論文 / 著書情報
Article / Book Information

題目(和文)	モデルメソッド駆動型アーキテクチャとそのモデリング環境の研究
Title(English)	
著者(和文)	倉田正
Author(English)	Tadashi Kurata
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第10663号, 授与年月日:2017年9月20日, 学位の種別:課程博士, 審査員:出口 弘,寺野 隆雄,新田 克己,三宅 美博,小野 功
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第10663号, Conferred date:2017/9/20, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

モデルメソッド駆動型アーキテクチャと
そのモデリング環境の研究

2017年8月

総合理工学研究科

知能システム科学専攻

倉田　正

目次

第1章 序論

本章では、プログラム開発手法としてモデル駆動型アーキテクチャが普及しない原因について考察を行った後、それに基づいて設定した本研究の目的について説明する．次に、本研究による成果の説明を行い、最後に本論文の構成について述べる．

1.1 はじめに

プログラム開発手法としてモデル駆動型アーキテクチャ(MDA)[Soley 2000]がある．これは例えば UML[Russ ほか 2006], [Martin 2003], [Warmer ほか 1998]のようなプラットフォームやプログラミング言語に依存しない抽象的な仕様記述言語で記述された仕様書から実際に動くプログラム(Java、C++、C#、Python 等)[Patrick ほか 2002], [Joshua ほか 2005], [Bjarne 1991], [Jesse ほか 2007], [Mark 1998]を自動生成させる開発手法であり、その目的は「特定領域専門家(特定のモデル化手法を用いる領域の専門家であり、その領域のモデル設計能力は持つがプログラムスキルは持たないユーザ)が、安全かつ効果的にソフトウェアに変更を加えて、変更に関するニーズやビジネスに対する理解を反映する能力を向上させる」[Den 2008]ことにある．この開発手法が登場した当時は「特定領域専門家が概念だけを記述した仕様書から実際に動くプログラムが作成出来る方法」として注目を集めたが、現在この開発手法によるソフトウェア開発は殆ど行われていないと言える状況に陥ってしまっている．特定領域専門家がアプリケーション仕様を記述するだけで実際に利用出来るアプリケーションを作ることが出来ればソフトウェア開発の生産性が飛躍的に向上することは間違いないが現実には何故うまくいっていないのであろうか？

本研究では、この原因について考察を行い、MDA に代わる新たなプログラム開発手法を考案し、更にそれに基づいた新たなアプリケーション開発環境の開発を行った．また、新たなアプリケーション開発環境の開発においては、その実装方法によりアプリケーション開発環境の変更作業効率の大幅な向上が可能であることを明らかにした．

1.2 研究の背景

1.2.1 モデル駆動型アーキテクチャの現状とその問題点

MDA は、プラットフォームやプログラミング言語に依存しない抽象的な仕様記述言語で記述された仕様書から実際に動くプログラム(Java、C++、C#、Python 等)を自動生成させる開発手法である．特定領域専門家がアプリケーション仕様を記述するだけで実際に利用出来るアプリケーションを作成出来るプログラム開発手法であるが、現在この開発手法によるソフトウェア開発は殆ど行われていない．その原因は、(1)MDA のモデル記述が特定モデル化手法ではなく広く一般のモデル化手法を対象としていること、及び(2)実際に動作しない UML のような抽象仕様記述言語を用いてモデルを記述することであると考えられる．MDA では、図 1-1 のように、UML のような汎用的な抽象仕様記述言語を使用して、全てのモデル化手法を対象としたモデル記述を行う．一般モデル記述手法により特定モデル化手法に基づくモデルを記述しなければならない為、モデル構築概念(特定モデル化手法に限定したモデル記述の為の論理思考形式)に基づいた特定モデル化手法によるモデル作成を行うことが出来ず、このことが特定領域専門家によるモデル作成を困難にしているのである．また更に UML のような抽象仕様記述言語から実際に動作する一般プログラミング言語(Java、C++、C#、Python 等)への言語間コンバートが非常に複雑で困難な為プログラム自動生成も実現出来ないのである．実際、MDA のプログラム自動生成ツールに出来ることは、せいぜい特定対象領域、特定プラットフォーム及び特定プログラミング言語限定でメンバ変数と中身が空のメソッドを持つクラス群を作成することぐらいで、実際に動くプログラムを生成することは出来ない．プログラムを実際に動くものとする為には、プログラマがテキストエディタを使用して空のメソッド内にコードを記述しなくてはならない．UML のような抽象言語をより詳細に記述すれば記述するコード量を減らすことは出来るが、それでは抽象言語記述がコードに近づいて複雑になってしまい、抽象言語記述が抽象的ではなくなり、MDA 本来の存在意義が失われてしまうことになる．

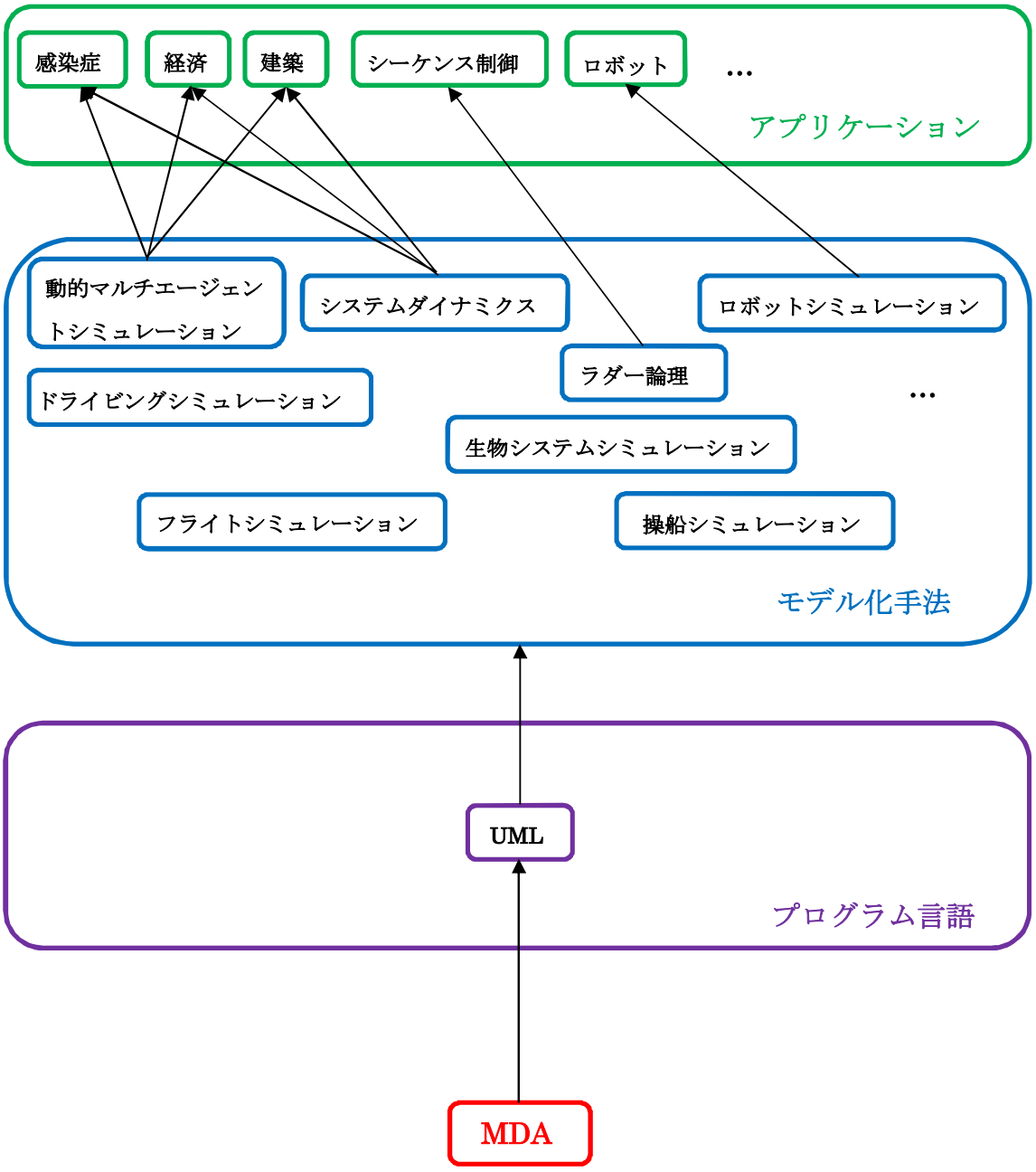


図 1-1 MDA によるプログラム開発の仕組み

例えば、図 1-2 のフローチャートで処理フローを表せるようなエージェントの感染症流行シミュレーションモデルを UML によって記述する例を考える．この例では、各エージェントは感染状態(未感染、感染、恢復及び死亡)及び位置情報(この例では職場と家)を持っている．各エージェントの位置情報はランダムに変化し、その間に SIR モデル[Kermack ほか 1927]等のアルゴリズムによりエージェント同士の接触に基づいて感染したエージェントが発生する．

ここで SIR モデルは、時刻 t における未感染者数を $S(t)$、感染者数を $I(t)$、恢復者数を $R(t)$とした時に、その変化速度を以下のように定義出来る．k は感染率、lは恢復率である．

$$\frac{d}{dt}S(t) = -kS(t)I(t)$$

$$\frac{d}{dt}I(t) = kS(t)I(t) - l\,I(t)$$

$$\frac{d}{dt}R(t) = l\,I(t)$$

この時、

$$\frac{d}{dt}\big(S(t) + I(t) + R(t)\big) = 0$$

であり、これは条件

$$S(t) + I(t) + R(t) = \text{一定}$$

を満たしている．

従って、時刻$t + \Delta t$における各人数はオイラー法により次のように求めることが出来る[水島ほか 2002], [川崎 2000], [長嶋 1995], [吉川 1994]．

$$S(t + \Delta t) = S(t) - \Delta t \cdot kS(t)I(t)$$

$$I(t + \Delta t) = I(t) + \Delta t \cdot (kS(t)I(t) - l\,I(t))$$

$$R(t + \Delta t) = R(t) + \Delta t \cdot l\,I(t)$$

$S(0)$、$I(0)$、$R(0)$、k、l及びΔtを指定して、繰り返し処理の中で毎回この計算を行うことによりシミュレーションは進行する．

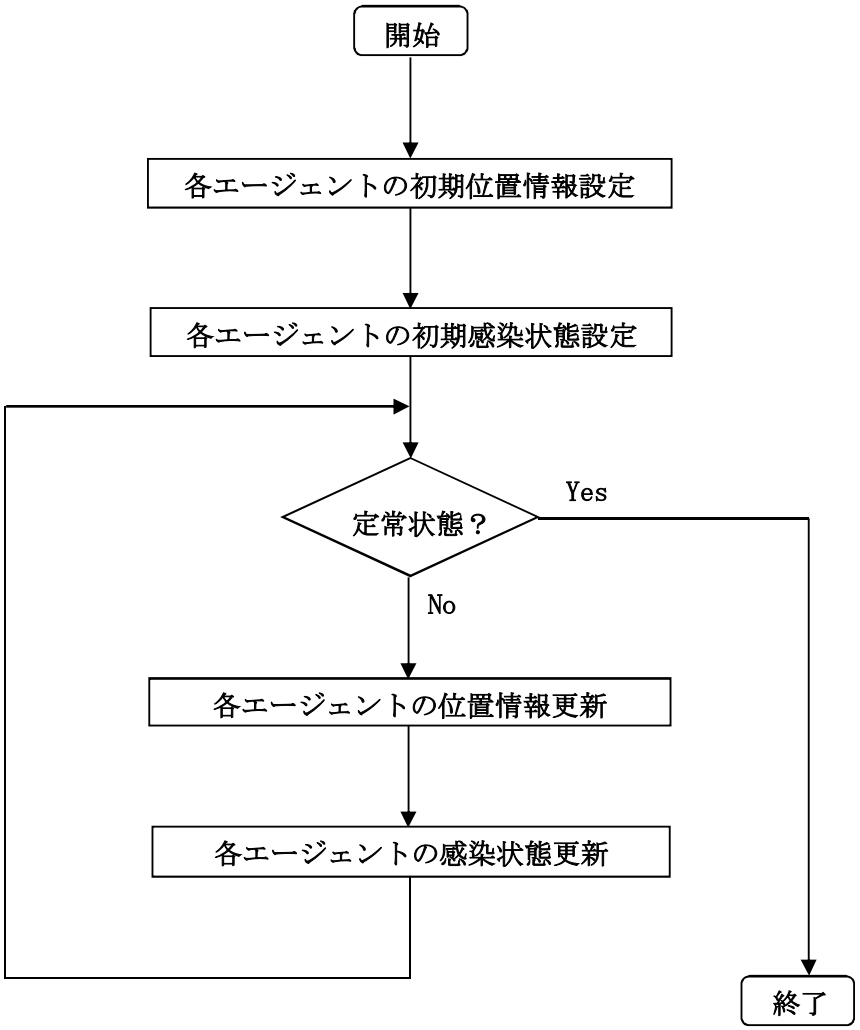


図 1-2 感染症流行シミュレーションのフローチャート

これを UML によって記述すると図 1-3 のユースケース図、図 1-4 のステートチャート図及び図 1-5 のクラス図のようになり、プラットフォーム及び使用する一般プログラミング言語に則ってエージェントクラス、変数クラス及び制御クラスを作成し、クラス図に基づいてメンバ変数及びメソッドをテキストエディタにより記述しなければならない[Martin 1996].

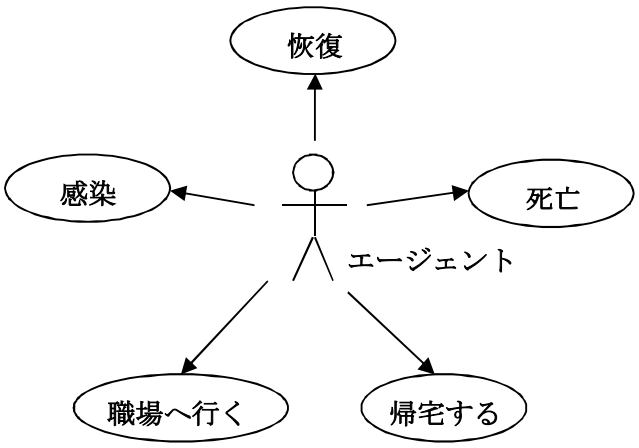


図 1-3　感染症流行シミュレーションのユースケース図

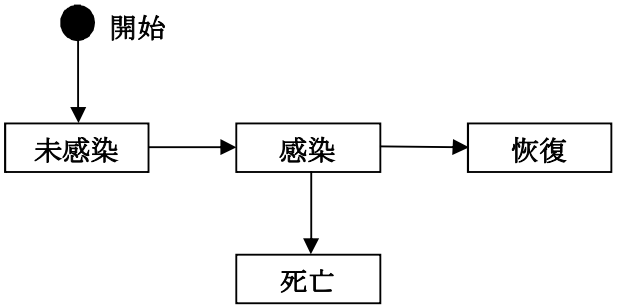


図 1-4 感染症流行シミュレーションのステートチャート図

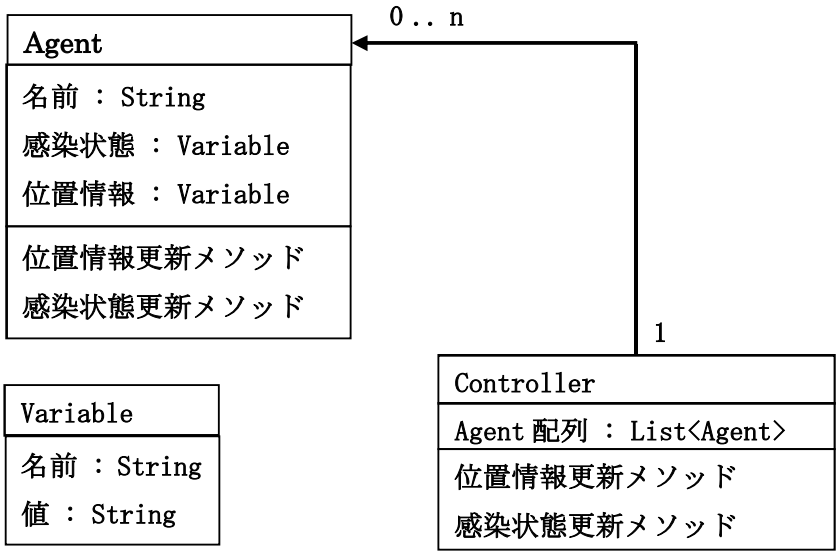


図 1-5 感染症流行シミュレーションのクラス図

仮に、プログラム自動生成機能が存在していたとしても、上述したように生成されるのはメンバ変数と中身が空のメソッドを持つ以下のようなクラス群程度となる.

```
public class Variable {
  public String name;
  public String value;
}

public class Agent {
  public String name;
  public Variable infection;
  public Variable location;

  public void updateLocation() {
    // Do something
  }
  public void updateInfectionStatus() {
    // Do something
  }
}

public class Controller {
  private List<Array> agents;

  public void updateLocation() {
    for (Agent agent:agents)
      agent.updateLocation ();
  }
  public void updateInfectionStatus() {
    for (Agent agent:agents)
      agent.updateInfectionStatus();
  }
}
```

従って、テキストエディタによるプログラミングが必要になり、これでは特定領域専門家がプログラムを作成して実行することは困難である.

本研究では、特定領域専門家がプログラムを作成して実行することが可能となる、MDA に代わる新たなプログラム開発手法について検討を行う.

1.3 本研究の目的と意義

本研究では、MDA に代わる新たなプログラム開発手法を、(1)モデル記述が広く一般のモデル化手法を対象とせず、特定のモデル構築概念に基づくモデル化手法を対象とすること、及び(2)UML のような抽象仕様記述言語や情報処理技術者向けの特定領域言語ではない特定領域専門家向けの特定領域言語、即ちモデル化手法特化型プログラム言語によりモデルを記述すること、更に(3)そのモデル化手法特化型プログラム言語が本論文の第２章で述べる９つの要件を満たしていること、と云う３つの要件が満たされた場合に実現可能となるプログラム開発手法であると定義し、この新たなプログラム開発手法においては、そのモデル化手法特化型プログラム言語によるプログラムを、テキストエディタではなくモデル化手法特化型プログラム言語に特化した GUI を持つ総合的なプログラム統合開発環境により作成出来ることを示す.

本研究では、本論文の第２章で述べる９つの要件を満たしているモデル化手法特化型プログラム言語を選択し、各々のプログラム言語によるプログラムを新たなプログラム開発手法により作成することが可能なプログラム開発環境の設計・実装を行った．新たなプログラム開発環境においては、テキストエディタを必要としないプログラム開発が可能であり、また作成したプログラムは実際に動作させることが可能である．この新たなプログラム開発環境では、MDA では不可能であった、特定領域専門家がプログラムを作成して実行することが可能になっている．また、この新たなプログラム開発環境では、実行結果の可視化が可能であり、更に開発環境自体の変更が容易であることも明らかにした.

毎年行われる、この開発環境を用いた Workshop では、僅か数時間のチュートリアルで誰もがプログラムを作成して実行することが出来るようになっている．また、省エネプロジェクトや既存の通信データ変換等、実際の現場において、現在この開発環境は利用されつつある.

1.4 本論文の構成

本論文の構成について説明する．第２章では、MDA に代わる新たなプログラム開発手法を定義し、そのプログラム開発手法によるプログラム開発を可能にするプログラム開発環境について述べる．第３章では、そのプログラム開発環境の詳細設計について述べる．第４章では、そのプログラム開発環境を使用してプログラム開発を行った事例について述べる．第５章では、本研究の結論と課題について述べる.

第2章 モデルメソッド駆動型アーキテクチャ

本章では、モデル駆動型アーキテクチャに代わるプログラム開発手法として、新たにモデルメソッド駆動型アーキテクチャを定義する．次に、この開発手法に適したプログラム言語の特徴を考察し、その特徴を持つプログラム言語を選択することが、この開発手法によるプログラム作成にとって必要不可欠であることを示す．最後に、その特徴を持つ２つのプログラム言語を選択し、そのプログラム言語によるプログラムをこの開発手法により作成することを可能にするプログラム開発環境の設計・実装を研究の目的として設定する．

2.1 新たなプログラム開発手法

MDA に代わる新たなプログラム開発手法は、図 2-1 のように、広く一般のモデル化手法を対象とせず、特定のモデル構築概念に基づくモデル化手法を対象とする．その記述言語は、UML のような抽象仕様記述言語や情報処理技術者向けの特定領域言語ではない特定領域専門家向けの特定領域言語、即ちモデル化手法特化型プログラム言語とする．そのプログラム言語で記述されたプログラムは実際に動作する．このプログラム開発手法では、モデル化手法特化型プログラム言語によるプログラムを、テキストエディタではなくモデル化手法特化型プログラム言語に特化した GUI により作成する．

モデル記述が特定モデル化手法に制限されていれば、そこには自ずとそのモデル構築概念が存在するので、そのモデル化手法の専門家がモデル記述を行うことが可能となる．例えば、システムダイナミクスのモデル化手法ではストックフローのような数値シミュレーションモデル構築概念に基づくモデル記述が可能となる．役割指向エージェントベースシミュレーション[Tanuma ほか 2004]のモデル化手法ではエージェント及びその振る舞いを定義するモデル構築概念に基づくモデル記述が可能となる．また、リアルワールドオペレーティングシステム(RWOS) [出口 2016], [竹林ほか 2016], [出口 2015], [出口 2014(1)], [出口 2014(2)]のモデル化手法では実世界及び仮想世界の双方を含む空間上のエージェント相互関係を表すモデル構築概念に基づくモデル記述が可能となる．

汎用的である一般プログラミング言語(Java、C++、C#、Python 等)には数多くの変数型が存在するので、これによるプログラム開発には Eclipse[Eclipse 1998], [水島 2003]や Microsoft Visual Studio[Microsoft Visual Studio 1997], [George ほか 1996]のようなプログラム統合開発環境(IDE)を用いることになる．これにはテキストエディタによるプログラム記述が必要となり、プログラマではない特定領域専門家がモデル記述を行うことは出来ない．Java を使用する MASON[MASON 2003]及び C++または Java を使用する Repast[Repast 2006]はこれにあたる．これに対して、モデル化手法特化型プログラム言語には特定モデル化手法の限られた種類のオブジェクトを表現する変数型しか存在しない．例えば、図 1-5 のクラス図における Agent クラス、Variable クラス及び Controller クラス

はこれに相当する．従って、変数型を選択形式に出来る等の特長を生かすことにより、テキストエディタを必要としないその言語に特化したモデル記述用の GUI を存在させることが可能となる．

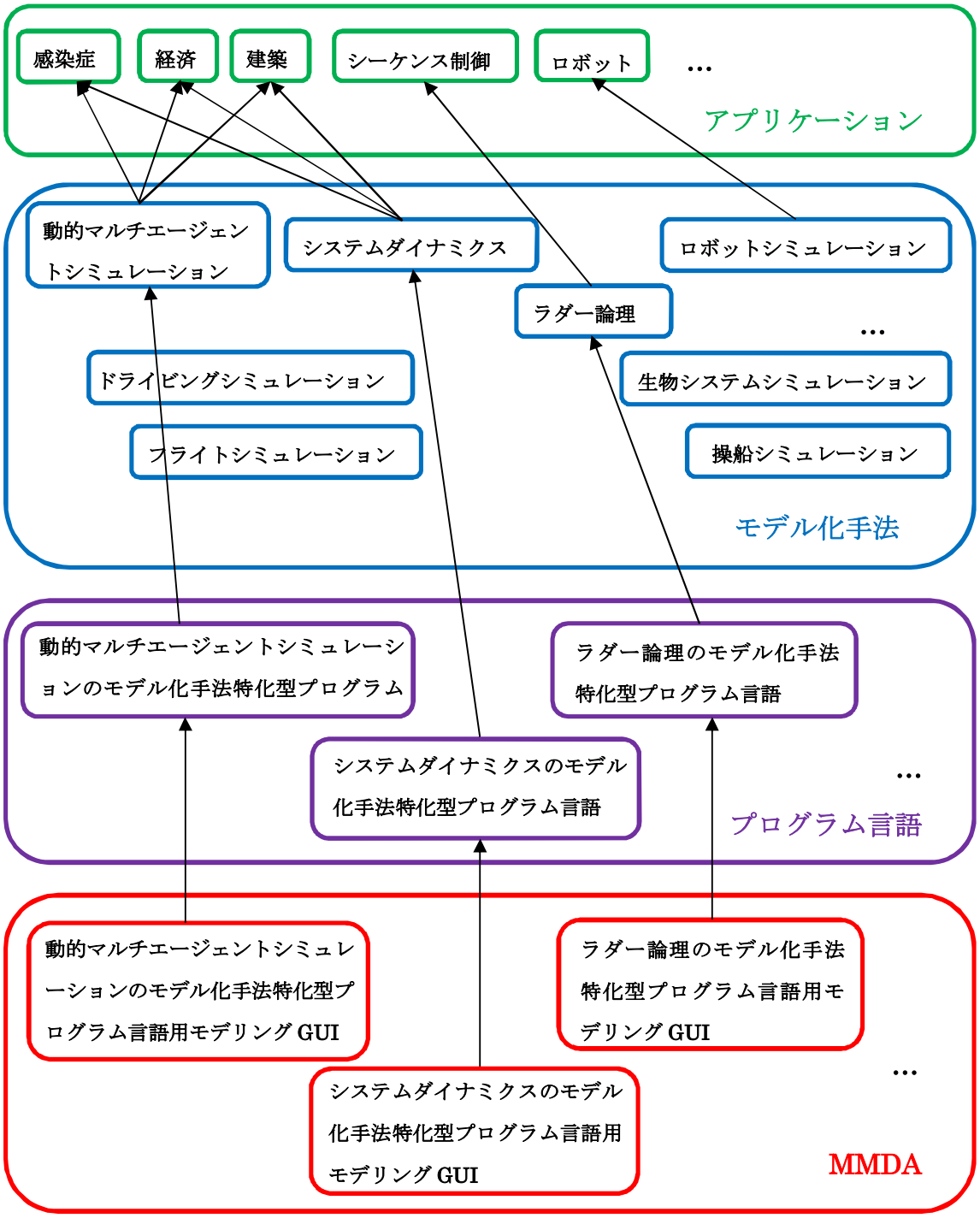


図 2-1 新たなプログラム開発手法によるプログラム開発の仕組み

また、そのモデル化手法特化型プログラム言語によって作成されたプログラムが実際に動作可能であれば、記述されたモデルから実際に動作するプログラムへのコンバートも可

能となる．その GUI に拡張性が備わっていればユーザによる GUI の拡張が可能となり、その GUI に可視化機能が備わっていればプログラム実行結果を可視化することが出来る．よって、この新たなプログラム開発環境を利用出来れば、特定領域専門家はモデル記述を行うだけでプログラムを実行することが可能となり、更に、その GUI の拡張及びプログラム実行結果の可視化までを行うことが可能となる．このモデル化手法特化型プログラム言語に特化した GUI によりモデル記述を行う総合的なプログラム統合開発環境を特にモデリング GUI と定義する．

即ち、MDA に代わる新たなプログラム開発手法では、特定モデル化手法とそのモデル化手法特化型プログラム言語によりモデル作成メソッドが確立され、処理フロー、オブジェクト及び変数の定義から命令文生成まで全てのモデル記述を GUI で行うことが出来、更にプログラム実行、GUI 拡張及びプログラム実行の可視化までも行える総合的なプログラム統合開発環境であるモデリング GUI を構築することが可能となる．そして、このモデリング GUI が存在することによって、特定領域専門家が安全且つ効果的にソフトウェアの作成及び変更を行うことが可能となるのである．

以上述べたように、モデル駆動型アーキテクチャとは異なり、この新たなプログラム開発手法では、モデル記述を特定モデル化手法に制限し、更に後述する 9 つの要件を満たすモデル化手法特化型プログラム言語が存在することによって、モデリング GUI によるプログラム開発が可能となる．本研究では、この新たなプログラム開発手法を Model Method Driven Architecture(MMDA)と定義し、後述する 9 つの要件を満たすモデル化手法特化型プログラム言語を選択することにより、MMDA によるプログラム開発を行うことが出来るモデリング GUI の構築について検討を行う．

表 2-1 MDA と MMDA の比較

比較項目	MDA	MMDA
モデル記述の対象	広く一般のモデル化手法	特定のモデル構築概念に基づくモデル化手法
プログラム言語	UML のような抽象仕様記述言語	特定領域専門家向けの特定領域言語(モデル化手法特化型プログラム言語)
プログラム言語により作成されるプログラム	実際には動作しない	実際に動作可能である
GUI	存在しない	モデリング GUI

2.2 モデル化手法特化型プログラム言語

MMDA では、モデル化手法特化型プログラム言語で記述されたプログラムを、モデリング GUI により作成する．従って、モデリング GUI の実現はそのモデル化手法特化型プログラム言語の仕様に依存する．モデリング GUI を実現する為には、そのモデル化手法特化型プログラム言語が以下の９つの要件を満たしている必要がある．

⑴処理フローを定義出来る

処理フローとは、モデルに於けるフローチャートであり、システムダイナミクスに於いては各フローの繋がり、役割指向エージェントベースシミュレーションやリアルワールドオペレーティングシステムに於いてはループ内処理手順を表すものである．処理フローを明確に定義出来る言語仕様であれば、GUI による処理フロー定義が可能となるので、時間軸に沿った直感的なモデル記述が可能となる．

⑵オブジェクトの種類が限られている

オブジェクトとは、モデルを構成する要素であり、システムダイナミクスに於いてはストック及びフロー、役割指向エージェントベースシミュレーションに於いてはエージェントやエージェントが保持する変数を表すものである．オブジェクトの種類が限られていれば、命令を作成する際に必要なオブジェクトをコンボボックス等により選択して指定することが出来る．

⑶命令の種類が限られている

命令の種類が限られていれば、命令をコンボボックス等により選択して指定することが出来る．命令とその引数であるオブジェクトを選択方式によって指定出来ることによりテキストエディタを使用しない命令文の作成が可能になる．

⑷オブジェクト間の依存関係が明確になっている

１つのオブジェクトが保持出来るオブジェクトの種類及び数が明確になっている必要がある．この条件が満たされていれば、命令文作成に際して、オブジェクトの選択を階層型にすることが出来る．

⑸オブジェクト集合を表すオブジェクトを定義出来る

オブジェクト集合を１つのオブジェクトとして定義出来る必要がある．このオブジェクト集合を処理出来る命令を存在させれば、命令文作成をシンプルに行うことが出来る．

⑹オブジェクトの振る舞いが限定されている

特定モデル化手法に基づいているので、各オブジェクトの振る舞いは限定されている筈である．このことにより、オブジェクト及び命令の種類が限定される．もし、この条件が満たされていなければ、モデリング GUI の構築は難しくなる．

⑺言語仕様に拡張性が備わっている

言語仕様に、新たな命令の追加機能や既存のプログラム利用機能等の拡張性が備わっていれば、モデリング GUI にもその拡張性を反映することが出来る．

⑻経過時間を表す変数を定義出来る

動的システムの場合、時間を表す変数を定義出来れば、チャートやアニメーションによるプログラム実行の可視化を行うことが出来る

⑼実行可能である

このプログラム言語で記述されたプログラムは実際に動作するものでなければならない.

以上、モデリング GUI は単なる GUI ではなく、モデリングの為の総合的なプログラム統合開発環境なので、そのモデル化手法特化型プログラム言語の仕様には多くの要件が求められる．例えば、標準的なプログラム言語である Fortran、Java、C、C++等はこれらの要件を満たしていない.

これらの要件のいくつかを満たすものとして、システムダイナミクスのプログラム言語である Stella[Stella 1987]及びプログラマブルロジックコントローラ[PLC 1968], [高来ほか 2014], [岡本 2014]のプログラム言語である ST 言語[Karl-Heinz ほか 2010]が存在する.

システムダイナミクスは自然科学、経済及び社会等の事象をシステムと捉えてその数値シミュレーションを行いシステムの状態遷移を観測／分析するものである. Stella はストックを表すオブジェクトを定義することが出来る．またフローを表すオブジェクトも定義出来るので全体の処理フローを定義することが可能であり、ストックフローの数値シミュレーションモデル構築概念に基づいた数値シミュレーションプログラムを作成することが出来る．Stella には、そのプログラムを開発する為のプログラム開発環境が存在している．図 2-2 は Stella のプログラム開発環境によるインフルエンザ流行シミュレーション編集画面である．システムダイナミクスについての数学的知識を持つユーザであれば、この GUI を使用することにより、モデル記述を行うだけでプログラムを作成して実行することが出来る.

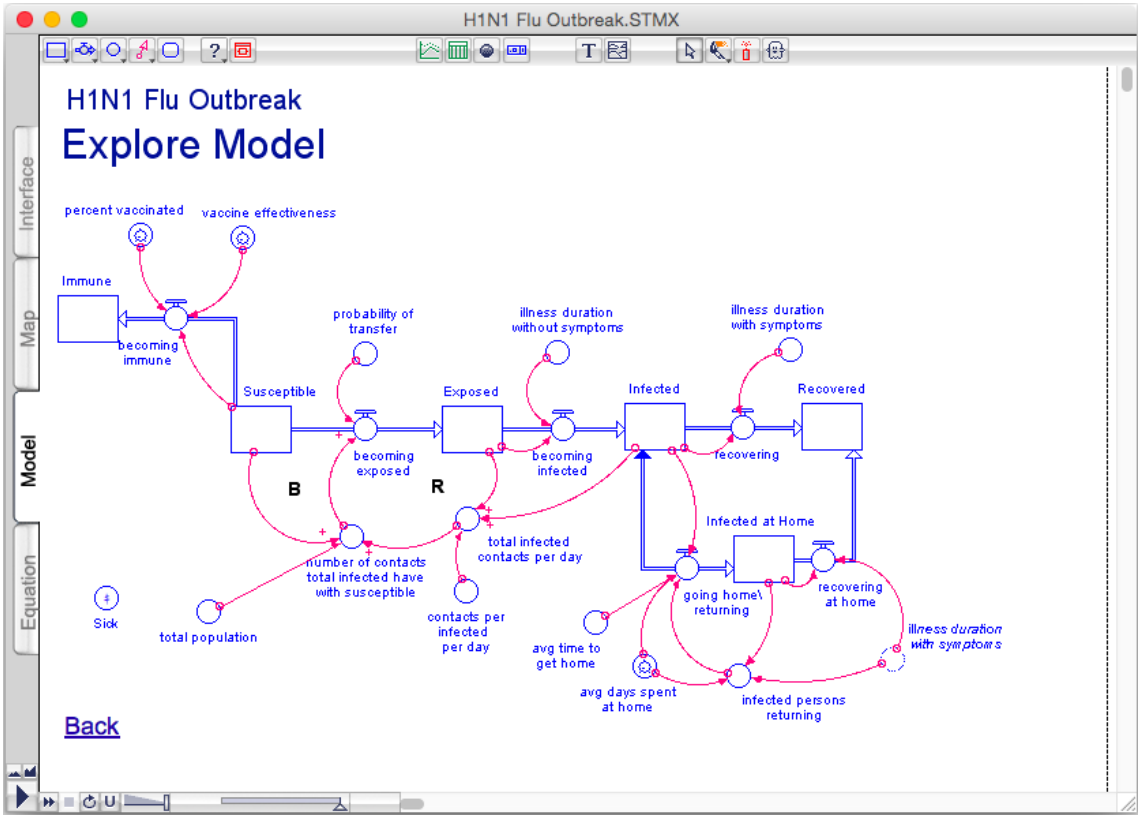


図 2-2 Stella によるインフルエンザ流行シミュレーション編集画面

Stella では図 2-3 のようにオブジェクトアイコンからストックオブジェクトを選択した状態で編集画面内をクリックすることにより新たなストックオブジェクトを生成される．ストックオブジェクトが保持する数式は図 2-4 のようにストックオブジェクト編集用ユーザインタフェースの数式エディタにより定義することが出来る．

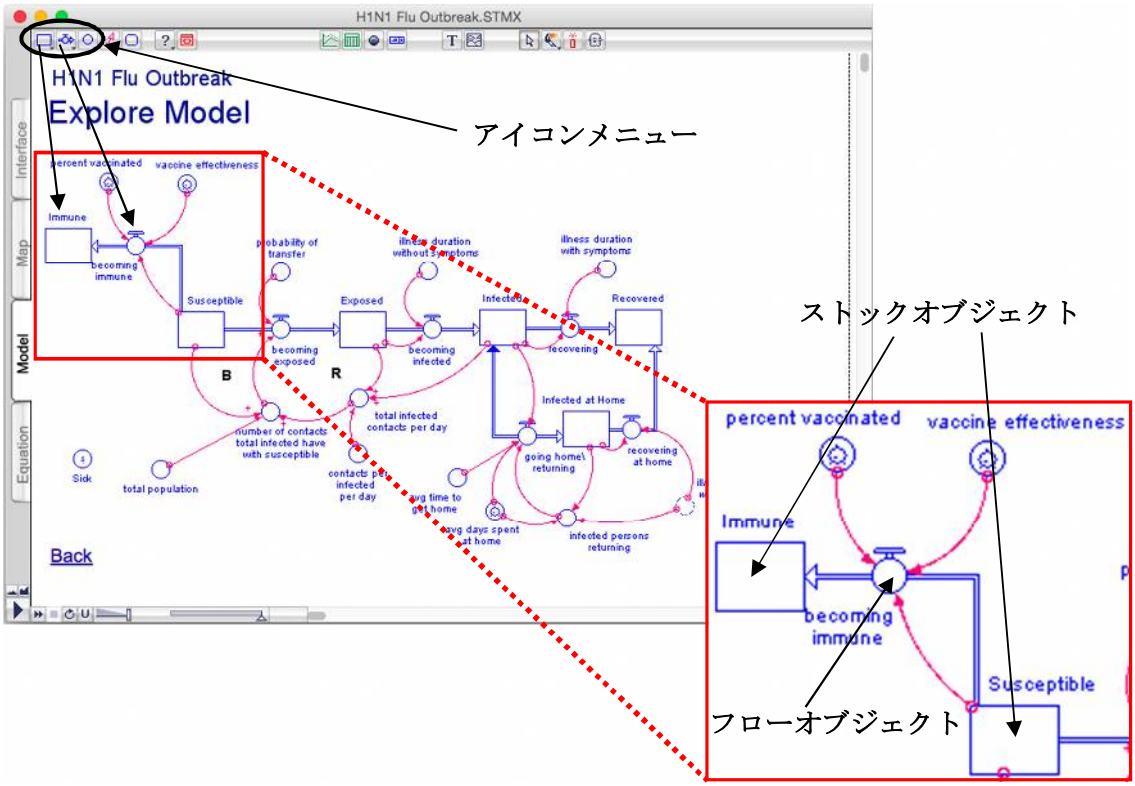


図 2-3 Stella のオブジェクト

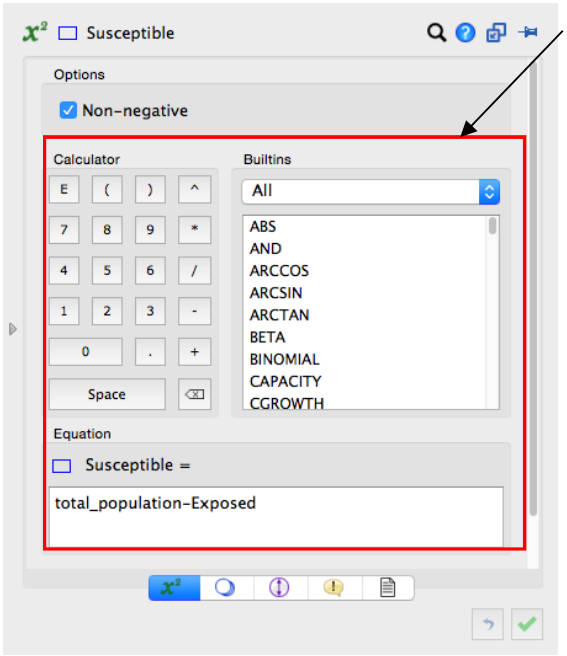


図 2-4 オブジェクト編集用 GUI

Stella では図 2-5 のようにオブジェクトアイコンからフローオブジェクトを選択し、編集画面内のストックオブジェクト間を繋ぐことにより新たな処理フローが定義される．フローオブジェクトが保持する数式は図 2-6 のようにフローオブジェクト編集用ユーザインタフェースの数式エディタにより定義することが出来る．

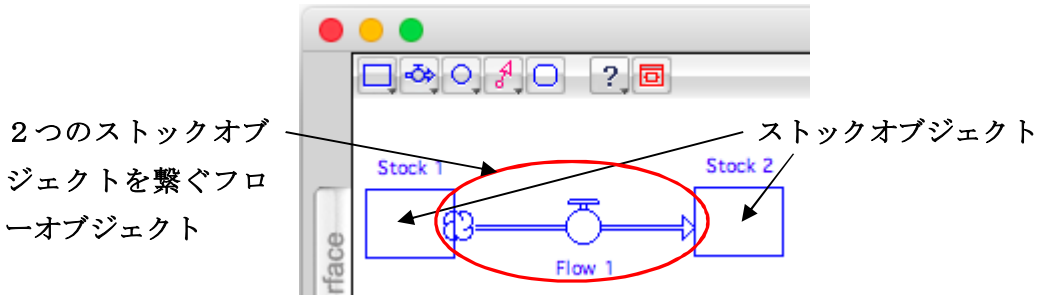


図 2-5 Stella のフローオブジェクト

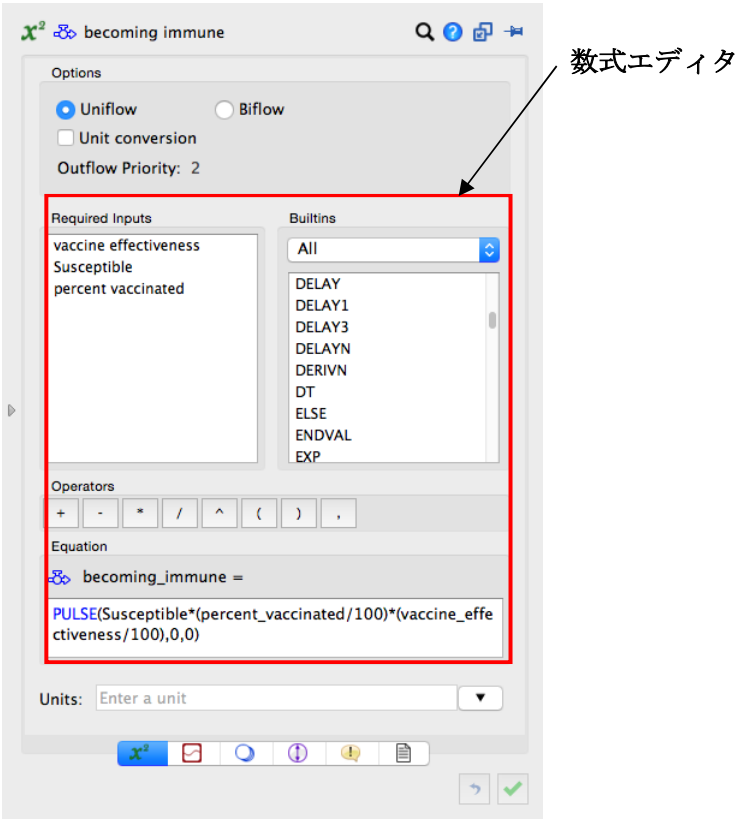


図 2-6 Stella のフローオブジェクト編集用 GUI

Stella では図 2-4 及び図 2-6 のようにストックオブジェクト及びフローオブジェクトへ定義する数式が命令文となる．既に述べたように、これらの数式は数式エディタによって定義出来る上、数式の書式は汎用的である為、ユーザが新たに書式を覚える必要性が無い．

