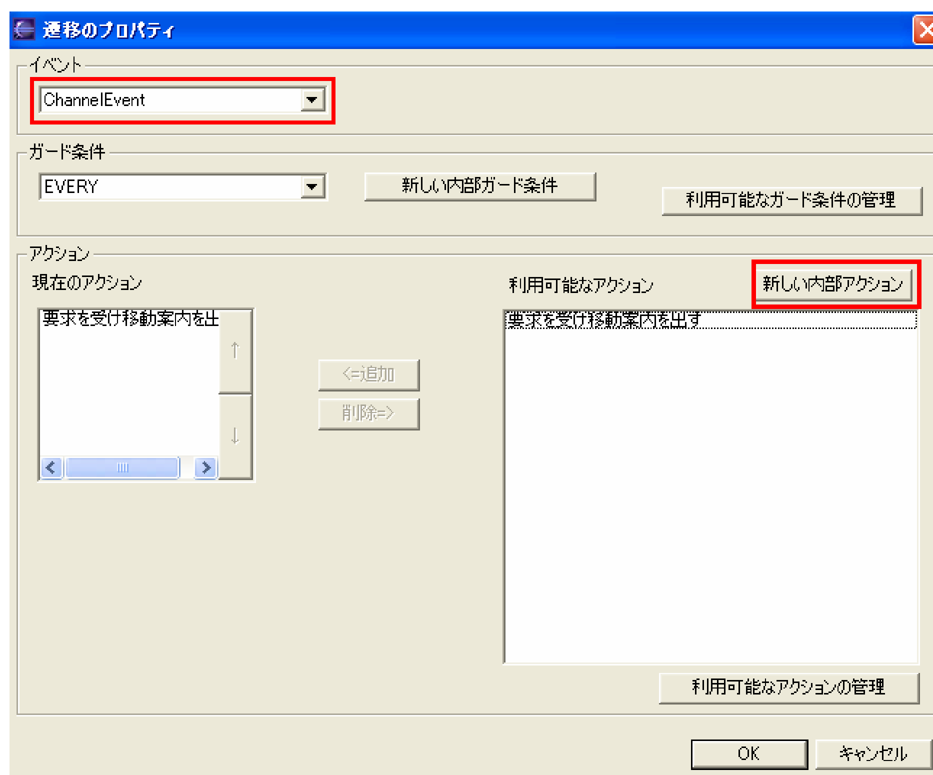

図 3.31　偏移のプロパティ(Channel Event)

キャンパスで右クリックして[Action Designer で開く]を選択。[Abstract 受付処理.java] と[受付処理.action]の生起を確認。

＜Caution＞

ここで[ソースコード生成]をするのだが、Channel Event のトリガーとなるものが存在しないのでエラーが発生する。このままトリガーとなる Information を作成する。

3.10　Information のデザイン　―受付処理のトリガー作成―

ランダムで選ばれた患者の要求を受付嬢に伝える。そのためには要求事項をいくつか(今回は 3 つ)用意して、リストに載せ、それぞれの患者がリスト内からランダムに要求を選択して[診察券]の Information として受付嬢に渡す。この一連の行動が受付嬢 Action のトリガーとなる。

3.10.1　ランダム要素の作成

[診察要求.action]を開く。

既存の乱数ジェネレータを用いて整数の乱数を生起させる。

図 3.32 を参照に右のボックスから[アクションパーツ]を選択する。[アクションパーツの選択]ダイアログの[計算]から[整数の乱数を生成する]を選択し[OK]を押す。

図 3.32　乱数生成

今回のケースは患者の要求を頭痛、腹痛、膝のけがの 3 つにする。よって乱数は 0 から 2 までとなる。図 3.33 で示すように乱数の名前は[診察要求乱数]とする。

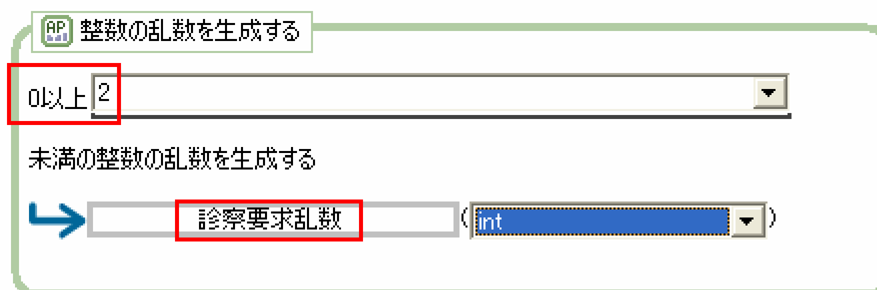

図 3.33　乱数設定

3.10.2　診察要求格納リストの作成　(空の ListInformation をつくる)

[アクションパーツ]の[Information の生成/操作]の中にある[空の List Information をつくる]を選択して配置する。図 3.34 で示すようにリストの名前は[患者要求格納リスト]とする。

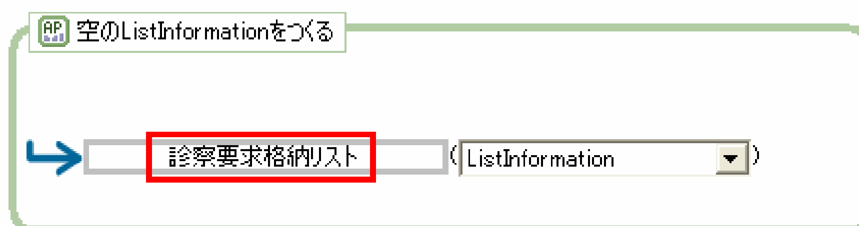

図 3.34　List Information 生成

診察要求事項を作る(文字列の生成)。

ボックスから[文字列]を配置。図 3.35･3.36 で示すように今回は「頭痛」「腹痛」「膝を痛めた」の 3 つ。それぞれの名前を[診察要求 1]～[診察要求 3]とする。

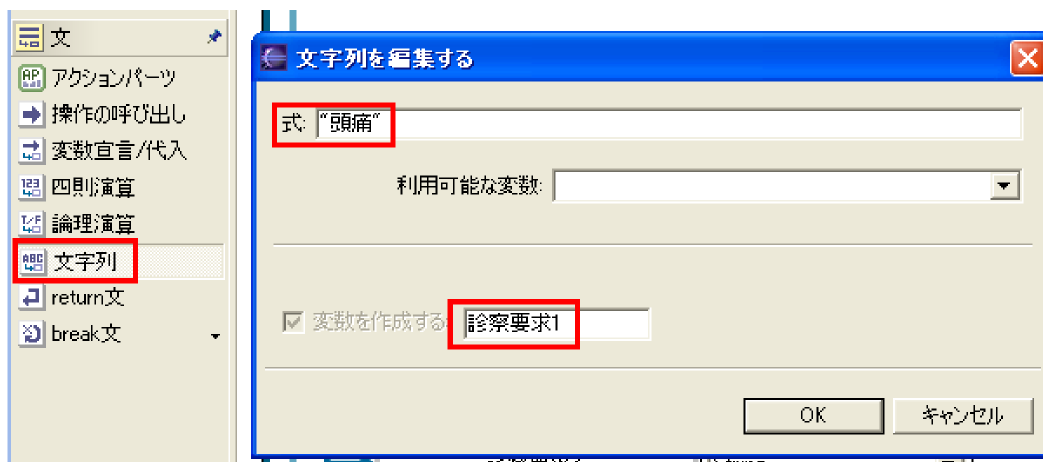

図 3.35　文字列生成

図 3.36　3 つの文字列

生成した文字列を受け渡し可能な Information に変換する。
[アクションパーツ]の[Information の生成/操作]から[String Information を作る]を選択。生成した文字列は 3 つあるのでこのパーツも 3 つ必要。図 3.37 で示すように対象をそれぞれ[診察要求 1]となるようにコンボボックスから選択する。名前は[診察要求 1-1]～[診察要求 3-3]とする。 図 4.33
String Information 生成

