Development Tools of Simulation Models with MDA

Nozomu Aoyama [1]
Rintaro Takeda [1]
Takashi Iba [2]
Hajime Ohiwa [3]

[1]Graduate School of Media and Governance, Keio University
5322 Endo, Fujisawa 252-8520, Japan
{bam, rintaro}@crew.sfc.keio.ac.jp
[2]Faculty of Policy Management, Keio University
5322 Endo, Fujisawa 252-8520, Japan
iba@sfc.keio.ac.jp
[3]Faculty of Environmental Information, Keio University
5322 Endo, Fujisawa 252-8520, Japan
ohiwa@sfc.keio.ac.jp

Abstract. In this paper, we propose Component Builder, which is a tool to develop simulation models and to share them with diagrams.

If a modeler has few experiences of programming, it is difficult to make a simulation model or to share it. It is because existing tools only support the modelers to make simulation models with programming. For solving the problem, Component Builder supports simulation development not with programming but with diagram. This tool generates executable program codes from a structure of models with diagrams. This tool enables the modeler to develop simulation without knowledge of programming.

Moreover, we propose the development methodology with diagrams independent of any programming language or software implementation. Existing tools depend on a specific programming language. It prevents a modeler who has knowledge of another programming language from developing simulation models. For solving the problem, we propose to apply a paradigm of software engineering to developing simulation models. The paradigm "MDA"(Model Driven Architecture) is to turn models with "UML"(Unified Modeling Language) into executable program code. Applying the paradigm, we can keep our mind on modeling without considering any programming language or software implementation.

Thus we implemented the develop environment for drawing diagrams and generating program codes from them. Some modeler tried developing simulation models with these tools. As a result, they could develop them without programming.

1 Introduction

Many researchers and we almost agree that the agent-based model (multi-agent model) is suitable for studying complex systems, however in the current state, there is a problem that needs to be resolved. The problem is the difficulty to develop simulation models and to share them. To study complex systems, we need not only to develop them but also to understand studies of other researchers. For solving the problem, we need a tool for developing and sharing simulation models.

In the current state in the study of simulation models, existing tools are not useful enough for using simulation models, because these tools only support modelers to make simulation models with programming. With these tools, the modelers have to program with specific programming language. Therefore, it is impossible for a modeler who has no knowledge of the language to make simulations or to understand them.

In this paper, we will propose a development tools for simulation models with diagrams to solve the problem. Using diagrams independent of any specific languages or software implementation, a modeler can make simulation models and share them without the knowledge of programming. In the second section of this paper, we will describe the background of simulation tools and software engineering. In the third, we will describe how to develop simulation models with MDA. In the forth, we will describe proposed tools.

2 Background

2.1 Existing Tools for Simulation Development

In the last some years, several languages, frameworks and tools for agent-based simulations have been proposed. For example, "Swarm Simulation System", which seems to be the most famous and to be used, provides the class library for the simulation of complex adaptive systems [Minar et al., 1996]. As well as Swarm, "RePast" provides the class library to make the agent-based model [Collier, 2003]. "Ascape" also provides the framework , and it is said that the amount of the code description can be less than that of Swarm or RePast [Parker, 2001].

However, it is difficult to make a simulation model or to share it with these tools, because these tools only support the modelers with programming. Though they are useful for the reduction of programming with general libraries and frameworks, still a modeler can not make it without knowledge of programming language which the tool depends on. Because they can neither understand basic literal nor control structure of program.