プログラマブルロジックコントローラは一般にシーケンサと呼ばれるリレー回路の制御装置であり、ST 言語と呼ばれるモデル化手法特化型プログラム言語により、その制御プログラムを記述する．ST 言語により処理フローを定義することが可能であり、ラダー論理のモデル構築概念に基づいたリレー制御プログラムを作成することが出来る．

ST 言語には、そのプログラムを開発する為のプログラム開発環境として三菱電機社製 GX Developer が存在している．図 2-7 は GX Developer によるシーケンサプログラム編集画面である．リレー回路及びラダー論理(論理回路記述手法)についての知識を持つユーザであれば、この GUI を使用することにより、モデル記述を行うだけでリレー制御プログラムを作成して実行することが出来る．

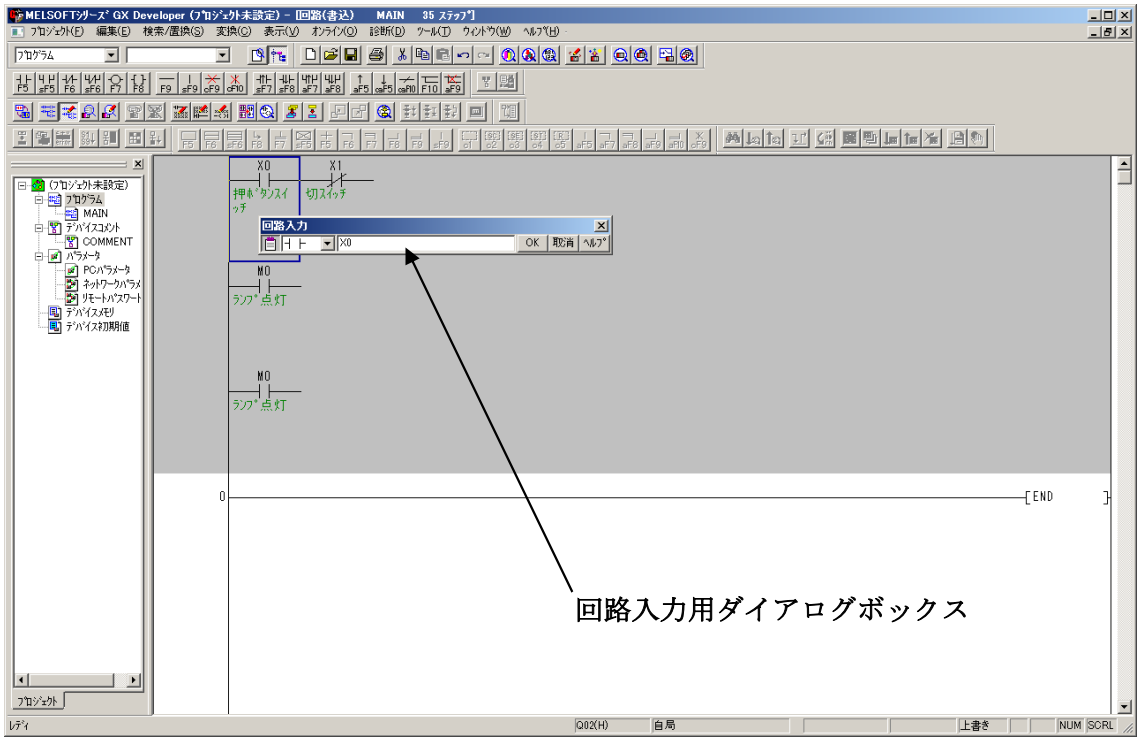


図 2-7 GX Developer による回路編集用 GUI

GX Developer では、図 2-7 のようにマウス操作により回路入力ダイアログボックスを表示させ、任意の回路パーツを選択して繋げることにより回路を作成することが出来る．

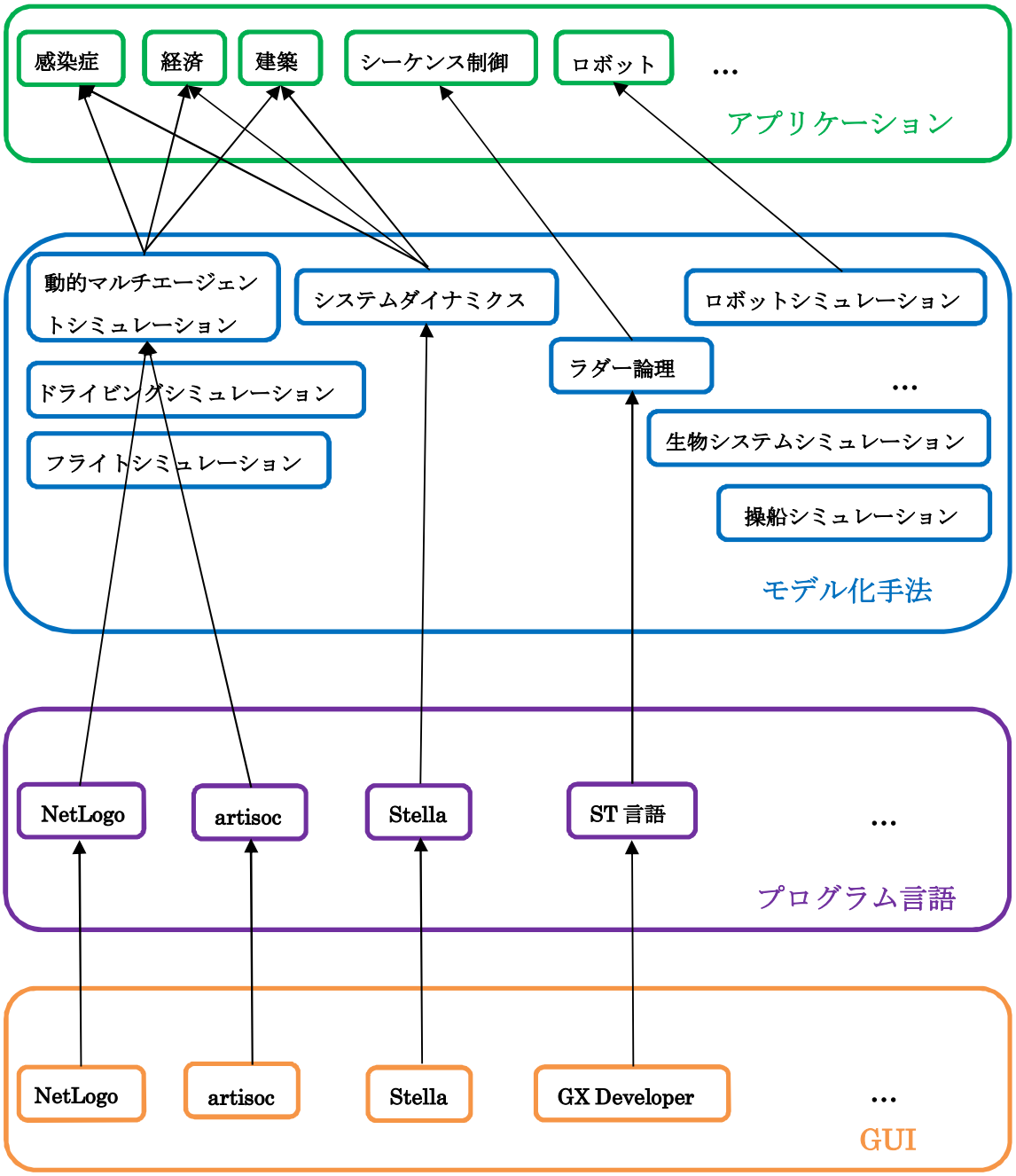


図 2-8 GUI によるモデリングが行えるプログラム開発環境の仕組み

Stella や ST 言語はモデリング GUI に必要な全ての条件を満たしているわけではない．また、そのプログラム開発環境である Stella や GX Developer は MMDA のような一貫したアーキテクチャ概念に基づいて設計・実装された GUI ではない．それらは、個別のプログラム言語に対してアドホックに設計･実装された固有の GUI である．これに対し､MMDA に基づくモデリング GUI は、Stella や GX Developer が持つ GUI の特長だけでなく、当該のモデル化手法特化型プログラム言語が持つ条件を全て生かすことで拡張性と柔軟性を持つ GUI のフレームワークとなっている．

Stella や GX Developer のような GUI を持つプログラム開発環境としては、他に座標系指向エージェントモデルシミュレーション用の NetLogo[NetLogo 1999] 及び artisoc[artisoc 2009], [山影 2007]がある．座標系指向エージェントベースシミュレーションは、平面座標で指定されるセルで構成された空間において、エージェントがセル上を移動し、相互に作用することにより各エージェントが保持する変数及びグローバル変数の値が変化して起こる空間の状態遷移を観測／分析するものである．図 2-8 はこれらのプログラム開発環境によるプログラム開発の仕組みを示したものである．

以上を踏まえて、本研究では、モデリング GUI に必要な条件を満たすモデル化手法特化型プログラム言語を選択し、そのプログラム言語によるプログラムを MMDA により開発する為のモデリング GUI を構築することについて検討を行う．そのモデリング GUI は、Stella や GX Developer が持つ GUI の特長だけでなく、そのモデル化手法特化型プログラム言語が持つモデリング GUI を実現する為に必要な条件を全て生かしたものでなければならない．

2.2.1 役割指向エージェントベースシミュレーション記述言語 SOARS

役割指向エージェントベースシミュレーションは、エージェントが任意の役割を選択し、空間内に配置されたスポット間を移動する間に、エージェント間相互作用により各エージェント／スポットが保持する変数の値が変化して起こる空間の複雑な状態遷移の観測／分析を行うものである．エージェントの役割を任意に変えられることで、非線形に変化する複雑なエージェント及び空間の状態をシミュレートすることが出来る．

SOARS(Spot Oriented Agent Role Simulator)[田沼ほか 2007], [Ichikawa ほか 2007]は東京工業大学出口研究室で開発された役割エージェントベースシミュレーション記述用のプログラム言語であり、現在も進化を続けている．

SOARS はオブジェクトとしてエージェント／スポット及びそれらの振る舞いを定義するロールによって構成されていて、エージェントは任意のスポット上を移動することが出来る．エージェント／スポットは変数(文字列、数値、配列、ハッシュテーブル等)を、ロールはエージェント／スポットが実行する命令(条件及びアクション)を保持している．エージェント／スポットが任意のロールを選択し、その変数を使用してロールの命令を実行することによりシミュレーションが進行する．図 2-9 はこの仕組みを示したものである．

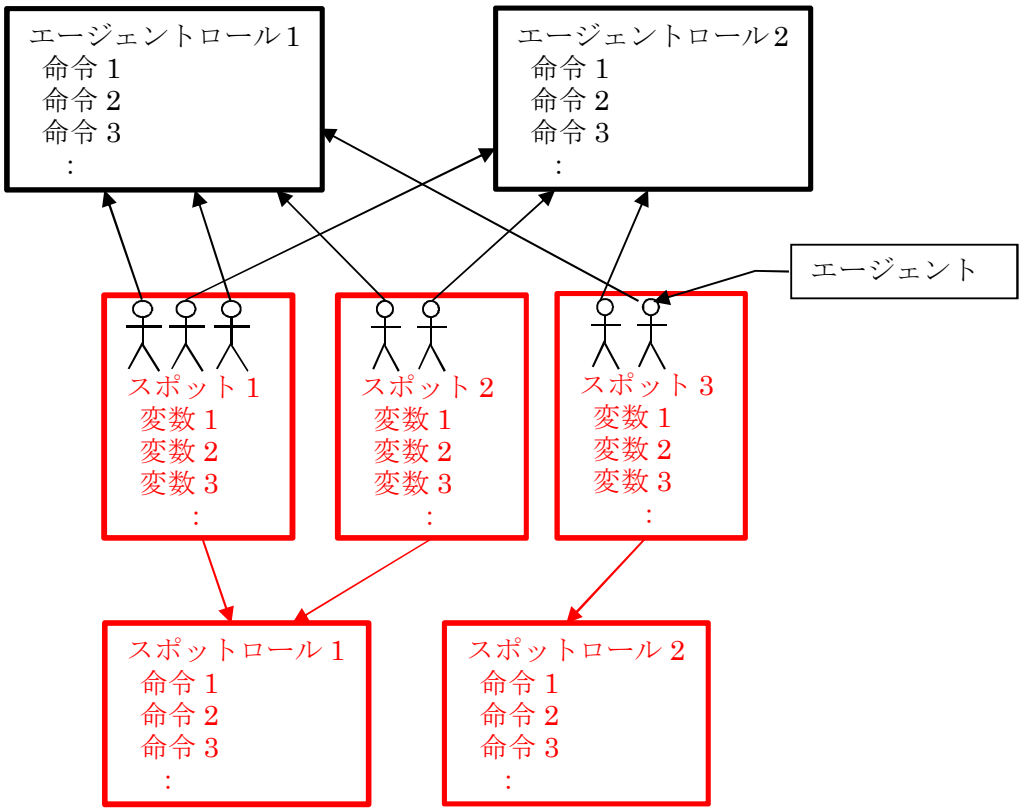


図 2-9 SOARS による役割エージェントベースシミュレーションの仕組み

SOARS では処理フロー定義手法としてステージ概念を使用している．ステージ概念では、シミュレーションの処理フローが 1 つ以上のステージの連続として表現される．各ステージでは、各エージェント及びスポットがそのステージで行うべき命令を一斉に実行するが、それらの命令は全て並列実行可能であるもののみとし、並列実行出来ない命令は別のステージで実行するようにする．このようにステージ概念に基づいて処理フローを設計することにより、シミュレーションにおける全ての命令が正確な順序で実行される図 2-10 のような仕組みとなっている．

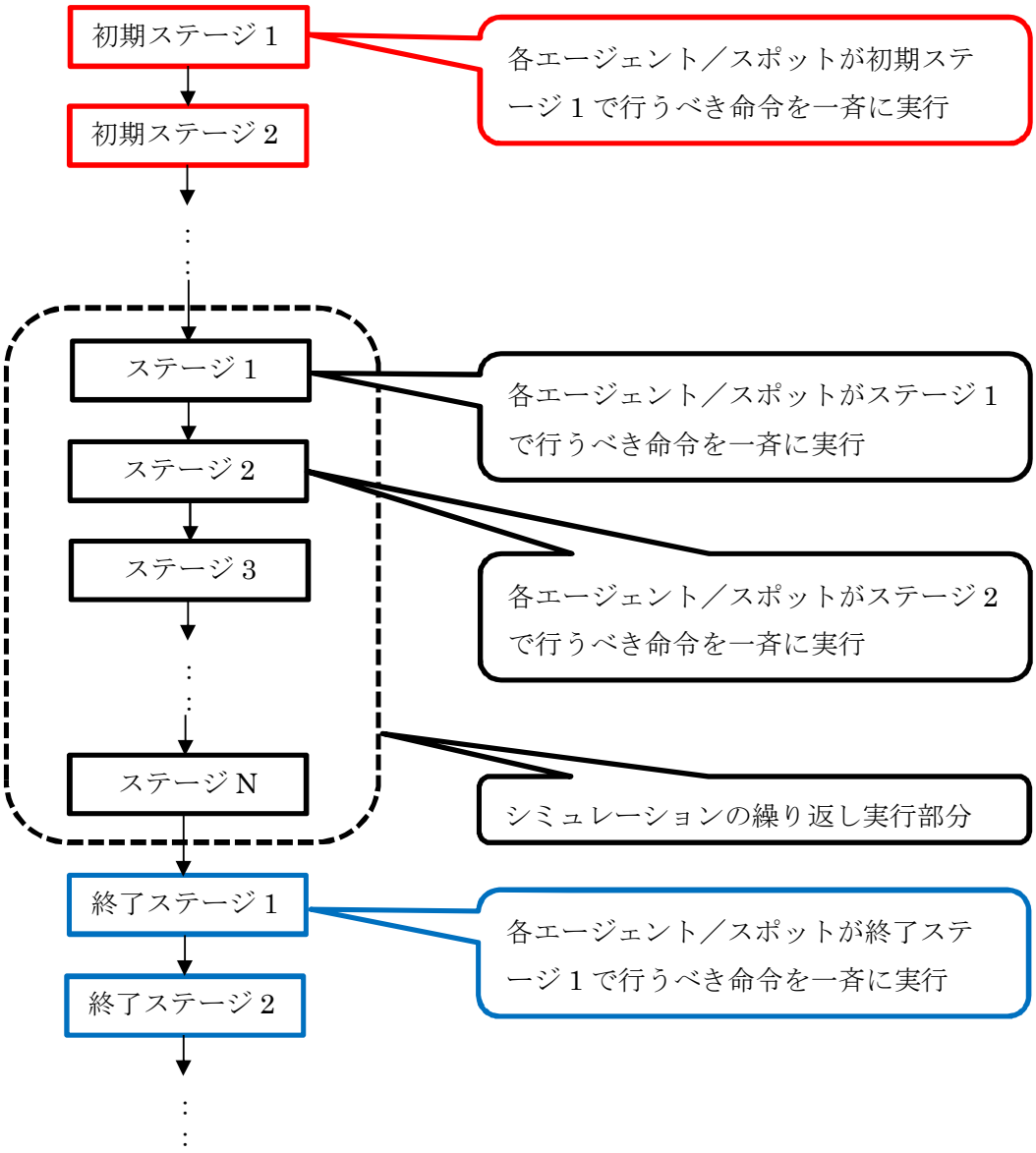


図 2-10 SOARS のステージ概念

SOARS は、役割指向エージェントベースシミュレーションのモデル構築概念に基づくモデル化手法の為のモデル化手法特化型プログラム言語であり、以下のようにモデリング GUI を実現する為の９つの要件を満たしている.

(1)ステージ概念により処理フローを定義出来る

(2)オブジェクトの種類が、エージェント、スポット、ロール、変数(文字列、数値、配列、ハッシュテーブル等)に限られている

(3)エージェント及びスポットが実行出来る命令の種類が限られている

(4)エージェント、スポット、ロール、変数間の依存関係が明確になっている

(5)エージェント、スポット、ロール、変数の集合を表すオブジェクトを定義出来る

(6)エージェント及びスポットの振る舞いが限定されている

(7)SOARS には、Java のリフレクション機能を利用して任意の Java プログラムを実行出来ると云う拡張性が備わっている

(8)SOARS では､経過時間を表す変数を定義出来るのでチャートやアニメーションのようなプログラム実行の可視化を行うことが出来る

(9)SOARS により記述されたプログラムは SOARS Simulator(Java プログラム)によって実際に動作させることが可能である

従って、SOARS によるプログラムを MMDA により開発する為のモデリング GUI を構築することが可能であり、役割指向エージェントベースシミュレーションのモデル構築概念を持つユーザであれば、プログラマではなくても、そのモデリング GUI を利用してエージェント間の複雑なやりとりを表現するプログラムを作成して実行することが可能となる.

2.2.2 実世界と仮想世界を繋ぐオペレーティングシステム記述言語 RWOS

RWOS(リアルワールドオペレーティングシステム)は、東京工業大学出口研究室で開発された実世界及び仮想世界の双方を含む空間上で動作するプログラム記述用のプログラム言語であり、現在も進化を続けている.

RWOS は、IoE[シスコ 2013]時代を迎えて、多様なデバイス(センサー、製造装置、情報端末等)上で動作するプログラム(ロールコンテナ)を記述すること、及び、ロールコンテナにより構成される実世界及び仮想世界の双方を含む空間上で、共通の通信プロトコルを使用し、処理フローに従って、ロールコンテナを正しい順序で実行させるプログラム(プロジェクト)を記述することを目的として開発された. 図 2-11 はこの仕組みを示したものである.

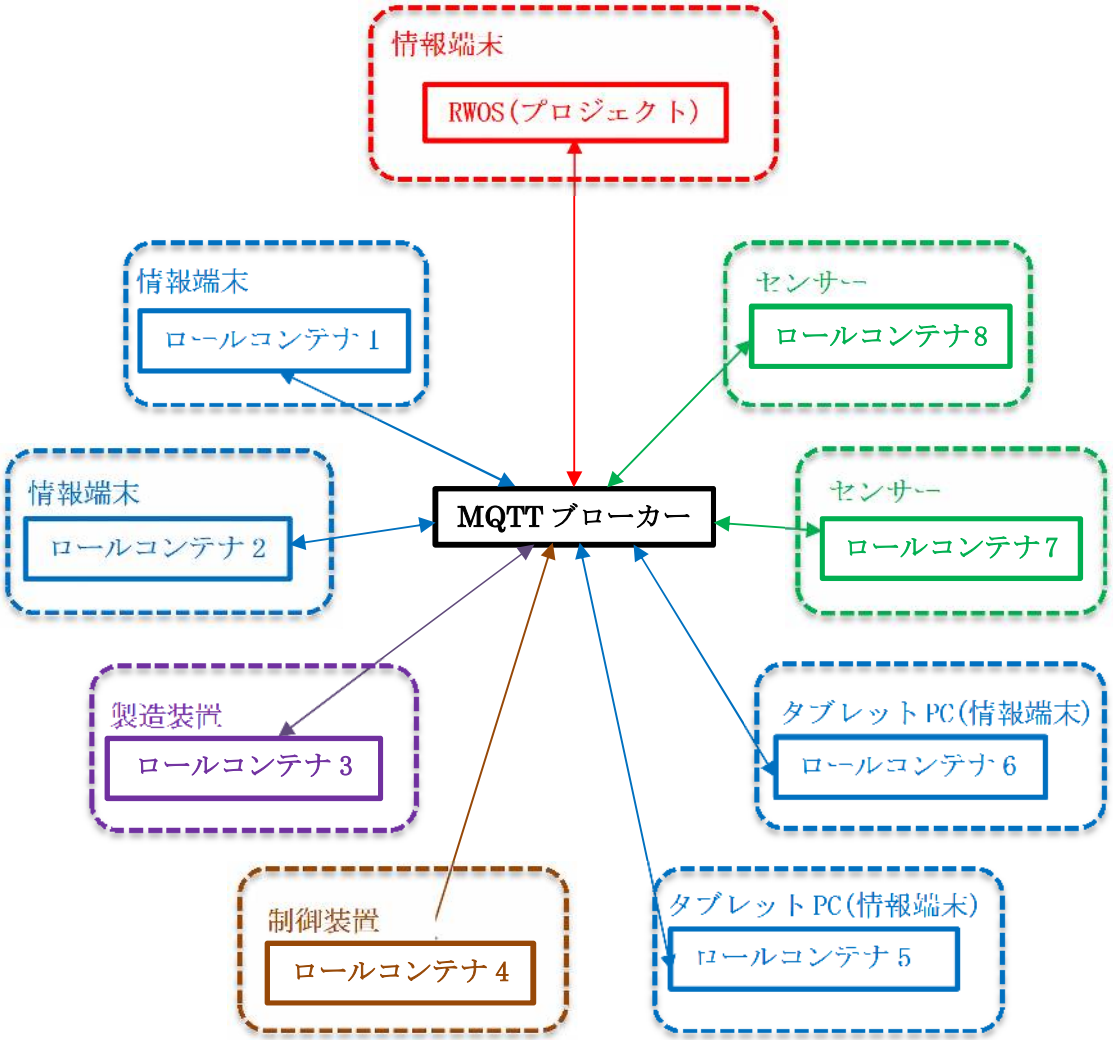


図 2-11 RWOS におけるプロジェクトとロールコンテナ

RWOS は SOARS と同様に処理フロー定義手法としてステージ概念を使用している．RWOS の処理フローは１つ以上のステージの連続として表現される．各ステージでは、各ロールコンテナがそのステージで行うべき命令を一斉に実行するが、それらの命令は全て並列実行可能であるもののみとし、並列実行出来ない命令は別のステージで実行するようにする．このようにステージ概念に基づいて処理フローを設計することにより、全てのロールコンテナの命令が正確な順序で実行される図 2-12 のような仕組みとなっている．RWOS のステージ概念は SOARS のそれよりも進化しており、ステージ自体の並列処理も行えるようになっている．

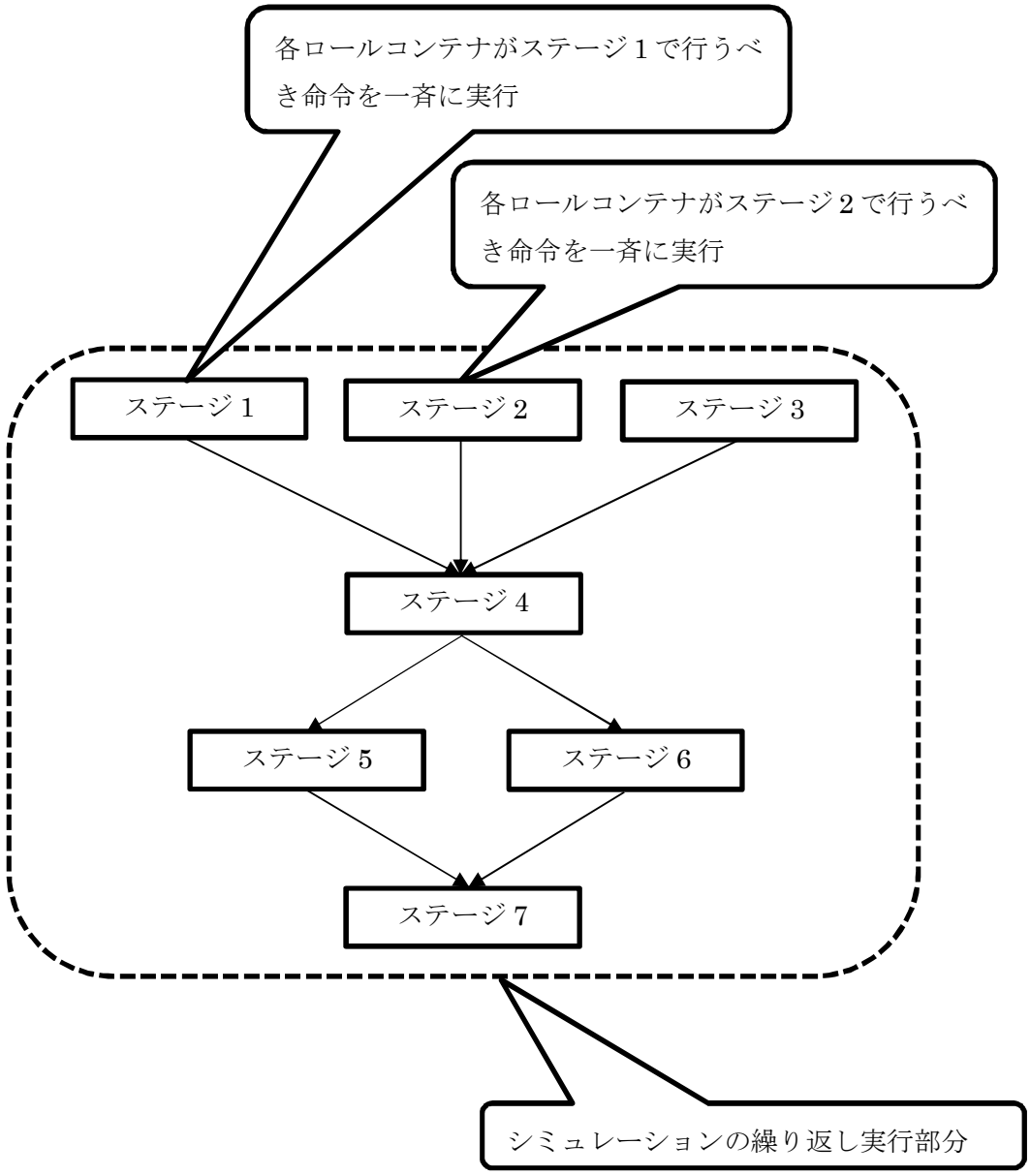


図 2-12 RWOS のステージ概念

また、RWOS では通信プロトコルとして MQTT(Message Queue Telemetry Transport) [MQTT 1999], [松本 2016]を使用している．MQTT は 1999 年に IBM 社と Eurotech 社のメンバにより考案されたプロトコルであり、その標準化はコンピュータと通信に関する標準化団体である OASIS によって行われている．MQTT は TCP/IP プロトコルの上で動作するパブリッシュ／サブスクライブ型のプロトコルを採用し、パブリッシャーとサブスクライバーが MQTT ブローカー(MQTT サーバ)を介して通信を行う仕様になっている．パブリッシャーはトピックを指定して MQTT ブローカーへメッセージを送信し、そのメッセージは、MQTT ブローカーに対してそのトピックでサブスクライブを要求しているサブスクライバーへと配信される．表 2-2 にネットワーク階層モデルにおける MQTT の位置付けを示す[William 1994], [田畑 1994]．図 2-13 は MQTT による通信の仕組みを示したものである．

表 2-2 ネットワーク階層モデルにおける MQTT の位置付け

階層	プロトコル	説明
アプリケーション層	MQTT	ブローカー・ベースの軽量なパブリッシュ／サブスクライブ型メッセージ送受信処理
	HTTP	Web ブラウザと Web サーバの間でのデータ送受信処理
	FTP	ファイル転送処理
	SMTP	メール送信処理
	NFS	ネットワーク・ファイル共有処理
トランスポート層	TCP	TCP パケットの送受信処理及び TCP パケット再送処理
	UDP	単にデータ送受信のみを行う処理 （送受信データのチェックを行わない）
ネットワーク層	IP	IP ルーティング処理と IP パケットの送受信処理
	ICMP	機器間の状態確認処理 （ネットワーク診断プログラム ping で使用される）
	IGMP	機器間のブロードキャスト処理
リンク層	Ethernet	送受信における物理的なハードウェア制御
	ARP	ARP 要求・応答処理

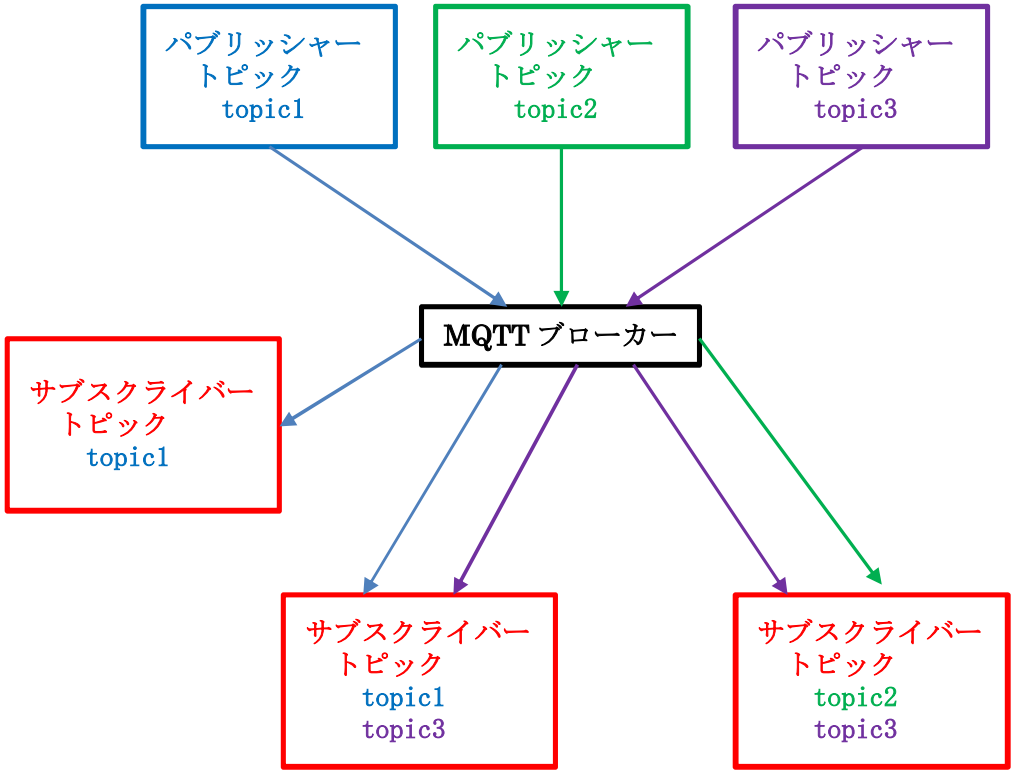


図 2-13 MQTT 通信の仕組み

RWOS では、プロジェクトと各ロールコンテナの間の MQTT 通信で使用する MQTT ブローカー(コントロールブローカー)とロールコンテナ間の MQTT 通信で使用する MQTT ブローカー(データブローカー)を指定する．前者は RWOS 制御用に、後者はロールコンテナ間のデータ交換用に使用される．この２つの MQTT ブローカーは同一のものでも差支えない．

RWOS は、コントロールブローカーを使用することにより、以下の方法で各ステージの開始／終了を制御する．

(1)各ステージ開始時に、プロジェクトがそのステージで命令を実行することになっているロールコンテナへ、命令実行を開始する旨のメッセージをパブリッシュする．

(2)プロジェクトから命令実行開始のメッセージをサブスクライブしたロールコンテナは、命令を実行し、全ての命令実行終了後、プロジェクトへ命令実行が完了した旨のメッセージをパブリッシュする．

(3)ロールコンテナから命令実行完了のメッセージをサブスクライブしたプロジェクトは、そのステージで実行されるべき全ての命令実行が完了するのを待って、そのステージを終了し、次のステージへと遷移する．

RWOS ではこのように実行中のステージが遷移することで処理が進行する．

図 2-14 は RWOS における通信の仕組みを示したものである

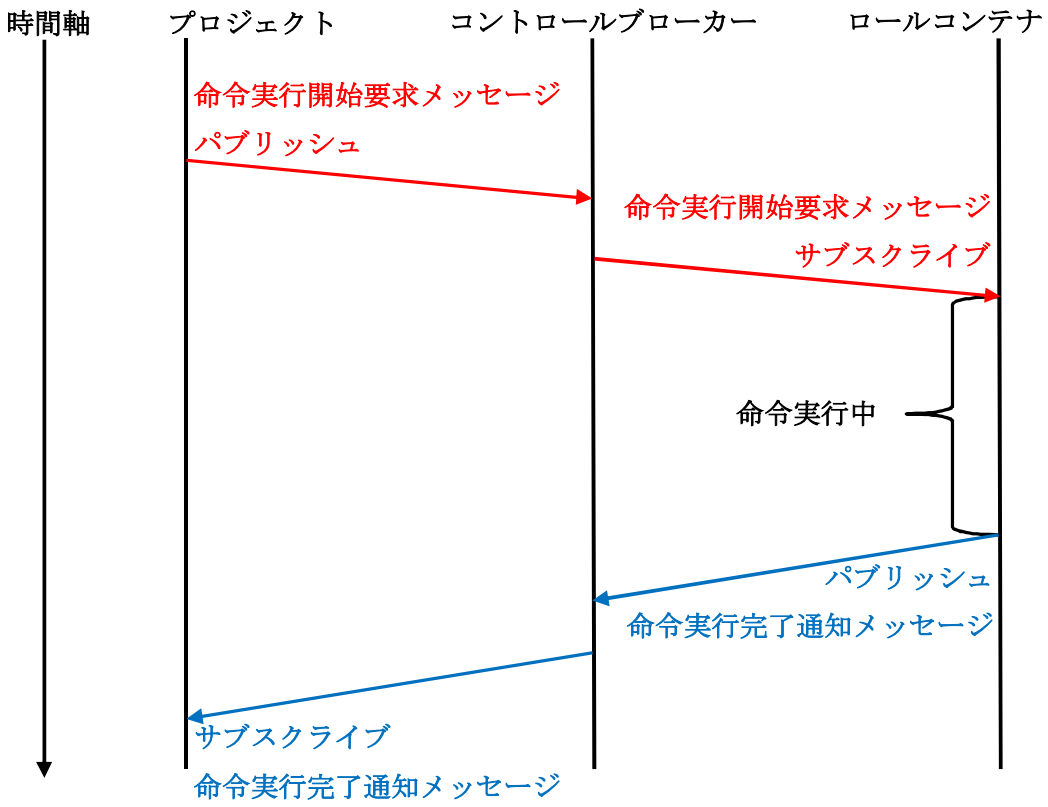


図 2-14 RWOS におけるステージ遷移の仕組み

RWOS は、実世界及び仮想世界の双方を含む空間上のエージェント相互関係を表すモデル構築概念に基づくモデル化手法の為のモデル化手法特化型プログラム言語であり、以下のようにモデリング GUI を実現する為の 9 つの要件を満たしている.

(1)ステージ概念により処理フローを定義出来る

(2)オブジェクトがロールコンテナに限られている

(3)ロールコンテナで実行出来る命令の種類が限られている

(4)プロジェクト内でのロールコンテナ間の依存関係が明確になっている

(5)ロールコンテナの集合を表すオブジェクトを定義出来る

(6)ロールコンテナの振る舞いが限定されている

(7)RWOS には、そのロールコンテナに任意のプログラムを実行出来ると云う拡張性が備わっている

(8)RWOS では、MQTT 通信の時系列変化に基づいてプログラム実行の可視化を行うことが出来る

(9)RWOS により記述されたプログラムは RWOS Project Manager(Java プログラム)によって実際に動作させることが可能である

従って、RWOS によるプログラムを MMDA により開発する為のモデリング GUI を構築することが可能であり、実世界と仮想世界を繋ぐオペレーティングシステムのモデル構築

概念を持つユーザであれば、プログラマではなくても、そのモデリング GUI を利用して実世界及び仮想世界の双方を含む空間上で複雑なやりとりを行うプログラムを作成することが可能となる.

2.3 本研究の目的と実現すべき内容の設定

MMDA は、以下の要件が満たされた場合に実現可能な、MDA に代わる新たなプログラム開発手法である.

(1)モデル記述が広く一般のモデル化手法を対象とせず、特定のモデル構築概念に基づくモデル化手法を対象とすること

(2)UML のような抽象仕様記述言語や情報処理技術者向けの特定領域言語ではない特定領域専門家向けの特定領域言語、即ちモデル化手法特化型プログラム言語によりモデルを記述すること

(3)そのモデル化手法特化型プログラム言語が 2.2 で述べた 9 つの要件を満たしていること

MMDA においては、そのモデル化手法特化型プログラム言語によるプログラムを、テキストエディタではなくモデル化手法特化型プログラム言語に特化した GUI を持つ総合的なプログラム統合開発環境により作成出来る.

SOARS 及び RWOS は、それぞれ役割エージェントベースシミュレーション及び実世界と仮想世界を繋ぐオペレーティングシステムのモデル記述を行うことが出来るモデル化手法特化型プログラム言語である. 特に、その処理フロー定義はステージ概念によって行うことが可能であり、また作成されたプログラムが実際に動作可能である等、モデリング GUI を実現する為の要件を満たしたモデル化手法特化型プログラム言語である. 従って、特定領域専門家が MMDA により SOARS 及び RWOS によるプログラムを作成して実行出来るモデリング GUI を構築することが可能である.

以上を踏まえて、本研究の目的と実現すべき内容を表 2-3 のように設定した.