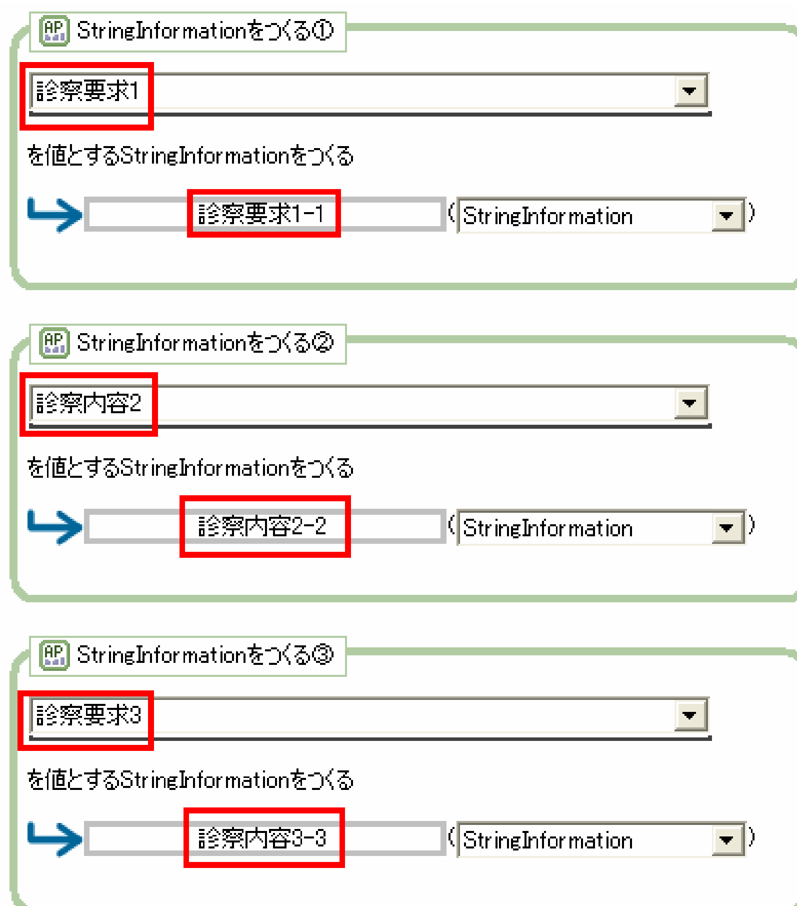

図 3.37 String Information 生成

3.10.3　診察要求格納リストにそれぞれの患者の供給を登録

図 3.38 で示すように[アクションパーツ]の[Information の生成/操作]から[List Information の N 番目に Information を追加する]を選択。リストに文字列から生成した String Information を追加していく。String Information は 3 つあるのでリストへの追加も 3 回行う。この作業は以下のような表 2 を作ったことになる。

図 3.38　List へ追加

表 2　要求項目

List　No	要求項目
0	膝を痛めた
1	頭痛
3	腹痛

3.10.4　リスト内の Information をランダムに選ぶ

[アクションパーツ]の[Information の生成/操作]から[List Information の N 番目の Information を取得する]を選択。リストから生成した乱数[診察要求乱数]で選ばれた Information を取り出す。図 3.39 で示すように名前を[選ばれた診察要求]をする。

図 3.39　リストからの参照

3.10.5　取得した Information を送る

[アクションパーツ]の[自分と他の Agent の間のやり取り]から[指定した Relation Type の Relation を持つ Agent 全員に Information を送る]を選択。病院関係を通じて相手の Agent に受付処理(受付嬢の Behavior)に診察券(受け渡し可能な Information)としてランダムに選ばれた診察要求を渡す。図 3.40 に具体例を示す。

図 3.40　Information の送信

3.10.6　ソースコードの作成

キャンパスで右クリックを押して、メニューを表示させる。[ソースコードの作成]を選択する。

3.10.7　Agent 患者に Information 診察券を持たせる

[boxhospital.world]を開き、最上段の[モデルを読み込む]を押す。 [Agent グループ]から[患者集団]を選択し[変更]を押す。図 3.41 で示すように[Agent グループの設定]ダイアログを開き、[Information Type]の診察券にチェックを入れる

図 3.41　Agent グループの設定　Information の付加

3.10.8　Agent の再設定

患者数を複数にする(今回は 3 人)。最下段の[World の生成]を押す。

3.10.9　シミュレーション(3 回目)

PBS を起動させ、動作確認を行う。それぞれランダムに選ばれた診察要求を Information として受付嬢に伝える様子が図 3.42 で示したように確認できる。

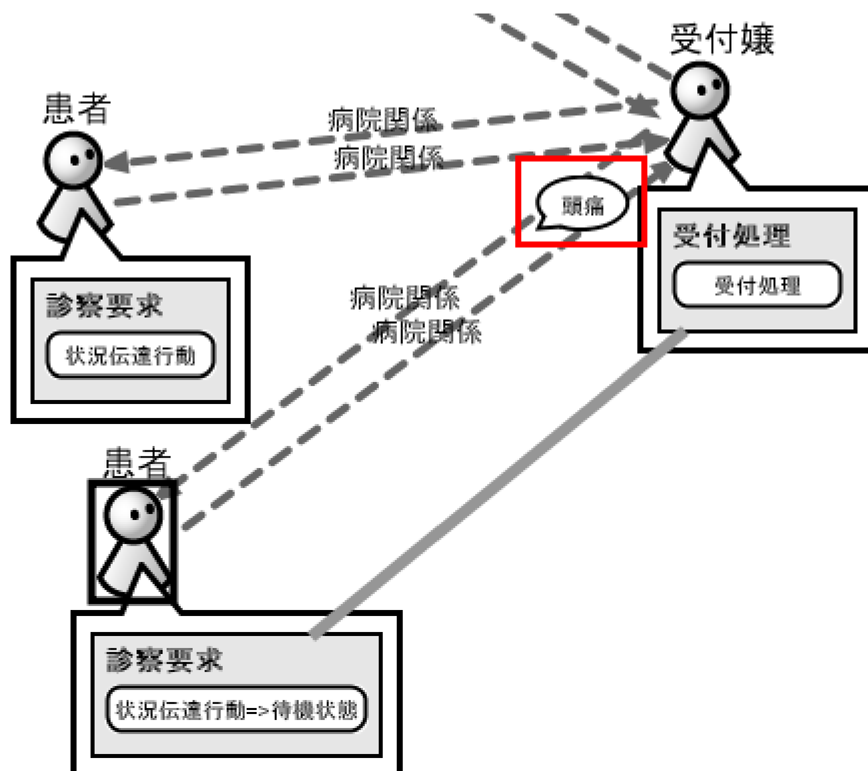

図 3.42　シミュレーション(3 回目)

3.11　受付嬢の Action デザイン

患者からの診察券(患者の症状を付加した Information)を受け取った受付嬢がそれぞれの要求に対して適切な診療科を伝える。適切な診療科を選択するために受付嬢は受け取った Information を文字列に変換し、自分の持っている文字列を論理演算にて比較する。その結果、それぞれの要求に適切な診療科を案内する。

図 3.43 を参照に[受付処理.behavior]を開き、キャンパスを右クリックして[Action Designer で開く]を選択する。[要求を受け移動案内を出す]に実際の行動を記述していく。