2.2 MDA: A New Paradigm of Software Development

In the field of software engineering, there is a trend toward considering the design models as the development artifacts that contribute directly to software development. "MDA" (Model Driven Architecture) and "Executable UML(Unified Modeling Language)" are proposed for the Model Driven Development [Frankel, 2003,Mellor and Balcer, 2002,Kleppe et al., 2003]. The point is "using

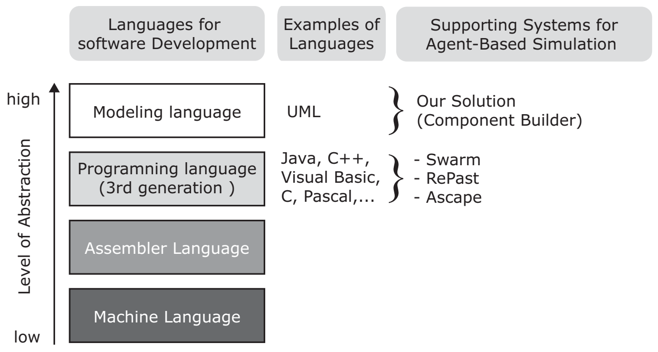

Languages for software Development
Examples of Languages
Supporting Systems for Agent-Based Simulation
high
Level of Abstraction
low
Modeling language
UML
Our Solution (Component Builder)
Programning language (3rd generation)
Java, C++, Visual Basic, C, Pascal,...
- Swarm
- RePast
- Ascape
Assembler Language
Machine Language

Fig. 1. Level of Abstraction, Language, and Supporting Systems

modeling languages as programming languages rather than merely as design languages."[Frankel, 2003]. As a result, "It makes it possible to raise the level of abstraction for software development"[Frankel, 2003] (Figure 1). History tells that the productivity and quality are improved in consequence of raising the level of abstraction.

We can use more high-level language for development with MDA, instead of writing in lower-level languages. It means that modelers describe models rather than program codes. Therefore, the modelers can keep their modeling in mind without considering the software implementation, because program codes will be extra translations of the design model.

MDA defines the model with height level of abstraction that is independent of any implementation technology. Then, we don't care which implementation technology should be used. It means that it is not important whether we can use a specific programming language.

A generator is necessary in order to generate program codes from models dependent on an implementation technology. The generator interprets the model described with diagram and generates executable program codes from it.

In this paper, we will propose a new paradigm and development tools adopting MDA.

3 Development of Simulation Model with MDA

3.1 A Language for Describing Model

To apply MDA, we have to describe a very detailed model to generate an executable program. For example, we have to define what elements have to be, what feature they have, and which algorithm or data structure they use. Though we use diagram to describe, this is not different from programming by hands in the necessary knowledge (Figure 2).

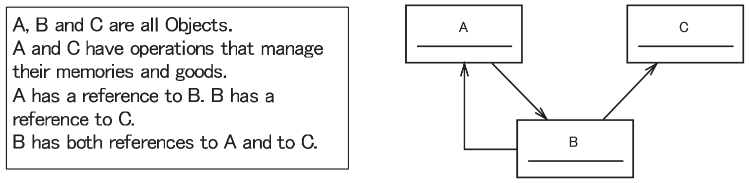

A, B and C are all Objects.
A and C have operations that manage their memories and goods.
A has a reference to B. B has a reference to C.
B has both references to A and to C.

Fig. 2. An Example of Describing Model Without Model Framework

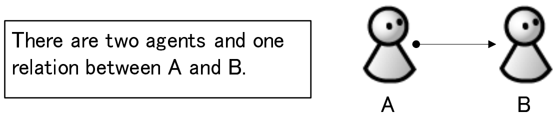

There are two agents and one relation between A and B.

Fig. 3. An Example of Describing model with Model Framework

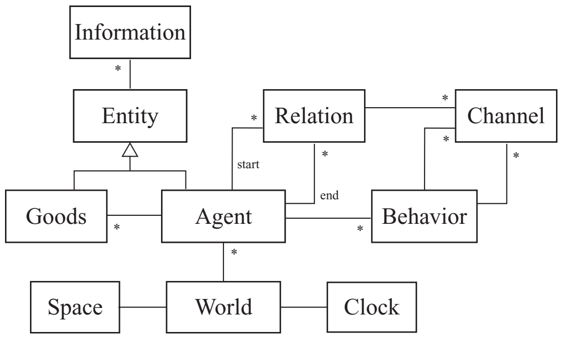


Fig. 4. Major Classes of Boxed Economy Foundation Model (BEFM)

For solving the problem, we describe a model based on a model framework, "Boxed Economy Foundation Model" (BEFM). BEFM can be a frame of reference for recognizing the target world (Figure 4). Using the model framework, the modeler can focus on the part of the target world [Iba et al., 2002,Iba et al., 2002].