表 2-3 本研究の目的と実現すべき内容

本研究の目的	実現すべき内容
プログラマではなくても役割エージェントベースシミュレーションのモデルを記述して実行出来るようにする	役割エージェントベースシミュレーションのモデル化手法特化型プログラム言語である SOARS によるプログラムを MMDA により開発する為のモデリング GUI を構築する
プログラマではなくても実世界と仮想世界を繋ぐオペレーティングシステムのモデルを記述して実行出来るようにする	実世界と仮想世界を繋ぐオペレーティングシステムのモデル化手法特化型プログラム言語である RWOS によるプログラムを MMDA により開発する為のモデリング GUI を構築する

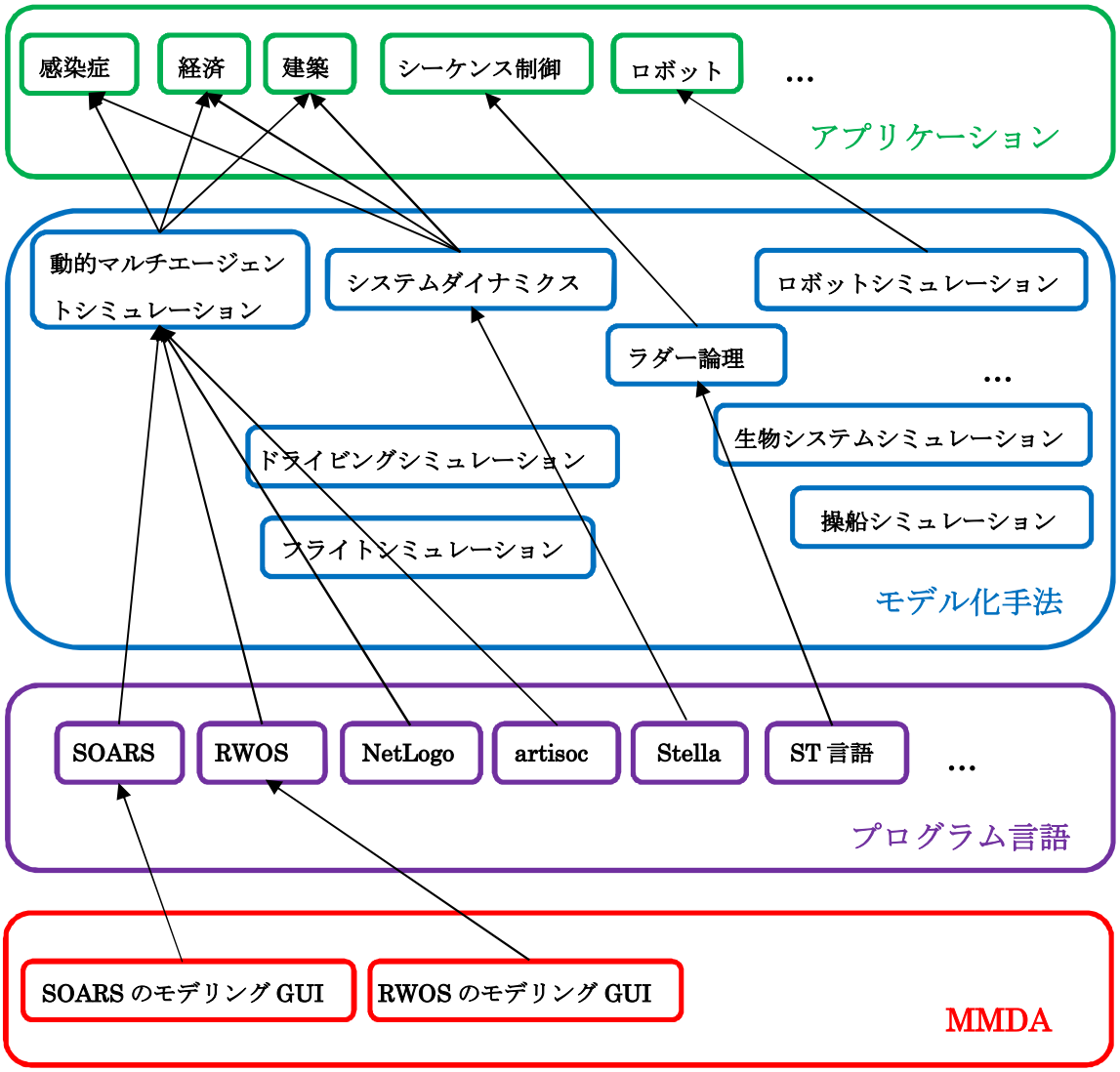


図 2-15 MMDA によるプログラ開発の仕組み

第3章 モデルメソッド駆動型アーキテクチャに基づくモデリング GUI の設計事例

本章では、モデリング GUI の設計原理を述べた後、それに基づいた SOARS 及び RWOS のモデリング GUI 構築設計について述べる．SOARS のモデリング GUI は SOARS VisualShell であり、RWOS のモデリング GUI は RWOS Program Editor である．まず、それぞれのクラス構成の設計について述べる．次に、各クラスについて説明を行い、モデリング GUI を実現する為に実装すべき機能の詳細設計について述べる．

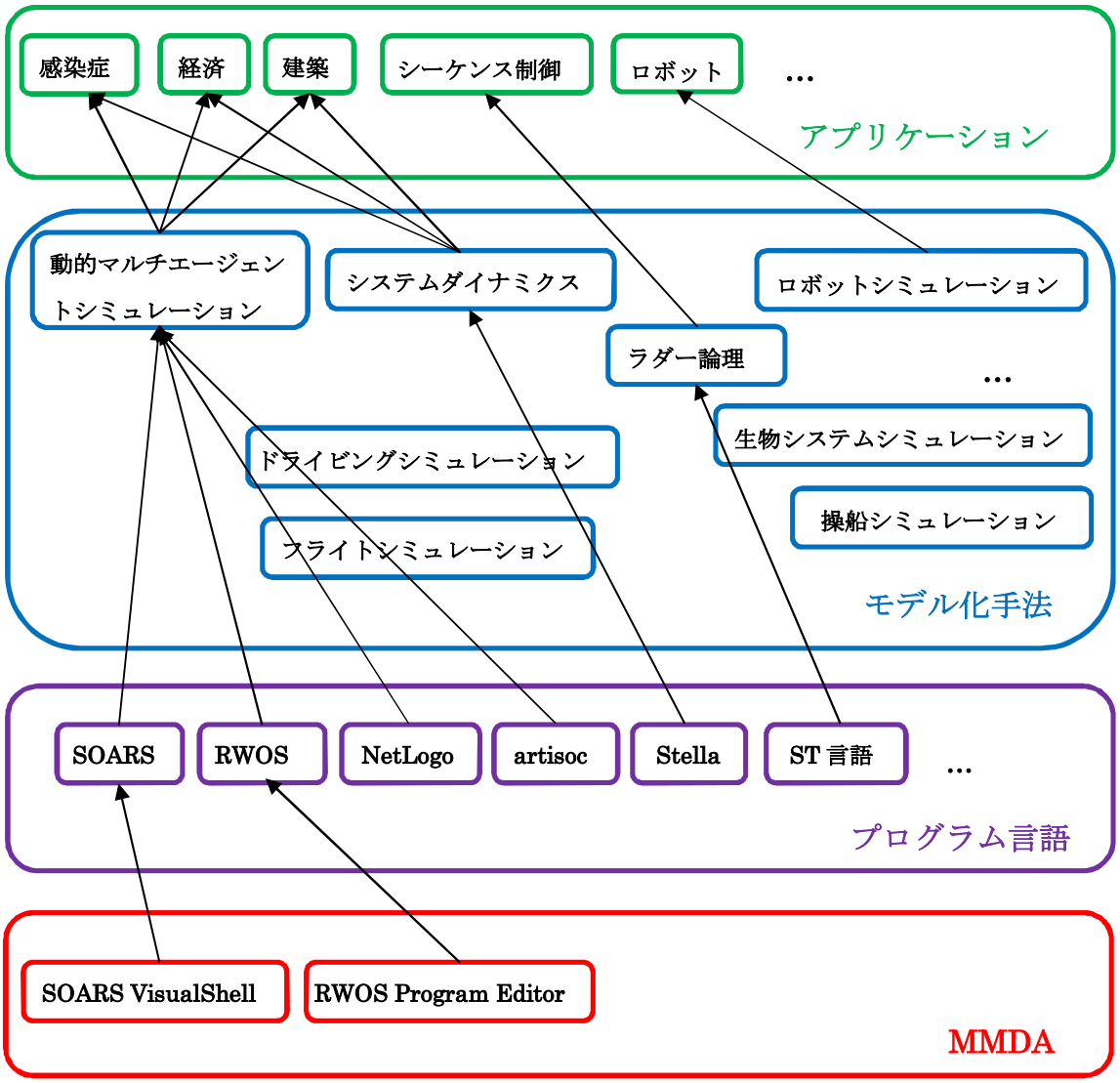


図 3-1 モデリング GUI としての SOARS VisualShel 及び RWOS Program Editor によるプログラム開発の仕組み

3.1 モデリング GUI の設計原理

モデリング GUI は、テキストエディタによるプログラミングを必要としないプログラム開発環境である．よって、その設計・実装は以下の原則に従って行う

(1)プログラム構成要素(命令、変数名及び即値)を指定するのみとする

(2)プログラム構成要素の指定は原則としてマウスにより行う

・コンボボックスやスピンコントロールによる選択

・ドラッグ＆ドロップによる設定

(3)キーボードによる入力はテキストフィールドによる変数名及び即値の定義のみとする

・テキストフィールドに入力された文字列を自動的にチェックしてシンタクスエラーを発生させない

(4)プログラム構成要素を正しく繋ぎ合わせて自動的にプログラム命令文を生成するプログラム生成機能を持つ

(5)生成したプログラムを実際に実行出来るメニューやボタンを持つ

(6)作成したモデルやモジュールを再利用することが可能なデータ構造及び GUI を持つ

(7)モデリング GUI の開発工数を抑えることが可能な実装にする

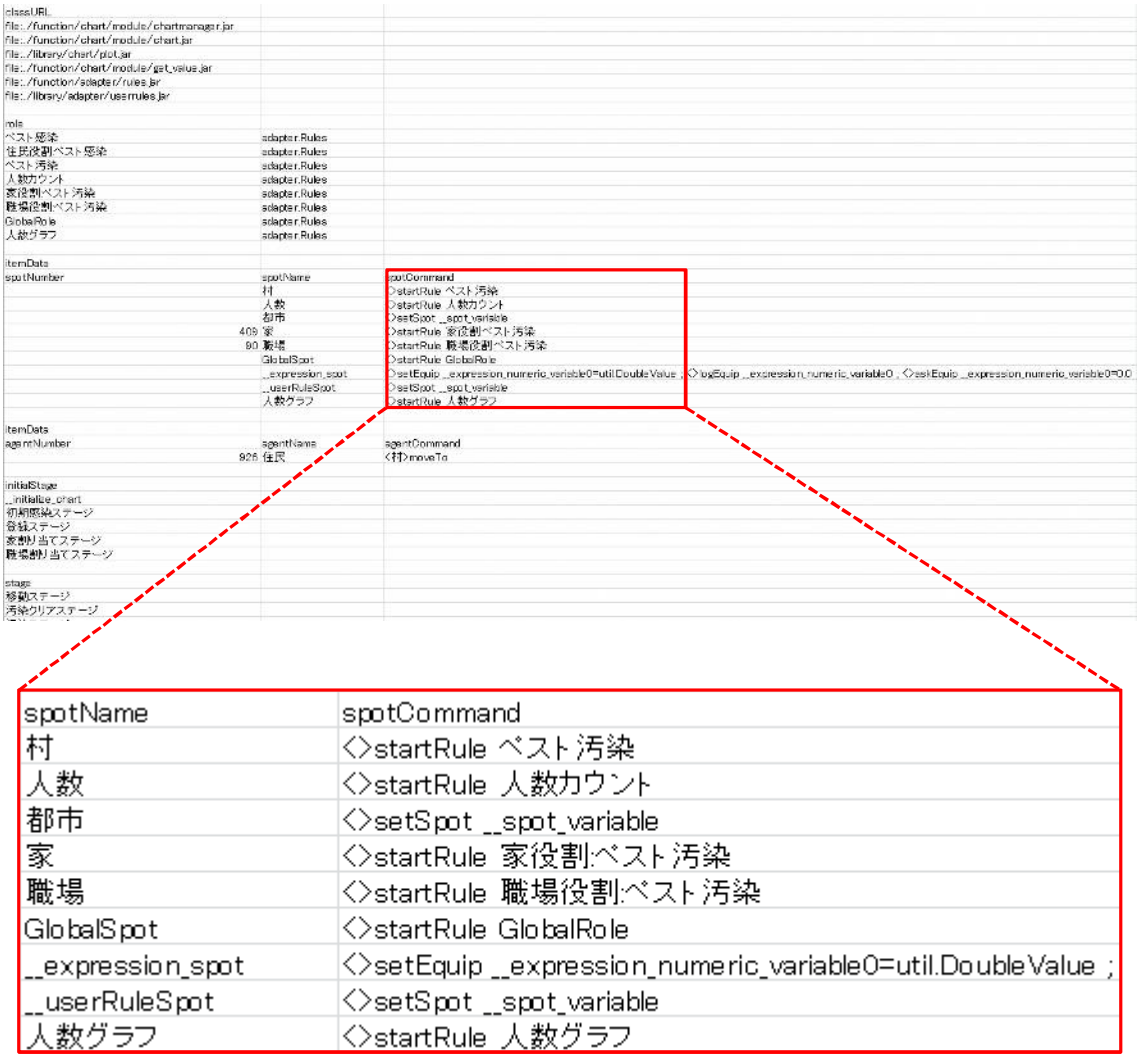
以上の原則に従って、SOARS VisualShell 及び RWOS Program Editor の設計・実装を行った．

3.2 SOARS VisualShell

3.2.1 プログラム言語 SOARS

図 3-2 は SOARS による SIR モデルを用いた感染症流行シミュレーションプログラムの一部である．このプログラムでは、最初に数人のエージェントが感染する．全てのエージェントは、家と職場と云う２つのスポット間を移動する．その間に、確率的に家または職場が汚染され、そこを訪れたエージェントが確率的に感染する．感染したエージェントは一定時間が経過すると恢復または死亡する．感染者数がゼロになった時点でシミュレーションは終了する．シミュレーション実行時には、指定された変数のログが表示され、未感染者数、感染者数及び恢復または死亡者数がチャートで表示される．

SOARS では、この図のようにテキストエディタによるプログラムを作成しなければならない．そこで、このようなプログラムを自動生成する為のモデリング GUI として Java による SOARS VisualShell の設計・実装を行った．



spotName	spotCommand
村	<>startRule ベスト汚染
人数	<>startRule 人数カウント
都市	<>setSpot __spot_variable
家	<>startRule 家役割:ベスト汚染
職場	<>startRule 職場役割:ベスト汚染
GlobalSpot	<>startRule GlobalRole
__expression_spot	<>setEquip __expression_numeric_variable0=util.DoubleValue ;
__userRuleSpot	<>setSpot __spot_variable
人数グラフ	<>startRule 人数グラフ

図 3-2 SOARS による感染流行シミュレーションプログラムの一部

3.2.2 SOARS VisualShell の設計

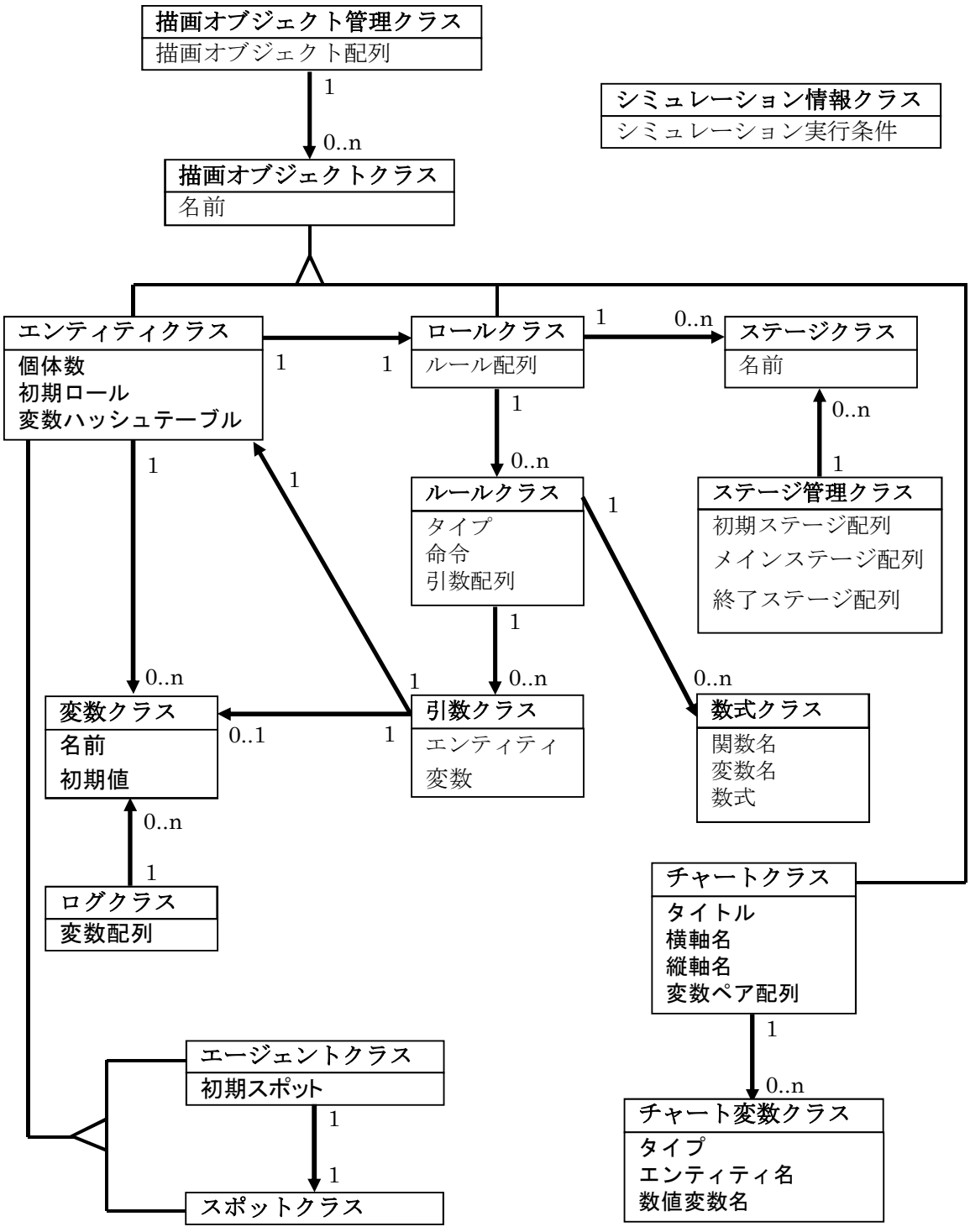


図 3-3 SOARS VisualShell のクラス構成

図 3-3 は SOARS VisualShell のクラス構成を示している．
エンティティ(エージェント／スポット)、ロール及びチャートは、図 3-4 のようにウィンドウへアイコンとして表示される．その為、共通の描画オブジェクトクラスから派生させ、描画オブジェクト管理クラスがそれらを全て保持する．アイコンを新規作成／削除することにより各オブジェクトの新規作成／削除が行える．各アイコンをダブルクリックすることにより、各オブジェクトの編集を行えるものとする．

エンティティは複数の変数及びロールを保持する．ロールは各ステージで実行する複数のルール(命令)を保持し、ルールはルールのタイプ、命令及びその引数を保持する．チャートはエンティティの変数を使用して行うチャート描画に必要な情報を保持する．数式クラスはルールの命令の引数として使用される複数の関数を保持し、ステージクラスはステージ情報を保持し、ステージ管理クラスが全てのステージを保持する．ログクラスはログ出力する変数を保持し、シミュレーション情報クラスはシミュレーション実行条件を保持する．

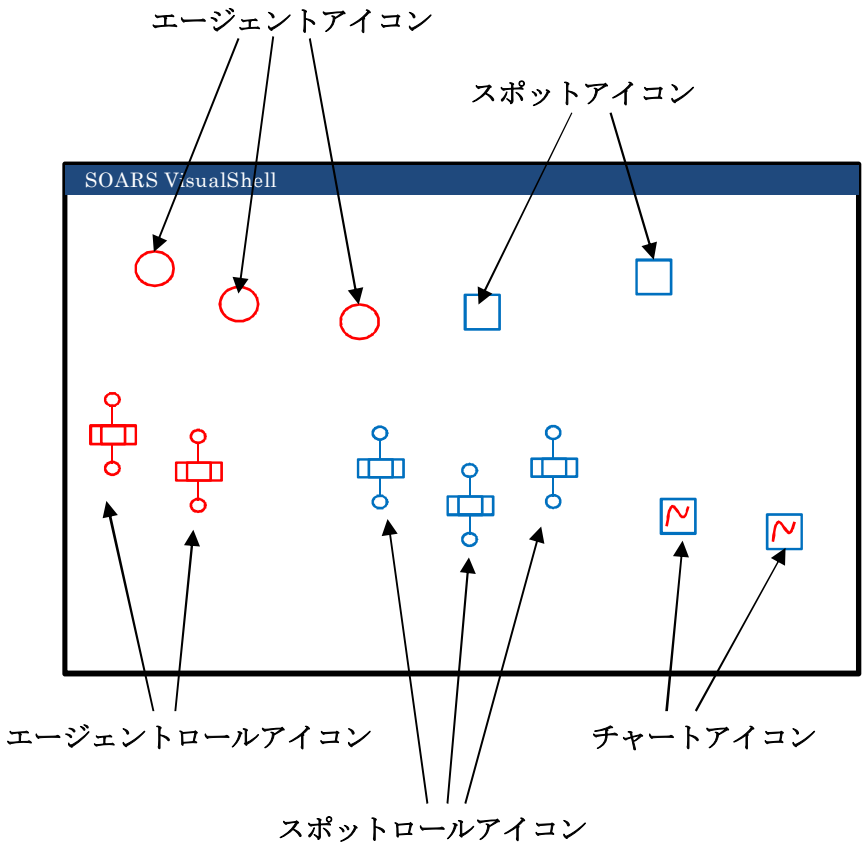


図 3-4 SOARS VisualShell ウィンドウ

3.2.2.1 ステージクラス

ステージクラスは、ステージ名を保持する．

ステージ管理クラスは、ステージクラスのインスタンスを、初期ステージ配列、メインステージ配列、または終了ステージ配列の要素として管理する．各ステージは、配列内の位置がその実行順序となる．従って、モデリング GUI を実現する為には、図 3-5 のようなステージの新規作成、ステージ名変更、ステージ削除、ステージの実行順序変更の為の GUI が必要となる．

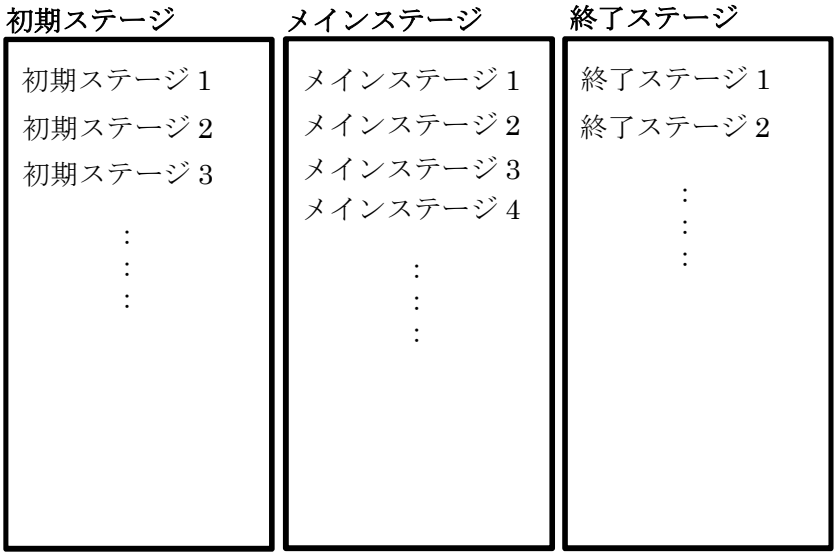


図 3-5 ステージ編集用 GUI

3.2.2.2 エンティティクラス

エンティティクラスは、その名前、個体数、ロール及び複数の変数を保持する．各変数名はユニークなので、エンティティは、自らが保持する変数を、名前をキー、変数クラスのインスタンスを値とするハッシュテーブルによって管理する．描画オブジェクト管理クラスは、エンティティクラスのインスタンスを配列及びその名前をキーとするハッシュテーブルによって管理する．モデリング GUI を実現する為には、エンティティ名変更、個体数変更、コンボボックスによる初期スポット及び初期ロール選択を行える図 3-6 のような GUI が必要となる．また、変数タイプ毎に、変数の新規作成、変数削除、変数名変更、初期値設定を行える図 3-7 のような GUI も必要となる．

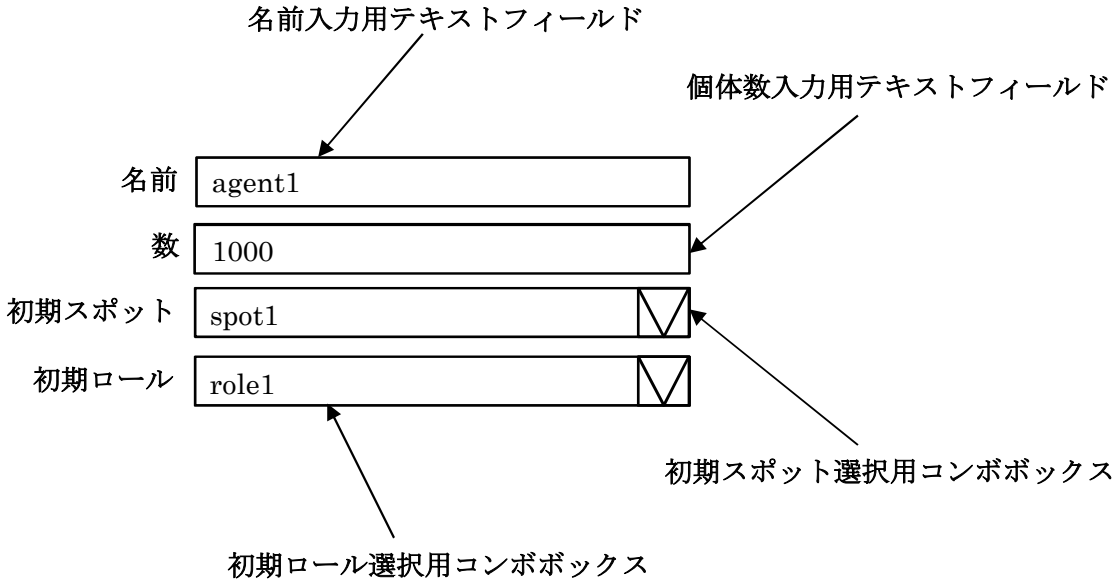


図 3-6 エンティティ編集用 GUI

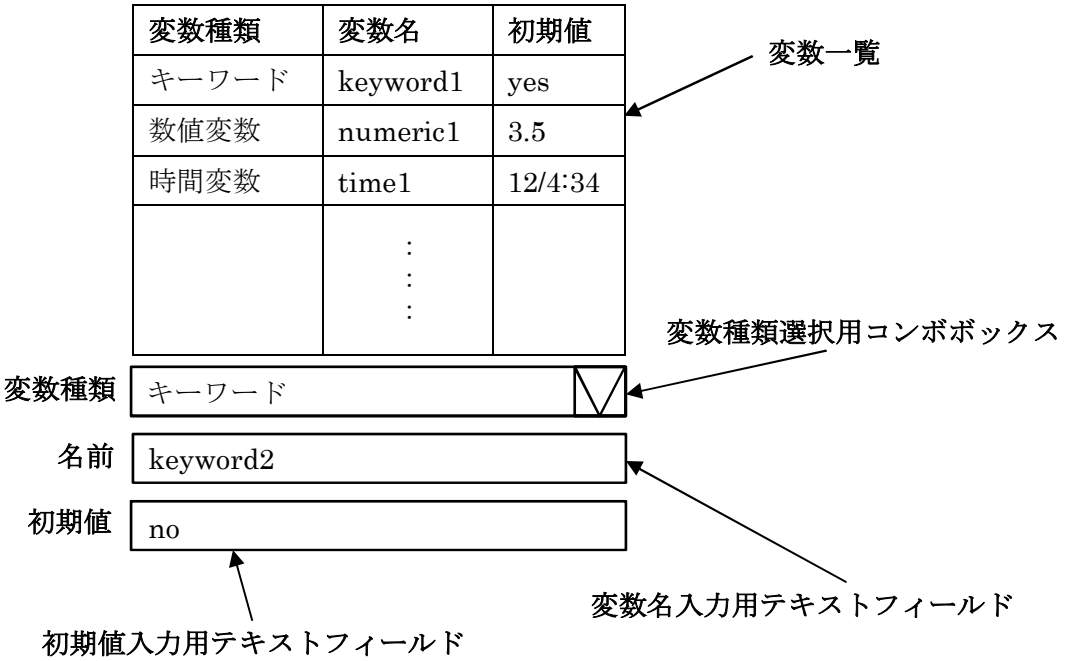


図 3-7 エンティティの変数編集用 GUI

3.2.2.3 変数クラス

変数クラスは変数名及び初期値を保持する．表 3-1 は各変数の内容を示している．

表 3-1 各変数の内容

変数タイプ	内容
キーワード	任意の文字列を保持する
数値変数	任意の整数または実数を保持する
役割変数	ロールクラスのインスタンスへの参照を保持する
時間変数	シミュレーション内の時刻及び時間を保持する
スポット変数	スポットクラスのインスタンスへの参照を保持する
集合変数	任意の変数または即値の集合を保持する
リスト変数	任意の変数または即値のリストを保持する
マップ変数	キーワードまたは文字列をキーとし、任意の変数を値として保持するハッシュテーブル
クラス変数	任意の Java クラスのインスタンスを保持する 全ての Java クラス及び任意の jar ファイルに含まれる任意のクラスを利用出来る シミュレーション実行時には、リフレクションにより、クラスのインスタンス生成及びメソッド実行を行う
ファイル変数	任意のファイルのフルパス(文字列)を保持する クラス変数のファイル処理を行うメソッドで引数として利用する
初期データファイル	指定の CSV 形式で記述された変数初期化用ファイルのフルパス(文字列)を保持する シミュレーション開始時に自動的に読み込まれて、内部記述に従って各変数の初期化を行う

3.2.2.3.1 初期データファイル

大規模なシミュレーションを記述する場合、多くの変数を初期化する必要があるので、タブ区切りの CSV ファイルによって各変数の初期値を設定する．ファイルはいくつでも指定出来るものとする．

文字列は全てタブ区切りで、１行目は２列目からコマンドを、２行目は２列目から変数名を、３列目以降は１列目にエンティティ名、２列目以降に初期値を指定する．

図 3-8 は初期データファイルの例であり、表 3-2 は利用可能なコマンド一覧である．

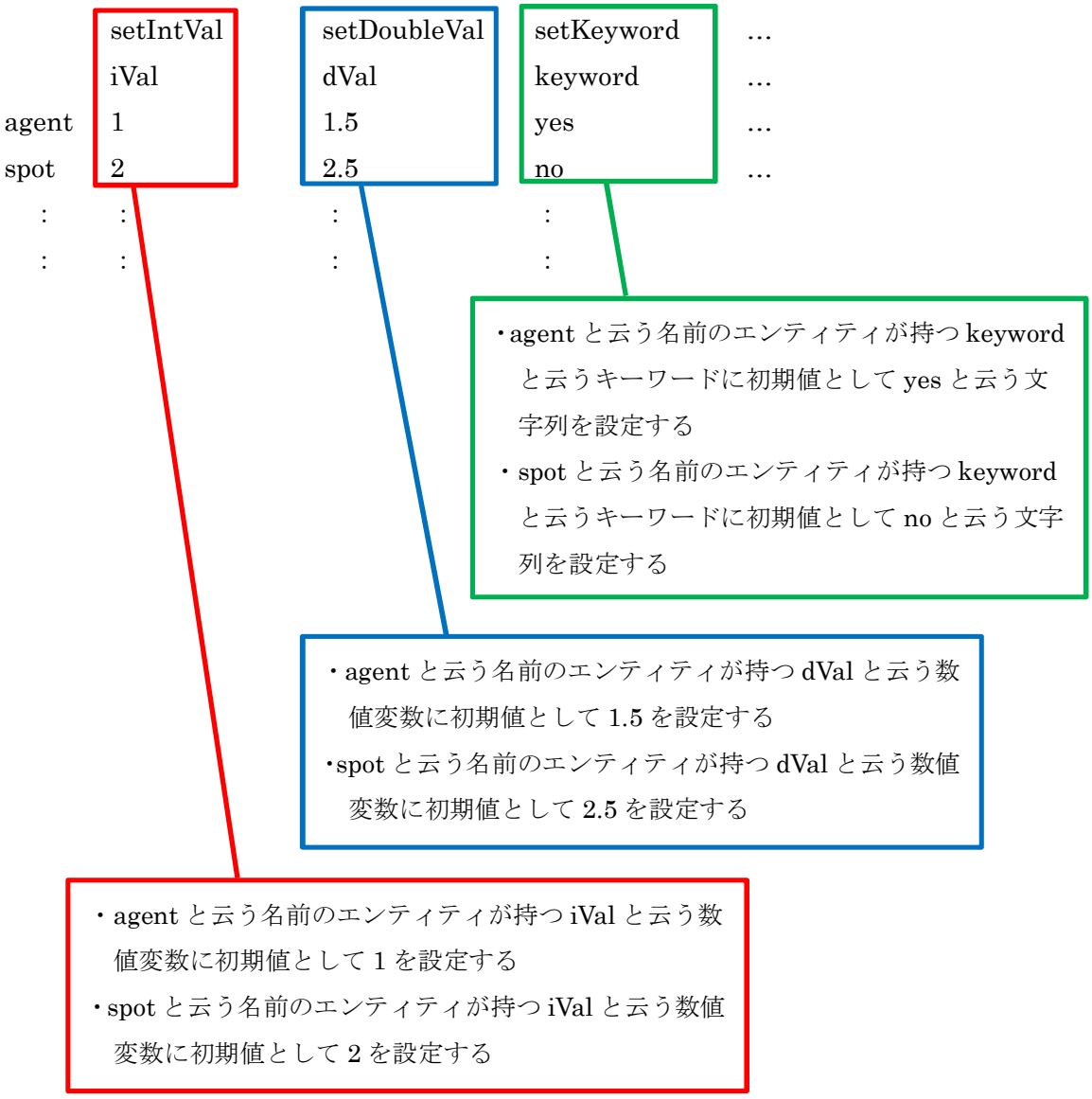


図 3-8 初期データファイルの例

表 3-2 初期データファイルで利用可能なコマンド一覧

コマンド	意味
setIntVal	整数型の数値変数に初期値を設定する
setDoubleVal	実数型の数値変数に初期値を設定する
setKeyword	キーワードに初期値を設定する
setRoleVal	役割変数に初期値を設定する
setTimeVal	時間変数に初期値を設定する
setSpotVal	スポット変数に初期値を設定する
addAgent	集合変数へエージェントを追加する
addSpot	集合変数へスポットを追加する
addKeyword	集合変数へキーワードを追加する
addIntVal	集合変数へ整数型数値変数を追加する
addDoubleVal	集合変数へ実数型数値変数を追加する
addRoleVal	集合変数へ役割変数を追加する
addTimeVal	集合変数へ時間変数を追加する
addSpotVal	集合変数へスポット変数を追加する
addMapVal	集合変数へマップ変数を追加する
addColVal	集合変数へ集合変数を追加する
addListVal	集合変数へリスト変数を追加する
addFirstAgent	リスト変数の先頭にエージェントを追加する
addFirstSpot	リスト変数の先頭にスポットを追加する
addFirstKeyword	リスト変数の先頭にキーワードを追加する
addFirstIntVal	リスト変数の先頭に整数型数値変数を追加する
addFirstDoubleVal	リスト変数の先頭に実数型数値変数を追加する
addFirstRoleVal	リスト変数の先頭に役割変数を追加する
addFirstTimeVal	リスト変数の先頭に時間変数を追加する
addFirstSpotVal	リスト変数の先頭にスポット変数を追加する
addFirstMapVal	リスト変数の先頭にマップ変数を追加する
addFirstColVal	リスト変数の先頭に集合変数を追加する
addFirstListVal	リスト変数の先頭にリスト変数を追加する
addLastAgent	リスト変数の最後にエージェントを追加する
addLastSpot	リスト変数の最後にスポットを追加する
addLastKeyword	リスト変数の最後にキーワードを追加する
addLastIntVal	リスト変数の最後に整数型数値変数を追加する

addLastDoubleVal	リスト変数の最後に実数型数値変数を追加する
addLastRoleVal	リスト変数の最後に役割変数を追加する
addLastTimeVal	リスト変数の最後に時間変数を追加する
addLastSpotVal	リスト変数の最後にスポット変数を追加する
addLastMapVal	リスト変数の最後にマップ変数を追加する
addLastColVal	リスト変数の最後に集合変数を追加する
addLastListVal	リスト変数の最後にリスト変数を追加する

3.2.2.4 チャートクラス及びチャート変数クラス

チャートクラスはプログラム実行時に、エンティティの変数を使用して行うチャート描画に必要な情報を保持する．その情報は、名前、タイトル、横軸名、縦軸名及び横軸と縦軸のチャート変数ペアの配列である．チャート変数クラスは、タイプ、エンティティ名及びエンティティが持つ数値変数名を保持する．タイプは、ステップ(時間)、エージェントの数値変数またはスポットの数値変数のいずれかを指定する．モデリング GUI を実現する為には、図 3-9 及び図 3-10 のような GUI により、これらの情報を指定する．

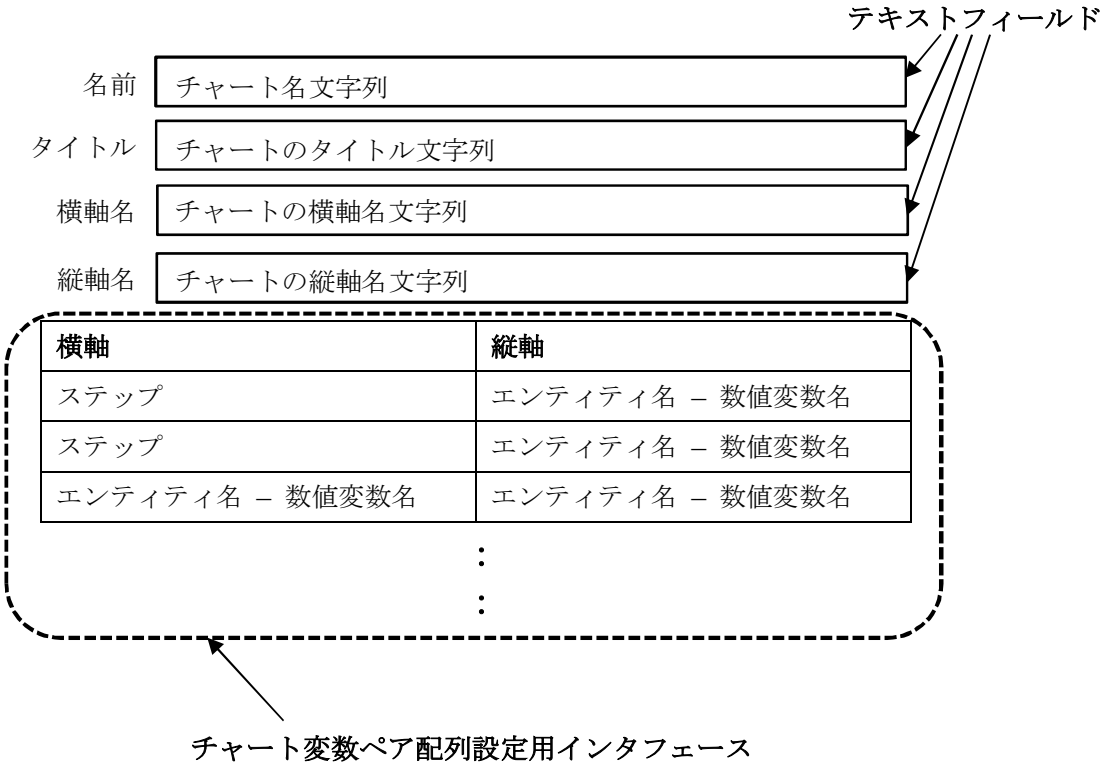


図 3-9 チャート情報編集用 GUI

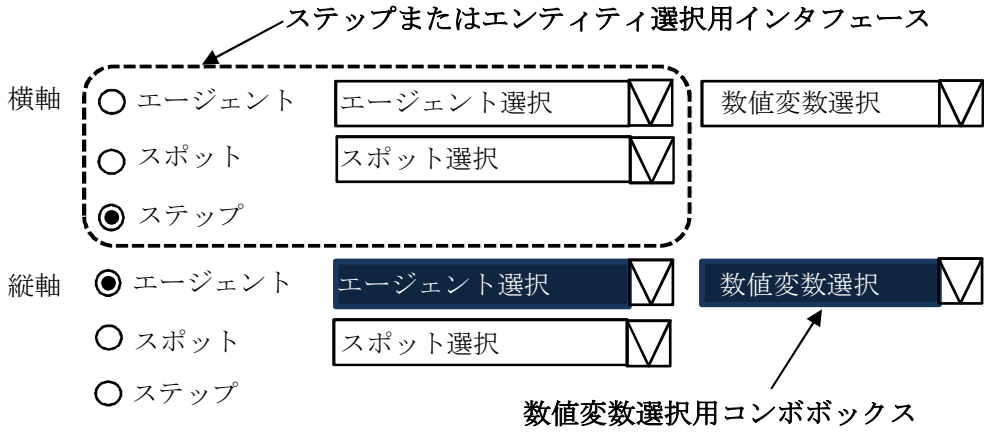


図 3-10 チャート変数ペア編集用 GUI

3.2.2.5 ルールクラス

ルールクラスは、ルールのタイプ、実行すべき命令、その命令を実行するエンティティ、命令を実行する為に必要な引数(エンティティの変数または即値)を保持する．モデリング GUI を実現する為には、命令、エンティティ及び引数をコンボボックスから選択することによってルールの編集を行える GUI が必要となる．

3.2.2.6 数式クラス

ルールで実行すべき命令の引数には数値計算結果が含まれる．SOARS VisualShell では、数値計算を全て関数により行うものとする．数式クラスは、関数をハッシュテーブルにより管理する．モデリング GUI を実現する為には、関数名、変数名及び数式をテキストフィールドにより入力出来る図 3-11 のような GUI が必要となる．数式テキストフィールドでは Java の Math クラスの全てのメソッドが使用出来るものとし、数式の構文チェックも自動的に行われるものとする．

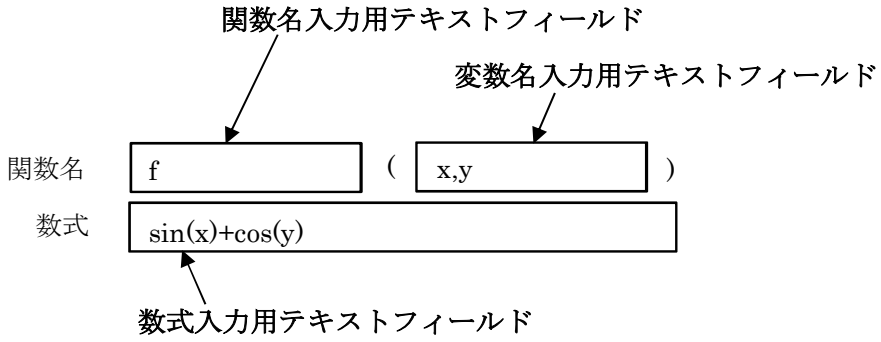


図 3-11 関数定義用 GUI

3.2.2.7 ロールクラス

ロールクラスはエンティティが実行するルールの集合であり、どのステージでどのルールを実行するのかと云う情報を保持する．ロールクラスのインスタンスは配列及び名前をキーとするハッシュテーブルによって管理される．モデリング GUI を実現する為に、ロールクラスの編集は図 3-12 のようなスプレッドシートにより行う．スプレッドシート上で選択したセルのルールをルール編集用 GUI で編集する．但し、ステージ名は、スプレッドシート上で、コンボボックスから選択出来るものとする．

この GUI 仕様に合わせて、ロールクラスのデータ構造は、複数の配列とし、１つの配列は先頭がステージ名で、２番目以降の要素がルールであるスプレッドシート上での１行を表すものとする．配列先頭のステージ名は、２番目以降のルールが実行されるステージである．図 3-13 はロールクラスのデータ構造を示したものである．

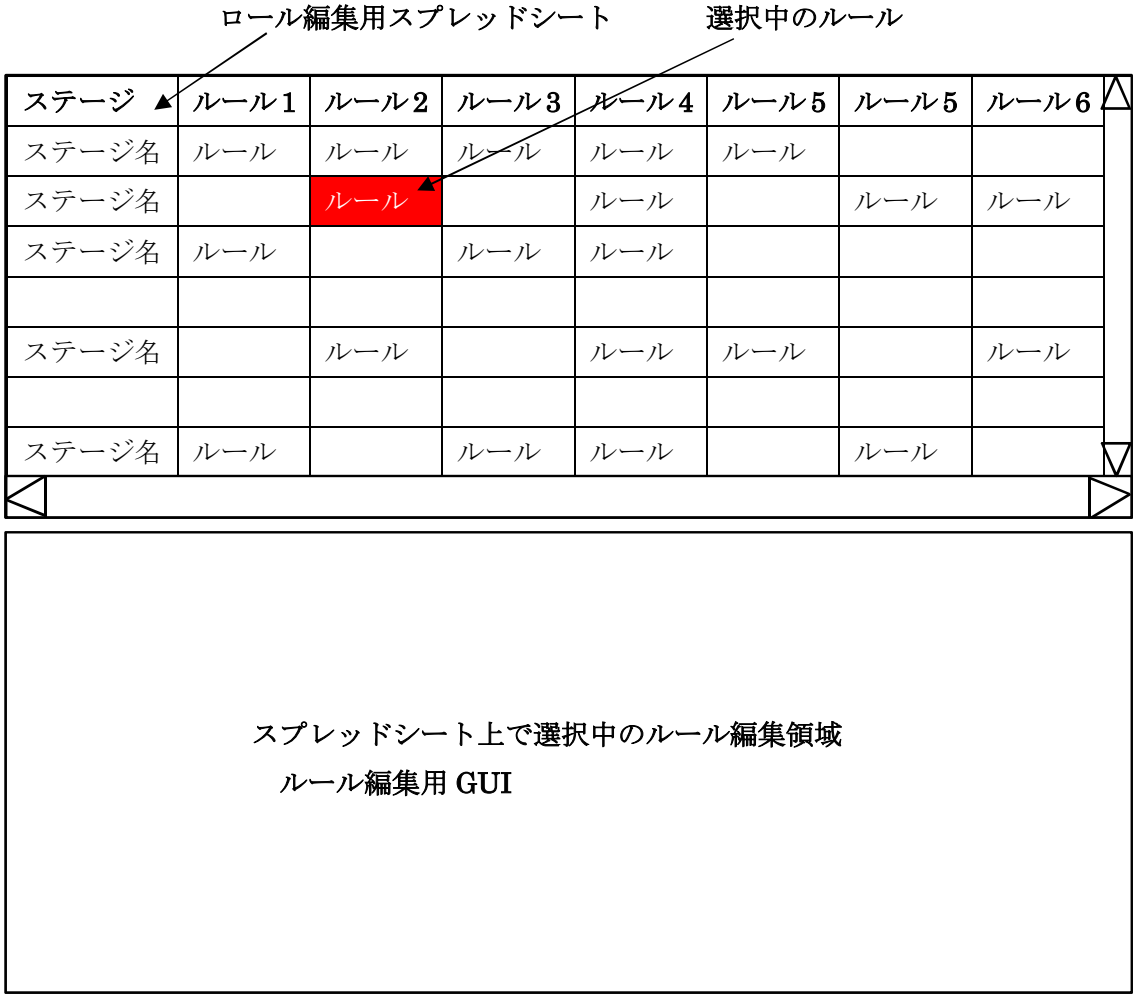


図 3-12 ロール編集用 GUI

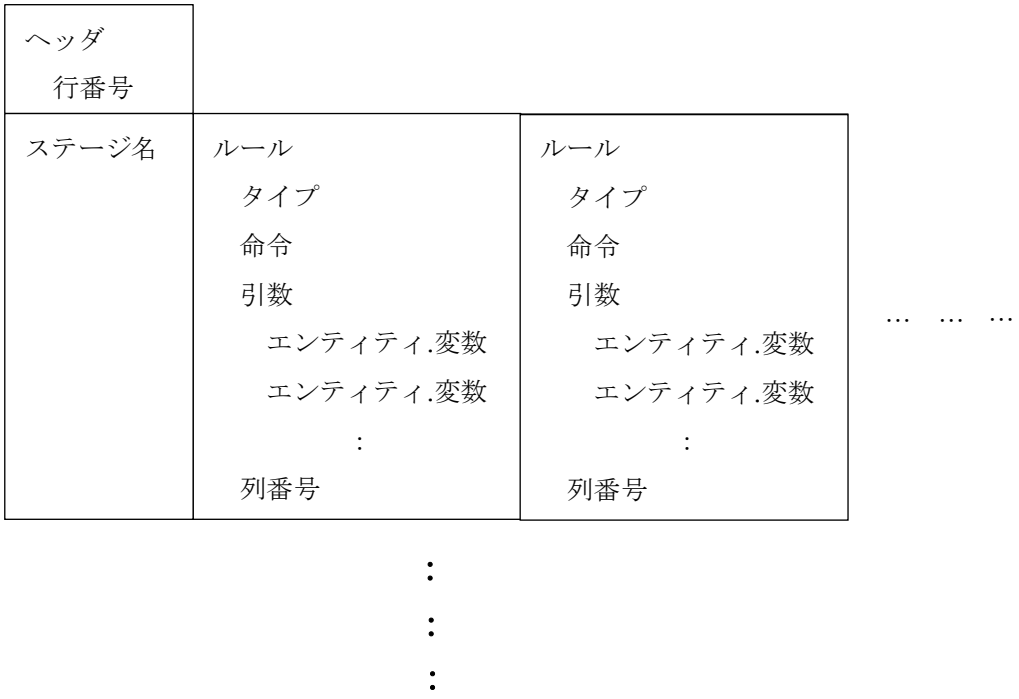


図 3-13 ロールクラスのデータ構造

3.2.2.7.1 ルール編集の GUI

ロールでは、スプレッドシートの各セルのルールを図 3-14 のような GUI により編集出来る機能が必要となる．ルールタイプ選択ツリーからルールタイプを選択し、命令選択用コンボボックスから命令を選択する．