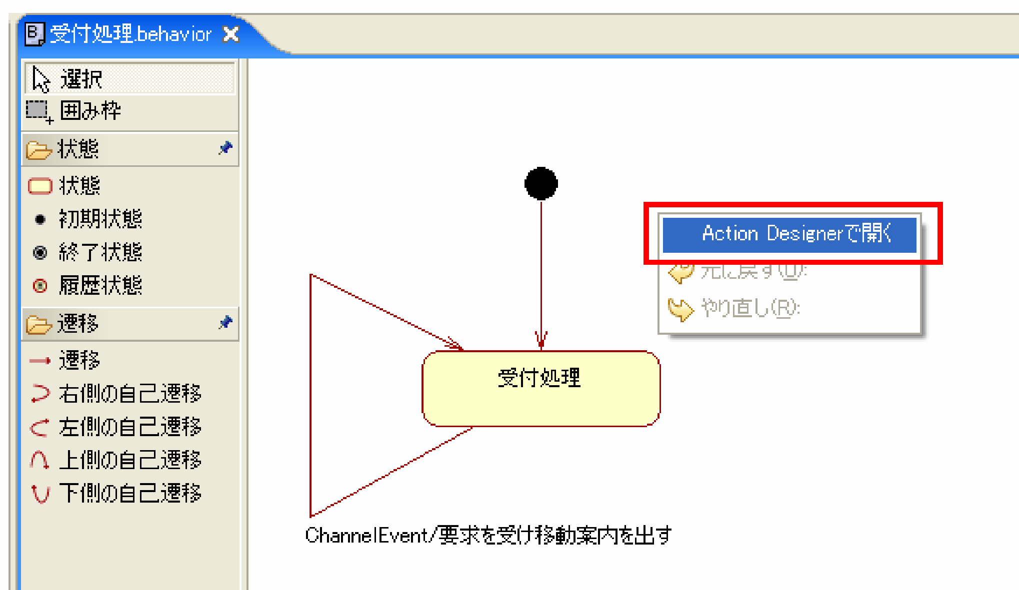

図 3.43　受付処理の Action Designer

3.11.1　診察券を受け取る

受け取った Information を記憶する。[アクションパーツ]の[自分と他の Agent の間のやりとり]から[最後に受け取った Information を取得する]を選択する。図 3.44 で示すように、名前を[受け取った診察券情報]とする。

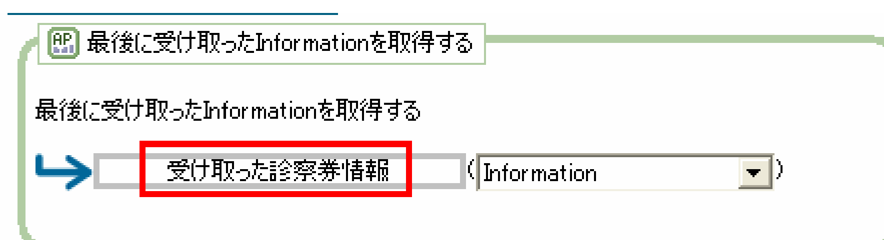

図 3.44　Information の取得

3.11.2　[受け取った診察券情報]を参照・比較可能な文字列に変換する

[アクションパーツ]の[Information の生成/操作]から[Information の内容を文字列として取得する]を選択する。図 3.45 で示すようにコンボボックスからさきほどの[受け取った診察券情報]を選択し、名前を[診察券情報]とする。

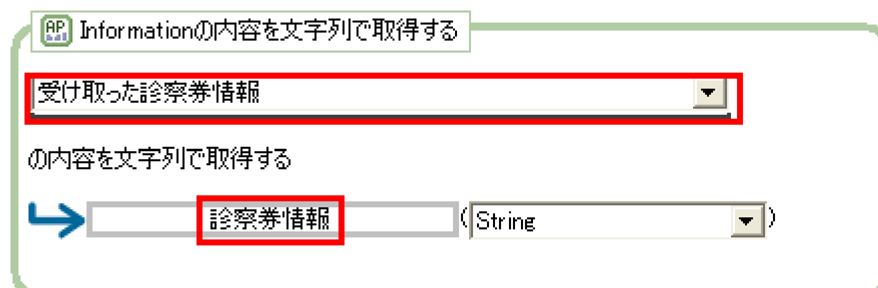

図 3.45　Information を文字列に変換

3.11.3　見比べる項目を作成する

比較･参照するために受付嬢側にも患者同様の症状例を作成する。

患者の症状と同じ文字列を作成する。ただし名前は図 3.46 で示すように[症状_脳]、[症状_腹]、[症状_脚]とする。

図 3.46　条件に使う文字列を生成

3.11.4　移動案内のセリフ(文字列)を作成する

受付嬢が患者に伝える Information 移動案内の雛形となる文字列を作成する。　セリフは[“脳神経外科に行ってください“] [“整形外科に行ってください“] [“内科に行ってください“]として、名前はそれぞれ[移動案内 1]、[移動案内 2]、[移動案内 3]とする。

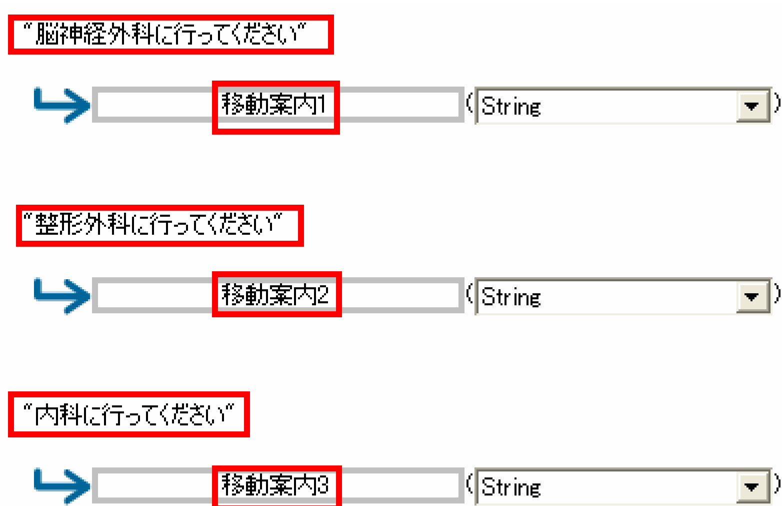

図 3.47　移動案内になる文字列生成

3.11.5　診察券の情報を比較検討する

[論理演算]を用いて[診察情報]とそれぞれの症状の項目([症状_脳]、[症状_腹]、[症状_脚])の分岐条件を作成する。図 3.48 で示すように[論理演算]を選択して[論理式を編集する]ダイアログを表示し、ディフォルトで書かれている[false]を消去する。

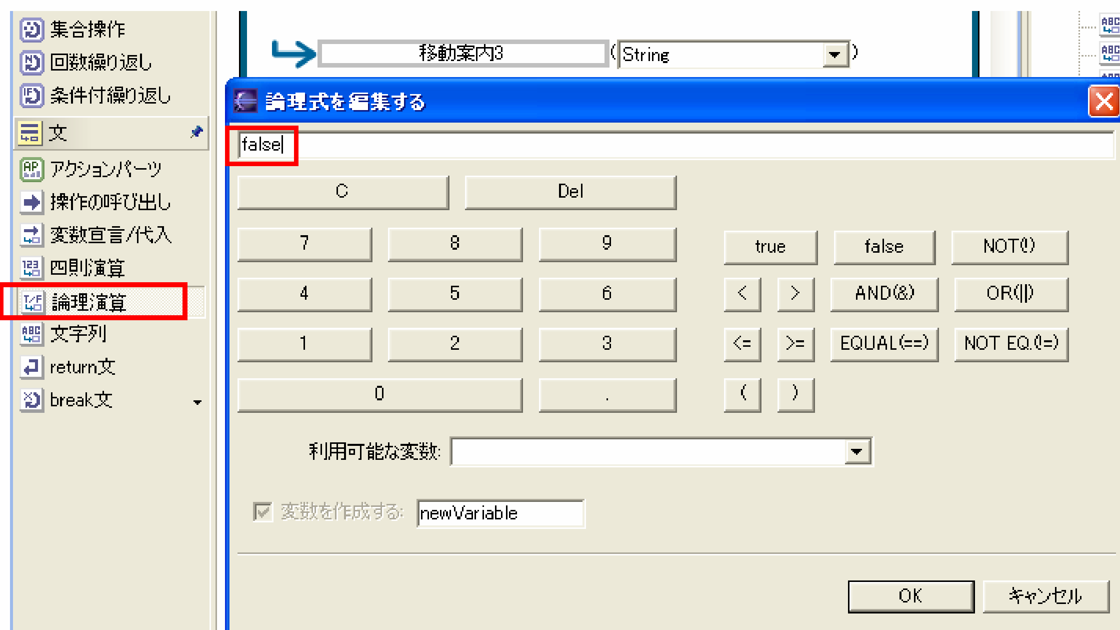

図 3.48　論理式記述①

[利用可能な変数]のコンボボックスから[症状_脳]を選択する。次に図 3.49 を参照に[EQUAL(==)]を押し、[利用可能な変数]のコンボボックスから[診察券情報]を選択する。条件式は[症状_脳==診察券情報]となる。つまり、受付嬢の持つ脳に関する情報と患者が伝えてきた症状(診察要求の情報)が等しいという意味である。