Using the model framework as language, we can keep our mind on simulation modeling (Figure 3). This makes it possible for modelers who have no skills of programming to develop or to understand a simulation model.

3.2 Elements Necessary in a Simulation Model

For generating codes, the generator needs a complete definition of the subject matter under study. We define specifications of a simulation model so that we can describe it completely. In other words, we define what a generator can generate automatically and what a modeler has to think of.

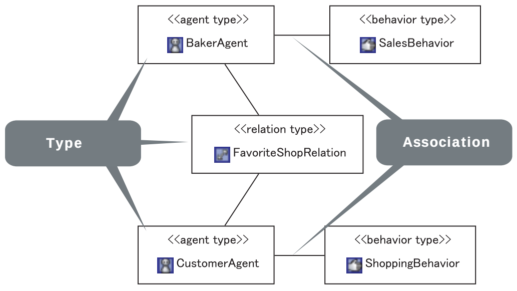
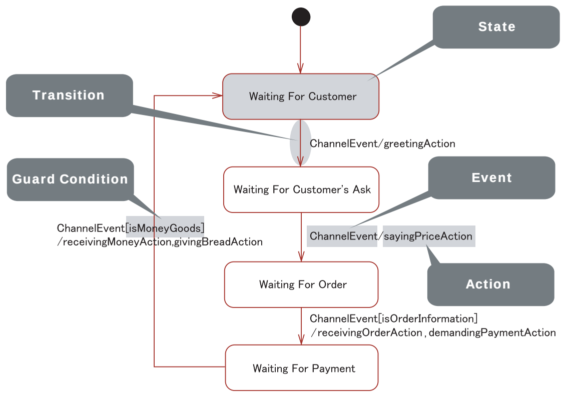


Fig. 5. An Example of Type Structure with Class Diagram



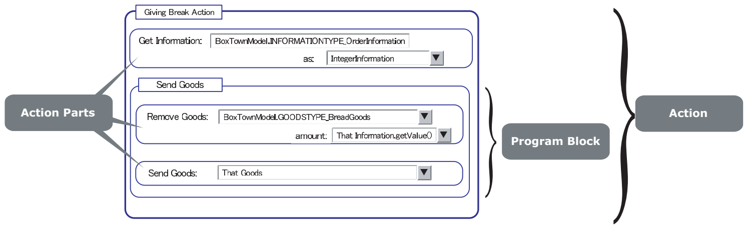
State
Transition
Waiting For Customer
ChannelEvent/greetingAction
Guard Condition
Waiting For Customer's Ask
Event
ChannelEvent[isMoneyGoods]
/receivingMoneyAction,givingBreadAction
ChannelEvent/sayingPriceAction
Waiting For Order
Action
ChannelEvent[isOrderInformation]
/receivingOrderAction, demandingPaymentAction
Waiting For Payment

Fig. 6. An Example of Behavior's Statemachine with Statechart Diagram

In the general software development, three models - the class model, the statemachines for classes, and the state's procedures - form a complete definition of the subject matter under study [Mellor and Balcer, 2002]. These models can match to elements in BEFM. When following models are defined, we can generate executable program codes.

Type Structure This is a class model in a simulation model. It represents defined types in the model. In BEFM, it is a structure of Agent, Relation, Behavior, Goods, and Information. We define it by drawing Class Diagram in UML [Object Management Group, 2000] .(Figure 5)

Behavior's Statemachine This is a statemachine of a class in a simulation model. It represents rules of agent's Behaviors. We define it by drawing Statechart Diagram in UML [Object Management Group, 2000] .(Figure 6)


Giving Break Action

Get Information: BoxTownModel.INFORMATIONTYPE_OrderInformation
as: IntegerInformation

Send Goods

Remove Goods: BoxTownModel.GOODSTYPE_BreadGoods
amount: That Information.getValue()