命令の引数はエンティティ選択用インタフェース、変数タイプ選択用コンボボックス、変数選択用コンボボックス及び即値入力用テキストフィールドにより指定する．エンティティ選択用インタフェースの「自分」はこのルールを含むロールを選択しているエンティティを、「現在のスポット」はこのルールを含むロールを選択しているエージェントが存在しているスポットを表す．選択用コンボボックス及び変数タイプは分かり易い説明文である必要がある．

命令の引数が関数である場合は、関数選択用コンボボックスから関数を選択し、関数の変数設定用インタフェースにより、関数の変数にエンティティの変数または即値を指定する．

モデリング GUI ではこのようにテキストエディタを使用せずに命令文を生成することが出来る．

しかし、ルールは頻繁に変更されることが予想される．新たなルールを追加したり、ルールの引数を変更する等の必要が発生した場合、SOARS VisualShell を改修することは非常に大きな負担となり、また利用者もすぐに新しい機能を使うことが出来ない．

そこで、ルール編集画面を定義する為の新たなスクリプト言語を作成し、SOARS VisualShell にはこの言語で記述されたスクリプトを解釈して GUI 画面を生成する機能を実装する． このスクリプト言語の書式は、タグに続いてカンマ区切りで記述するものとする．表 3-3 はこの書式の仕様を、表 3-4 は引数に指定出来る変数タイプ予約語を示している．

この機能を実装することにより、ルールの変更を全てユーザが行うことが出来るようになる．

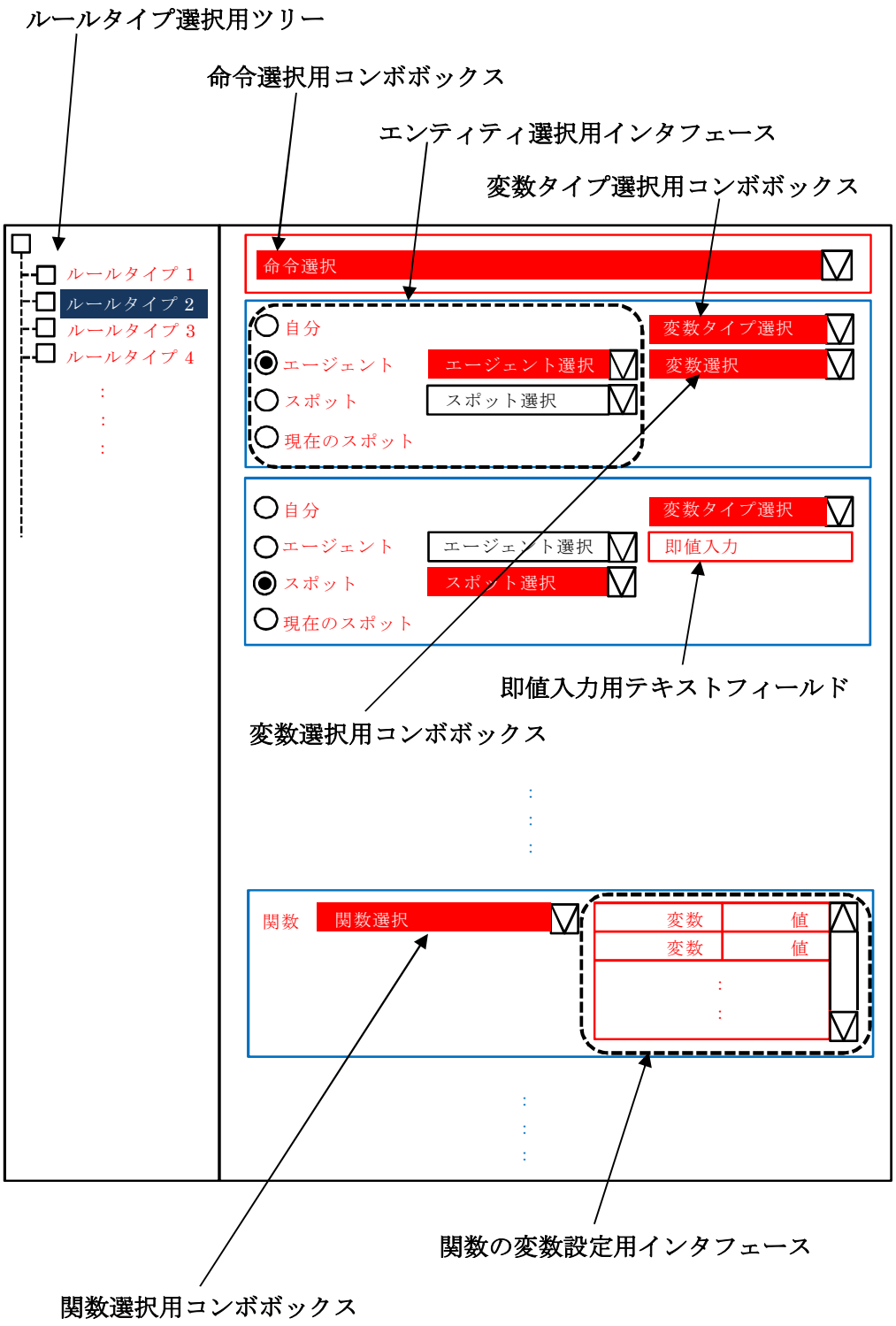


図 3-14 ルール編集用 GUI

表 3-3 ルール編集画面定義用スクリプト言語仕様

タグ	書式	意味
type:	type:[ルールタイプ ID]	ルールタイプを表すユニークな文字列を指定する
title:	title:[ルールタイプ名]	ルールタイプ選択ツリーに表示されるルールタイプ名を指定する
object:	self、agent、spot、currentspot、spotvariable のうち１つ以上をカンマで区切って記述する 例)object:self,agent	引数として使用する変数を持つエンティティを指定する
expression:	self、agent、spot、currentspot、spotvariable のうち１つ以上をカンマで区切って記述する 例)expression:self,agent	引数に関数を指定する場合、関数の変数として使用する変数を持つエンティティを指定する
variable:	variable:[変数タイプ予約語],[変数タイプの説明]	引数に指定出来る変数タイプ予約語と変数タイプ選択用コンボボックス上での文字列を指定する
verb:	verb:[命令 ID],[インデクス],[コマンド],[命令の説明] または verb:[命令 ID],[インデクス],[クラス名:メソッド名],[命令の説明]	・命令 ID には、命令を表すユニークな文字列を指定する ・インデクスには、命令選択用コンボボックスでの表示位置を指定する ・コマンドには、実際にプログラムを実行する際の SOARS コマンドを指定する ・クラス名:メソッド名には、実際にプログラムを実行する際の Java クラス名とメソッド名を指定する ・命令の説明には、命令選択用コンボボックス上での文字列を指定する

表 3-4 ルール編集画面定義用スクリプトの引数に指定出来る変数タイプ予約語

予約語	意味
keyword	キーワード選択
number object	数値変数選択
role variable	役割変数選択
time variable	時間変数選択
spot variable	スポット変数選択
class variable	クラス変数選択
file	ファイル変数選択
map	マップ変数選択
collection	集合変数選択
list	リスト変数選択
integer number object	整数型数値変数選択
real number object	実数型数値変数選択
agent	エージェント選択
spot	スポット選択
role	役割選択
agent role	エージェント役割選択
spot role	集合変数へリスト変数を追加する
stage	ステージ選択
immediate keyword	キーワード即値入力
immediate integer	整数即値入力
immediate real number	実数即値入力
immediate time variable	時間即値入力
immediate data	任意文字列入力

以下の例は、SOARS コマンド setKeyword を使用して、キーワードに他のキーワードまたは即値を代入するルール編集画面定義スクリプトである.

```
type:set1
title:変数設定 1
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード
variable:immediate keyword,文字列
verb:id1,1,setKeyword,キーワードに代入
```

以下の例は、関数による計算結果を数値変数へ代入するルール編集画面定義スクリプトである.

type:set1
title:変数設定 1
object:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
expression:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
variable:immediate integer,整数値
variable:immediate real number,実数値
verb:numeric.expression.substitution,4,,数値変数に計算値を代入

以下の例は、soars.library.adapter.pubsub.PubSub クラスの initialize メソッドを使用して、MQTT 通信の初期化を行うルール編集画面定義スクリプトである.

type:pubsub
title:Publish/Subscribe
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[ブローカー],empty
object:self,agent,spot,currentspot,spotvariable
variable:integer number object,数値変数[ポート番号],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[クライアント ID],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[ユーザ名],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[パスワード],empty
verb:id1,1,soars.library.adapter.pubsub.PubSub:initialize,指定された接続パラメータでサーバに接続し、バッファリング可能な新しいセッションを開始する

3.2.2.8 ログクラス

ログクラスは、プログラム実行時にログ出力する変数を配列で保持する.
従って、モデリング GUI を実現する為には、画面に変数一覧を表示し、チェックボックスによりログ出力の有無を指定出来る図 3-15 のような GUI が必要となる.

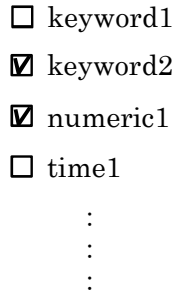


図 3-15 ログ出力指定用 GUI

3.2.2.9 シミュレーション条件クラス

シミュレーション条件クラスは、シミュレーションの開始時刻、実行間隔及び終了時刻を保持する. SOARS のシミュレーション時間は、仮想時間となっており、“日”(0 以上の整数)、“時”(0 以上 23 以下の整数)、“分”(0 以上 59 以下の整数)によって指定する仕様になっている.
従って、モデリング GUI を実現する為には、シミュレーションの開始時刻、実行間隔及び終了時刻それぞれについて、“日”を数値入力用テキストフィールドにより、“時”及び“分”をコンボボックスにより指定出来る図 3-16 のような GUI が必要となる.

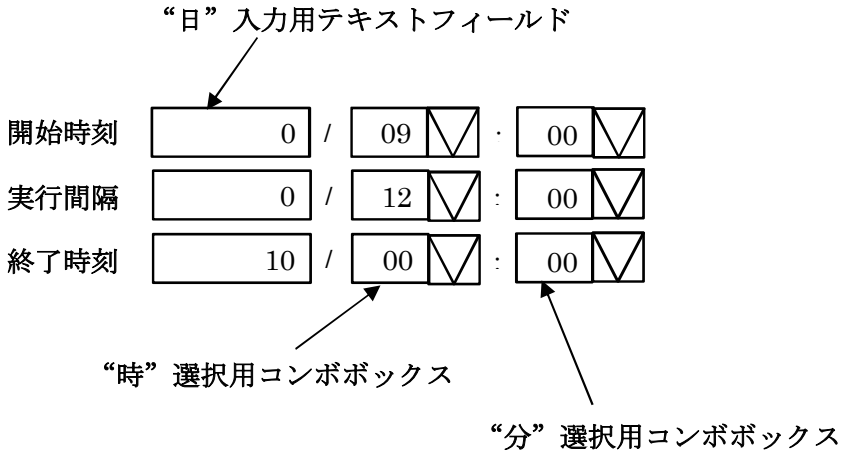


図 3-16 シミュレーション条件設定用 GUI

3.2.2.10 データファイルフォーマット

モデリング GUI を実現する為には、ファイル変数及び初期データファイルに必要なファイルを直接編集出来る機能、またドラッグ＆ドロップによって外部からファイルをインポート出来る機能を実装する必要がある．更に、これを可能にする為に、SOARS VisualShell のデータファイルを圧縮形式とし、ファイル変数及び初期データファイルに必要なファイルを、図 3-17 のような内部の独自ファイルシステムに持たせることも必要となる．

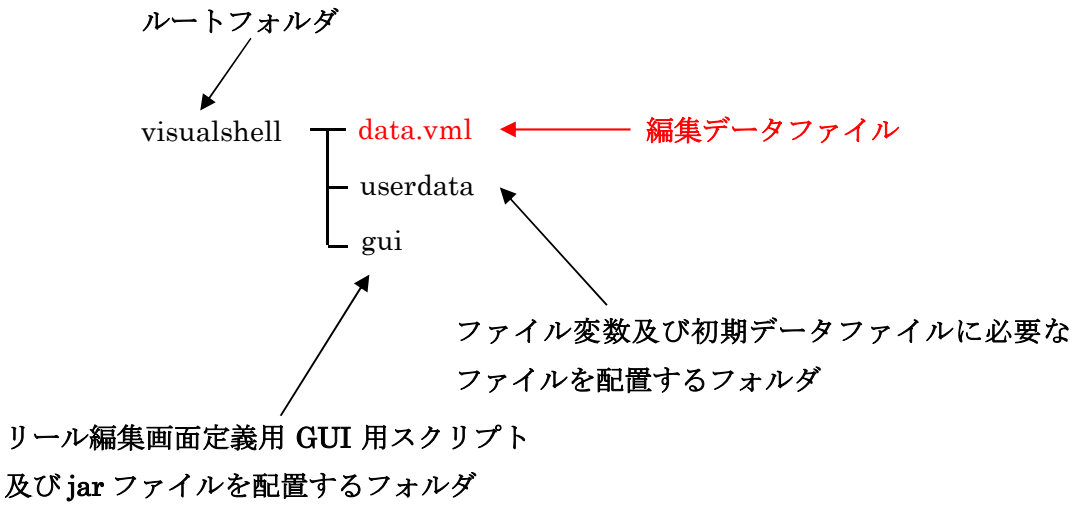


図 3-17 SOARS VisualShell データファイル内部のファイルシステム

3.2.2.11 シミュレーションプロセス起動

SOARS VisualShell では、ボタンを押すだけで、そのモデリング GUI により入力された全てのデータから SOARS プログラムを自動生成し、SOARS 実行プログラムである SOARS Simulator にこれを実行させる．この時、SOARS VisualShell は、一時的な作業フォルダを作成してそこに SOARS プログラムファイルを配置し、また、SOARS プログラムが参照しているファイル(ファイル変数が参照しているファイル、初期データファイル、jar ファイル)も全てそのフォルダ以下へ配置する．

3.2.2.12 シミュレーションログの可視化

SOARS プログラムが出力したログファイルを使用してアニメーションを行う SOARS Animator を開発する．SOARS Animator は、シミュレーション終了後に、SOARS Simulator からボタンを押すだけで起動出来るものとする．

SOARS Animator は、各エージェントのスポット間移動記録をログファイルから読み出し、２次元上に各スポットを表すアイコンを配置し、各エージェントを表すアイコンがスポットアイコン間を移動するアニメーションによって、その移動状況を可視化する．また、エンティティの持つ１つの変数の値変化をログファイルから読み出し、２次元上にその変数値集合の各要素を表すアイコンを配置し、各エンティティを表すアイコンが変数値アイ

コン間を移動するアニメーションによって、その変化状況を可視化する.

従って、SOARS Animator は、複数のアニメーションウィンドウを持つ図 3-18 のようなマルチドキュメントインタフェース(MDI)アプリケーションとし、各ウィンドウ内のアニメーションは正しく同期して実行されるものとする.

ここで、SOARS のログファイルが巨大である場合、全てのログファイルをメモリ上へ読み込むことが出来なくなるケースが考えられる. そこで、SOARS Animator は、各ログファイルから常にアニメーションに必要な部分のログだけをメモリ上へ読み込むことによってこの問題を解決する. 即ち、ログファイル読み込みスレッド、描画スレッド及びユーザインタフェーススレッドと云う 3 つのスレッドが互いに調和して動作する仕組みを実装する.

更に、各エンティティが保持する 3 つの数値変数の値から 3 次元座標を生成し、その 3 次元座標集合を Java FX により 3DCG として可視化を行う SOARS LogViewer の実装も行う.

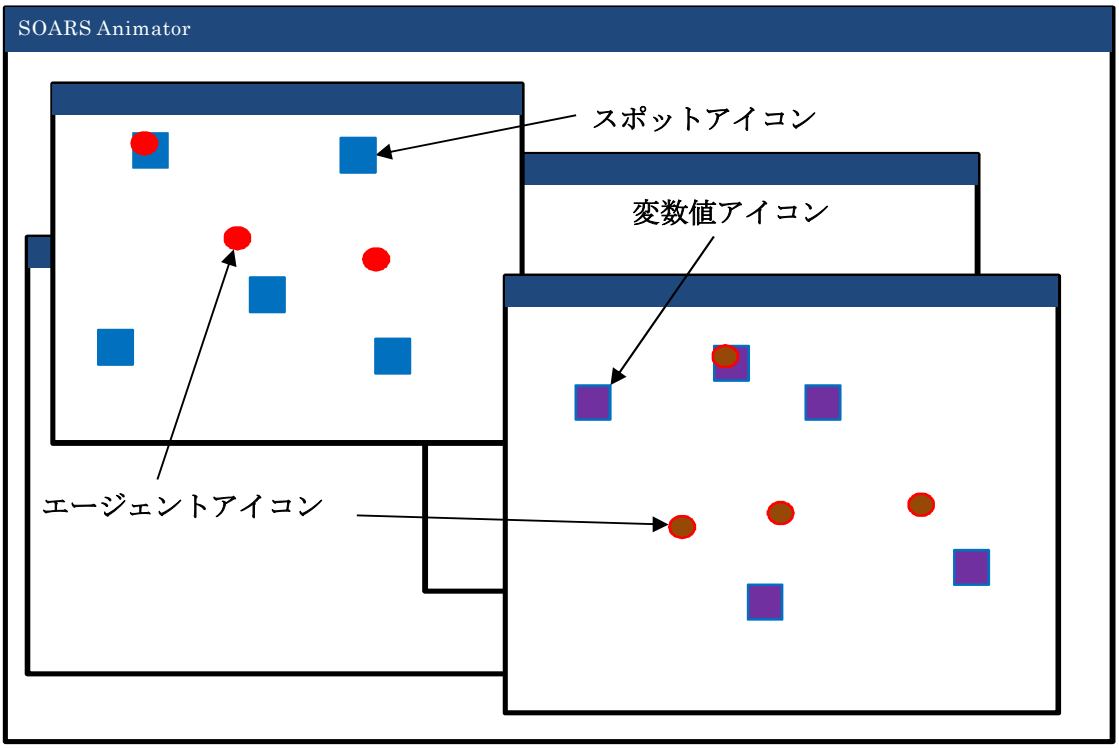


図 3-18 SOARS Animator ウィンドウ

3.3 RWOS Program Editor

3.3.1 プログラム言語 RWOS

図 3-19～図 3-24 は RWOS により記述されたプログラムである．このプログラムは、温度センサーからのデータ及び Web ブラウザからのデータに基づいてパトライト[パトライト 1947]点灯色を決定して点灯させると云うもので、以下の処理を行っている．

(1)sensor1、sensor2、human と云う３つのステージで、それぞれロールコンテナ sensor1 の命令 getSensorData、ロールコンテナ sensor2 の命令 getSensorData、ロールコンテナ human の命令 getHumanData を実行する．これらの命令は並列に実行されて差支えない．命令 getSensorData は温度センサーから JSON フォーマット[RFC4627 2007]でパブリッシュされたメッセージ(温度データ)をサブスクライブし、json2csv.py と云う Python プログラムで CSV フォーマットへ変換してパブリッシュする．命令 getHumanData は Web ブラウザから JSON フォーマットでパブリッシュされたメッセージ(パトライト点灯色)を同様に Python プログラムで CSV フォーマットへ変換してパブリッシュする．

(2)sensor1、sensor2、human と云う３つのステージが終了したら、calc ステージで、ロールコンテナ calc の命令 doCalc を実行する．命令 doCalc は、命令 getSensorData がパブリッシュしたメッセージ(温度データ)及び命令 getHumanData がパブリッシュしたメッセージ(パトライト点灯色)をサブスクライブし、QueueingCalc.med と云うプログラムで処理する．QueueingCalc.med は、パトライト点灯色が指定されていればその色をパブリッシュする．パトライト点灯色が指定されていなければ、２つの温度データの平均値によりパトライト点灯色を決定してその色をパブリッシュする．

(3)calc ステージが終了したら、actuator ステージで、ロールコンテナ actuator の命令 actuate を実行する．命令 actuate は、命令 doCalc がパブリッシュしたメッセージ(パトライト点灯色)をサブスクライブし、patlite.py と云う Python プログラムでパトライト制御を行う．

RWOS では、このようにテキストエディタによるプログラム作成が必要となる．そこで、このプログラムを自動生成する為のモデリング GUI として Java による RWOS Program Editor の設計・実装を行った．

```
# Project definition file example for RWOS.
# environment
ControlBroker {
        url             tcp://localhost:1883
}
DataBroker {
        url             tcp://localhost:1883
}
# project
project[loop=on] {
        sensor1 -> calc
        sensor2 -> calc
        human -> calc
        calc -> actuator
}
fail {
}
# conditions
# role
role {
        "role.rwrole"
}
```

```
# Role definition file example for RWOS.
# stages
stage[sensor1::sensor1] {
        call.container          "getSensorData"
}
stage[sensor2::sensor2] {
        call.container          "getSensorData"
}
stage[human::human] {
        call.container          "getHumanData"
}
stage[calc::calc] {
        call.container          "doCalc"
}
stage[actuator::actuator] {
        call.container          "actuate"
}
```

図 3-19 プロジェクト定義ファイル

```
# Role-container definition file example for RWOS.
# environment
ControlBroker {
        url             tcp://localhost:1883
}
DataBroker {
        url             tcp://localhost:1883
}
Queueing {
        "home/1"
}
# container name
name        sensor1
# functions
function getSensorData {
        getLastToBinaryFile   "home/1"    "sensor1/rawdata1.json"          "timeout=0"
        "limit=1"
        retainQueueLast       "home/1"
        exec.command          "python"    "sensor1/json2csv.py"  "temperature"          "sensor1"
        "sensor1/rawdata1.json"           "sensor1/rawdata1.csv"
        pushFromBinaryFile  "sensor1/rawdata1.csv"          "data/calc/sensor1"     "emptyFinal=true"
}
```

図 3-20 sensor1 ロールコンテナ定義ファイル

```
# Role-container definition file example for RWOS.
# environment
ControlBroker {
        url         tcp://localhost:1883
}
DataBroker {
        url         tcp://localhost:1883
}
Queueing {
        "office/0"
}
# container name
name    sensor2
# functions
function getSensorData {
        getLastToBinaryFile   "office/0"   "sensor2/rawdata2.json"          "timeout=0"
        "limit=1"
        retainQueueLast       "office/0"
        exec.command          "python"   "sensor2/json2csv.py"   "temperature"          "sensor2"
        "sensor2/rawdata2.json"          "sensor2/rawdata2.csv"
        pushFromBinaryFile   "sensor2/rawdata2.csv"          "data/calc/sensor2"     "emptyFinal=true"
}
```

図 3-21 sensor2 ロールコンテナ定義ファイル

```
# Role-container definition file example for RWOS.
# environment
ControlBroker {
        url         tcp://localhost:1883
}
DataBroker {
        url         tcp://localhost:1883
}
Queueing {
        "human/1"
}
# container name
name    human
# functions
function getHumanData {
        getLastToBinaryFile   "human/1" "human/rawdata1.json"          "timeout=0"
        "limit=1"
        retainQueueLast       "human/1"
        exec.command          "python"   "human/json2csv.py"   "command""human"
        "human/rawdata1.json"          "human/rawdata1.csv"
        pushFromBinaryFile   "human/rawdata1.csv""data/calc/human"     "emptyFinal=true"
}
```

図 3-22 human ロールコンテナ定義ファイル

```
# Role-container definition file example for RWOS.
# environment
ControlBroker {
        url         tcp://localhost:1883
}
DataBroker {
        url         tcp://localhost:1883
}
Queueing {
        "data/calc/sensor1"
        "data/calc/sensor2"
        "data/calc/human"
}
# container name
name    calc
# functions
function doCalc {
        popFirstToBinaryFile  "data/calc/sensor1"    "calc/sensor1.csv"    "timeout=-1"
        "limit=0"   "ignoreEmpty=false"
        popFirstToBinaryFile  "data/calc/sensor2"    "calc/sensor2.csv"    "timeout=-1"
        "limit=0"   "ignoreEmpty=false"
        popFirstToBinaryFile  "data/calc/human"      "calc/human.csv"      "timeout=-1"
        "limit=0"   "ignoreEmpty=false"
        exec.filter   "calc/QueueingCalc.med"        "calc/sensor1.csv"    "calc/sensor2.csv"
        "calc/human.csv"      "calc/result.csv"
        pushFromBinaryFile   "calc/result.csv"       "data/actuate"        "emptyFinal=true"
}
```

図 3-23 calc ロールコンテナ定義ファイル

```
# Role-container definition file example for RWOS.
# environment
ControlBroker {
        url         tcp://localhost:1883
}
DataBroker {
        url         tcp://localhost:1883
}
Queueing {
        "data/actuate"
}
# container name
name    actuator

# functions
function actuate {
        popFirstToBinaryFile  "data/actuate"         "actuator/actuatelog.csv"          "timeout=-1"
        "limit=0"   "ignoreEmpty=false"   "append=false"
        exec.command          "python"    "actuator/patlite.py"    "actuator/actuatelog.csv"
}
```

図 3-24 actuator ロールコンテナ定義ファイル

3.3.2 RWOS Program Editor の設計

3.3.2.1 ステージ編集

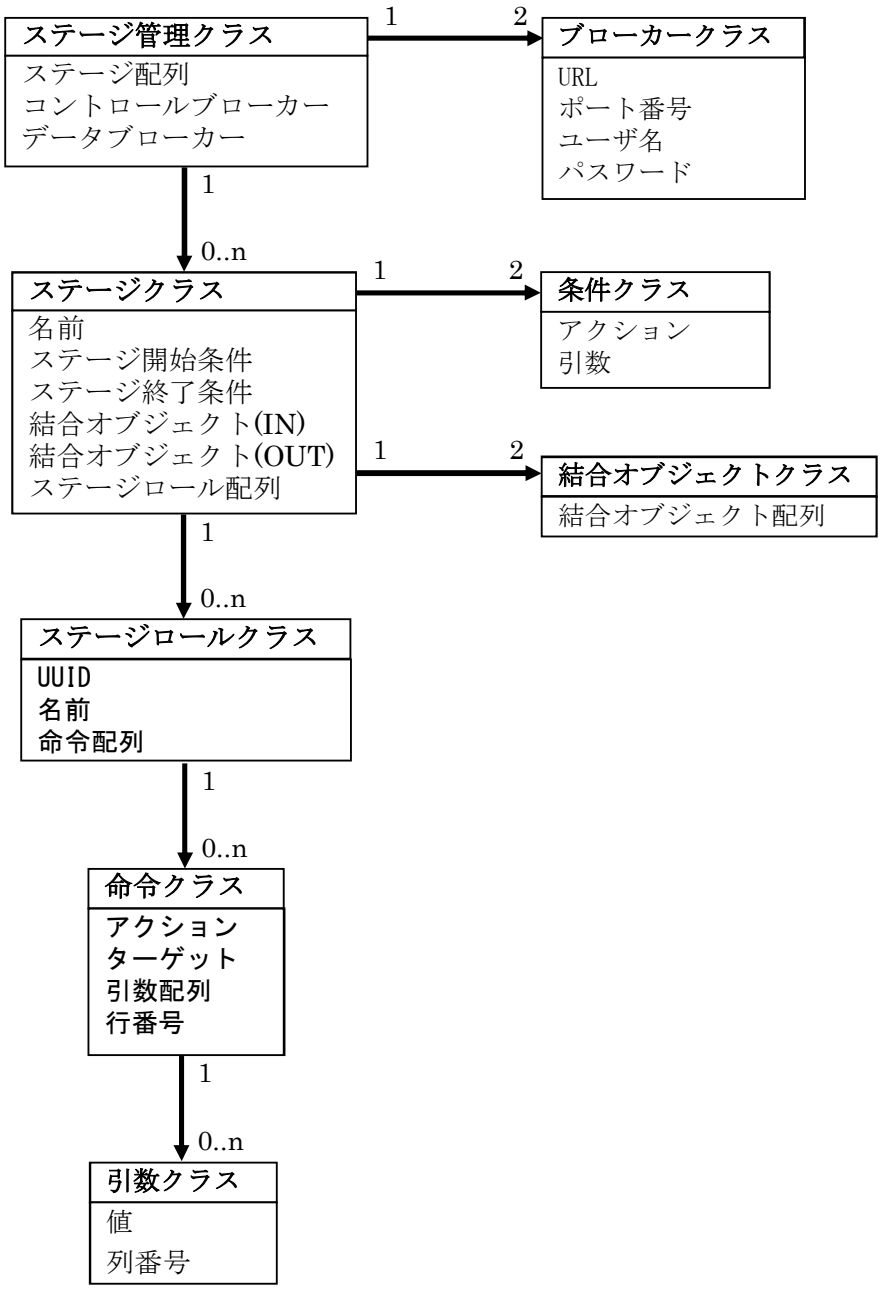


図 3-25 RWOS Program Editor ステージクラス構成

図 3-25 は RWOS Program Editor ステージ編集機能のクラス構成を示している.

各ステージは、図 3-26 のようにウィンドウへアイコンとして表示され、アイコンを１つ選択すると、そのアイコンが表すステージの内容を編集出来るものとする.

ステージ実行順序は、マウスのドラッグ＆ドロップによりアイコンどうしを繋ぐことにより指定出来るものとする.

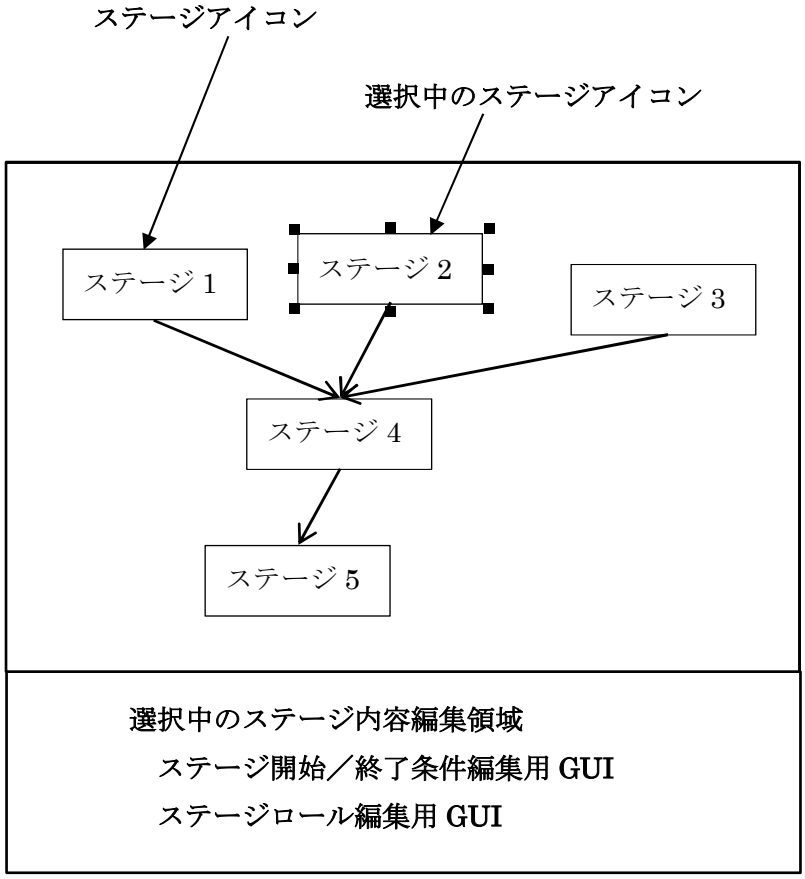


図 3-26 ステージ編集用 GUI

3.3.2.1.1 ステージクラス

ステージクラスは、その名前、結合オブジェクト、複数のステージロール、ステージ開始条件及びステージ終了条件を保持する．ステージ管理クラスは、ステージクラスのインスタンスを配列の要素として管理する.

3.3.2.1.2 結合オブジェクトクラス

結合オブジェクトクラスは、ステージアイコンどうしを繋いでステージ実行順序定義を行える GUI 構築の為に使用する．そのメンバである配列に、結合先の結合オブジェクトクラスを複数保持する.

3.3.2.1.3 ステージロールクラス

ステージロールクラスは、UIID[RFC4122 2005]、名前及び命令配列を保持する．命令は各ロールコンテナが保持しているものである．ステージロールの名前はユニークではない為、UUID によりインスタンスを識別する．

ステージロールは、並列実行出来ない命令の配列を保持しており、各命令は配列内の順序に従って実行される．並列実行可能な命令は別のステージロールへ含める．この仕組みにより、同一ステージ内のステージロールは並列実行が可能となる．

モデリング GUI を実現する為、図 3-27 のようにステージロールはツリーで編集し、命令配列はスプレッドシートで編集するものとする．

3.3.2.1.4 命令クラス及び引数クラス

命令クラスは、命令のタイプを表すアクション、実際の命令を表すターゲットとその引数配列及びスプレッドシート上での行番号を保持し、図 3-27 のスプレッドシート上の１行を表す．引数クラスは、引数の値及びスプレッドシート上での列番号を保持する．アクションは、スプレッドシート上で、コンボボックスにより選択出来るものとする．

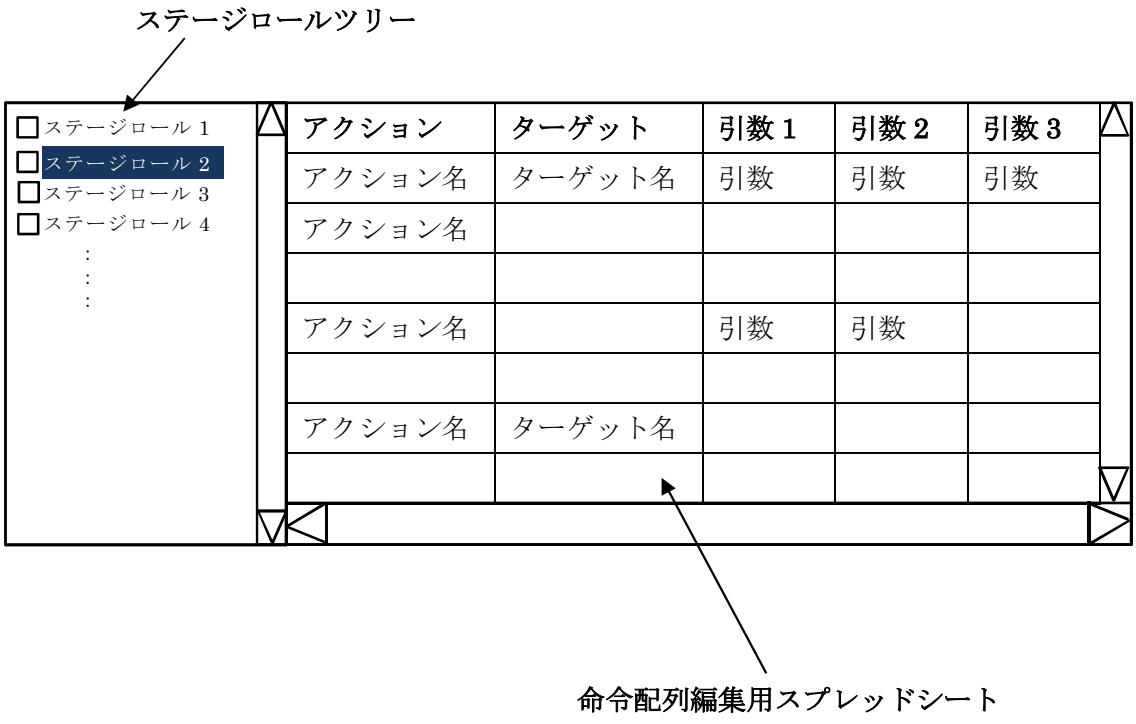


図 3-27 ステージロール編集用 GUI

3.3.2.1.5 条件クラス

条件クラスは、各ステージの開始／終了条件を保持する．RWOS 実行時に、各ステージは開始条件が満たされていなければ、そのステージロールを実行しない．また、各ステージは終了条件が満たされなければ、プロジェクトへステージ完了通知のメッセージをパブリッシュしない．条件クラスは、条件を表すアクションとその引数文字列を保持する．図 3-28 のように、アクションはコンボボックスにより選択し、引数はテキストフィールドにより入力出来るものとする．

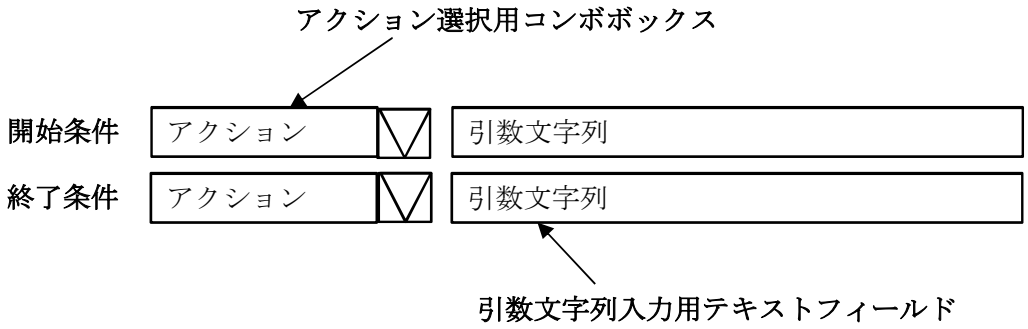


図 3-28 ステージ開始条件／終了条件編集用 GUI

3.3.2.1.6 ブローカークラス

ブローカークラスは、プロジェクト実行時に使用する MQTT ブローカーの情報として、URL、ポート番号、ユーザ名及びパスワードを保持する．図 3-29 のように、テキストフィールドによりこれらを編集する為の GUI が必要になる．

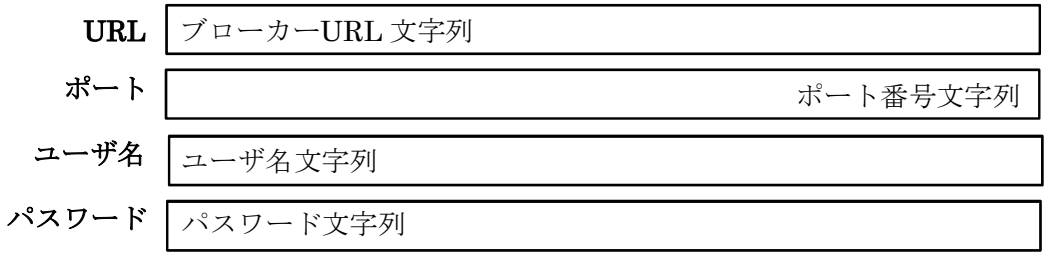


図 3-29 MQTT ブローカー編集用 GUI

3.3.2.1.7 プロジェクトプロセス起動

RWOS Program Editor では、ボタンを押すだけで、そのモデリング GUI により入力された全てのデータから RWOS プログラムを自動生成し、RWOS 実行プログラムである RWOS Project Manger にこれを実行させる．この時、RWOS Program Editor は、一時的な作業フォルダを作成してそこに RWOS プロジェクト定義ファイルを配置する．

3.3.2.2 ロールコンテナ編集

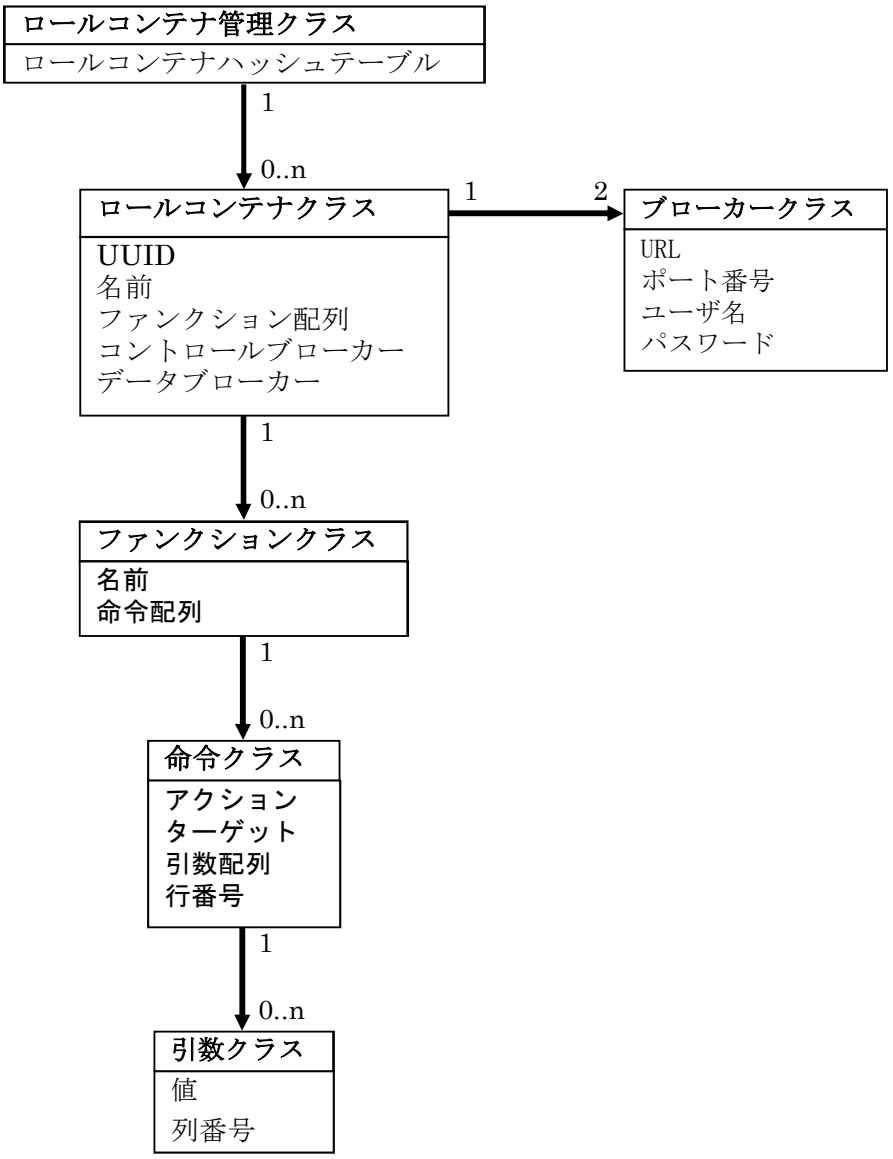


図 3-30 RWOS Program Editor ロールコンテナクラス構成

図 3-30 は RWOS Program Editor ロールコンテナ編集機能のクラス構成を示している．ロールコンテナは、図 3-31 のようにロールコンテナツリーでその編集を行う．ロールコンテナツリーで選択されているロールコンテナの内容を編集出来るものとする．モデリング GUI により入力された全てのデータから RWOS ロールコンテナ定義ファイルを自動生成し、更に実行形式の生成も行えるようにする必要がある．