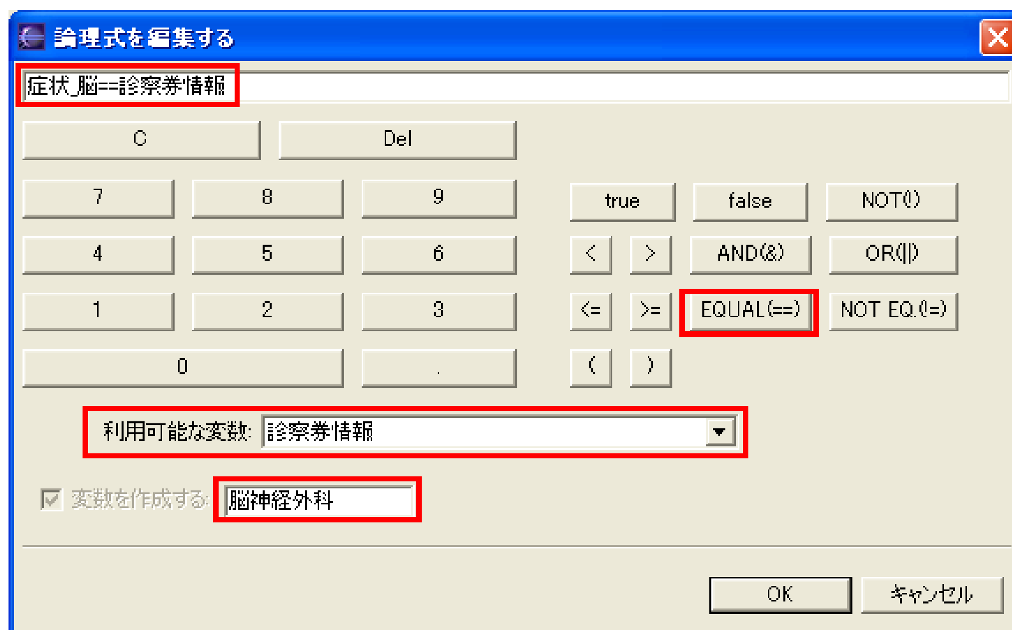

図 3.49　論理式記述②

上と同様に残り二つの症状も条件式を作成する。
図 3.50 で示すように条件名は[脳神経外科]、[整形外科]、[内科]とする。

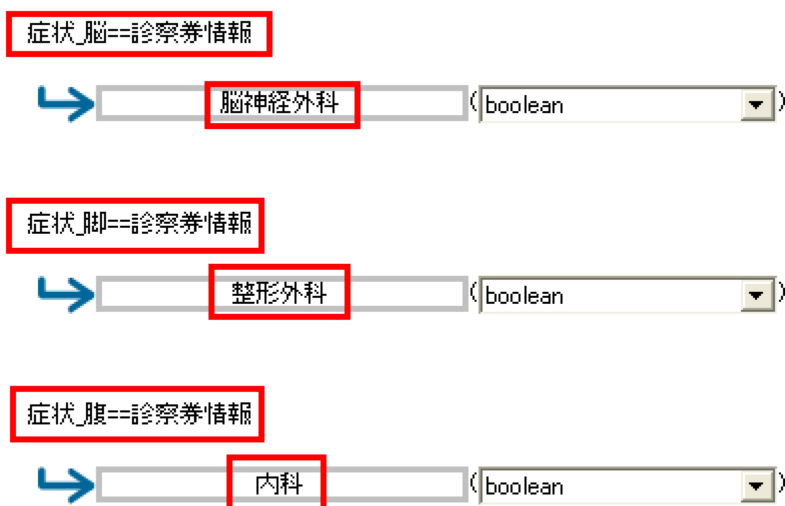

図 3.50　条件となる論理式の完成

3.11.6　それぞれの症状に合った診療科を移動案内として伝える

[分岐条件]を配置する。図 3.51 を参照詩歌の操作を行う。[条件]のコンボボックスから先程作成した分岐条件[脳神経外科]を選択する。[アクションパーツ]の[自分と他の Agent の間のやりとり]の中から[Goods や Information を送ってきた相手にメッセージを送る]を選択する。伝える Information を[移動案内 1]とし、名前を[移動案内 1-1]とする。

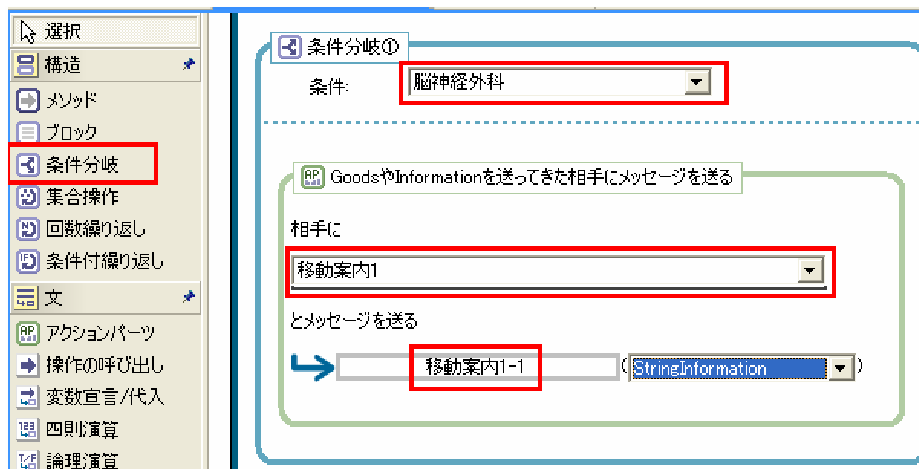

図 3.51　条件分岐作成①

残りの二つも図 3.52 を参照に以下のように作成する。

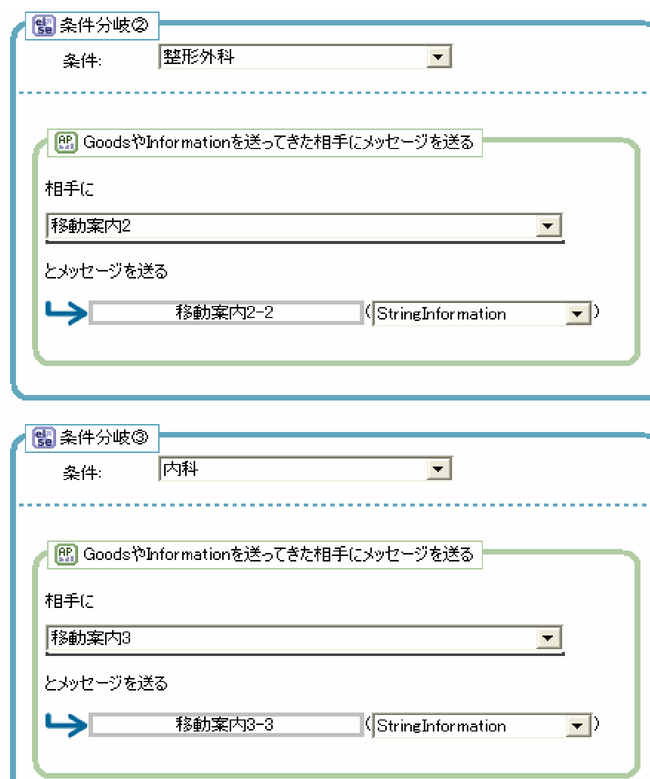

図 3.52　条件分岐作成②

3.11.7　ソースコード作成

ソースコード生成する。

3.11.8　受付嬢に Information 移動案内を持たせる

[boxhospital.world]を開き、最上段の[モデルを読み込む]を押す。図 3.53 で示すように[Agent グループ]から[受付嬢集団]を選択し[変更]を押す。[Agent グループの設定]ダイアログを開き、[Information Type]の[移動案内]にチェックを入れる。