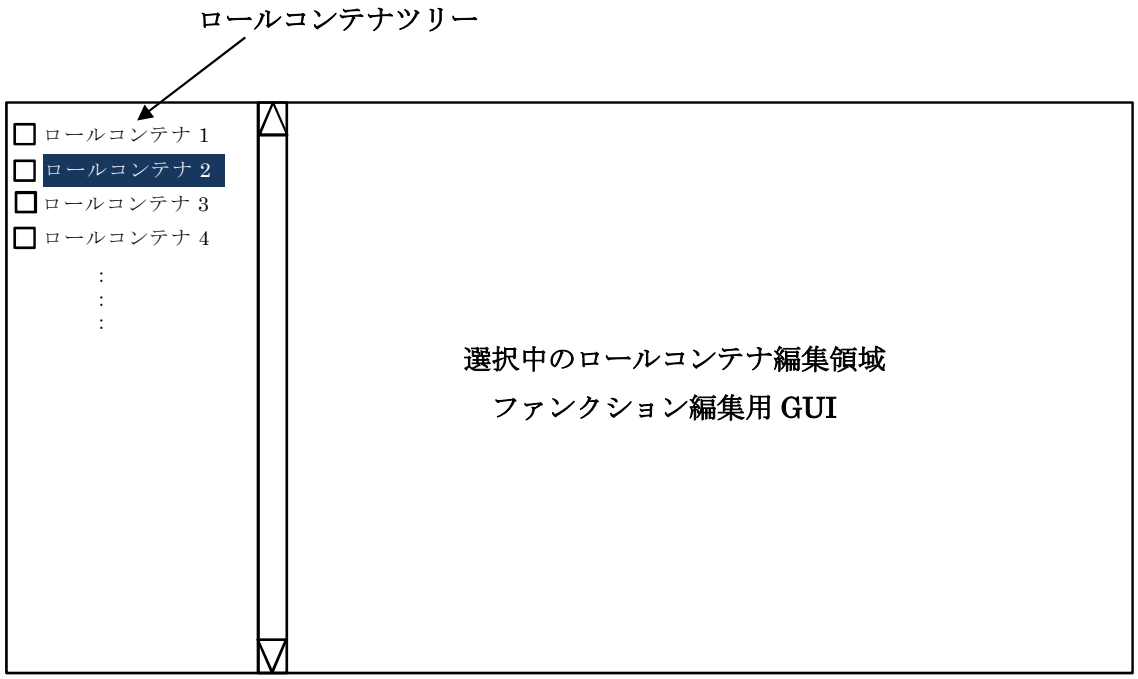


図 3-31 ロールコンテナ編集用 GUI

3.3.2.2.1 ロールコンテナクラス

ロールコンテナクラスは、UUID、名前、複数のファンクション及び MQTT ブローカー情報を保持する．ロールコンテナの名前はユニークではない為、UUID によりインスタンスを識別する．ロールコンテナ管理クラスは、UUID をキー、ロールコンテナクラスのインスタンスを値とするハッシュテーブルによって管理する．

3.3.2.2.2 ファンクションクラス

ファンクションクラスは、名前及び命令配列を保持する．
ファンクションは、並列実行出来ない命令の配列を保持しており、各命令は配列内の順序に従って実行される．並列実行可能な命令は別のファンクションへ含める．この仕組みにより、同一ロールコンテナ内のファンクションは並列実行が可能となる．

モデリング GUI を実現する為、図 3-32 のようにファンクションはツリーで編集し、命令配列はスプレッドシートで編集するものとする．

3.3.2.2.3 命令クラス及び引数クラス

命令クラスは、命令のタイプを表すアクション、実際の命令を表すターゲットとその引数配列及びスプレッドシート上での行番号を保持し、図 3-32 のスプレッドシート上の 1 行を表す．引数クラスは、引数の値及びスプレッドシート上での列番号を保持する．アクションは、スプレッドシート上で、コンボボックスにより選択出来るものとする．また、ターゲットがプログラムファイルである場合や引数にフォルダやファイルを指定する場合、フォルダまたはファイルをドラッグ＆ドロップによりスプレッドシートのセルへ設定出来るようなファイルマネージャも必要となる．ファイルマネージャは Windows の Explorer や Mac の Finder のようなファイル操作を行えるものとする．

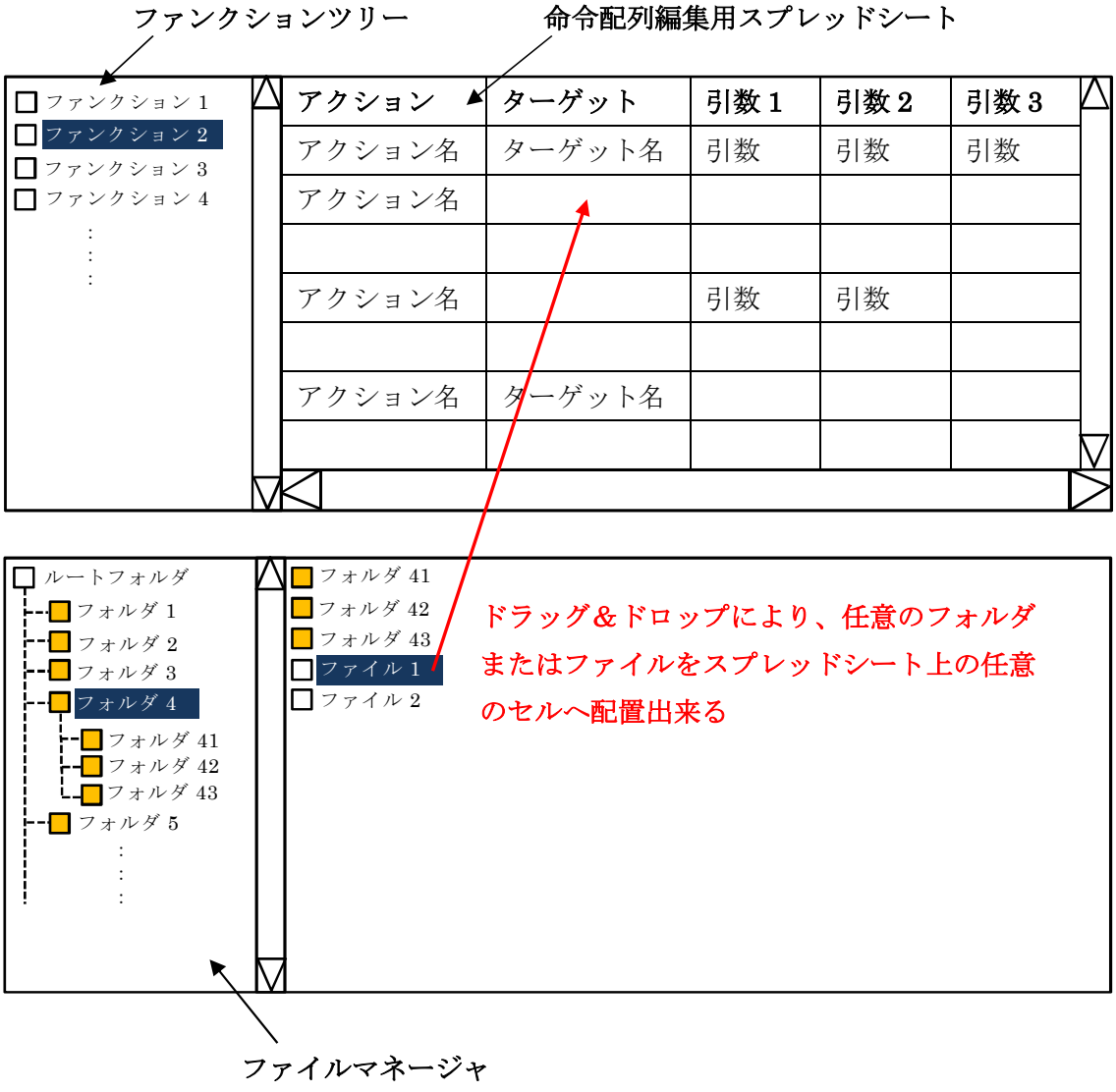


図 3-32 ファンクション編集用 GUI

3.3.2.2.4 ブローカークラス

ブローカークラスは、ロールコンテナ実行時に使用する MQTT ブローカーの情報として、URL、ポート番号、ユーザ名及びパスワードを保持する．図 3-29 のように、テキストフィールドによりこれらを編集する為の GUI が必要になる．

3.3.3 データファイルフォーマット

RWOS Program Editor のデータファイル(RWOS プロジェクトファイル)は、編集データだけでなく、各ロールコンテナが使用するファイル(プログラムファイルやデータファイル)もその内部に含むことが出来る必要がある．従って、図 3-33 のようなファイルシステムを持つ圧縮ファイルとする．

各ロールコンテナが使用するファイルは、ロールコンテナクラスの保持する UUID を名前とするフォルダ直下の root フォルダ以下に配置される．この root フォルダは、図 3-32 のファイルマネージャにおけるルートフォルダに相当する．

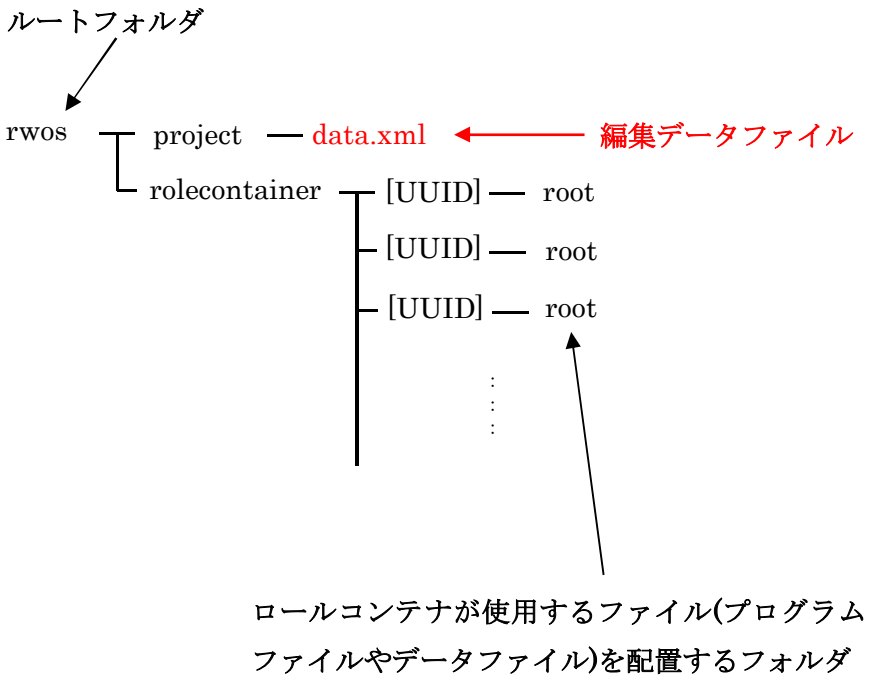


図 3-33 RWOS Program Editor データファイル内部のファイルシステム

3.3.4 ライブラリ化

RWOS プロジェクトファイルをライブラリ化して、RWOS Program Editor 上で管理出来る機能を実装する.

RWOS プロジェクトファイルは､図 3-34 のようにプロジェクトツリーにより管理する. プロジェクトツリーで選択された RWOS プロジェクトファイルのステージ編集及びロールコンテナ編集をタブで切り替えて行えるようにする.

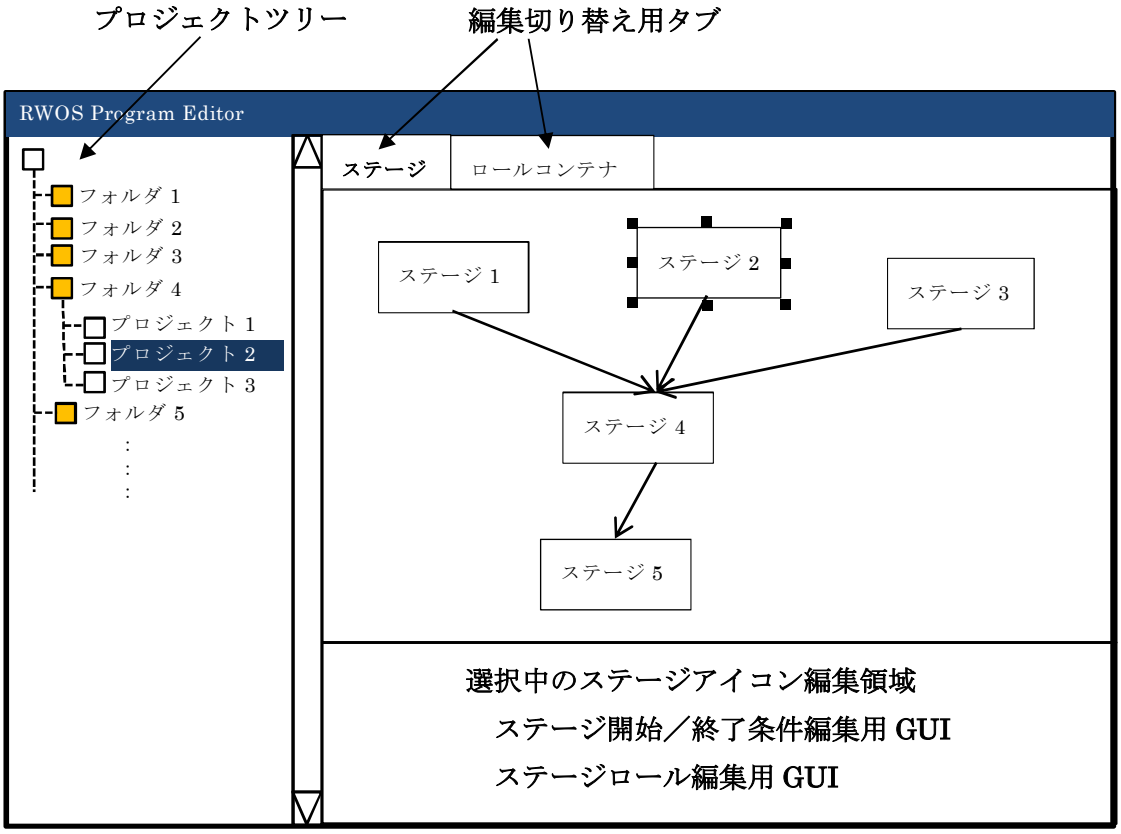


図 3-34 RWOS Program Editor ウィンドウ

また、ロールコンテナ管理用の機能として、RWOS Role Container Library Manager を実装する. RWOS Role Container Library Manager は、RWOS Program Editor と同様のロールコンテナ編集を行えるものとし、ロールコンテナツリーによりロールコンテナのデータファイル(RWOS ロールコンテナファイル)を管理する. RWOS ロールコンテナファイルは内部に図 3-36 のようなファイルシステムを持つ圧縮ファイルとする. RWOS Program Editor と RWOS Role Container Library Manager の間では、図 3-35 のようにドラッグ＆ドロップによりロールコンテナのやりとりを行えるようにする. これにより編集データの再利用及び配布が容易に行えるようになる.

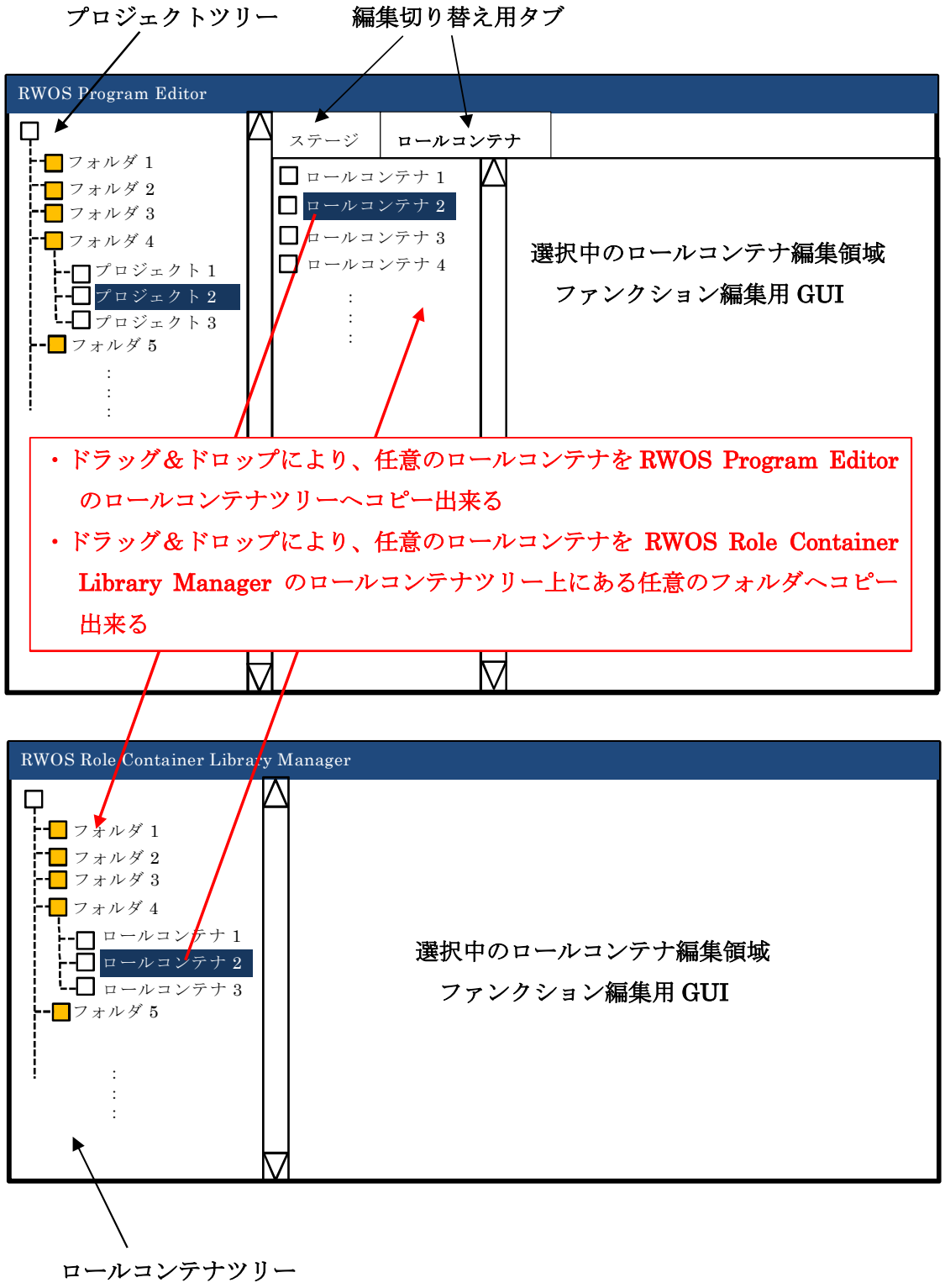


図 3-35 RWOS Program Editor と RWOS Role Container Library Manager

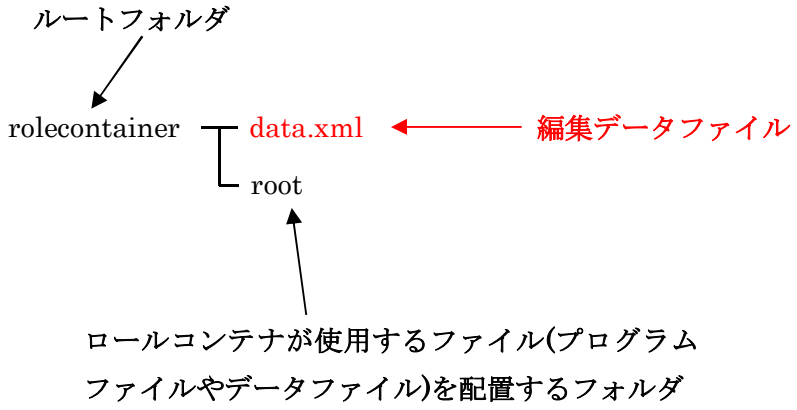


図 3-36 RWOS ロールコンテナファイル内部のファイルシステム

第4章 モデルメソッド駆動型アーキテクチャに基づくモデリング GUI の実装事例

この章では、前章の設計に基づいて実装した SOARS 及び RWOS のモデリング GUI である SOARS VisualShell 及び RWOS Program Editor を実際に使用した事例について述べる．SOARS VisualShell の事例では、SOARS VisualShell により、3.2.1 で述べた SIR モデルを用いた感染症流行シミュレーションの SOARS プログラムを作成して実行し､その実行結果をアニメーションにより可視化した事例について述べる．RWOS Program Editor の事例では、RWOS Program Editor により、3.3.1 で述べた温度センサーからのデータ及び Web ブラウザからのデータに基づいてパトライト点灯色を決定して点灯させると云う RWOS プログラムを作成して実行した事例について述べる．

4.1 モデリング GUI としての SOARS VisualShell

SOARS VisualShell により、3.2.1 で述べた SIR モデルを用いた感染症流行シミュレーションの SOARS プログラムを作成して実行し､アニメーションによりその実行結果の可視化を行った．この事例では、エージェント「住民」が､「家」と「職場」と云う２つのスポット間を移動する．住民は、キーワード「病状」に、未感染、感染及び恢復または死亡を表す「”S”」､「”I”」及び「”R”」のいずれかの文字列を保持する．まず、シミュレーション開始時に数人の住民が感染する．以後、感染した住民が訪れたスポットが確率的に汚染され、未感染の住民は、汚染されたスポットを訪れることにより確率的に感染する．感染した住民は一定時間が経過すると恢復または死亡する．

シミュレーションは、その繰り返し処理の中で、まず全ての住民の病状が更新される．次に､未感染者数､感染者数及び恢復または死亡者数が､スポット「人数」の数値変数「S」､「I」及び「R」にそれぞれ保存される．感染者数、即ち数値変数「I」がゼロになると繰り返し処理から抜けてシミュレーションは終了する．

スポット「人数」の数値変数「S」､「I」及び「R」は、チャート「人数グラフ」で、その時系列変化をチャート表示するように指定する．従って、シミュレーション実行時にはこれらの数値変化をチャートにより可視化出来る．

4.1.1 SOARS VisualShell のユーザインタフェース

4.1.1.1 オブジェクトの生成

SOARS VisualShell では、図 4-1 のようにアイコンメニューから編集領域へアイコンをドラッグ＆ドロップするだけでオブジェクト(エンティティ、ロール及びチャート)を生成することが出来る．各アイコンをダブルクリックすることにより、各オブジェクトの編集を

行うことが出来る.

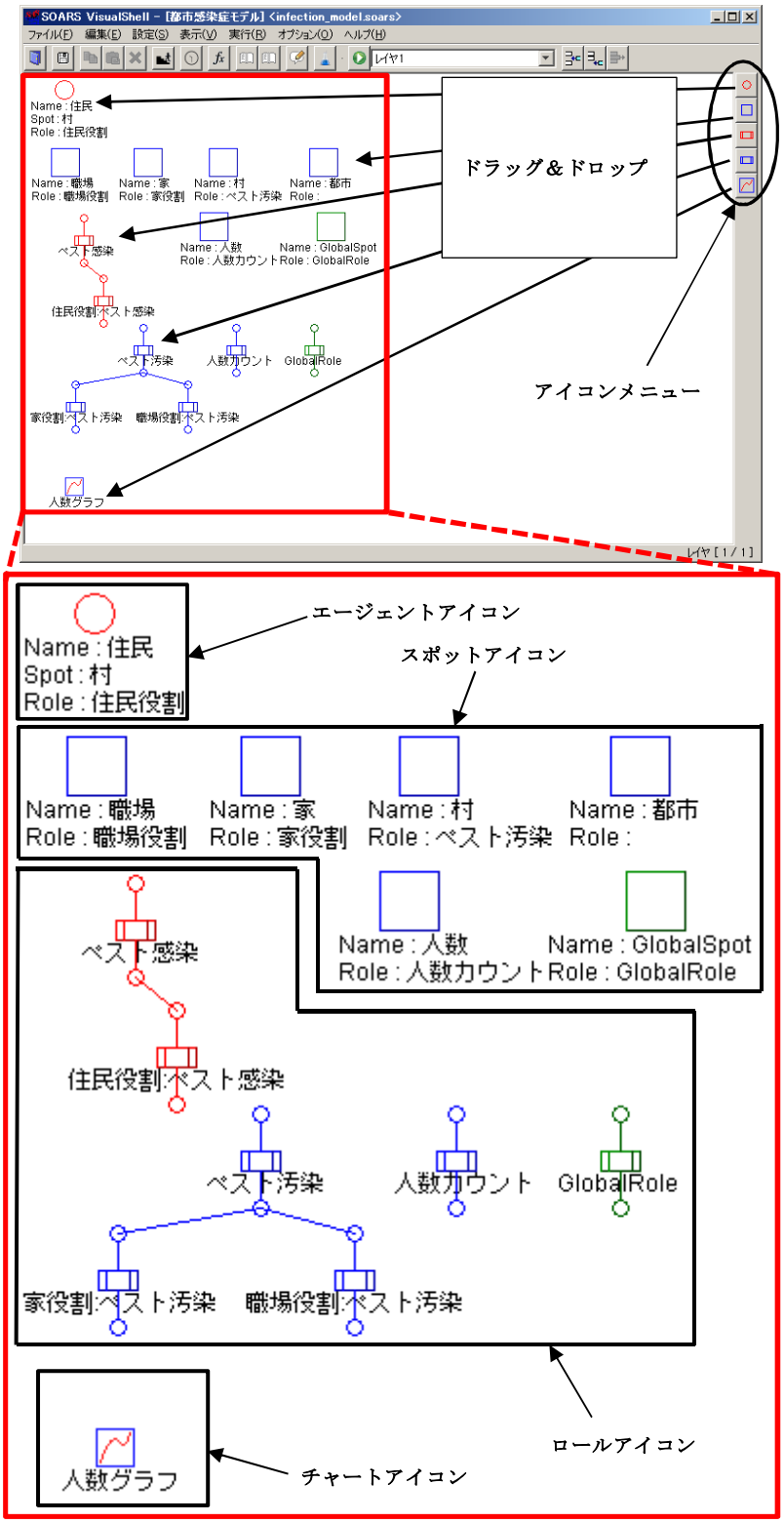


図 4-1 SOARS VisualShell におけるオブジェクトの生成

4.1.1.2 ステージ編集

図 4-2 は、この事例のステージ編集用 GUI である．初期ステージでは、初期感染ステージで数人の住民が感染する．登録ステージ、家割り当てステージ及び職場割り当てステージで、家及び職場の初期化、住民の家と職場の割り当てが完了する．

メインステージは、シミュレーションの繰り返し処理部分である．移動ステージでは、住民は朝９時に職場へ移動し、夜９時に家へ戻る．汚染クリアステージ及び汚染ステージで家または職場が確率的に汚染される．感染ステージでは、汚染されたステージの住民が確率的に感染する．人数クリアステージ及び人数カウントステージでは、未感染者数、感染者数及び恢復または死亡者数が更新される．終了判定ステージでは、感染者数がゼロであれば繰り返し処理から抜けてシミュレーションが終了する．

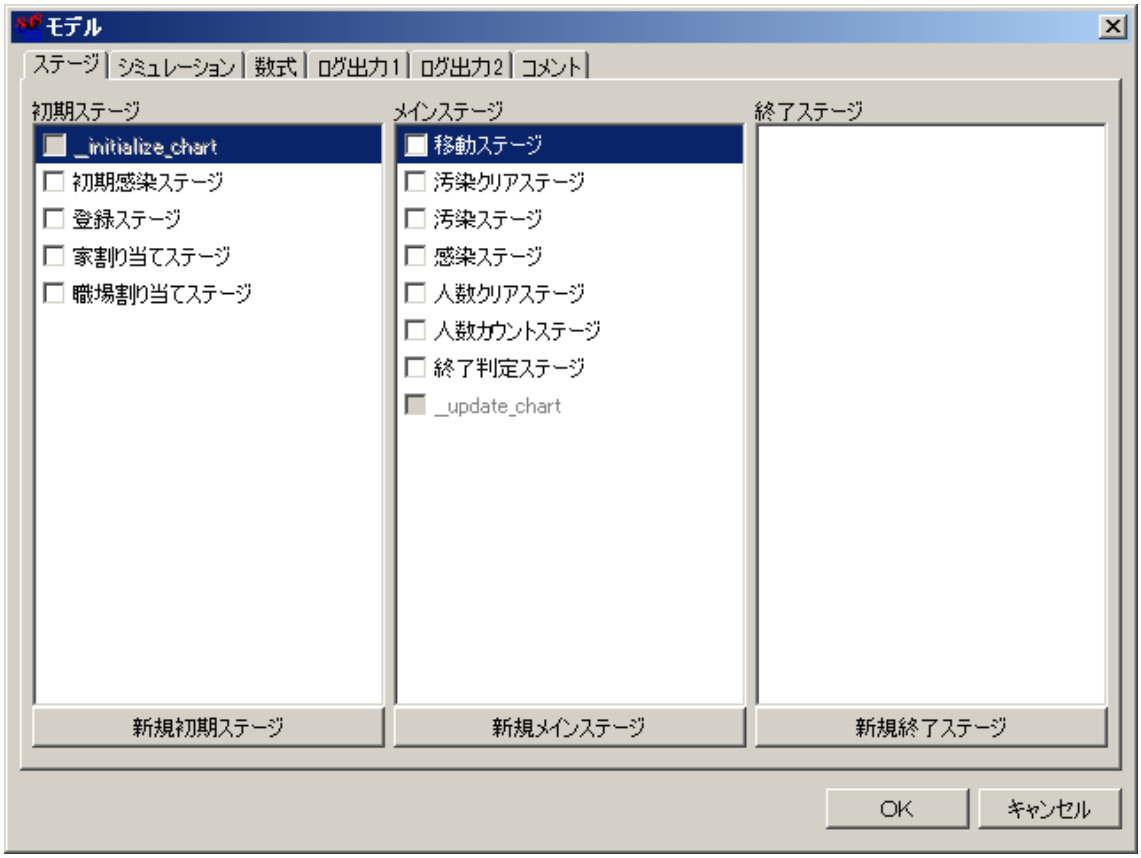


図 4-2 ステージ編集用 GUI

4.1.1.3 エンティティの編集

図 4-3 はエージェント「住民」の編集用 GUI である．名前及び個体数はテキストフィールドにより入力する．初期スポット及び初期ロールはコンボボックスにより選択する．

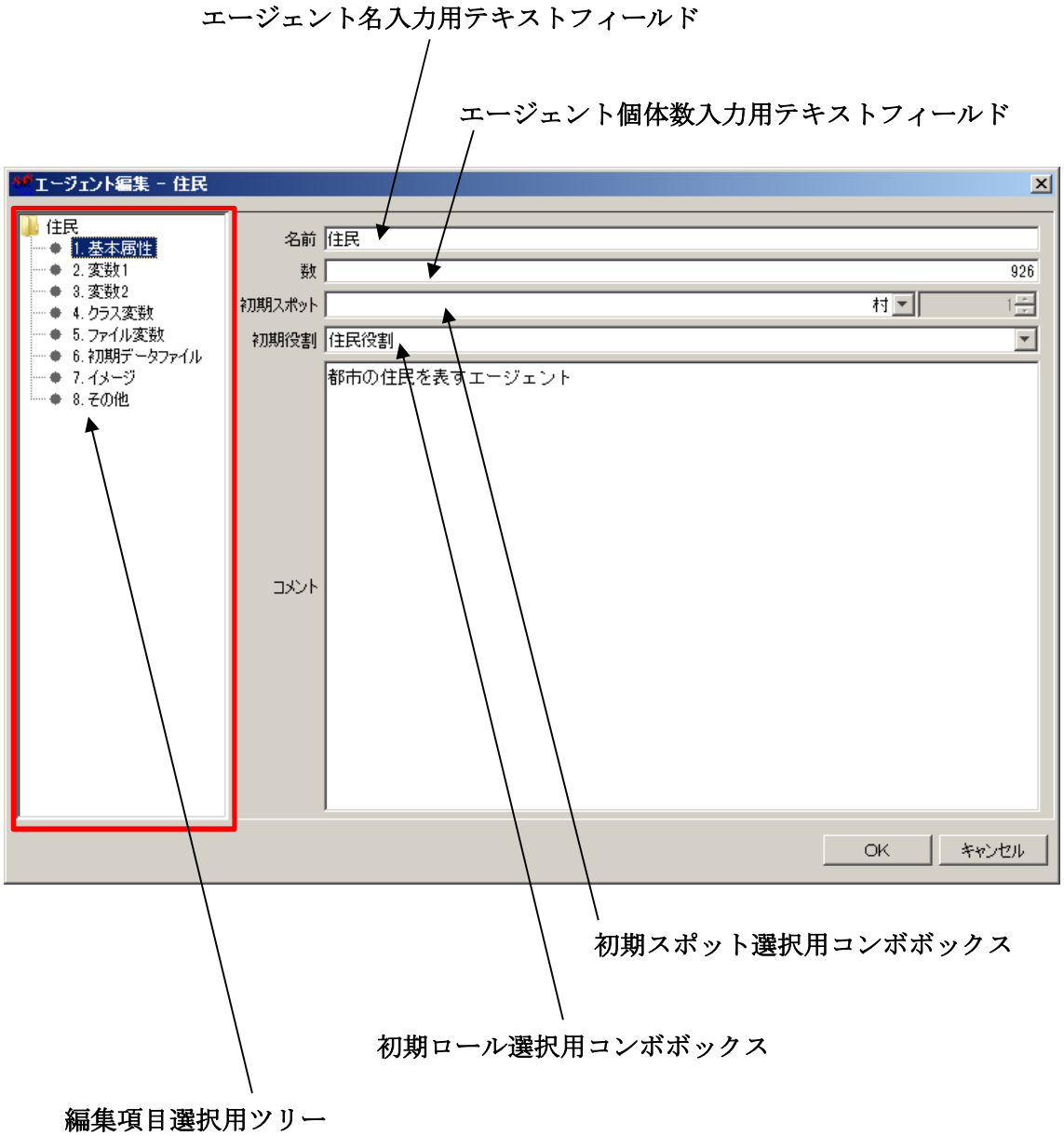


図 4-3 SOARS VisualShell におけるエンティティ編集用 GUI

4.1.1.4 エンティティの変数編集

図 4-4 はエージェント「住民」の変数編集用 GUI である．コンボボックスから変数の種類を選択する．この例では「キーワード」である．キーワードの名前及び初期値はテキストフィールドにより入力する．

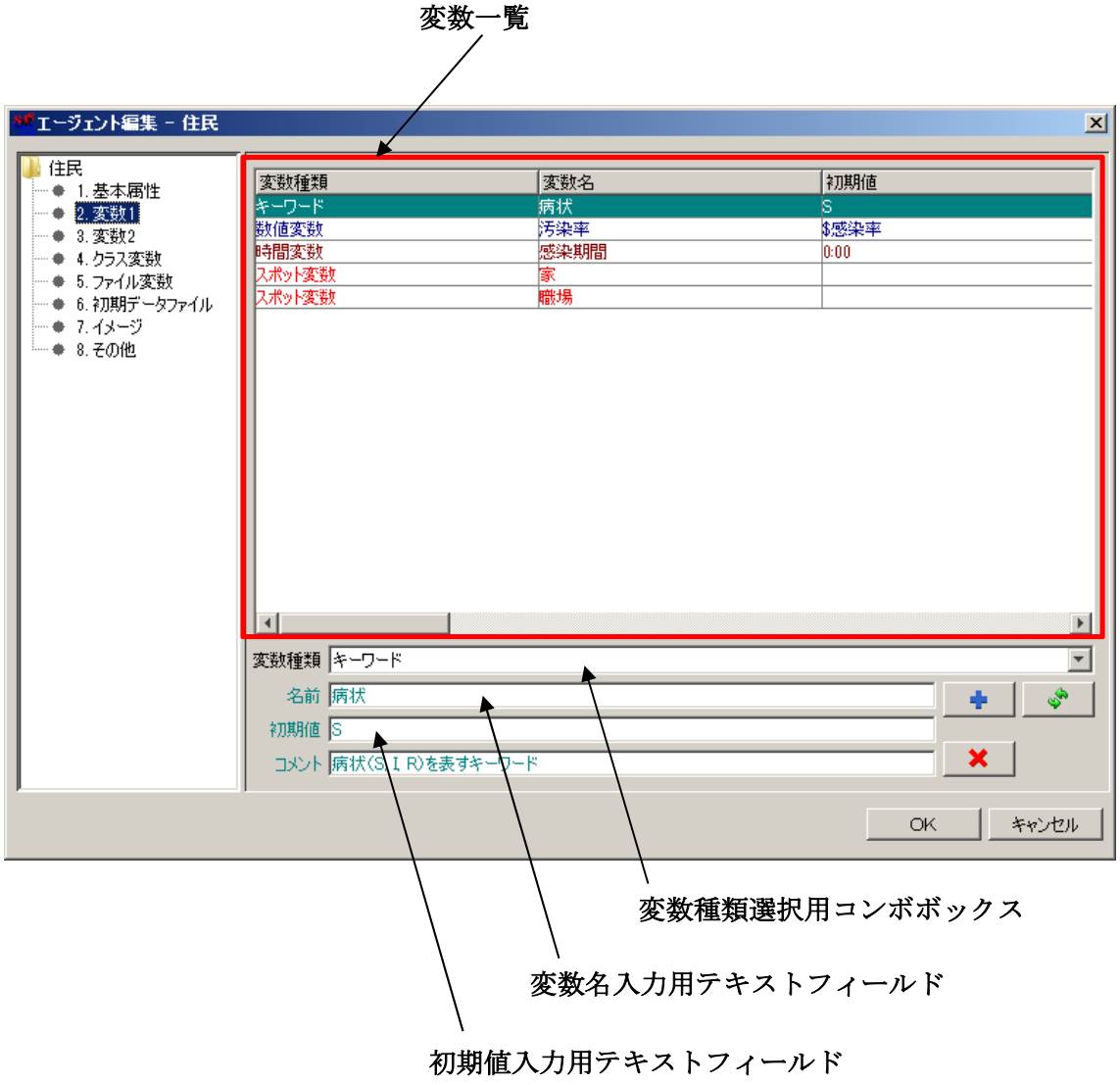


図 4-4 SOARS VisualShell におけるエンティティの変数編集用 GUI

図 4-5 はスポット「都市」の変数編集用 GUI である．コンボボックスから変数の種類を選択する．この例では「リスト変数」である．リスト変数の名前はテキストフィールドにより入力する．この例ではリスト変数の初期値を設定していないが、初期値には任意の変数をいくつでも追加することが出来る．初期値は、種類選択用インタフェースにより初期値の型及び変数を指定する．指定した初期値は、初期値操作用インタフェースにより追加出来る．初期値操作用インタフェースのボタンは、左から順に以下の機能に対応している．

・指定した初期値を最後尾に追加
・初期値一覧で選択されている部分へ挿入
・初期値一覧で選択されている部分の要素を置き換える
・初期値一覧で選択されている部分の要素を削除
・初期値一覧で選択されている部分の要素を１つ上へ移動
・初期値一覧で選択されている部分の要素を１つ下に移動

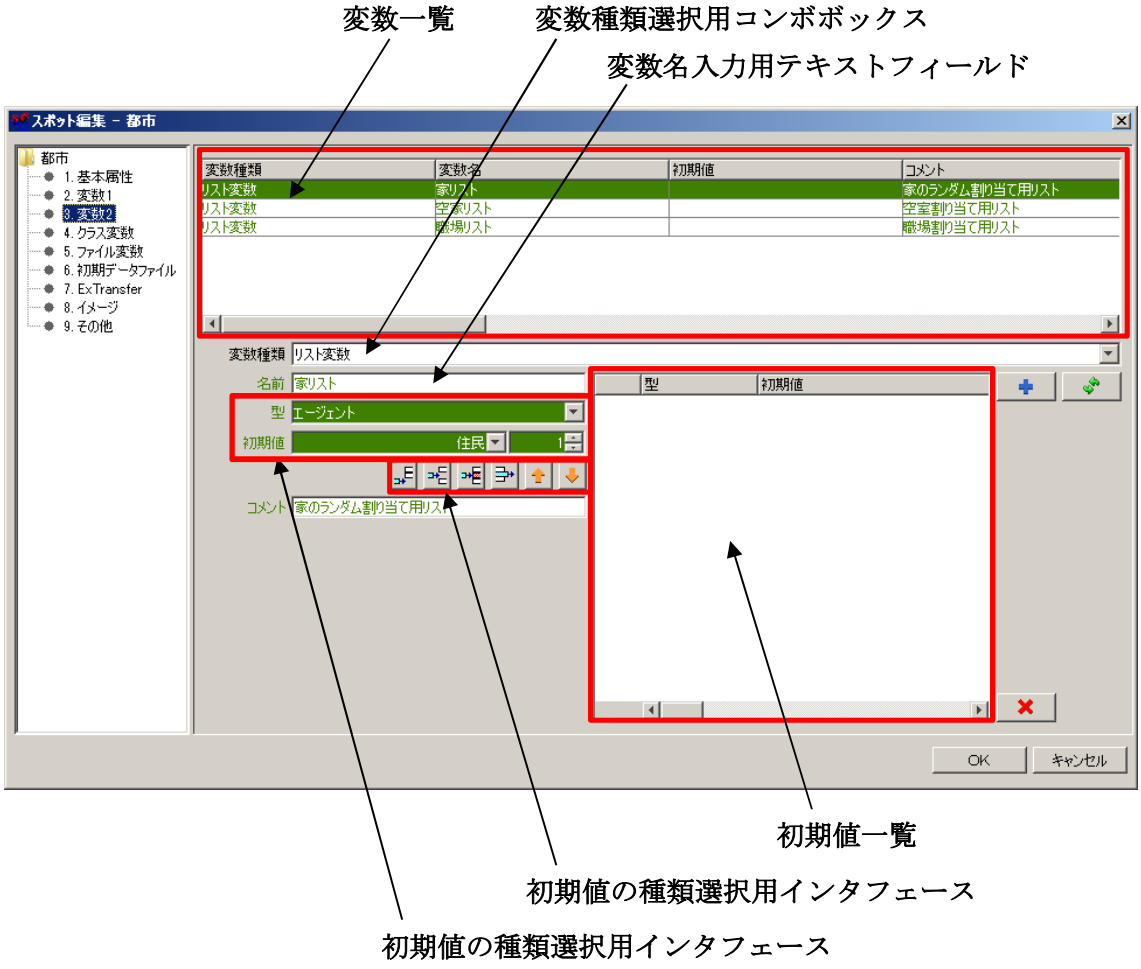


図 4-5 SOARS VisualShell におけるエンティティの変数編集用 GUI

4.1.1.5 チャート編集

図 4-6 は、チャート「人数グラフ」の情報編集用 GUI である．スポット「人数」の数値変数「S」、「I」及び「R」の時系列変化をチャート表示するように指定する．チャート一覧のチャート変数ペアは組み合わせ編集用 GUI で指定出来る．

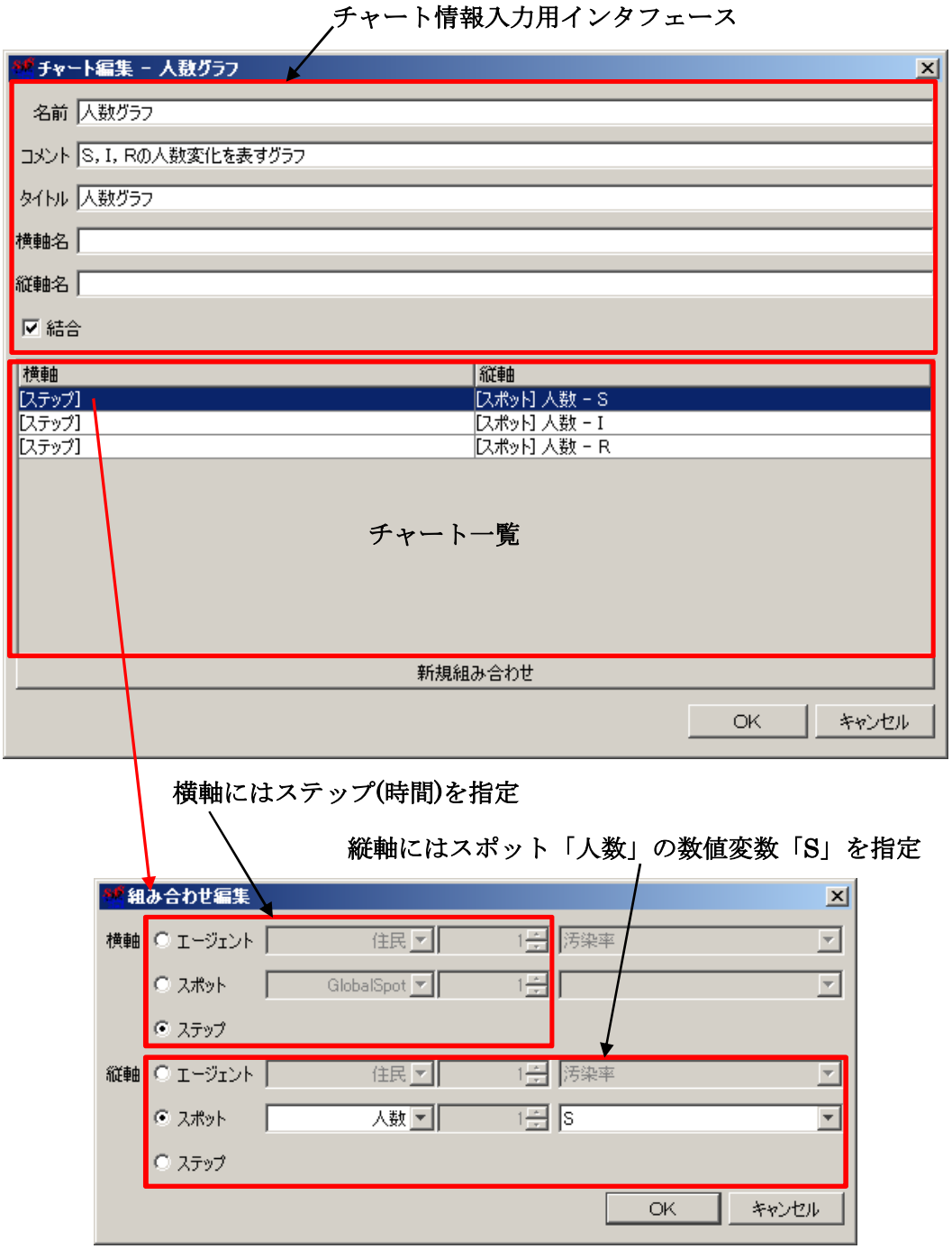


図 4-6 チャート編集用 GUI

4.1.1.6 数式編集

図 4-7 では、この事例で使用する関数定義を行っている．関数は数式編集用 GUI により行うことが出来る．

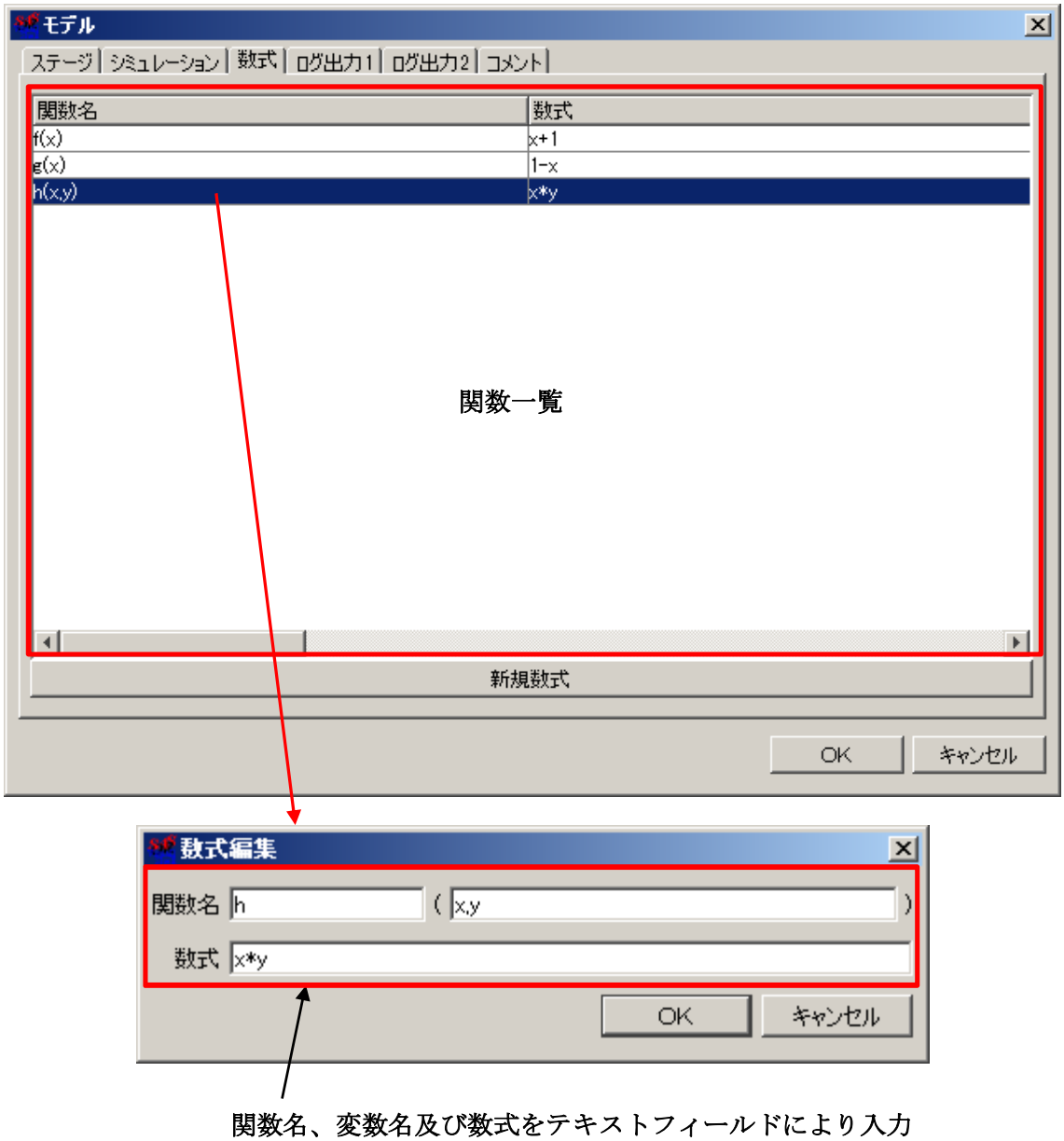


図 4-7 関数定義用 GUI

4.1.1.7 ロール編集

図 4-8 は、エージェントロール「ペスト感染」のルール編集用 GUI である．住民の各ステージでの振る舞いを記述している．スプレッドシートの１列目はコンボボックスからステージを選択する． ２列目以降には、そのステージで住民が行うルールを指定する．

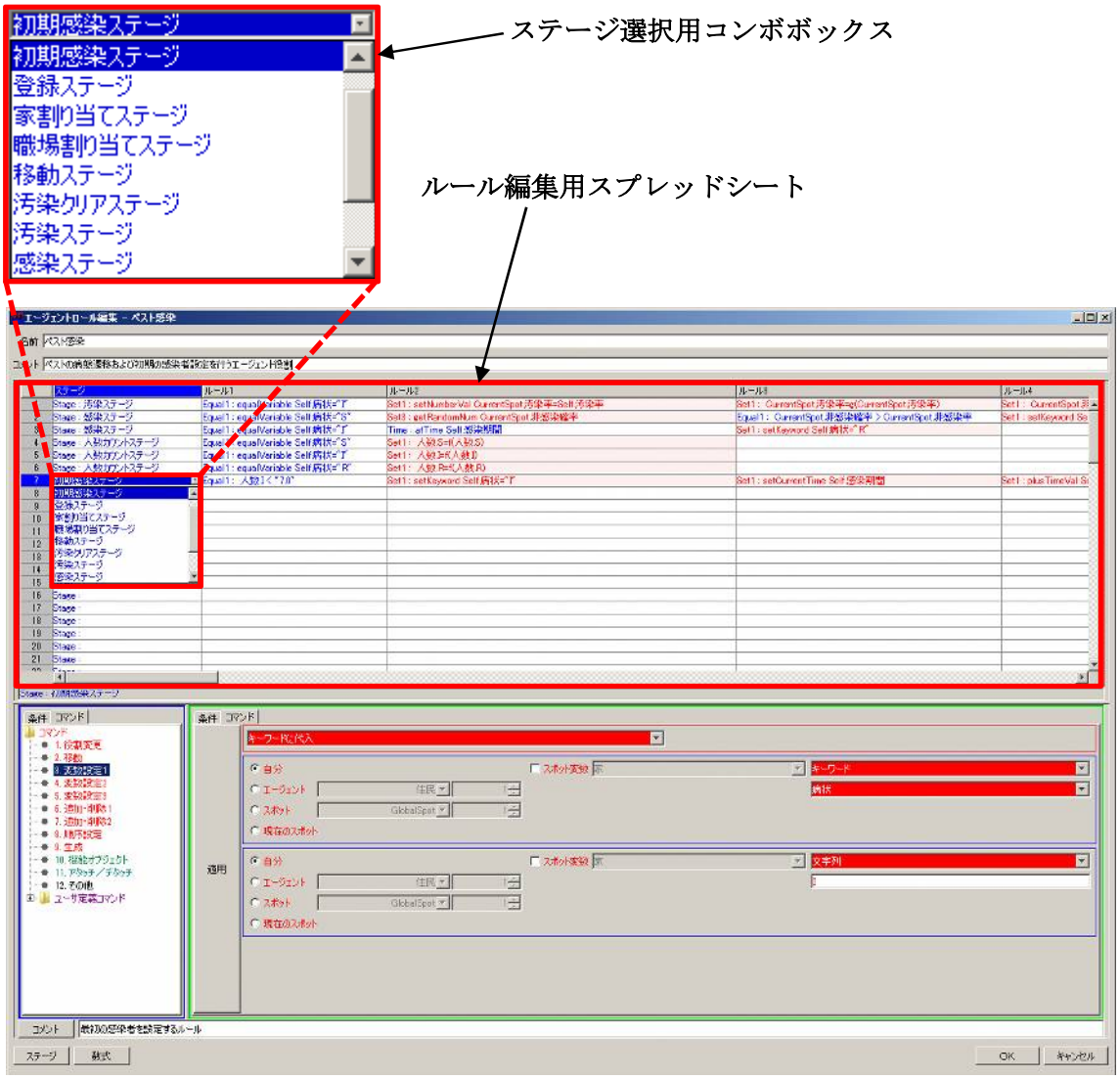


図 4-8 ロール編集用 GUI におけるステージ選択

図 4-9 は住民が持つキーワード「病状」に感染状態を表す文字列「"I"」を設定するルールの編集を行っている．図 4-10 に命令選択用コンボボックスを、図 4-12 にこのルールタイプの GUI 定義用スクリプトを示す．

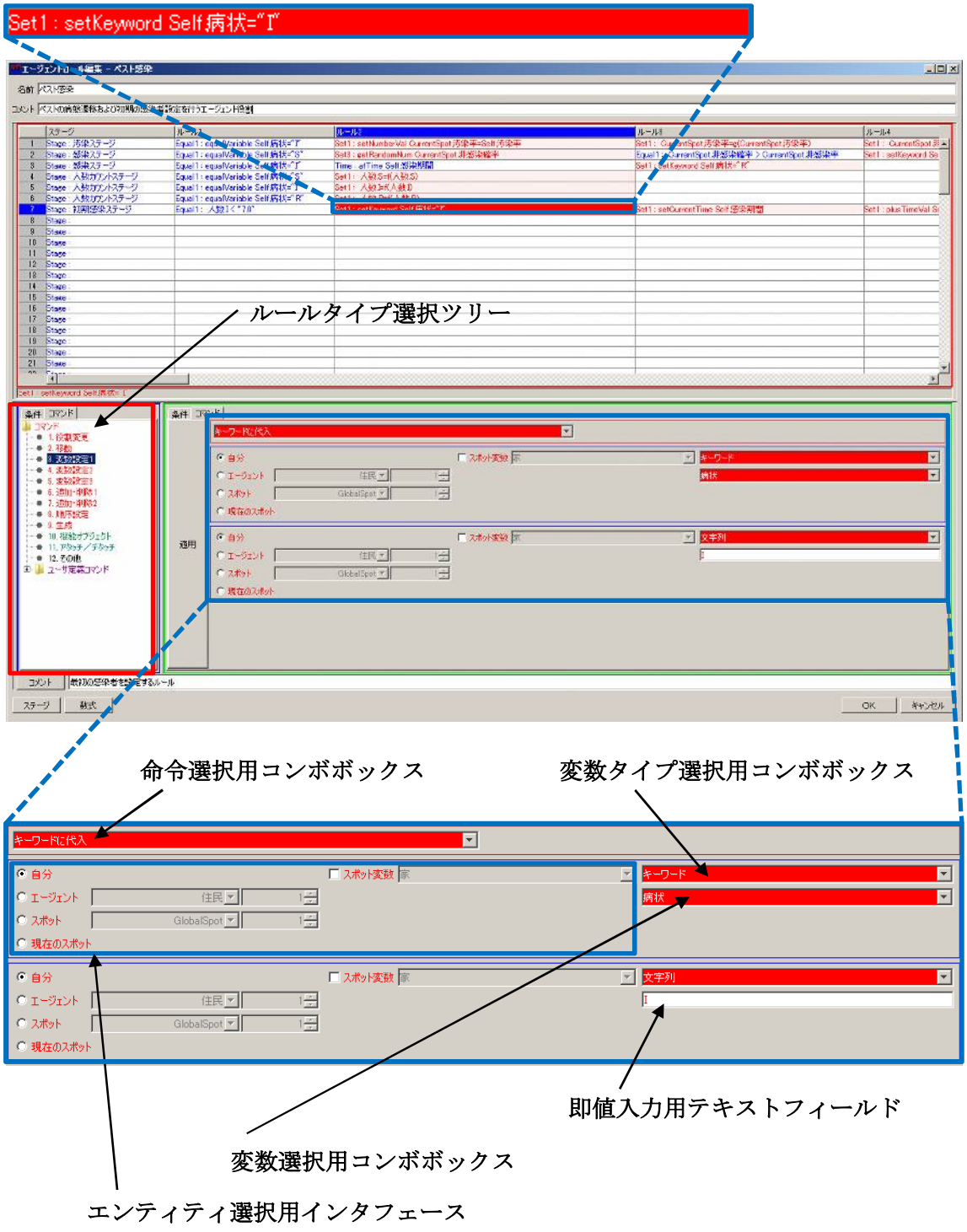


図 4-9 ルール編集用 GUI

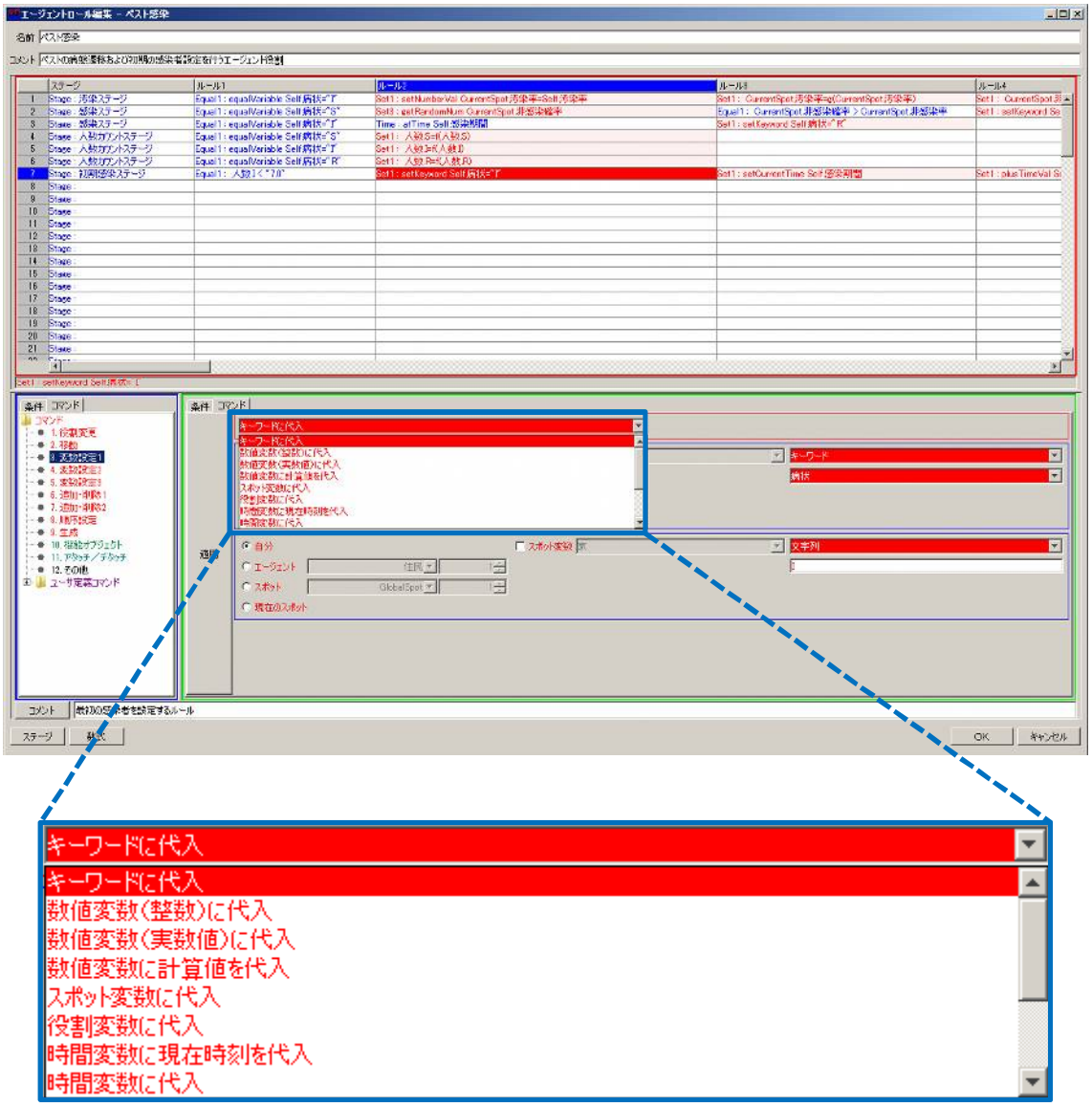


図 4-10 ルール編集における命令選択用コンボボックス

図 4-11 は、関数 f(x)=x+1 を利用してスポット「人数」の数値変数「S」の値を 1 つ増やすルールの編集を行っている．関数の引数編集は値編集 GUI により行うことが出来る．このルールの GUI 定義も、図 4-12 の GUI 定義用スクリプトに含まれている．

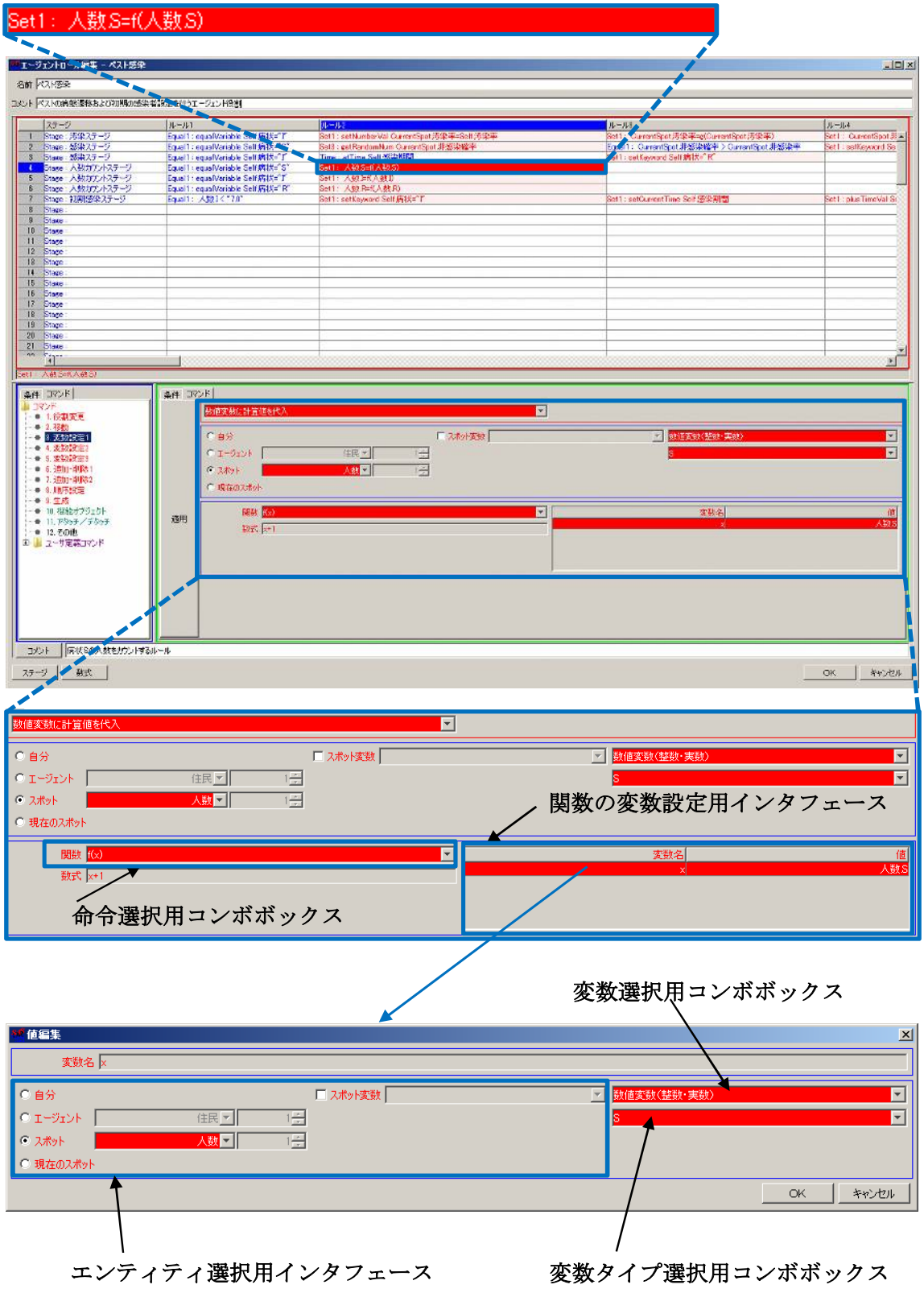


図 4-11 ロール編集における関数利用

```
entity:both
type:set1
name:Set1
color:255,0,0
title:変数設定 1
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード
variable:immediate keyword,文字列
verb:id1,1,setKeyword,キーワードに代入
object:self,agent,spot,currentspot,spotvariable
variable:integer number object,数値変数（整数）
object:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
variable:immediate integer,整数値
verb:id2,2,setNumberVal,数値変数（整数）に代入
object:self,agent,spot,currentspot,spotvariable
variable:real number object,数値変数（実数）
object:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
variable:immediate real number,実数値
verb:id3,3,setNumberVal,数値変数（実数値）に代入
object:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
expression:self,agent,spot,currentspot,spotvariable
variable:number object,数値変数（整数・実数）
variable:immediate integer,整数値
variable:immediate real number,実数値
verb:numeric.expression.substitution,4,,数値変数に計算値を代入
object:self,agent,spot,currentspot,spotvariable
variable:spot variable,スポット変数
object:self,agent,spot,currentspot,spotvariable
verb:id4,5,setSpotVal,スポット変数に代入
object:self,agent,spot,currentspot,spotvariable
variable:role variable,役割変数
object:self,agent,spot,currentspot,spotvariable
variable:role,役割
variable:role variable,役割変数
verb:id5,6,setRoleVal,役割変数に代入
```

図 4-12 「変数設定 1」の GUI 定義用スクリプト

今回の事例では使用していないが、ロールのルール編集では、任意の Java プログラム(jar ファイル)に含まれるメソッドを呼ぶことが出来る．図 4-13 は、この機能を利用して、MQTT 通信を行うルールを使用出来るようにしたものである．図 4-14 が、この GUI 定義用スクリプトである．

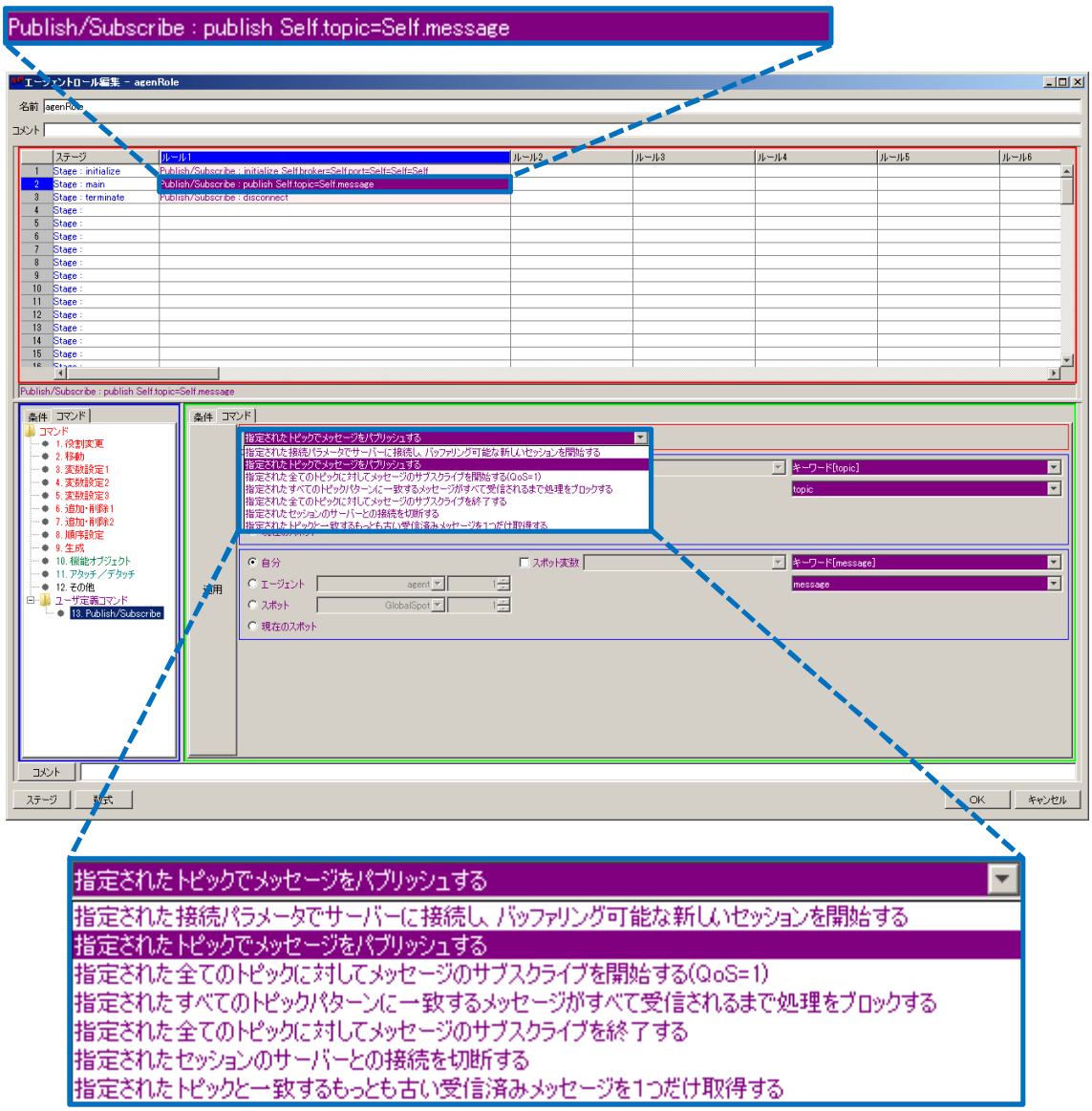


図 4-13 ロール編集における Java プログラム利用

```
entity:both
type:pubsub
name:Publish/Subscribe
color:128,0,128
title:Publish/Subscribe
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[ブローカー],empty
object:self,agent,spot,currentspot,spotvariable
variable:integer number object,数値変数[ポート番号],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[クライアント ID],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[ユーザ名],empty
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[パスワード],empty
verb:id1,1,soars.library.adapter.pubsub.PubSub:initialize,指定された接続パラメータでサーバーに接続し、バッファリング可能な新しいセッションを開始する
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[topic]
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[message]
variable:number object,数値変数[message]
verb:id2,2,soars.library.adapter.pubsub.PubSub:publish,
指定されたトピックでメッセージをパブリッシュする
object:self,agent,spot,currentspot,spotvariable
variable:list,リスト
verb:id3,3,soars.library.adapter.pubsub.PubSub:subscribe,指定された全てのトピックに対してメッセージのサブスクライブを開始する(QoS=1)
object:self,agent,spot,currentspot,spotvariable
variable:list,リスト
verb:id4,4,soars.library.adapter.pubsub.PubSub:waitAllMessages,指定されたすべてのトピックパターンに一致するメッセージがすべて受信されるまで処理をブロックする
object:self,agent,spot,currentspot,spotvariable
variable:list,リスト
verb:id5,5,soars.library.adapter.pubsub.PubSub:unsubscribe,指定された全てのトピックに対してメッセージのサブスクライブを終了する
verb:id6,6,soars.library.adapter.pubsub.PubSub:disconnect,指定されたセッションのサーバーとの接続を切断する
object:self,agent,spot,currentspot,spotvariable
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[topic]
object:self,agent,spot,currentspot,spotvariable
variable:keyword,キーワード[message]
verb:id7,7,soars.library.adapter.pubsub.PubSub:popFilteredString,指定されたトピックと一致するもっとも古い受信済みメッセージを 1 つだけ取得する
```

図 4-14 「Publish/Subscribe」の GUI 定義スクリプト

4.1.1.8 ログ出力指定

図 4-15 は、この事例でのログ出力指定である．チェックボックスにより、「$Spot」「病状」及び「S」を出力するように指定する．「$Spot」は住民のいる場所(家または職場)、「病状」は住民の状態を表すキーワード(S=未感染、I=感染、R=恢復または死亡)、「S」はスポット「人数」の数値変数で未感染者数である．ログ出力指定された変数は、シミュレーション実行時に画面上でその遷移を確認することが出来る．

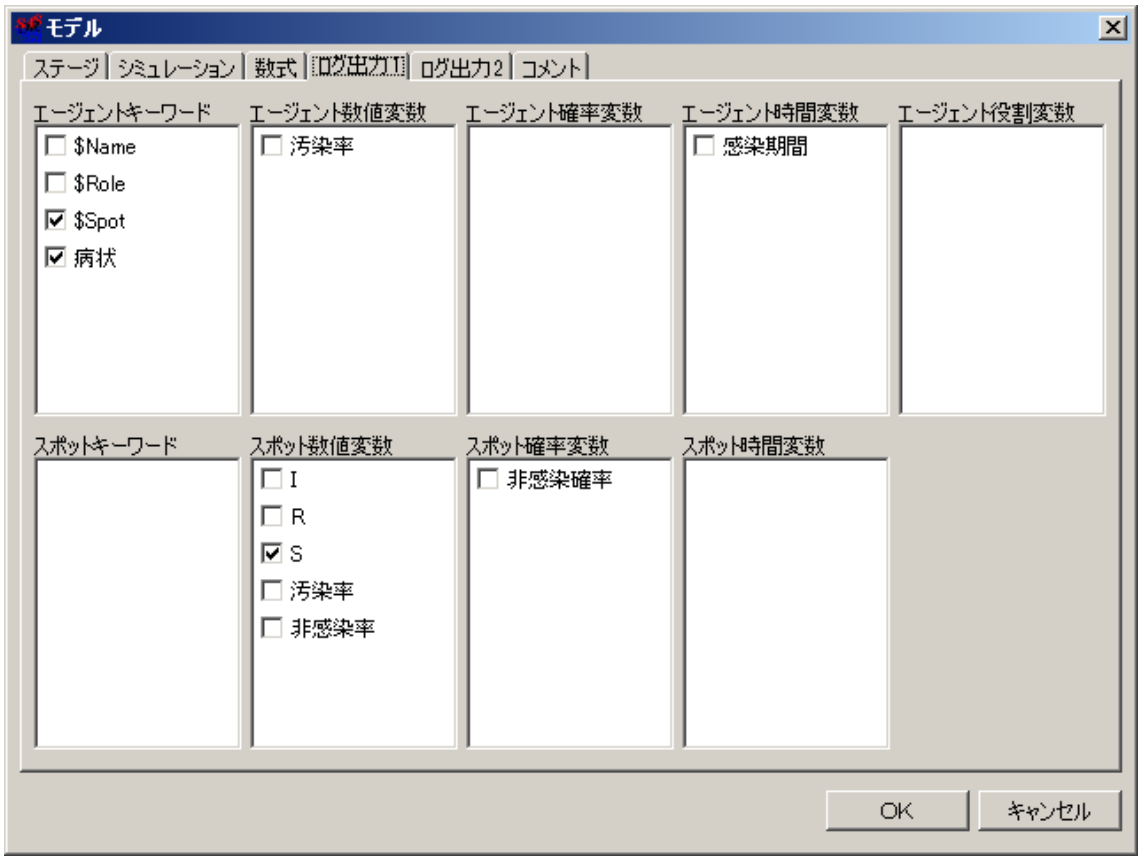


図 4-15 ログ出力指定用 GUI

4.1.1.9 シミュレーション条件編集

図 4-16 は、この事例のシミュレーション条件指定である．シミュレーション開始時刻は、初日の 9 時 0 分、繰り返し処理は 12 時間おきと指定する．終了時刻は 10 日 0 時 0 分としているが、実際には感染者数がゼロになるまで実行される．乱数シードを指定することにより、シミュレーション実行時の確率値の再現性を確保することが出来る．

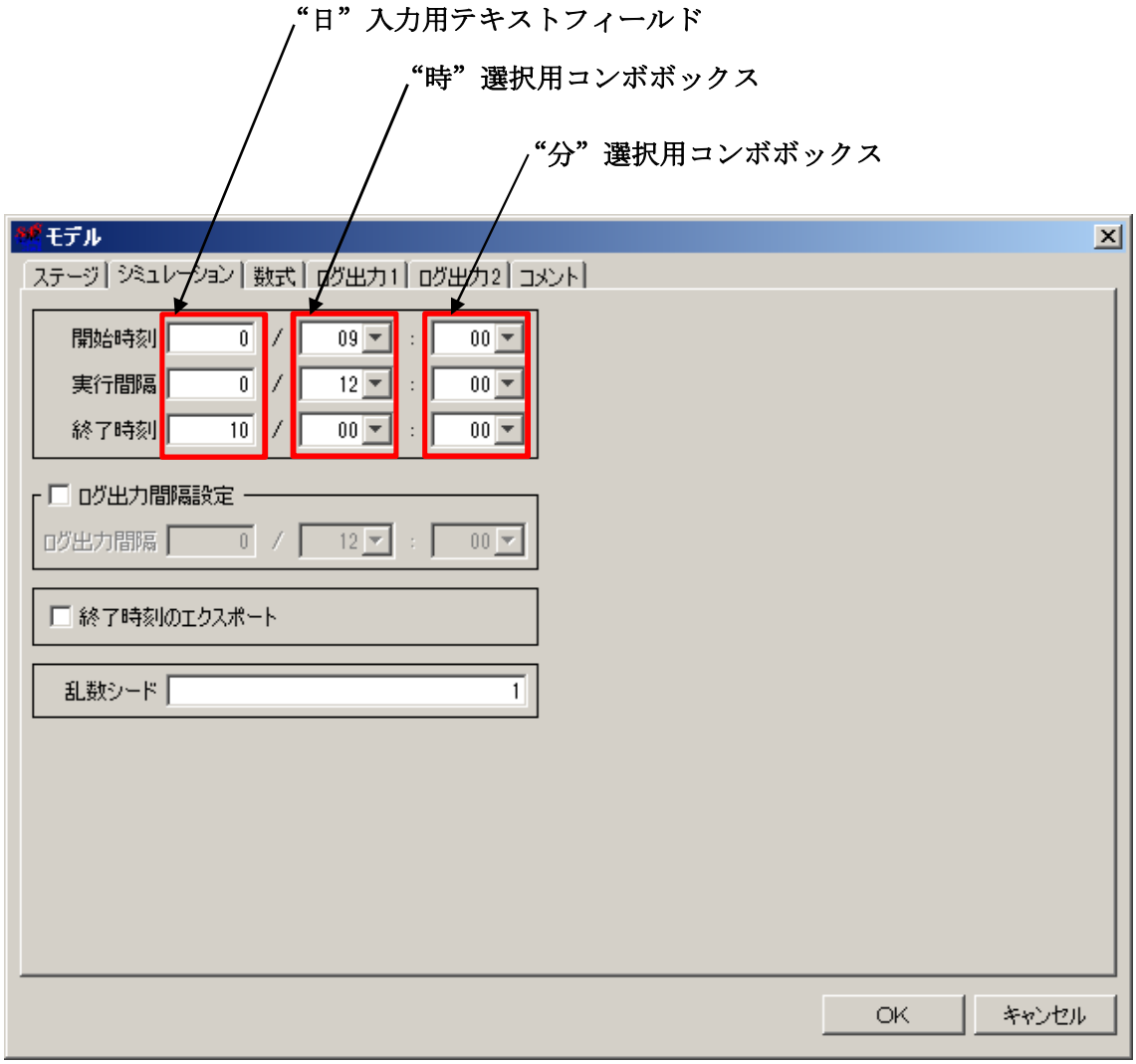


図 4-16 シミュレーション条件設定用 GUI

4.1.1.10 シミュレーションプロセス起動

図 4-17 に示すツールバーの「シミュレーション実行ボタン」を押すと、SOARS プログラムが自動的に生成されてシミュレーションが開始される．左側のボタンを押すとコンソールでの実行、右側のボタンを押すと GUI を持つ SOARS Simulator での実行となる．

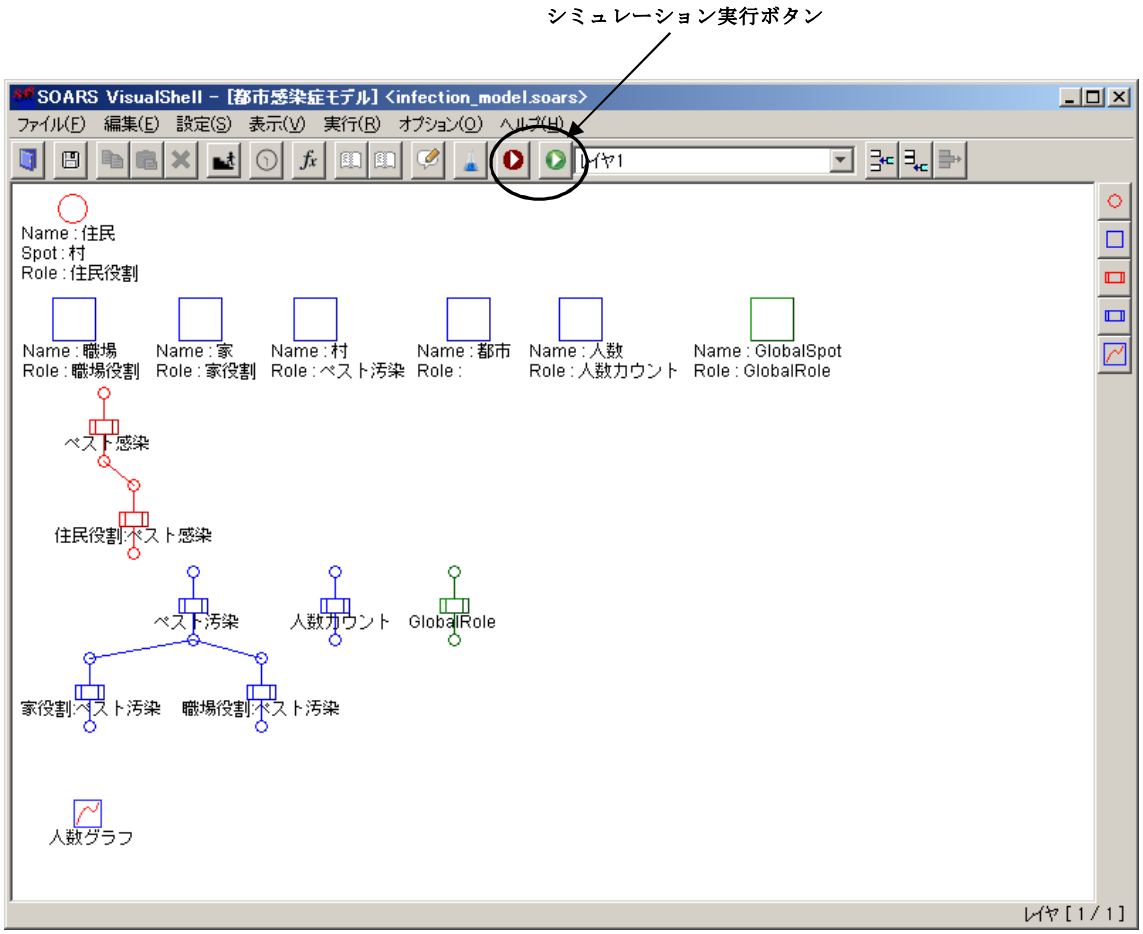


図 4-17 シミュレーションプロセス起動

4.1.2 SOARS Simulator のユーザインタフェース

図 4-18 は、この事例のシミュレーションが終了した時の SOARS Simulator の画面である．Log viewer ウィンドウでは、タブを切り替えることにより、図 4-19 のように、出力されたログの内容を確認することが出来る．人数グラフウィンドウには、チャート編集で表示指定した数値変数「S」、「I」及び「R」のチャートが表示されている．アニメーション開始ボタンを押すと SOARS Animator が起動する．SOARS Animator により、シミュレーションの内容は可視化することが出来る．