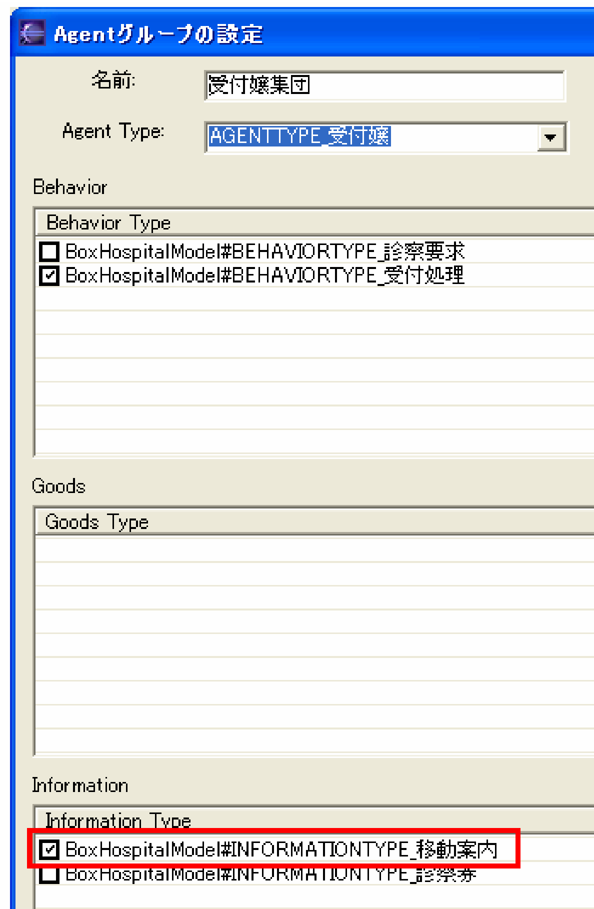

図 3.53　Agent グループの設定　Information Type の付加

3.12　シミュレーション(3 回目)

PBS を起動させ、動作確認を行う。

患者から頭痛という診察要求が来れば、脳神経外科に行ってくださいと返すことが図 5.54 で示すように確認できる。

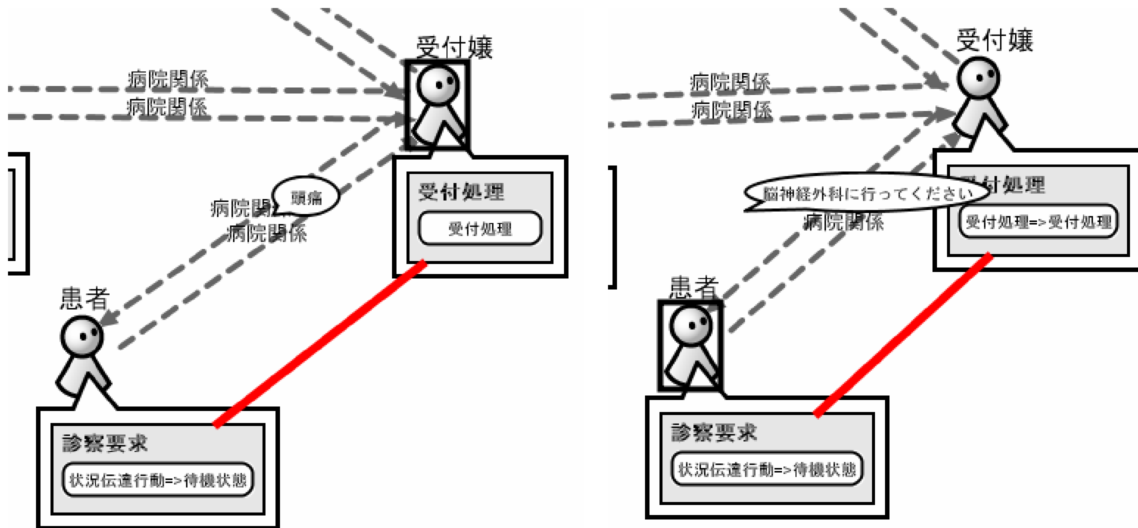

図 3.54　シミュレーション(3 回目)

4. 評価

Action パーツを選択する部分が習得するのに最も時間を要した。なぜなら、Action は 17 種類のカテゴリーがあり、一つのカテゴリー内に少なくて 3、多い場合は 30 ほどのパーツが存在する。各 Action パーツはそのタイトルから機能を直感的に理解できないものでもあり、また同じ機能を担う Action が複数混在するため、試行錯誤的に Action パーツ選択をする必要がある。このため、Action 選択が最も PBS 特有の経験を要する操作であると経験的に考えられる。

例えば、論理演算を用いた条件分岐を記述する場合は、P.33「4.10.5　診察券の情報を比較検討する」で行ったように、条件となる文字列を作成して、その文字列を用いて論理演算の式を作成する。その後に、条件分岐した場合の Action を作成するといった複数の機能を組み合わせなければならない。直接、プログラミングを行った場合には、If 文や While 文などを用いて数行の記述ですむ。その代わり、特定のプログラミング言語特有の文法や書式を新しく習得する必要がなく、直感的に分岐構造や繰り返し構造などの通常のプログラミングに必須であるこれらの概念を習得することが可能である。

5. まとめ

実施に PBS によるシステム開発の経験を通したオブジェクト指向概念の学習効率に関して、初学者の立場からの評価を行った。

通常のシステム開発過程は分析⇒設計⇒実装だが、PBS の場合は設計と実装を同時進行的に行うので初学者にとっては特定のプログラミング言語を習得せずにモデル作成できる。しかし、作業効率や設計の緻密性を考えると条件分岐や繰り返しといったプログラミングの基本構造を十二分に理解する必要があり、そのためには適切な指導者が必要である。

また、PBS を用いてオブジェクト指向の重要な要素である「クラスとインスタンス」「ポリモーフィズム」「継承」について理解することは難しい。オブジェクト指向に関して初学者でも扱えるよう PBS は設計されているため、こういった概念がどのように扱われ実装されているのか読み取りにくい。

6. 今後の課題

6.1 評価項目について

今回の最大の反省点は評価項目を明確に決めていなかったことである。なにかを評価する場合、文章にしても数値にしても評価項目が具体的に挙がっていなければならない。

6.2 作業時間について

今回のケース「Box Hospital」の作成に当てた時間は概ね以下の通りである。

表 3 作業時間

作業項目	作業時間[h]
ミーティング(アドバイス、模擬練習)	2
オブジェクト抽出・設計	1
Model のデザイン	1
Behavior のデザイン	2
Action のデザイン	7
シミュレーション(1 回目)･修正	2
拡張(リスト、分岐条件の追加)	4
シミュレーション(2 回目)･修正	2
評価	2
合計	23

初学者である自分が始めて Plat Box に触れてから約 23 時間で Box Hospital を完成させた。この結果とプログラミングをおこなったことがない人間が通常のプログラミング(手続型プログラミング)を学習して Box Hospital を作成した場合を比較し、学習効率について考えなければならない。

6.3 複数人による実施について

今回の評価は自分一人で実施したものだ。今後は複数人で、別のシステムと比較した評価も行いたい。つまり、今回の主観的評価に加えて、客観的評価を行い、より説得力を増すことが今後の課題になる。

7. 謝辞

この研究を行うにあたり、直接指導してくださった副テーマ指導教官である橋本敬先生、橋本先生と共にUMLやオブジェクト指向に関する指導をしてくださった畠山剛臣さん、副テーマの題材を提供してもらい橋本先生や畠山さんと知り合う機会を与えていただいた上原安浩さん、ミーティング場所を提供していただいた橋本研究室の皆様、私自身が所属する研究室の皆様には並々ならぬ御尽力を賜り、感謝の念に耐えません。

8. 参考文献・公式マニュアル

[1] 樫山友一，日野泰臣，2002，「わかりやすいUML」オーム社.