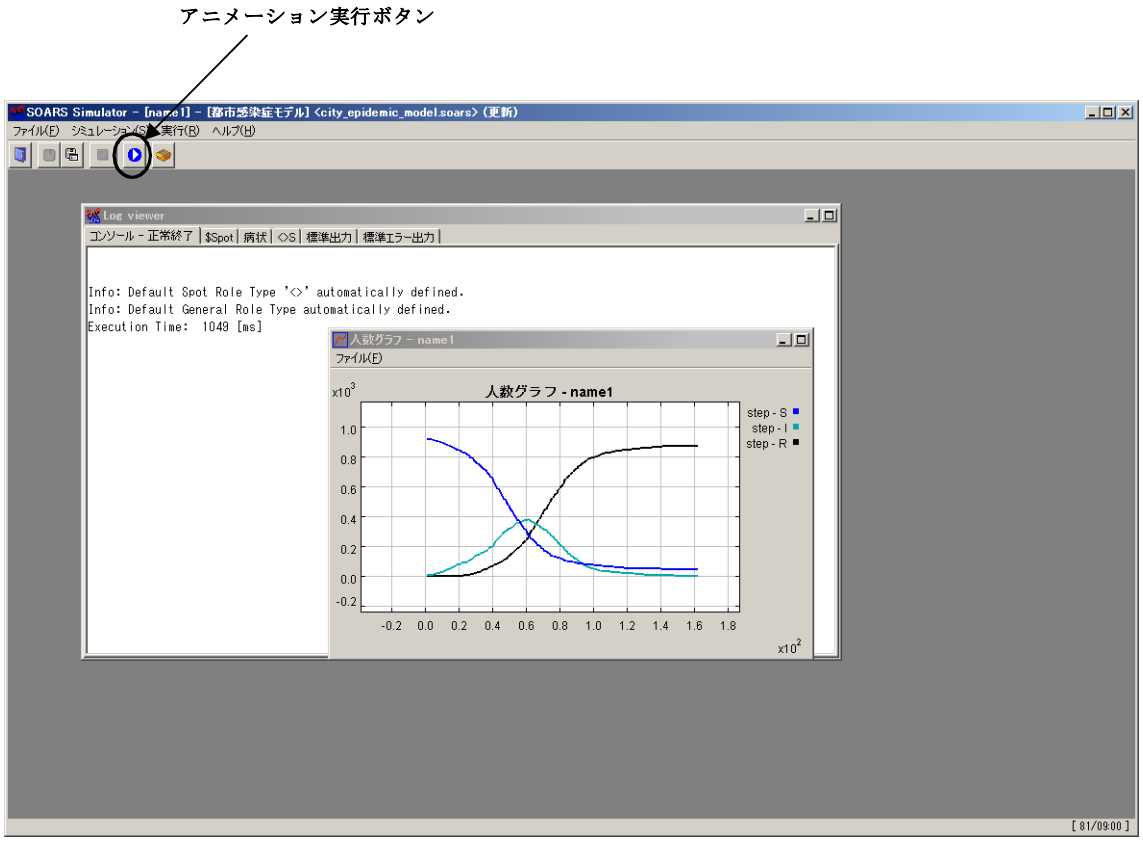
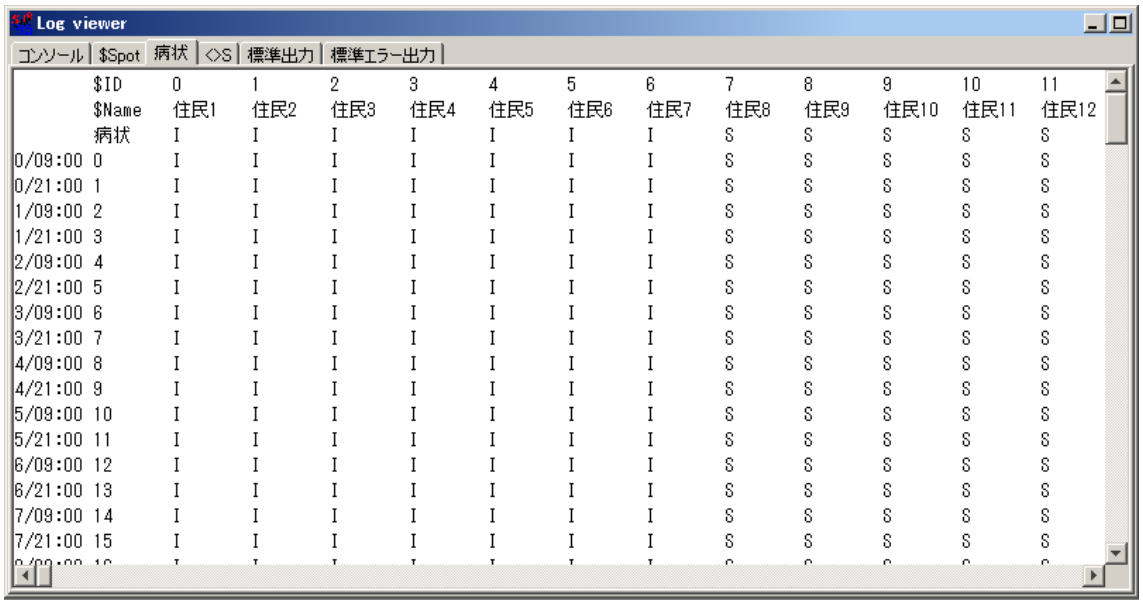


図 4-18 SOARS Simulator ウィンドウ



	$ID	0	1	2	3	4	5	6	7	8	9	10	11
	$Name	住民1	住民2	住民3	住民4	住民5	住民6	住民7	住民8	住民9	住民10	住民11	住民12
	病状	I	I	I	I	I	I	I	S	S	S	S	S
0/09:00	0	I	I	I	I	I	I	I	S	S	S	S	S
0/21:00	1	I	I	I	I	I	I	I	S	S	S	S	S
1/09:00	2	I	I	I	I	I	I	I	S	S	S	S	S
1/21:00	3	I	I	I	I	I	I	I	S	S	S	S	S
2/09:00	4	I	I	I	I	I	I	I	S	S	S	S	S
2/21:00	5	I	I	I	I	I	I	I	S	S	S	S	S
3/09:00	6	I	I	I	I	I	I	I	S	S	S	S	S
3/21:00	7	I	I	I	I	I	I	I	S	S	S	S	S
4/09:00	8	I	I	I	I	I	I	I	S	S	S	S	S
4/21:00	9	I	I	I	I	I	I	I	S	S	S	S	S
5/09:00	10	I	I	I	I	I	I	I	S	S	S	S	S
5/21:00	11	I	I	I	I	I	I	I	S	S	S	S	S
6/09:00	12	I	I	I	I	I	I	I	S	S	S	S	S
6/21:00	13	I	I	I	I	I	I	I	S	S	S	S	S
7/09:00	14	I	I	I	I	I	I	I	S	S	S	S	S
7/21:00	15	I	I	I	I	I	I	I	S	S	S	S	S

図 4-19 住民の病状遷移ログ

4.1.3 シミュレーションログの可視化機能

4.1.3.1 SOARS Animator による可視化

図 4-20 は、SOARS Animator により、この事例のシミュレーション結果をアニメーション表示している画面である．ウィンドウ①では、住民を表すアイコンが、家と職場を表すアイコンの間を移動する．ウィンドウ②では、住民の病状「"S"」、「"I"」及び「"R"」 を表すアイコンの間を、住民を表すアイコンが移動する．ウィンドウ③では、スポット「人数」の数値変数「S」の値集合に含まれる各値がアイコンで表され、それらの間を、スポット「人数」を表すアイコンが移動する．チャート表示ウィンドウでは、アニメーションの再生位置がチャート上に赤い点で表される．この赤い点はアニメーションの進行に合わせて移動する．

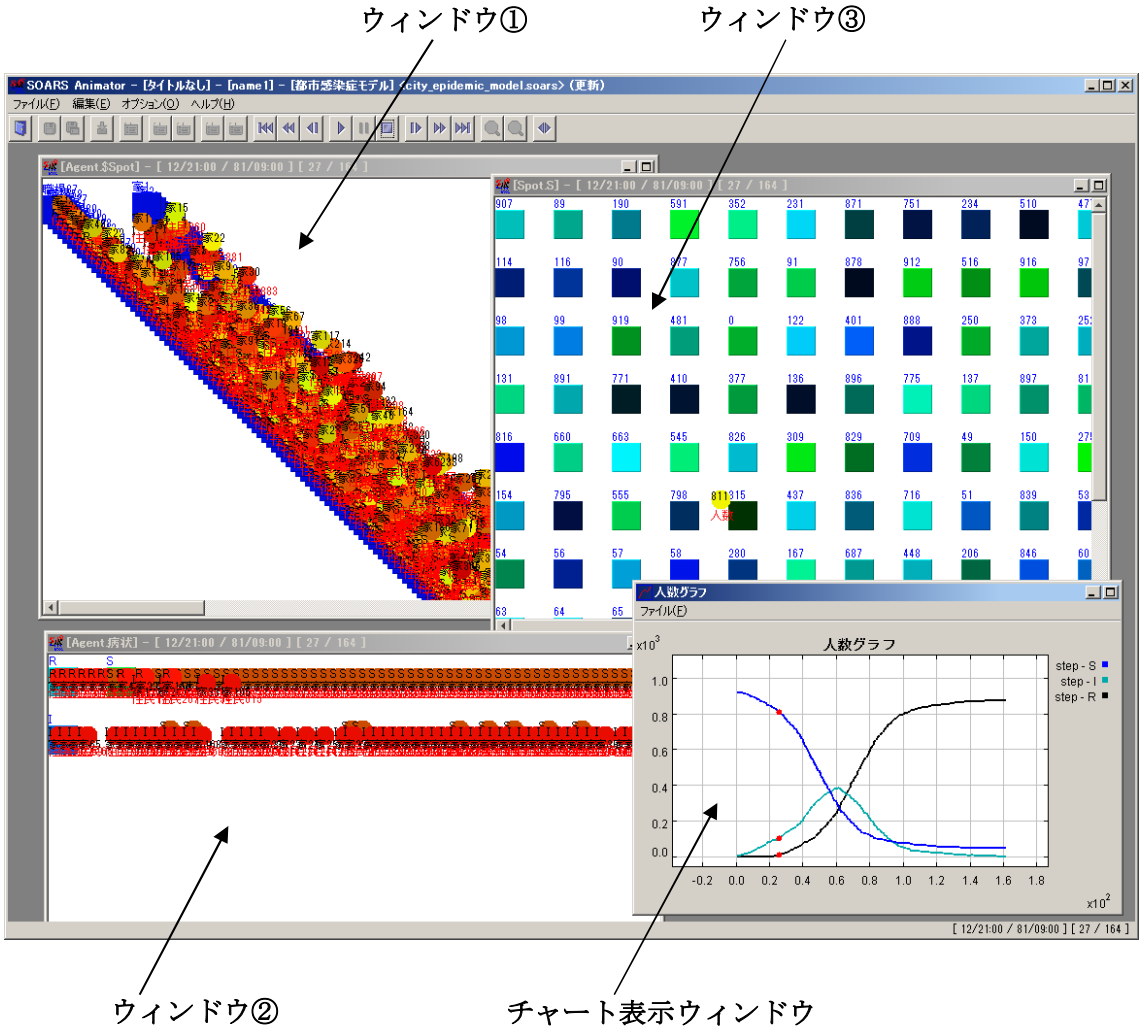


図 4-20 SOARS Animator ウィンドウ

4.1.3.2 SOARS LogViewer による可視化

図 4-21 は、この事例のログ出力を可視化したものではないが、ログ出力内容を３次元可視化出来る SOARS LogViewer の画面である．各エンティティが保持する３つの数値変数の値から３次元座標を生成し、その３次元座標集合を 3DCG として可視化することが出来る．

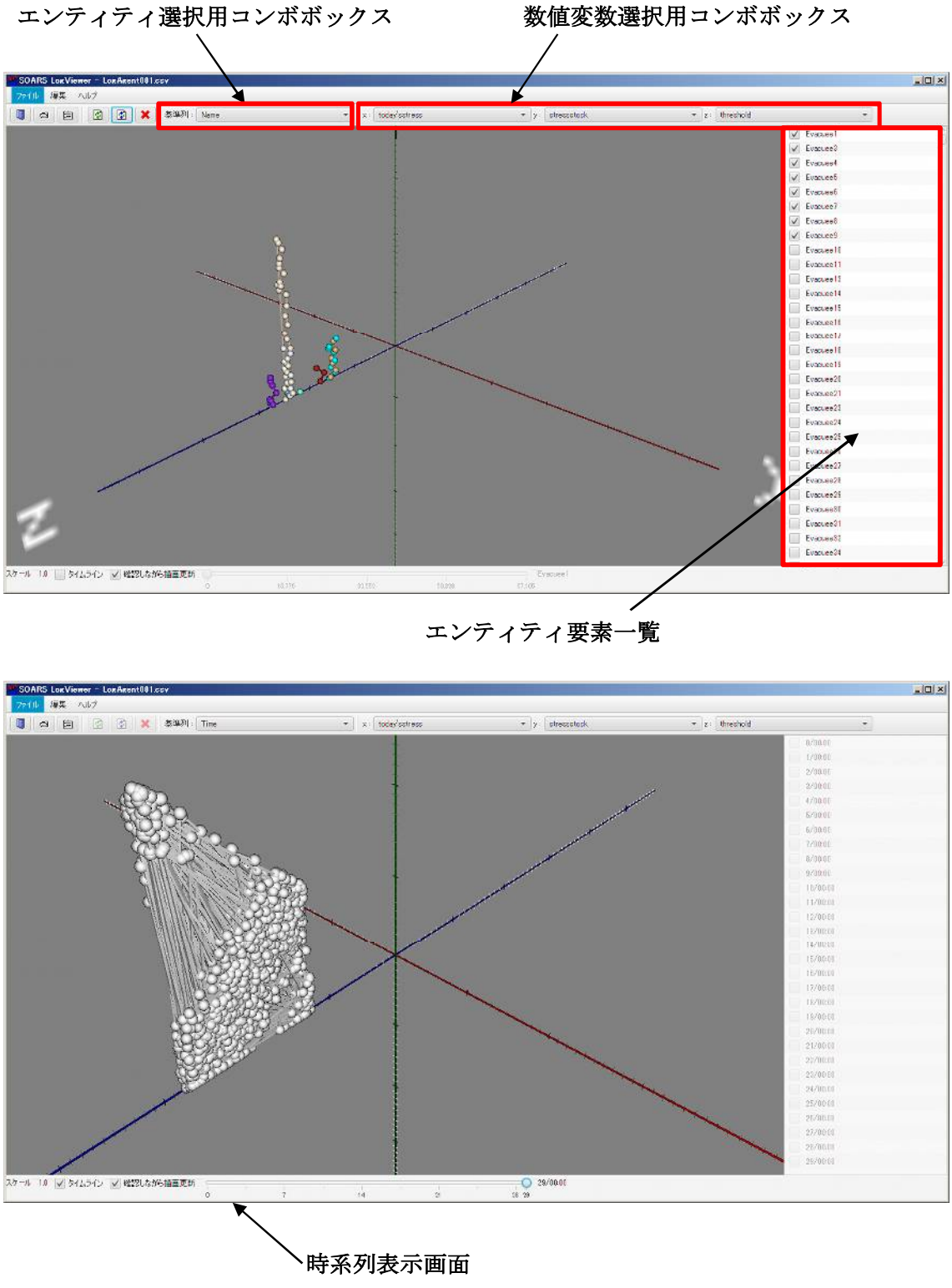


図 4-21 SOARS LogViewer ウィンドウ

4.2 モデリング GUI としての RWOS Program Editor

RWOS Program Editor により、3.3.1 で述べた、温度センサーからのデータ及び Web ブラウザからのデータに基づいてパトライト点灯色を決定して点灯させると云う実験事例、及びの RWOS プログラムを作成して実行した.

4.2.1 システム構成

プログラム事例実行は、MQTT ブローカーにより、神奈川県横浜市南区、神奈川県横浜市鶴見区及び長野県飯田市の 3 箇所を繋いで行った. MQTT ブローカーには mosquitto[mosquitto 2009]、HTTP サーバには apache[apache 1995], [Ben ほか 2003]、温度センサ回路には Arduino[Arduino 2005], [Massimo 2009], [高橋 2013]互換マイコン ESP-WROOM-02 及び温度センサ AE-HDC1000 をした. 表 4-1 及び図 4-22～図 4-24 にそのシステム構成を示す.

表 4-1 RWOS 事例実行のシステム構成

場所	デバイス	ソフトウェア	説明
神奈川県横浜市南区	PC(Debian Wheezy)	mosquito	MQTT ブローカー
		apache	HTTP サーバ
	ESP-WROOM-02 AE-HDC1000	Arduino スケッチ	・無線 LAN 機能付きの Arduino 互換マイコンと温度センサの回路 ・トピック「home/1」で 5 秒に 1 回温度データを JSON フォーマットのメッセージでブローカーへパブリッシュ
神奈川県横浜市鶴見区	PC(Debian Wheezy)	ロールコンテナ sensor1 sensor2 human calc	sensor1、sensor2、human 及び calc の各ステージで使用されるロールコンテナ
	ESP-WROOM-02 AE-HDC1000	Arduino スケッチ	・無線 LAN 機能付きの Arduino 互換マイコンと温度センサの回路 ・トピック「office/0」で 5 秒に 1 回温度データを JSON フォーマットのメッセージでブローカーへパブリッシュ

長野県飯田市	PC(MacBook Air)	RWOS Program Editor	RWOS プログラムの編集と RWOS Project Manager の起動
		RWOS Project Manager	RWOS プログラムの実行
		ロールコンテナ actuator	actuator ステージで使用されるロールコンテナ
	パトライト		ロールコンテナ actuate から制御される
	iPhone	Java script	Web ブラウザ上から、HTTP サーバ経由、トピック「human/1」でパトライト点灯色を JSON フォーマットのメッセージでブローカーへパブリッシュ

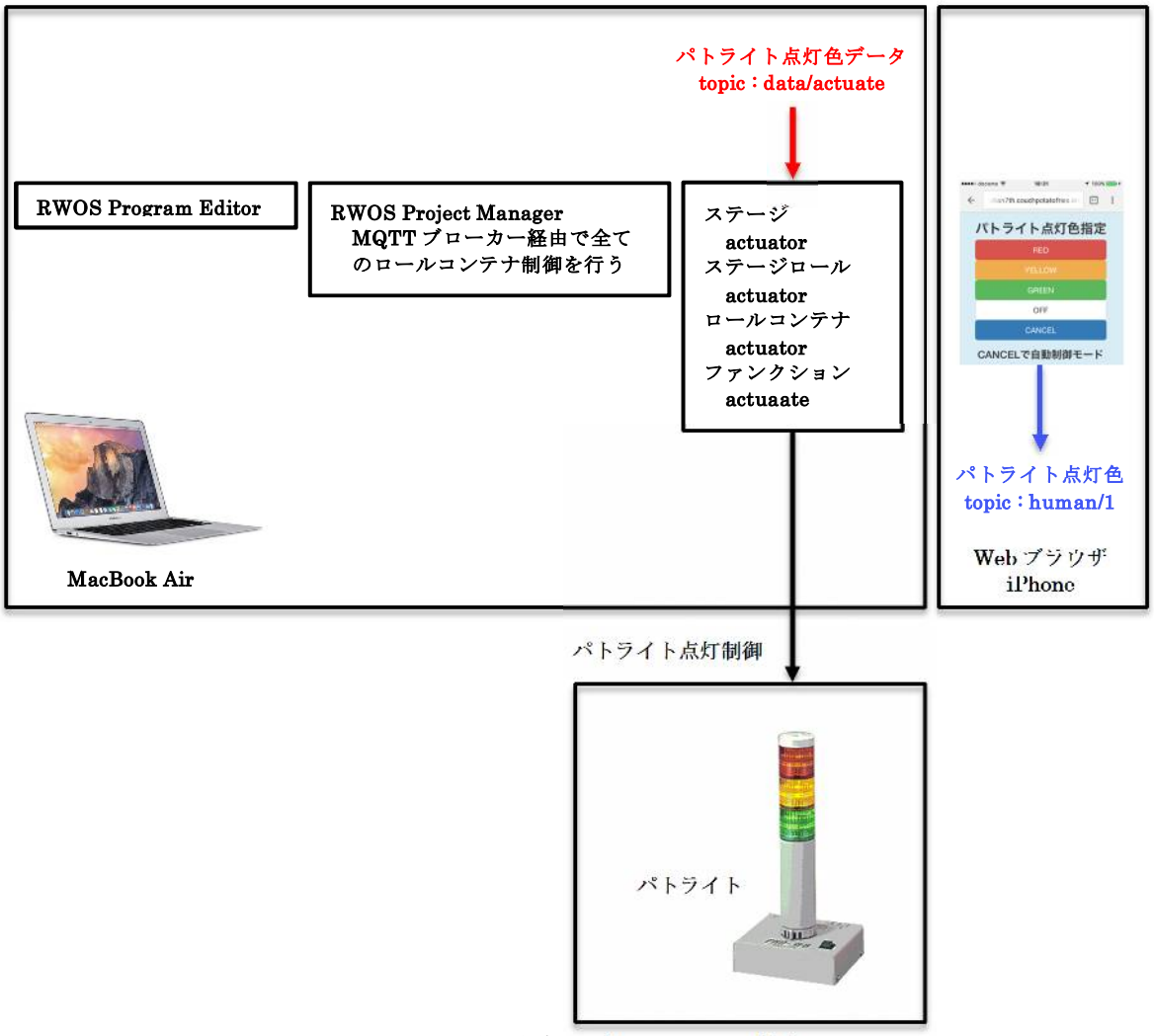


図 4-22 長野県飯田市のシステム構成

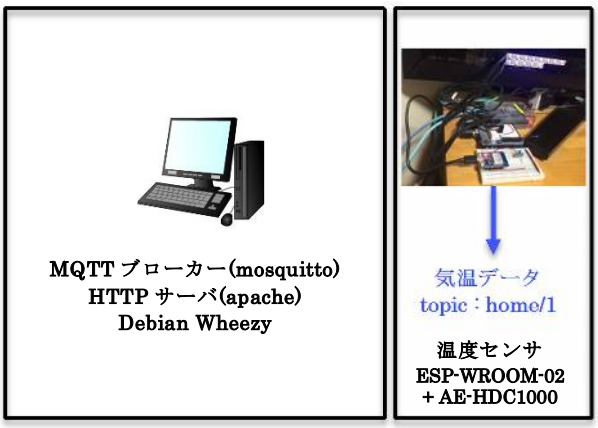


図 4-23 神奈川県横浜市南区のシステム構成

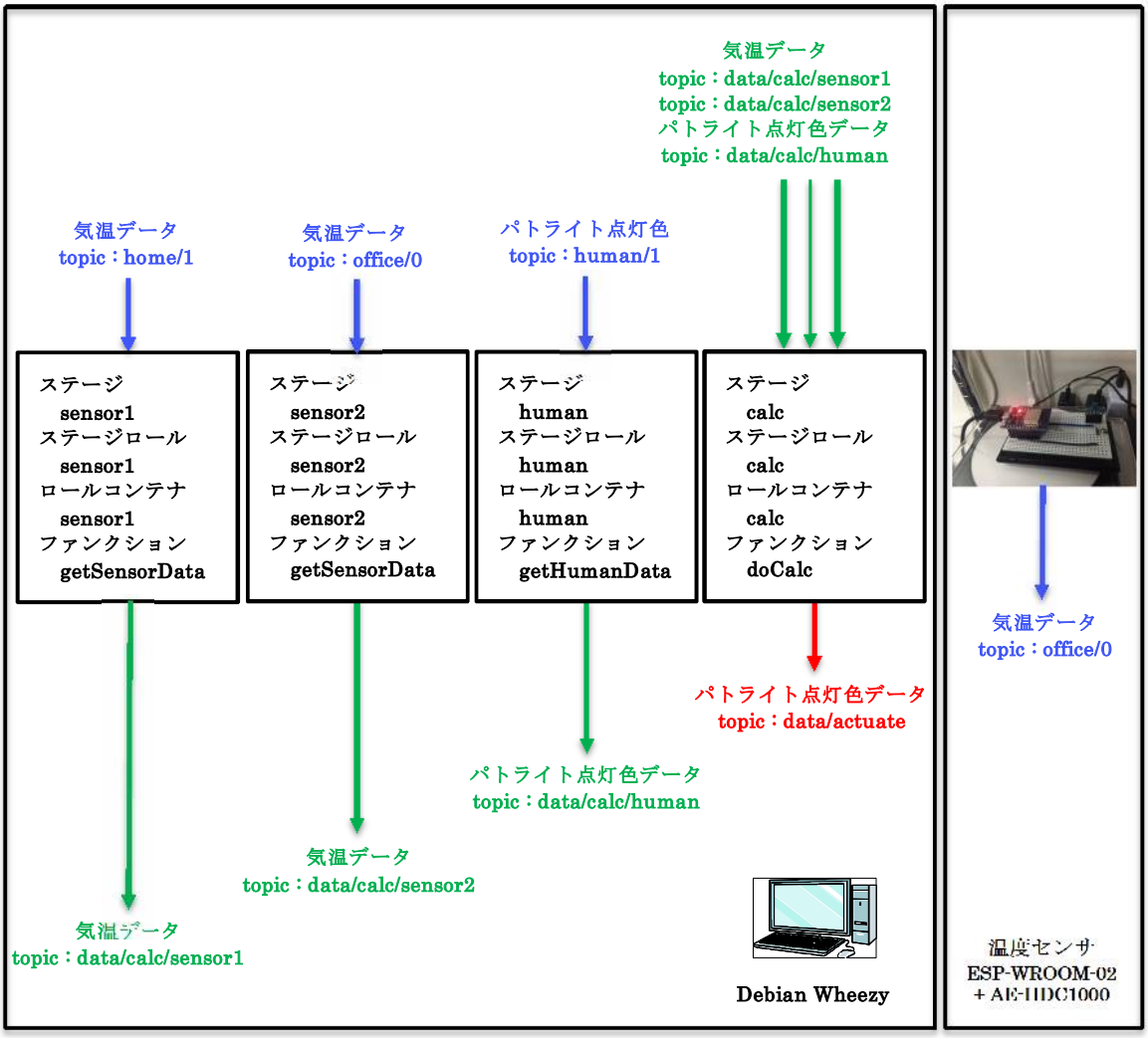


図 4-24 神奈川県横浜市鶴見区のシステム構成

4.2.2 RWOS Program Editor のユーザインタフェース

4.2.2.1 ステージ編集

図 4-25 は、RWOS Program Editor における、この事例のステージ編集画面である．RWOS Program Editor では、プロジェクトツリーからプロジェクトを選択して編集を行う．

この事例では、sensor1、sensor2、human、calc 及び actuator と云う 5 つのステージを作成する．では、ステージアイコン calc が選択されているので、ステージ内容編集領域ではステージ calc の内容を編集することが出来る．

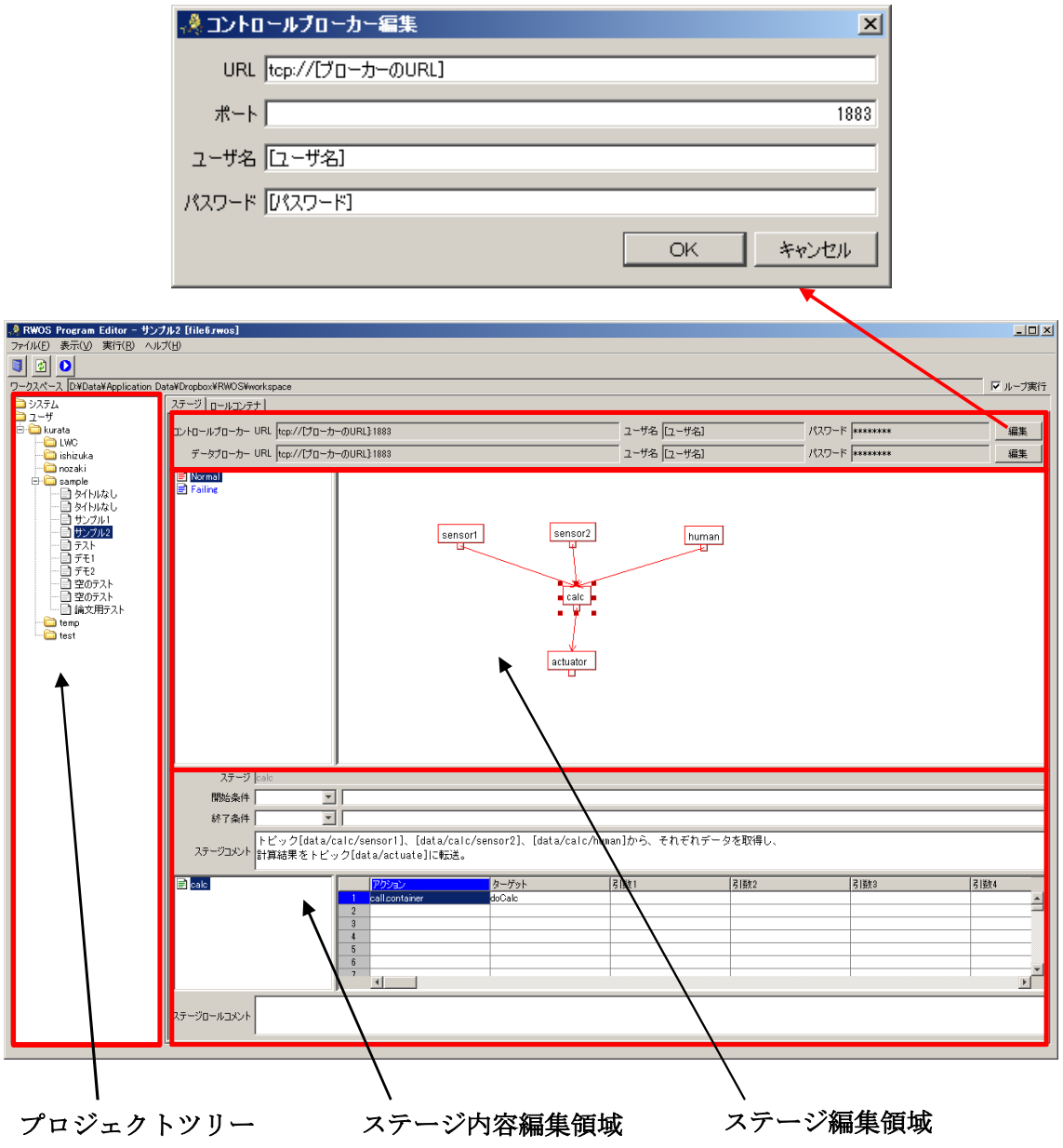


図 4-25 ステージ編集用 GUI

4.2.2.2 ロールコンテナ編集

図 4-26 は、RWOS Program Editor における、この事例のロールコンテナ actuator 編集画面である．ロールコンテナ actuator は、唯一の命令配列であるファンクション actuate を保持している．ファンクション actuate は、サブスクライブにより受信済みのパトライト点灯色があれば、その点灯色でパトライトを点灯させる．

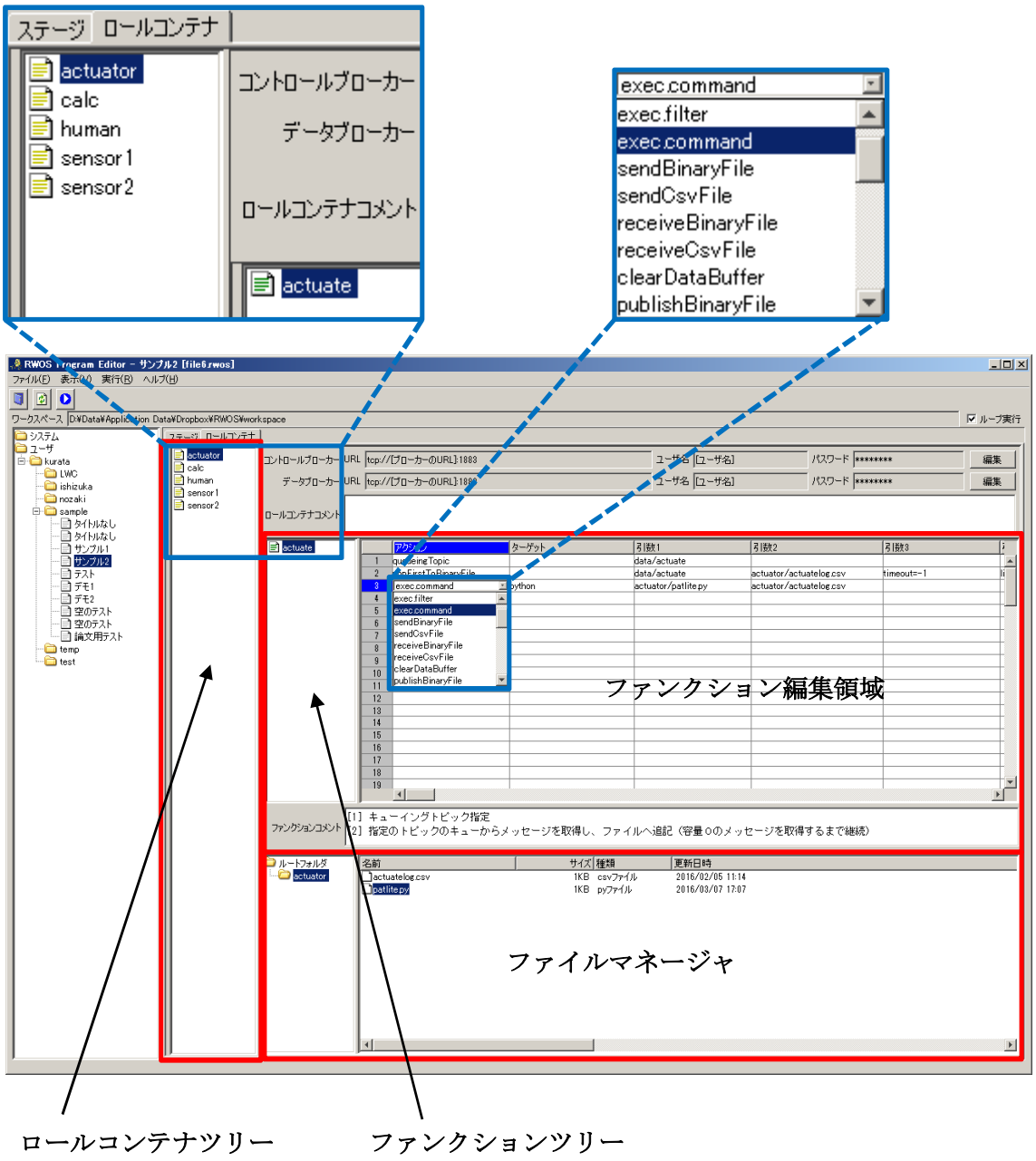


図 4-26 ロールコンテナ編集用 GUI

パトライトを点灯させるプログラムは patlite.py と云う Python プログラムであり、図 4-27 のようにドラッグ＆ドロップにより、正しい相対パスでファンクション編集領域のスプレッドシート上のセルへ設定することが出来る.

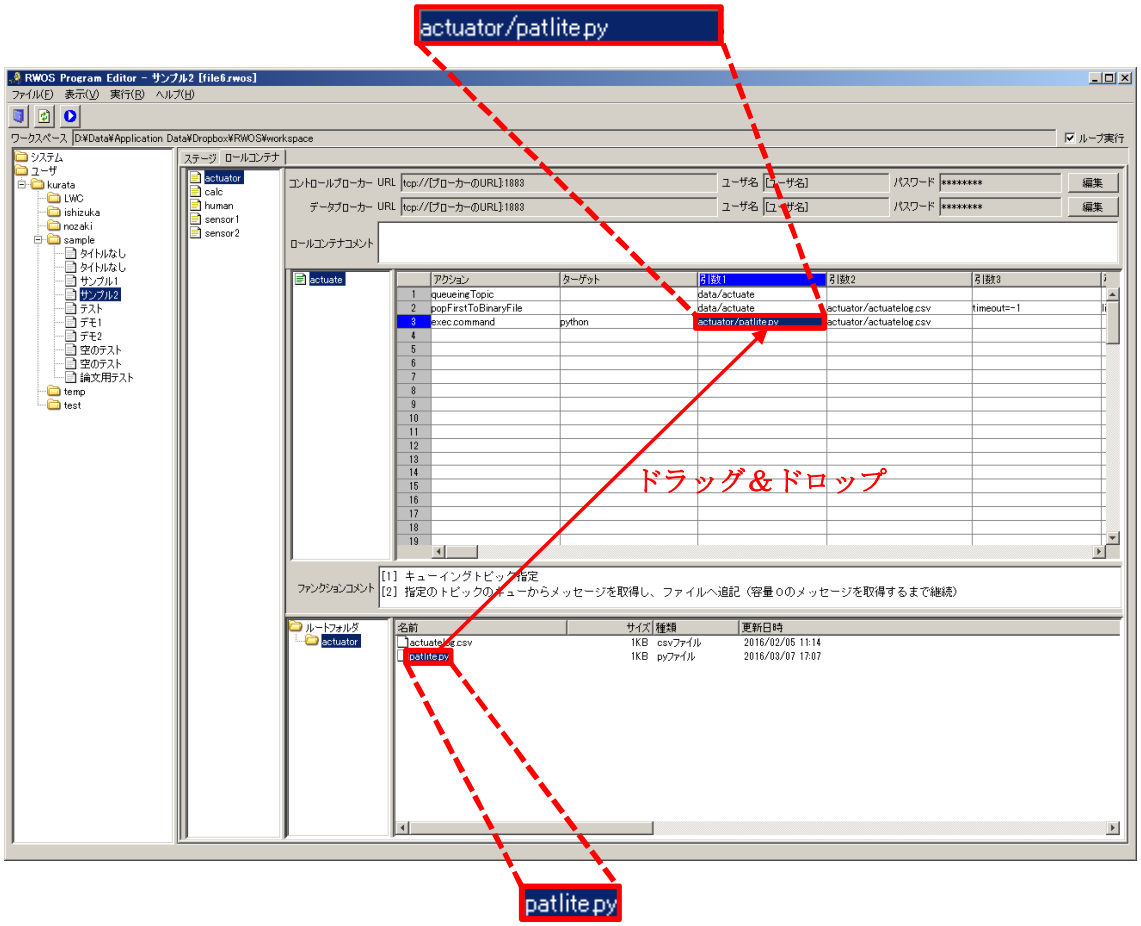


図 4-27 ドラッグ＆ドロップによるスプレッドシート上のセルへのファイル設定

表 4-2 ロールコンテナ及びそのファンクションに、この事例におけるロールコンテナとそのファンクションを示す.

表 4-2 ロールコンテナ及びそのファンクション

ロールコンテナ	ファンクション	処理内容
sensor1	geSensorData	温度センサーから、トピック「home/1」でパブリッシュされた JSON フォーマットのメッセージ(温度データ)をサブスクライブし、json2csv.py と云う Python プログラムで CSV フォーマットへ変換してパブリッシュする

sensor2	geSensorData	温度センサーから、トピック「office/0」でパブリッシュされた JSON フォーマットのメッセージ(温度データ)をサブスクライブし、json2csv.py と云う Python プログラムで CSV フォーマットへ変換してパブリッシュする
human	getHumanData	Web ブラウザ上から、HTTP サーバ経由、トピック「human/1」でパブリッシュされたメッセージ(パトライト点灯色)をサブスクライブし、json2csv.py と云う Python プログラムで CSV フォーマットへ変換してパブリッシュする
calc	doCalc	getSensorData がパブリッシュしたメッセージ(温度データ)及び getHumanData がパブリッシュしたメッセージ(パトライト点灯色)をサブスクライブし、QueueingCalc.med と云うプログラムで処理する．QueueingCalc.med は、パトライト点灯色が指定されていればその色をパブリッシュする．パトライト点灯色が指定されていなければ、2 つの温度データの平均値によりパトライト点灯色を決定してその色をパブリッシュする
actuator	actuate	doCalc がパブリッシュしたメッセージ(パトライト点灯色)をサブスクライブし、patlite.py と云う Python プログラムでパトライト制御を行う．

ロールコンテナツリー上で選択されているロールコンテナについては、メニュー操作によりその実行形式(jar ファイル)の生成が行えるようになっている．

4.2.2.3 ステージ内容編集

図 4-28 は、RWOS Program Editor における、この事例のステージ calc のステージ内容編集画面である．ステージ calc で実行されるステージロール calc を作成している．このステージロール calc には、アクションとして call.container を、ターゲットとして doCalc を指定しているので、プロジェクト実行時にはいずれかのロールコンテナのファンクション

doCalc が実行される．この事例のプロジェクトにおいて、doCalc と云うファンクションを持つロールコンテナは calc 唯一つである．従って、プロジェクト実行時に、ステージ calc においては、ロールコンテナ calc のファンクション doCalc が実行される．

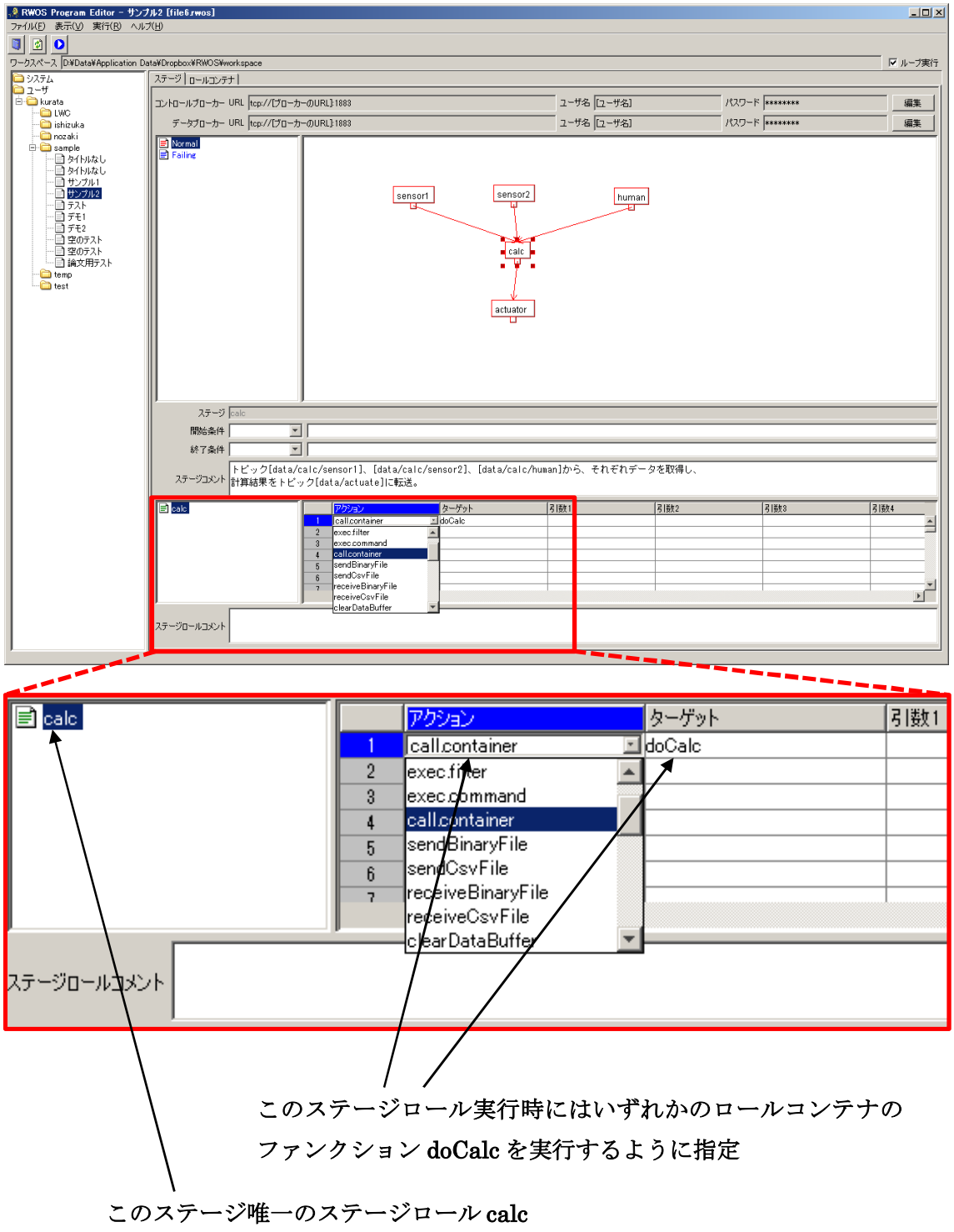


図 4-28 ステージ内容編集用 GUI

表 4-3 ステージ、ステージロール、ロールコンテナ及びそのファンクションの関係に、この事例におけるステージ、ステージロール、ロールコンテナ及びそのファンクションの関係を示す.