[2] 小森裕介，2006，「なぜ、あなたはJavaでオブジェクト指向開発ができないのか―Javaの壁を克服する実践トレーニング―」技術評論社.

[3] 平澤章，2004，「オブジェクト指向でなぜつくるのか」日経BP社.

[4] Plat Box Simulator & Component Builder インストールガイド
http://platbox.sfc.keio.ac.jp/jp/document/200508PlatBoxInstallGuide.pdf

[5] Plat Box Simulator マニュアル
http://platbox.sfc.keio.ac.jp/jp/document/200508platbox-manual.pdf

[6] モデル作成リファレンスガイド
http://platbox.sfc.keio.ac.jp/jp/document/200508modeling-reference.pdf

[7] モデル作成チュートリアル
http://platbox.sfc.keio.ac.jp/jp/document/200508modeling-tutorial.pdf

9. 資料

作成した Box Hospital のソースコードを資料として掲載する。

9.1 Abstract 受付受理.java

```
/*
 * 作成日： 2006/05/28
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
package boxhospital;

import org.platbox.simulator.model.fmfw.behavior.AbstractBehavior;

import org.platbox.simulator.model.fmfw.behavior.CompositeState;
import org.platbox.simulator.model.fmfw.behavior.State;
import org.platbox.simulator.model.fmfw.behavior.StateMachineFactory;
import org.platbox.simulator.model.fmfw.behavior.Transition;

import org.platbox.simulator.model.fmfw.TimeEvent;
import org.platbox.simulator.model.fmfw.behavior.Action;

import org.platbox.simulator.model.fmfw.ChannelEvent;

/**
 * Abstract 受付処理
 */
public abstract class Abstract 受付処理 extends AbstractBehavior {

  /**
   * This method automatically generated from PlatBox Behavior Editor.
   * Don't modify this method.
   */
  protected void initializeStateMachine() {
        //factory
        StateMachineFactory factory = this.getStateMachine();

        //states
        State initialState = factory.createInitialState();
        CompositeState 受付処理 = factory.createCompositeState("受付処理");
```

```
		//actions
		Action 要求を受け移動案内を出す = new Action() {
			public void doAction() {
				要求を受け移動案内を出す();
			}

			public String toString() {
				return "要求を受け移動案内を出す";
			}
		};

		//guard-conditions

		//transitions
		Transition transition受付処理To受付処理 = factory.createTransition();
		Transition transitionInitialStateTo受付処理 = factory.createTransition();

		//states setting

		//structure of states
		this.setInitialState(initialState);
		this.addState(受付処理);

		//transitions setting
		transition受付処理To受付処理.setEvent(ChannelEvent.class);
		transition受付処理To受付処理.addAction(要求を受け移動案内を出す);

		//connection of transitions
		transition受付処理To受付処理.setSourceState(受付処理);
		transition受付処理To受付処理.setTargetState(受付処理);
		transitionInitialStateTo受付処理.setSourceState(initialState);
		transitionInitialStateTo受付処理.setTargetState(受付処理);

	}

	/**
	 *要求を受け移動案内を出す
	 */
	protected abstract void 要求を受け移動案内を出す();
}
```

9.2 Abstract 診察要求.java

```
/*
 * 作成日： 2006/05/28
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
package boxhospital;

import org.platbox.simulator.model.fmfw.behavior.AbstractBehavior;

import org.platbox.simulator.model.fmfw.TimeEvent;
import org.platbox.simulator.model.fmfw.behavior.Action;
import org.platbox.simulator.model.fmfw.behavior.CompositeState;
import org.platbox.simulator.model.fmfw.behavior.State;
import org.platbox.simulator.model.fmfw.behavior.StateMachineFactory;
import org.platbox.simulator.model.fmfw.behavior.Transition;

import org.platbox.simulator.model.fmfw.ChannelEvent;

/**
 * Abstract 診察要求
 */
public abstract class Abstract 診察要求 extends AbstractBehavior {

    /**
     * This method automatically generated from PlatBox Behavior Editor.
     * Don't modify this method.
     */
    protected void initializeStateMachine() {
            //factory
            StateMachineFactory factory = this.getStateMachine();

            //states
            State initialState = factory.createInitialState();
            CompositeState 状況伝達行動 = factory.createCompositeState("状況伝達行動");
            CompositeState 待機状態 = factory.createCompositeState("待機状態");
            CompositeState 移動状態 = factory.createCompositeState("移動状態");

            //actions
```

```
Action 自分の状態を受付嬢に伝える = new Action() {
	public void doAction() {
		自分の状態を受付嬢に伝える();
	}

	public String toString() {
		return "自分の状態を受付嬢に伝える";
	}
};
Action 移動案内を聞き移動の準備をする = new Action() {
	public void doAction() {
		移動案内を聞き移動の準備をする();
	}

	public String toString() {
		return "移動案内を聞き移動の準備をする";
	}
};

//guard-conditions

//transitions
Transition transition 待機状態 To 移動状態 = factory.createTransition();
Transition transition 状況伝達行動 To 待機状態 = factory.createTransition();
Transition transitionInitialStateTo 状況伝達行動 = factory.createTransition();

//states setting

//structure of states
this.setInitialState(initialState);
this.addState(状況伝達行動);
this.addState(待機状態);
this.addState(移動状態);

//transitions setting
transition 待機状態 To 移動状態.setEvent(ChannelEvent.class);
transition 待機状態 To 移動状態.addAction(移動案内を聞き移動の準備をする);
transition 状況伝達行動 To 待機状態.setEvent(TimeEvent.class);
transition 状況伝達行動 To 待機状態.addAction(自分の状態を受付嬢に伝える);
```

```
        //connection of transitions
        transition 待機状態 To 移動状態.setSourceState(待機状態);
        transition 待機状態 To 移動状態.setTargetState(移動状態);
        transition 状況伝達行動 To 待機状態.setSourceState(状況伝達行動);
        transition 状況伝達行動 To 待機状態.setTargetState(待機状態);
        transitionInitialStateTo 状況伝達行動.setSourceState(initialState);
        transitionInitialStateTo 状況伝達行動.setTargetState(状況伝達行動);

    }

    /**
     *自分の状態を受付嬢に伝える
     */
    protected abstract void 自分の状態を受付嬢に伝える();

    /**
     *移動案内を聞き移動の準備をする
     */
    protected abstract void 移動案内を聞き移動の準備をする();
}
```