表 4-3 ステージ、ステージロール、ロールコンテナ及びそのファンクションの関係

ステージ	ステージロール	ロールコンテナ	ファンクション
sensor1	sensor1	sensor1	geSensorData
sensor2	sensor2	sensor2	geSensorData
human	human	human	getHumanData
calc	calc	calc	doCalc
actuator	actuator	actuator	actuate

4.2.2.4 RWOS プログラムの実行

RWOS プログラムを実行する前に、必要な全てのロールコンテナ実行形式(jar ファイル)を起動し、図 4-29 に示すツールバーの「プロジェクト実行ボタン」を押すと RWOS プログラムが自動的に生成されて、RWOS Project Manager がそのプログラムを実行する.

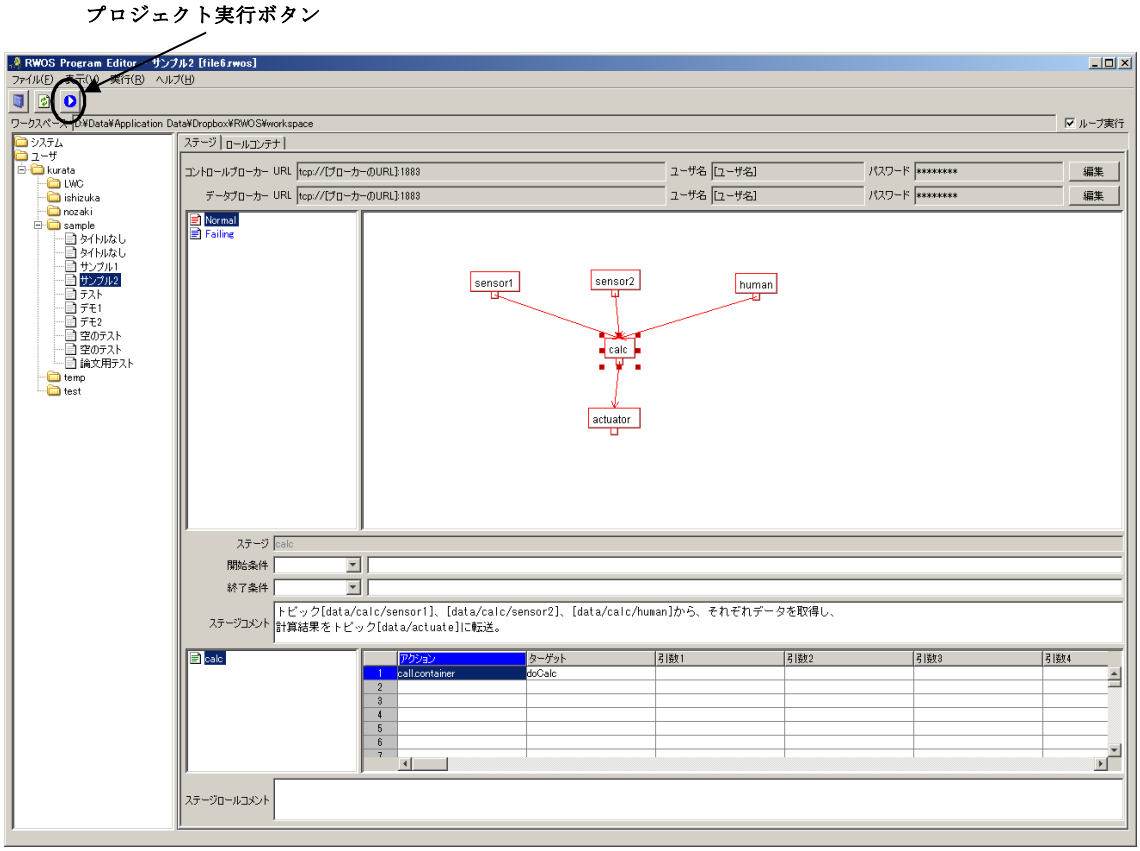


図 4-29 RWOS プログラムの実行

RWOS Project Manager は、図 4-30 のように、実行中のステージをリアルタイムに表示する．また、エラーが発生した場合には、情報表示用コンソールにそのエラー内容が表示される．

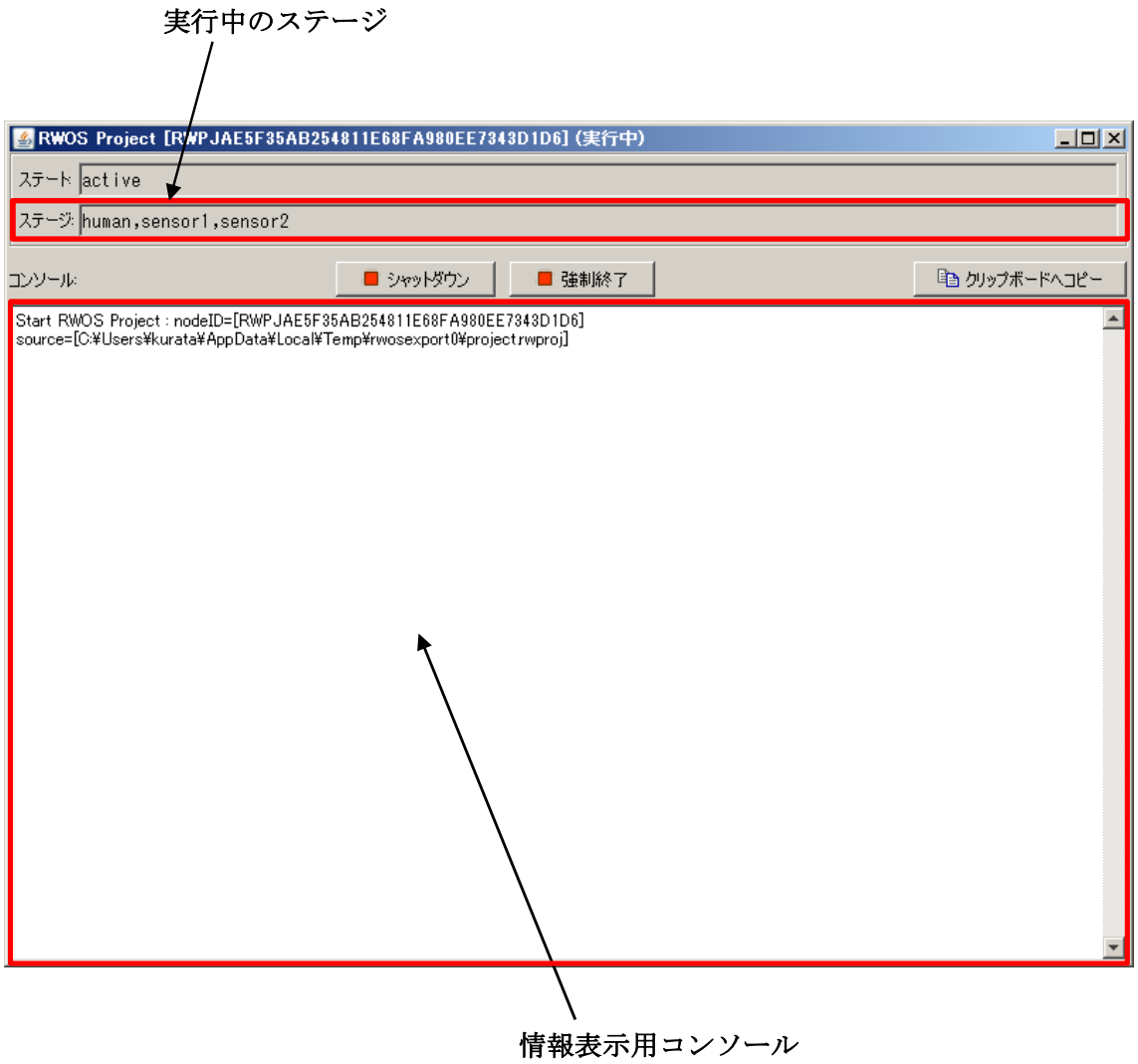


図 4-30 RWOS Project Manager ウィンドウ

第5章 結論

本章では、本研究で設計・実装を行った SOARS VisualShell 及び RWOS Program Editor の使用実績について述べた後、今後の課題について述べる.

5.1 本研究の成果

本研究では、特定モデル化手法におけるモデル化手法特化型プログラム言語により、モデル作成メソッドが確立され、そのプログラム言語によるプログラムを MMDA により開発することを可能にするモデリング GUI を構築することが可能であることを示した. このモデリング GUI は、特定領域専門家がマウス操作と僅かなキーボード入力によりモデル記述を行うだけでプログラムを作成して実行することを可能にする. また、GUI の拡張及びプログラム実行の可視化も可能にする.

モデリング GUI はそのモデル化手法特化型プログラム言語の命令や書式を覚えなくても利用出来るので、そのモデル化手法の専門家でなくても容易に利用することが出来る. このことはモデリング GUI を通して特定モデル化手法を学ぶことも出来ることを意味していると言える.

更に、特定モデル化手法において、モデリング GUI を実現出来るモデル化手法特化型プログラム言語を作成することが出来れば、そのプログラム言語によるプログラムを MMDA により開発することを可能にするモデリング GUI を設計・実装することが可能であることが明らかになった.

5.1.1 SOARS Workshop での実績

本研究では、役割エージェントベースシミュレーションのモデル化手法特化型プログラム言語である SOARS によるプログラムを MMDA により開発する為のモデリング GUI として SOARS VisualShell の設計・実装を行った. エージェントベースモデリングの特定領域専門家は SOARS VisualShell により役割指向エージェントベースシミュレーションモデルを作成してシミュレーションを実行することが出来る. 必要とされるのは、マウス操作と僅かなキーボード入力だけであり、テキストエディタによるプログラミングが不要なので命令及びその書式を覚える必要が無い. また、キーボード入力による文字列は自動的にチェックされるので構文エラーは起こり得ない. 従って、ユーザはプログラムのセマンティックなバグの解析のみに専心出来る.

SOARS プロジェクト[SOARS プロジェクト 2004]では毎年 SOARS Workshop を開催しているが、SOARS VisualShell を利用することにより、少人数の TA(Teaching Assistant)による僅か数時間のチュートリアルで誰もが役割指向エージェントベースシミュレーションを行えるようになっている.

5.1.2 SOARS VisualShell の改修作業工数短縮の実績

モデリング GUI は、その設計・実装に比較的大きな工数が発生する．これは GUI 実装やエラー処理等の煩雑なプログラミングが必要となるからである．SOARS ではルールが頻繁に変更される．新たなルールが必要になるケース、既存のルールの引数を変更する必要が発生するケース等がこれにあたる．それに伴い、SOARS VisualShell のルール編集用 GUI の改修を行う必要が発生するが、これは非常に大きな負担であり、また利用者もすぐに新しい機能を使うことが出来ない．そこで、本研究では、SOARS が持つ言語仕様の拡張性を利用して、SOARS VisualShell のルール編集用 GUI を定義する為の新たなスクリプト言語を作成し、SOARS VisualShell にはこの言語で記述されたスクリプトを解釈して GUI 画面を生成する機能を実装した．通常、SOARS VisualShell のルール編集用 GUI の改修には数日～２週間を要するが、この機能によりルールの変更は全て利用者が短時間で行うことが出来るようになった．

また、SOARS VisualShell にこのスクリプト言語のインタプリタを実装したことにより、ルール編集用 GUI に、利用者が任意の Java プログラム(jar ファイル)に含まれるメソッドを呼び出す機能を自由に追加出来るようになった．これにより、利用者は SOARS には無い機能をすぐに使えることになった．

5.1.3 RWOS Program Editor の実装例での稼働実績

本研究では、実世界と仮想世界を繋ぐオペレーティングシステムのモデル化手法特化型プログラム言語である RWOS によるプログラムを MMDA により開発する為のモデリング GUI として RWOS Program Editor の設計・実装を行った．RWOS Program Editor では、プロジェクト及びロールコンテナがライブラリ化されている為、これらをブロックのように組み合わせて RWOS プログラムを作成して実行することが出来る．

RWOS Program Editor は既に実際の現場で利用されつつある

竹中工務店は、プロジェクト「次世代型技術でできる省エネ＜ゼロカーボンをめざして＞」の中で、既に RWOS を利用した照明機器制御実験を行っている．今後は、RWOS Program Editor を利用することにより、更に他の分野へも RWOS 利用を展開する為に現在共同で研究を進めている．

また、長野県の或る企業は、気象庁が設置した気象センサーから得られるデータを加工してエンドユーザへ配信するシステムの中で RWOS を利用する予定である．既に RWOS Program Editor を使用してプロジェクト及びロールコンテナを作成し、その正常動作確認も完了している．この RWOS プログラムが行う処理は、15 年程前に気象庁が設置した長野県八方尾根スキー場の気象センサーから６秒間隔で出力されるシリアルデータをサブスクライブし、これを指定のフォーマットへ変換してパブリッシュすると云うものである．作業は RWOS Program Editor 上での編集及びフォーマット変換用 Python プログラムの作成だけなので１時間程度で完了した．ステージ名、ステージロール名及びロールコンテナ名

は conversion、ファンクション名は convert である．図 5-1 は RWOS Program Editor のステージ編集画面、図 5-2 は RWOS Program Editor のロールコンテナ編集画面、図 5-3 はフォーマット変換用 Python プログラムである．また、図 5-4 は実行中の RWOS Project Manager、図 5-5 は動作確認画面である．

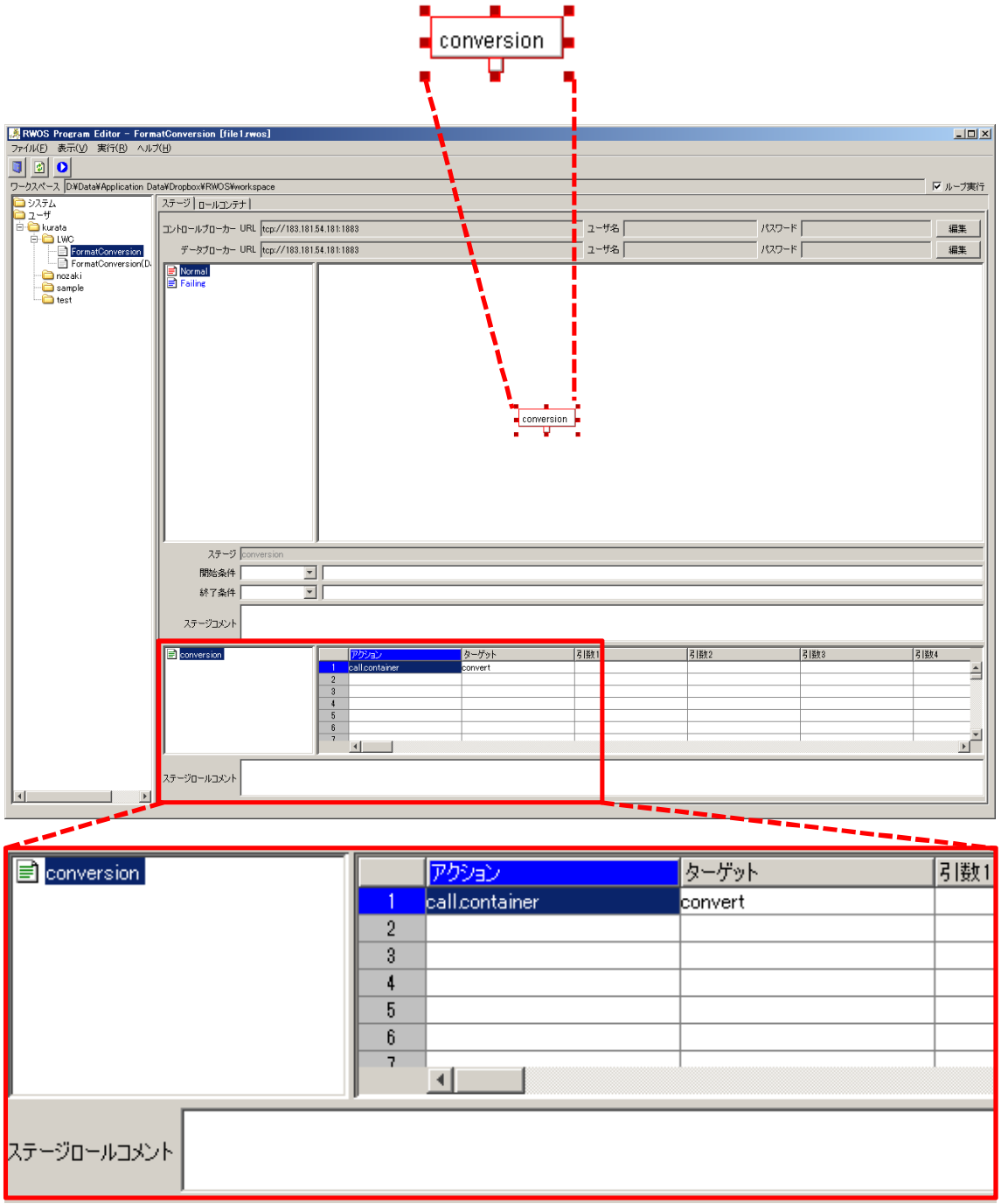


図 5-1 ステージ編集画面

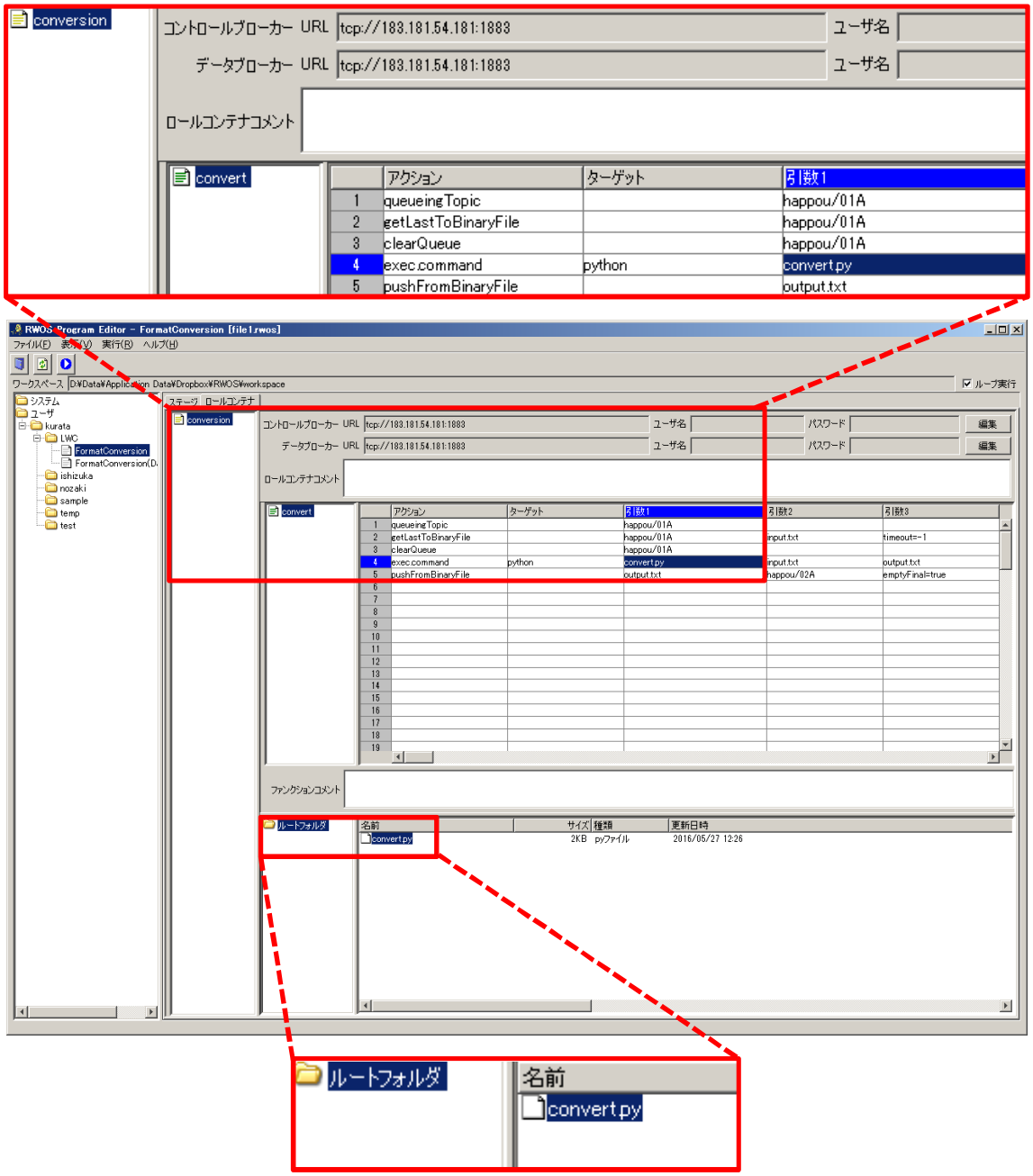


図 5-2 ロールコンテナ編集画面

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import os.path
import sys


def convert(ifilename, ofilename):
	if False == os.path.exists( ifilename):
		return

	ifile = open( ifilename, 'r')
	str = ifile.readline()
	str = str.rstrip( '¥n')
	ifile.close()

	if 56 > len( str):
		return

	line = str[1]
	line += str[3:54]
	line += str[55]

	ofile = open( ofilename, 'w')

	ofile.write( "対象文字列  " + line + "¥n")
	ofile.write( "[地点番号]          " + line[1:4] + '¥n')
	ofile.write( "[FM500 時計]      " + line[4:14] + '¥n')
	ofile.write( "[構成要素]          " + line[14:16] + '¥n')
	ofile.write( "[瞬間風向]          " + line[16:19] + '¥n')
	ofile.write( "[平均風向]          " + line[19:22] + '¥n')
	ofile.write( "[瞬間風速]          " + line[22:25] + '¥n')
	ofile.write( "[平均風速]          " + line[25:28] + '¥n')
	ofile.write( "[温度]              " + line[28:32] + '¥n')
	ofile.write( "[湿度]              " + line[32:36] + '¥n')
	ofile.write( "[日積算降水量]    " + line[36:40] + '¥n')
	ofile.write( "[オプション入力] " + line[40:44] + '¥n')
	ofile.write( "[オプション入力] " + line[44:48] + '¥n')
	ofile.write( "[オプション入力] " + line[48:52] + '¥n')

	ofile.close()

if __name__ == '__main__':
	argv = sys.argv
	convert( argv[ 1], argv[ 2])
```

図 5-3 フォーマット変換用 Python プログラム

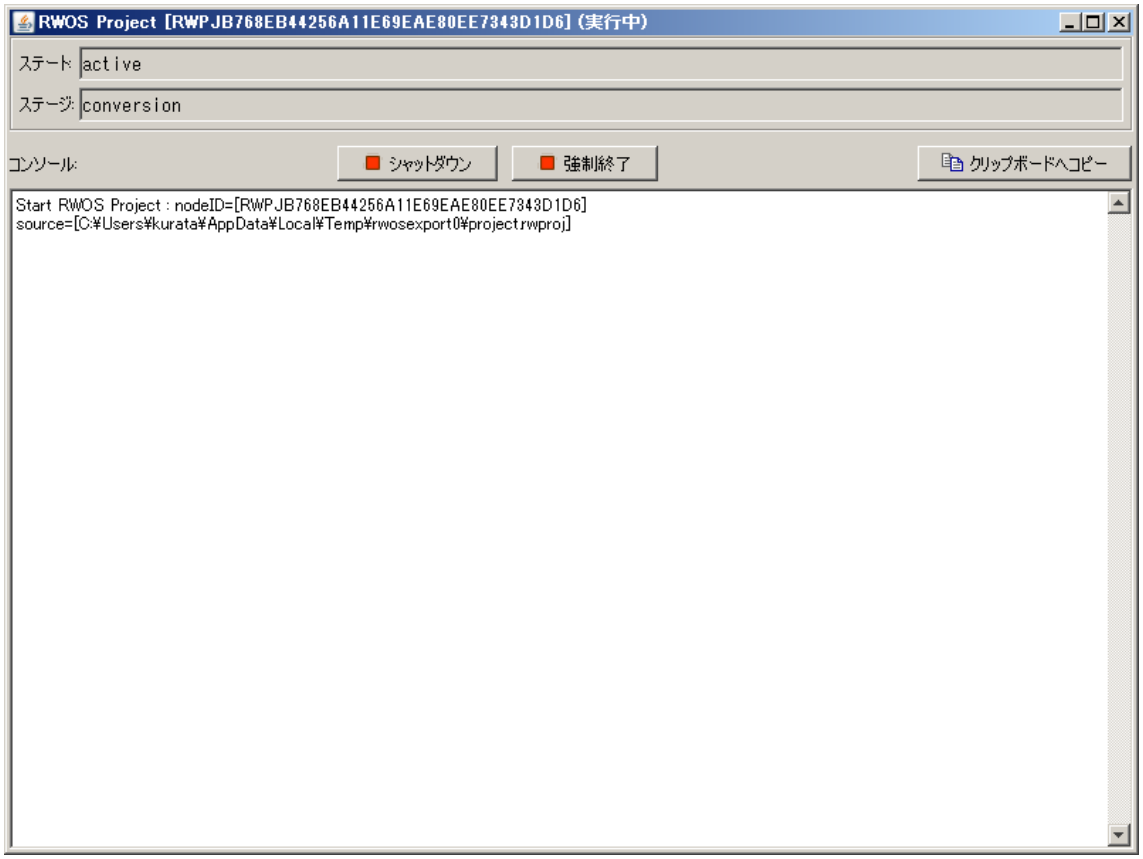


図 5-4 実行時の RWOS Project Manager ウィンドウ

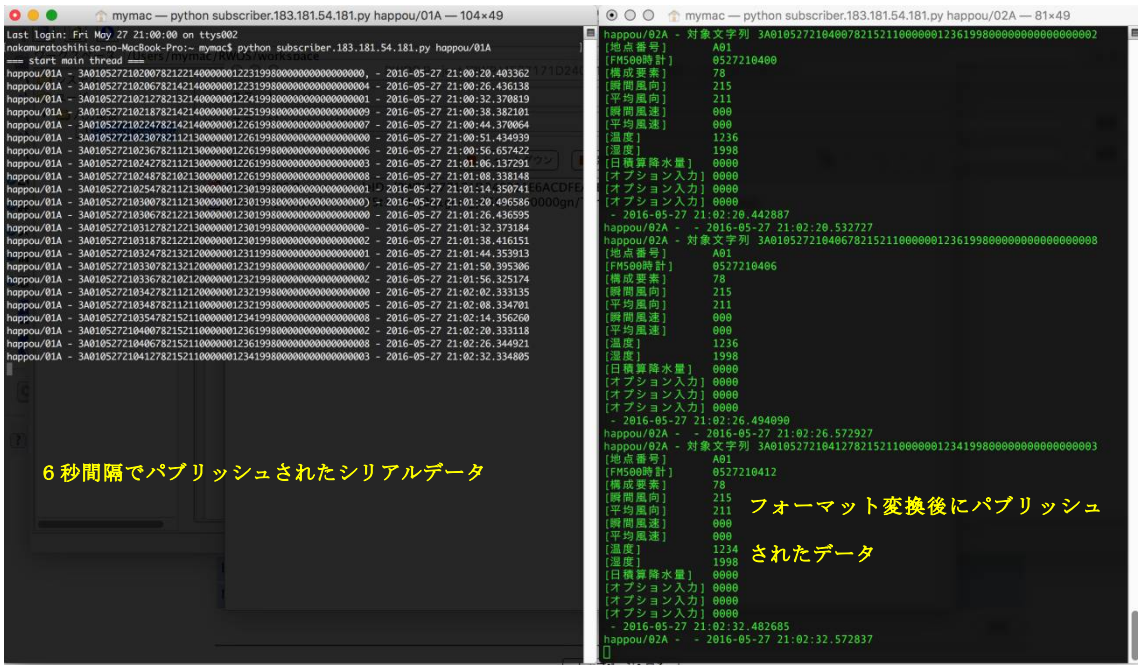


図 5-5 動作確認画面

5.2 今後の課題

本研究では、特定の特定モデル化手法におけるモデル化手法特化型言語を選択し、そのプログラム言語によるプログラムを MMDA により開発することが出来るモデリング GUI を開発した．このことは、特定モデル化手法において、モデリング GUI を実現出来るモデル化手法特化型プログラム言語を作成すれば、そのプログラム言語によるプログラムを MMDA により開発出来るモデリング GUI を設計・実装することが可能であることを意味している．

これを踏まえて、今後は、ソフトウェア開発と云う自らの業務の中で、モデル化手法特化型プログラム言語を作成することにも挑戦したいと考えている．ソフトウェア開発案件において、顧客に要求されたソフトウェアを、単に一般プログラム言語(Java、C++、C#、Python 等)の利用を前提に設計・実装するのではなく、要求分析に基づいて、可能であれば、モデリング GUI を実現出来る新たなモデル化手法特化型プログラム言語を提案する．その設計及び実装を行った上で、そのプログラム言語によるプログラムを MMDA により開発出来るモデリング GUI を設計・実装する．これにより、自らのソフトウェア設計・実装能力向上だけでなく、顧客満足度の向上も期待することが出来る．ソフトウェア開発は、顧客の想定する開発期間及び予算枠の中で行う必要があるので、実際には、乗り越えなければならない多くの問題があると考えられるが、是非実現したいと考えている．

また、今後は、MMDA を拡張し、このモデル化手法特化型言語の上位に UML や SysML[西村ら 2015]のような抽象記述言語を設計し、その言語によるプログラムを開発する為の GUI を設計・実装したいと考えている．MDA では、抽象記述言語で記述されたモデルからいきなり実行可能なテキストベースのプログラムを作成しようとして失敗していた．そこで、MMDA では、抽象記述言語で記述されたモデルから、モデリング GUI のデータファイルを生成する．モデル化手法毎のプラグインを用意して選択出来るようにすれば、選択されたモデル化手法のモデリング GUI 用データファイルを生成することは可能である．その後は、本論文で述べたように、ユーザがモデリング GUI によりこのモデリング GUI 用データファイルの編集を行う．これが実現できれば、抽象記述によるモデル作成が可能になり、プログラムの抽象設計から開発までを一貫して行える特定領域専門家の為の総合的なプログラム統合開発環境が完成する．それは、巨大なソフトウェア開発プロジェクトにおいても、その開発効率向上に貢献出来る筈であり、そこに至って初めて MDA の問題点を完全に解決した開発方式を確立したことになりうると考えている．

更に、本研究において設計・実装を行った SOARS VisualShell については、4.1.3.2 で述べたように既にプログラム実行結果の 3 次元可視化を行っているが、今後は、この機能を更に向上させ FlexSim[FlexSim 1993]のような描画が行えるようにすることが望ましい．同様に、RWOS Program Editor についても、Unity[Unity 2005]を利用する等により、実世界及び仮想世界の状態を Exaplog[結城 2002]のようにリアルタイムに解析及び可視化出

来るようにすることが望ましい.

謝辞

本論文の作成にあたっては、多くの方々からご指導とご支援を頂きました.

東京工業大学大学院の出口弘先生には、指導教員として、研究内容についてだけでなく、研究と向き合う為の姿勢から自らが行った研究を論文としてまとめる為の心構えに至るまで、広く深くご指導を頂いたことに心から感謝致します.

また、東京工業大学大学院の寺野隆雄先生、新田克己先生、三宅美博先生、小野功先生には、論文審査を通じて大変貴重なご助言を賜り、丁寧なご指導を頂いたことに深く感謝致します。

また、研究を進める上でいつも貴重な助言をしてくださった国立保健医療科学院の市川学先生、筆者が出口研究室と出会う契機を与えてくださった田沼英樹氏、筆者のプログラム開発に際して的確な助言をしてくださった法政大学の佐々木晃先生、そして入学以来ずっと御世話になった出口研究室のメンバーの方々には、改めて深く感謝致します.

最後に、筆者の本学への研究活動を理解し協力してくださった勤務先の株式会社パイケークの木寺重樹氏、石塚康成氏、助台良之氏、そして研究活動から論文作成に至るまで長きに渡って筆者を支えてくれた妻に感謝致します.

2017 年 8 月

参考文献

[apache 1995] apache: http://www.apache.org/, 1995.

[Arduino 2005] Arduino: https://www.arduino.cc/, 2005.

[artisoc 2009] artisoc: http://mas.kke.co.jp/modules/tinyd0/index.php?id=11, 2009.

[Ben ほか 2003] Ben Laurie, Peter Laurie (2003): Apache Handbook. (大川佳織訳，田辺茂也監訳 『Apache ハンドブック』，オライリージャパン, 2003 年)

[Bjarne 1991] Bjarne Stroustrup (1991): The C++ Programming Language (2nd Edition). (望月康司, 谷口功訳『プログラミング言語 C++ 第 2 版』，トッパン, 1993 年)

[Den 2008] Den Haan, J. "Reasons Why Model-Driven Approaches (will) Fail [documento en línea]. 2008." (8).

[Eclipse 1998] Eclipse: https://eclipse.org/, 1998.

[FlexSim 1993] FlexSim: https://www.flexsim.com/, 1993.

[George ほか 1996] George Shepherd, Scot Wingo. (1996): MFC Internals. (玉井浩訳『MFC インターナル』，アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1997 年)

[Ichikawa ほか 2007] Manabu Ichikawa, Hideki Tanuma, Yuhsuke Koyama, Hiroshi Deguchi. “SOARS: introduction as a social microscope for simulations of social interactions and gaming.” Organizing and Learning through Gaming and Simulation, pp.149- 158, Eburon, 2007.

[Jesse ほか 2007] Jesse Liberty, Donald Xie (2007): Programming C# 3.0, 5th Edition. (鈴木幸敏, 首藤一幸, 株式会社情報技研訳『プログラミング C# 第 5 版』，オライリー・ジャパン, 2009 年)

[Joshua ほか 2005] Joshua Marinacci, Chris Adamson (2005): Swing Hacks Tips and Tools for Killer GUIs. (神戸博之, 島田秋雄, 加藤慶彦訳『Java Swing Hacks 今日から使える驚きの GUI プログラミング集』, オライリー・ジャパン, 2006 年)

[Karl-Heinz ほか 2010] Karl-Heinz John, Michael Tiegelkamp (2010): IEC 61131-3: programming industrial automation systems. (PLCOPEN JAPAN 訳『IEC 61131-3 を用いた PLC プログラミング ：PLC 言語の国際規格の解説と応用』, 丸善出版, 2012 年)

[Kermack ほか 1927] Kermack, William O., and Anderson G. McKendrick. "A contribution to the mathematical theory of epidemics." Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. Vol. 115. No. 772. The Royal Society, 1927.

[Mark 1998] Mark Lutz (1998): Introduction to Python. (村山敏夫訳, 飯坂剛一監訳『Python 入門』, オライリー・ジャパン, 1998 年)

[Martin 1996] Martin Fowler (1996): Analysis Patterns Reusable Object Models. (堀内一監訳, 児玉公信, 友野昌夫訳『アナリシスパターン 再利用可能なオブジェクトモデル』, アジソン・ウェスレイ・パブリッシャーズ・ジャパン株式会社, 1998 年)

[Martin 2003] Martin Fowler (2003): UML Distilled Third Edition. (羽生田栄一訳『UML モデリングのエッセンス 第 3 版 (Object Oriented SELECTION)』, 翔泳社, 2005 年)

[MASON 2003] MASON: https://cs.gmu.edu/~eclab/projects/mason/, 2003.

[Massimo 2009] Massimo Banzi (2009): Getting Started with Arduino. (船田巧訳『Arduino をはじめよう』, オライリー・ジャパン, 2009 年)

[Microsoft Visual Studio 1997] Microsoft Visual Studio: https://www.microsoft.com/ja-jp/dev/default.aspx, 1997.

[mosquito 2009] mosquito: http://mosquitto.org/, 2009.

[MQTT 1999] MQTT: http://mqtt.org/documentation, 1999.

[NetLogo 1999] NetLogo: https://ccl.northwestern.edu/netlogo/, 1999.

[Patrick ほか 2002] Patrick Niemeyer, Jonathan Knudsen (2002): Learning Java, 2nd Edition. (滝沢徹, 鈴木憲子, 牧野祐子訳『詳解 Java プログラミング 第 2 版』, オライリー・ジャパン, 2003 年)

[PLC 1968] PLC: http://www.mitsubishielectric.co.jp/fa/products/cnt/plc/, 1968.

[Repast 2006] Repast: http://repast.sourceforge.net/, 2006.

[RFC4122 2005] RFC4122: https://www.ietf.org/rfc/rfc4122.txt, 2005.

[RFC4627 2007] RFC4627: https://www.ietf.org/rfc/rfc4627.txt, 2007.

[Russ ほか 2006] Russ Miles, Kim Hamilton (2006): Learning UML 2.0. (原隆文訳『入門 UML 2.0』, オライリー・ジャパン, 2007 年)

[SOARS プロジェクト 2004] SOARS プロジェクト: http://www.soars.jp, 2004.

[Soley 2000] Soley, Richard. "Model driven architecture." OMG white paper 308.308 (2000): 5.

[Stella 1987] Stella: http://www.iseesystems.com/, 1987.

[Tanuma ほか 2004] Tanuma H, Deguchi H, Shimizu T (2004) “SOARS: spot oriented agent role simulator: design and implementation.” In:Post-proceedings of AESCS04. Springer, Japan, pp 49–56.

[Unity 2005] Unity: https://unity3d.com/jp/, 2005.

[Warmer ほか 1998] Warmer, Jos B., and Anneke G. Kleppe. "The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series)." (1998).

[William 1994] William Richard Stevens (1994): TCP/IP Illustrated, Volume 1: The Protocols. (橘康雄訳, 井上尚司監訳『詳細 TCP/IP Vol.1 プロトコル』, 株式会社ピアソン・エデュケーション, 2000 年)

[岡本 2014] 岡本裕生 (2014): 『やさしいリレーとシーケンサ』, オーム社

[川崎 2000] 川崎晴久 (2000): C & FORTRAN による数値解析の基礎, 共立出版

[シスコ 2013] シスコシステムズ合同会社 IoT インキュベーションラボ (2013): Internet of Everything の衝撃, インプレス R&D

[高来ほか 2014] 高来一彦, 張紅, 萩原光則, 芦田政之, 渡辺康雄, 半田執 (2014):『プログラマブルコントローラ(PLC)原理と設計法(設計技術シリーズ)』, 科学情報出版

[高橋 2013] 高橋隆雄 (2013): 『たのしい電子工作 Arduino で電子工作をはじめよう！』, 秀和システム

[竹林ほか 2016] 竹林康太、出口弘、倉田正、石塚康成、木寺重樹、"IoT アーキテクチャとしてのリアルワールドＯＳアーキテクチャデザイン"、第 10 回社会システム部会研究会 -SICE 社会システム部会、Mar.2016.

[田沼ほか 2007] 田沼英樹, 出口弘. “エージェントベース社会シミュレーション言語 SOARS の開発.” 電子情報通信学会論文誌 D, 2007, 90.9: 2415-2422.

[田畑 1994] 田畑孝一 (1994): 『OSI - 明日へのコンピュータネットワーク』, 日本規格協会

[出口 2014(1)] 出口弘. “マルチエージェントシミュレーション: 2. 社会シミュレーションと組織・社会の情報処理のアーキテクチャ・デザイン.” 情報処理, 2014, 55.6: 539-548.

[出口 2014(2)] 出口弘. “POE(Point of Event)データとその利活用-IOE 時代に向けての個人・企業・政府のデータ利活用のための三つの原則.” システム／制御／情報, 58-7, 274/281, 2014.

[出口 2015] 出口弘. “IoE 時代のもの・サービスの生産支援システム-代数的多元簿記に基づく自律分散協調型システムとして-.” 経営情報学会 2015 年秋季大会予稿集, 2015.

[出口 2016] 出口弘. “組織・産業・経済システムの人工物としてのデザイン論 -IoE 時代の組織・産業・経済システムの現実の再構築に向けて-.” 計測と制御, Vol.55 No.1 2016.

[長嶋 1995] 長嶋秀世 (1995): 数値計算法, 槇書店

[西村 2015] 西村秀和, 藤倉俊幸 (2015): モデルに基づくシステムズエンジニアリング, 日経 BP 社

[パトライト 1947] パトライト: http://www.patlite.co.jp/, 1947.

[松本 2016] 松本直人 (2016): 『モノのインターネットのコトハジメ』, 翔泳社

[水島ほか 2002] 水島二郎, 柳瀬眞一郎 (2002): 理工学のための数値計算法, 数理工学社

[水島 2003] 水島和憲 (2003):『オープンソース徹底活用 Eclipse による Java アプリケーション開発』, 秀和システム

[山影 2007] 山影進 (2007): 『人工社会構築指南 - artisoc によるマルチエージェント・シミュレーション入門(人工社会の可能性)』, 書籍工房早山

[結城 2002] 結城義敬. "データマイニングの最前線 プロセスイベント記録の「バランス」に着目したデータマイニング." 計測と制御 41.5 (2002): 350-353.

[吉川 1994] 吉川敏則 (1994): C 言語実用数値処理プログラム集, 近代科学社

以上

業績目録

査読付き学会誌論文

○出口弘, 竹林友善, 吉田宏章, 梅宮茂良, 紺野剛史, 石塚康成, 木寺重樹, 倉田正, Shuang Chang, エネルギー会計によるエネルギー運用計画デザイン, Journal of IAP2M Vol.10No.1, pp191-214, Oct.2015（研究論文）

○Tadashi KURATA, Hiroshi DEGUCHI, Manabu ICHIKAWA, "A Study of the Model Method Driven Architecture (MMDA) and its modeling environment, SICE Journal of Control, Measurement, and System Integration, SICE (Accepted)

査読付き国際会議論文及び査読付きプロシーディング

○ Tadashi Kurata, Hiroshi Deguchi, Manabu Ichikawa, "GUI for Agent Based Modeling", Social Modeling and Simulations + Econophysics Colloquium 2014(SMSEC2014), Proceedings of the International Conference on Social Modeling and Simulation, plus Econophysics Colloquium 2014, pp.275-286, Nov.2014

国内会議プロシーディング

○竹林康太、出口弘、倉田正、石塚康成、木寺重樹、"IoT アーキテクチャとしてのリアルワールドＯＳアーキテクチャデザイン"、第 10 回社会システム部会研究会-SICE 社会システム部会、Mar.2016

○青松祐亮、出口弘、倉田正、石塚康成、"スケジューリング問題のための動的支援システムの構築"、第 10 回社会システム部会研究会- SICE 社会システム部会、Mar.2016

ポスター発表等

○倉田正、出口弘、市川学、"モデルメソッド駆動型アーキテクチャ(MMDA)とそのモデリング環境の研究"、計測自動制御学会・システム・情報部門 学術講演会 2015(SSI2015)、函館アリーナ、Nov.2015

以上