9.3 BoxHospitalModel.java

```
/*
 * BoxHospitalModel.java
 */
package boxhospital;

import org.platbox.simulator.container.PlatBoxSimulationContainer;
import org.platbox.simulator.container.PlatBoxPlugin;
import org.platbox.simulator.model.ModelContainer;
import org.platbox.simulator.model.fmfw.AgentType;
import org.platbox.simulator.model.fmfw.GoodsType;
import org.platbox.simulator.model.fmfw.BehaviorType;
import org.platbox.simulator.model.fmfw.InformationType;
import org.platbox.simulator.model.fmfw.RelationType;

/**
 * Model for PlatBox Foundation Model Framework
 *
 * @generated modifiable
 * @version $$Id$$
 */
public class BoxHospitalModel implements PlatBoxPlugin {

  public static AgentType AGENTTYPE_受付嬢;
  public static AgentType AGENTTYPE_患者;
  public static BehaviorType BEHAVIORTYPE_診察要求;
  public static BehaviorType BEHAVIORTYPE_受付処理;
  public static RelationType RELATIONTYPE_病院関係;
  public static InformationType INFORMATIONTYPE_移動案内;
  public static InformationType INFORMATIONTYPE_診察券;

  /**
  * This method will be modified automatically by Model Designer
  *
  * @generated
  */
  public static void initializePlugin(PlatBoxSimulationContainer container) {
          installTypes(container);
          buildStructure(container);
          setPriority(container);
```

```
    }

    /**
    * This method will be modified automatically by Model Designer
    *
    * @generated
    */
    private static void installTypes(PlatBoxSimulationContainer container) {
            //Get the model container
            ModelContainer modelContainer = container.getModelContainer();

            AGENTTYPE_受付嬢=modelContainer.installAgentType("boxhospital.受付嬢");
            AGENTTYPE_患者=modelContainer.installAgentType("boxhospital.患者");
      BEHAVIORTYPE_診察要求=modelContainer.installBehaviorType("boxhospital.診察要求");
      BEHAVIORTYPE_受付処理=modelContainer.installBehaviorType("boxhospital.受付処理");
      RELATIONTYPE_病院関係=modelContainer.installRelationType("boxhospital.病院関係");
INFORMATIONTYPE_移動案内=modelContainer.installInformationType("boxhospital.移動案内");
INFORMATIONTYPE_診察券=modelContainer.installInformationType("boxhospital.診察券");
    }

    /**
    * This method will be modified automatically by Model Designer
    *
    * @generated
    */
    private static void buildStructure(PlatBoxSimulationContainer container) {
            ModelContainer modelContainer = container.getModelContainer();

    }
    /**
    * Set priority
    *
    * @generated
    */
    private static void setPriority(PlatBoxSimulationContainer container) {
            ModelContainer modelContainer = container.getModelContainer();



    }
}
```

9.4　boxhospitalworldWorld.java

```
/*
 * 作成日： 2006/05/28
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
package boxhospital;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.log4j.Logger;
import org.platbox.components.cell.Cell;
import org.platbox.components.stepclock.StepClock;
import org.platbox.simulator.container.PlatBoxSimulator;
import org.platbox.simulator.model.fmfw.Agent;
import org.platbox.simulator.model.fmfw.RandomNumberGenerator;
import org.platbox.simulator.model.fmfw.World;

/**
 * boxhospitalworldWorld
 */
public class boxhospitalworldWorld extends World {

    /************************************
     * Class variable and main method for running on the PlatBox Simulator
     ************************************/
    private static final long serialVersionUID = 1L;

    private static final Logger logger = Logger
                .getLogger(boxhospitalworldWorld.class.getName());

    public static void main(String[] args) {
            PlatBoxSimulator.main(new String[] { "-model",
                    boxhospitalworldWorld.class.getName() });
    }

    /************************************
```

```
 * Parameters
 *************************************/

//List of agents created from AgentGroup
private List 患者集団 s = new ArrayList();

private List 受付嬢集団 s = new ArrayList();

/*************************************
 * Initialize
 *************************************/

/**
 * Initialize World.
 *
 * @see org.platbox.simulator.model.fmfw.World#initializeWorld()
 */
public void initializeWorld() {
		logger.info("Initialize World.");
		super.initializeWorld();

		//set the clock
		this.setClock(new StepClock());
}

/**
 *   Initialize Agents.
 *
 * @see org.platbox.simulator.model.fmfw.World#initializeAgents()
 */
public void initializeAgents() {
		logger.info(" Initialize Agents.");
		super.initializeAgents();

		this.createAgents();
		this.addRelations();
		this.initializeByHands();
}

/**
```

```
 * this method is not overrided for automaticaly.
 */
private void initializeByHands() {
}

/**
 * Create Agents.
 */
private void createAgents() {
	logger.info("Create Agents.");

	this.create 患者集団 Agents();
	this.create 受付嬢集団 Agent();
}

/**
 * Add Relations
 */
private void addRelations() {
	logger.info("Add Relations");

	this.add 患者と受付嬢の病院関係 Relations();
}

/************************************
 * Subroutines to create agents
 ************************************/

/**
 * Create  患者集団  Agents
 */
private void create 患者集団 Agents() {
	logger.info("Create  患者集団  Agents");

	for (int i = 0; i < 3; i++) {
		this.create 患者集団 Agent();
	}
}

/**
```

```
  * Create 患者集団 Agent
  */
private void create 患者集団 Agent() {
        logger.info("Create 患者集団 Agent");

        //create agent
        Agent agent 患者集団 = super.createAgent(BoxHospitalModel.AGENTTYPE_患者);

        //put information
        agent 患者集団.putInformation(BoxHospitalModel.INFORMATIONTYPE_診察券, this
                        .create 診察券 ForAgent 患者集団(agent 患者集団));

        //add behavior(s)
        agent 患者集団.addBehavior(BoxHospitalModel.BEHAVIORTYPE_診察要求);

        患者集団 s.add(agent 患者集団);
}

/**
  * Create 受付嬢集団 Agent
  */
private void create 受付嬢集団 Agent() {
        logger.info("Create 受付嬢集団 Agent");

        //create agent
        Agent agent 受付嬢集団 = super.createAgent(BoxHospitalModel.AGENTTYPE_受付
嬢);

        //put information
        agent 受付嬢集団.putInformation(BoxHospitalModel.INFORMATIONTYPE_移動案内,
this
                        .create 移動案内 ForAgent 受付嬢集団(agent 受付嬢集団));

        //add behavior(s)
        agent 受付嬢集団.addBehavior(BoxHospitalModel.BEHAVIORTYPE_受付処理);

        受付嬢集団 s.add(agent 受付嬢集団);
}

/************************************
```

```
 * Subroutines to add relations
 ************************************/

/**
 * add 患者と受付嬢の病院関係 relations
 */
private void add 患者と受付嬢の病院関係 Relations() {
		logger.info("add 患者と受付嬢の病院関係 relations");

		//set the source and target agents
		List sourceAgents = new ArrayList(患者集団 s);
		List targetAgents = new ArrayList(受付嬢集団 s);

		Agent targetAgent = (Agent) targetAgents.get(0);

		//Add Relations
		for (int i = 0; i < sourceAgents.size(); i++) {
			Agent sourceAgent = (Agent) sourceAgents.get(i);

			if (sourceAgent != targetAgent) {
				//add relation from source to target
				sourceAgent.addRelation(BoxHospitalModel.RELATIONTYPE_病院
関係,
						targetAgent);
				//add reverse relation from target to source
				targetAgent.addRelation(BoxHospitalModel.RELATIONTYPE_ 病 院
関係,
						sourceAgent);
			}

		}

}

/************************************
 * Name and Desctription for World
 ************************************/

/**
 * Returns the description of World.
```

```
 *
 * @see org.platbox.simulator.model.fmfw.World#getDescription()
 */
public String getName() {
        return "boxhospitalworldWorld";
}

/**
 * Returns the name of World.
 *
 * @see org.platbox.simulator.model.fmfw.World#getName()
 */
public String getDescription() {
        return "";
}

/**
 * create  移動案内 ForAgent 受付嬢集団
 */
private Cell create 移動案内 ForAgent 受付嬢集団(Agent agent) {
        return null;
}

/*************************************
 * Factory method(s) for Information
 *************************************/

/**
 * create  診察券 ForAgent 患者集団
 */
private Cell create 診察券 ForAgent 患者集団(Agent agent) {
        return null;
}

/*************************************
 * Utility methods for generating random number
 *************************************/

/**
 * Generate random number by double.
```

```
 *
 * @param key of random number generator
 * @param minimum
 * @param max
 * @return random number
 */
private double generateRandomNumberByDouble(String generatorName,
                double max, double minimum) {
        RandomNumberGenerator generator = super
                        .getRandomNumberGenerator(generatorName);

        //generate random number.
        double randomNumber = generator.generate();
        double result = minimum + randomNumber * (max - minimum);

        return result;
}

/**
 * Generate random number by int.
 *
 * @param key of random number generator
 * @param minimum
 * @param max
 * @return random number
 */
private int generateRandomNumberByInt(String generatorName, int max,
                int minimum) {
        RandomNumberGenerator generator = super
                        .getRandomNumberGenerator(generatorName);

        //generate random number.
        double randomNumber = generator.generate();
        int result = (int) (minimum + randomNumber * (max - minimum));

        return result;
}

/**
 * Generate random number by long.
```

```
  *
  * @param key of random number generator
  * @param minimum
  * @param max
  * @return random number
  */
private long generateRandomNumberByLong(String generatorName, long max,
                long minimum) {
        RandomNumberGenerator generator = super
                        .getRandomNumberGenerator(generatorName);

        //generate random number.
        double randomNumber = generator.generate();
        long result = (long) (minimum + randomNumber * (max - minimum));

        return result;
}

/**
  * generate elected numbers excluded the value at random
  *
  * @param key of random number generator
  * @param number of elected one
  * @param max
  * @param exclude number
  * @return elected numbers
  */
private int[] generateElectedRandomNumbers(String generatorName,
                int number, int max, int excludedNumber) {
        assert (number < max);
        RandomNumberGenerator generator = super
                        .getRandomNumberGenerator(generatorName);
        int[] electedNumbers = new int[number];
        Arrays.fill(electedNumbers, -1);

        //generate elected number
        for (int i = 0; i < number; i++) {
                int generatedNumber;
                if (excludedNumber >= 0) {
                        generatedNumber = generator.generate(max - 1);
```

```
                    //exclude the number
                    if (generatedNumber >= excludedNumber) {
                        generatedNumber++;
                    }

                } else {
                    generatedNumber = generator.generate(max);
                }
                if (isElectedNumber(electedNumbers, generatedNumber)) {
                    i--;
                    continue;
                } else {
                    electedNumbers[i] = generatedNumber;
                }
            }

            return electedNumbers;
        }

        /**
         * Returns whether target number is elected or not.
         *
         * @param elected numbers
         * @param target number
         * @return whether the number is elected
         */
        private boolean isElectedNumber(int[] electedNumbers, int target) {
            int[] copy = new int[electedNumbers.length];
            for (int i = 0; i < copy.length; i++) {
                copy[i] = electedNumbers[i];
            }

            ;

            Arrays.sort(copy);
            int result = Arrays.binarySearch(copy, target);

            return result >= 0;
        }
```

9.5　受付処理.java

```
/*
 * 作成日： 2006/05/28
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
package boxhospital;

import org.apache.log4j.Logger;
import org.platbox.simulator.model.fmfw.Information;
import org.platbox.simulator.model.fmfw.informations.StringInformation;

public class 受付処理 extends Abstract受付処理 {

    /******************************************
     * Attribute(s).
     ******************************************/
    private static transient Logger logger = Logger
                                .getLogger("boxhospital.受付処理");

    /******************************************
     * Method(s).
     ******************************************/

    /**
     * initialize
     **/
    public void initialize() {
    }
    /**
     * terminate
     **/
    public void terminate() {
    }

    /**
     * 要求を受け移動案内を出す
     **/
    public void 要求を受け移動案内を出す() {
```

```
        //最後に受け取った Information を取得する
        Information 受け取った診察券情報 = (Information) this.getReceivedInformation();

        //Information の内容を文字列で取得する
        String 診察券情報 = 受け取った診察券情報.toString();
        String 症状_脳 = "頭痛";
        String 症状_腹 = "腹痛";
        String 症状_脚 = "膝を痛めた";
        String 移動案内 1 = "脳神経外科に行ってください";
        String 移動案内 2 = "整形外科に行ってください";
        String 移動案内 3 = "内科に行ってください";
        boolean 脳神経外科 = 症状_脳 == 診察券情報;
        boolean 整形外科 = 症状_脚 == 診察券情報;
        boolean 内科 = 症状_腹 == 診察券情報;
        //条件分岐①
        if (脳神経外科) {

                //Goods や Information を送ってきた相手にメッセージを送る
                StringInformation 移動案内 1_1 = new StringInformation(移動案内 1);
                this.sendInformation(移動案内 1_1);
        }

        //条件分岐②
        else if (整形外科) {

                //Goods や Information を送ってきた相手にメッセージを送る
                StringInformation 移動案内 2_2 = new StringInformation(移動案内 2);
                this.sendInformation(移動案内 2_2);
        }

        //条件分岐③
        else if (内科) {

                //Goods や Information を送ってきた相手にメッセージを送る
                StringInformation 移動案内 3_3 = new StringInformation(移動案内 3);
                this.sendInformation(移動案内 3_3);
        }

}
```

9.6　診察要求.java

```
/*
 * 作成日： 2006/05/28
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
package boxhospital;

import org.apache.log4j.Logger;
import org.platbox.simulator.model.fmfw.Information;
import org.platbox.simulator.model.fmfw.informations.ListInformation;
import org.platbox.simulator.model.fmfw.informations.StringInformation;

public class 診察要求 extends Abstract 診察要求 {

    /******************************************
     * Attribute(s).
     ******************************************/
    private static transient Logger logger = (Logger) Logger
                            .getLogger("boxhospital.診察要求");

    /******************************************
     * Method(s).
     ******************************************/

    /**
     * initialize
     **/
    public void initialize() {
    }

    /**
     * terminate
     **/
    public void terminate() {
    }

    /**
     * 自分の状態を受付嬢に伝える
```

```
  **/
 public void 自分の状態を受付嬢に伝える() {

		//整数の乱数を生成する
		int 診察要求乱数 = this.getWorld().getRandomNumberGenerator().generate(2);

		//空の ListInformation をつくる
		ListInformation 診察要求格納リスト = new ListInformation();

		//診察要求を生成
		String 診察要求 1 = "膝を痛めた";
		String 診察内容 2 = "頭痛";
		String 診察要求 3 = "腹痛";
		//	StringInformation をつくる①
		StringInformation 診察要求 1_1 = new StringInformation(診察要求 1);

		//	StringInformation をつくる②
		StringInformation 診察内容 2_2 = new StringInformation(診察内容 2);

		//	StringInformation をつくる③
		StringInformation 診察内容 3_3 = new StringInformation(診察要求 3);

		//診察要求をリストに追加する

		//	ListInformation の N 番目に Information を追加する
		診察要求格納リスト.add(0, 診察要求 1_1);

		//	ListInformation の N 番目に Information を追加する
		診察要求格納リスト.add(1, 診察内容 2_2);

		//	ListInformation の N 番目に Information を追加する
		診察要求格納リスト.add(2, 診察内容 3_3);

		//診察要求格納リストからランダム選び受付嬢に送る

		//	ListInformation の N 番目の Information を取得する
		StringInformation 選ばれた診察要求 = (StringInformation) 診察要求格納リスト.get(診察要求乱数);

		//	指定した RelationType の Relation を持つ Agent 全員に Information を送る
```

```
		int 情報を送った人数 = (int) this.sendInformation(
				BoxHospitalModel.RELATIONTYPE_病院関係,
				BoxHospitalModel.BEHAVIORTYPE_受付処理,
				BoxHospitalModel.INFORMATIONTYPE_診察券, 選ばれた診察要
求, false);

	}

	/**
	 * 移動案内を聞き移動の準備をする
	 **/
	public void 移動案内を聞き移動の準備をする() {

		//最後に受け取った Information を記憶する
		Information 受け取った Information = this.getReceivedInformation();
		this.getAgent().putInformation(BoxHospitalModel.INFORMATIONTYPE_移動案内,
				受け取った Information);
	}

}
```