Send Goods: That Goods

Fig. 7. An Example of Action

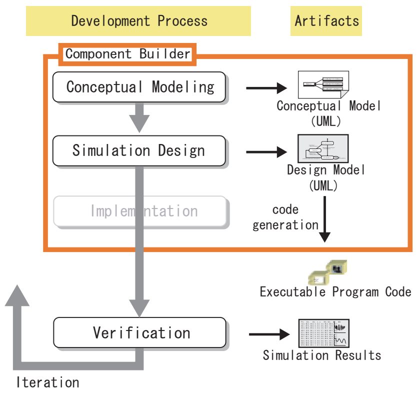


Fig. 8. Development Process with Component Builder

Action This is a state's procedure in a simulation model. It represents a detailed process in a behavior. We define it by drawing a hierarchy of operations in BEFM (Figure 7).

4 Simulation Development Tools: Component Builder

In order to support developing simulations with MDA, we would like to propose "Component Builder(CB)". CB consists of six tools. Especially, four tools of them are to generate program codes just by drawing a diagram and by setting parameters with a graphical user interface.

4.1 Modeling Process Supported by Component Builder

We proposed a simulation development process from conceptual modeling to verification [Iba et al., 2004]. CB supports "Conceptual Modeling Phase" and "Simulation Design Phase" in the process (Figure 8).

CB provides functions to draw diagrams in Conceptual Modeling Phase. We can make a conceptual model with three tools (Figure 9).

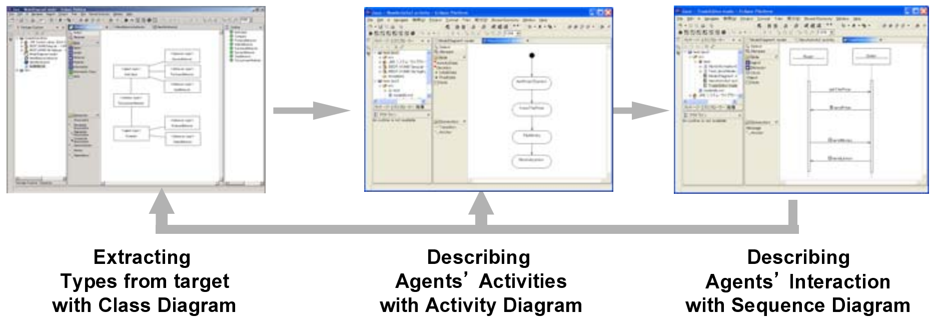


Fig. 9. Conceptual Modeling Phase

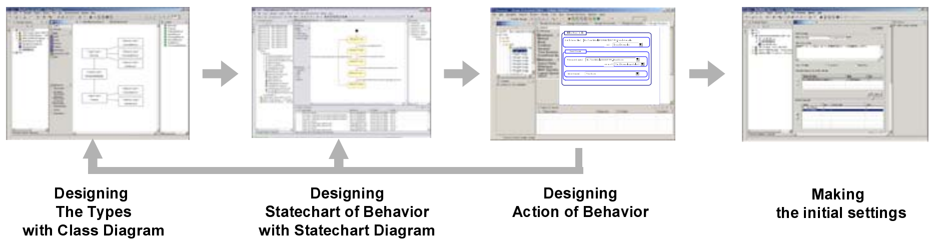


Fig. 10. Simulation Design Phase

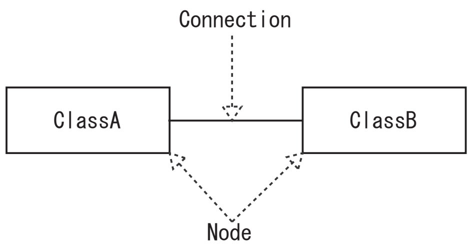


Fig. 11. An Example of Node and Connection

In Simulation Design Phase, CB provides functions to draw diagrams, to generate program codes from them, and to make initial settings of a simulation. We can make a design model with four tools (Figure 10).

4.2 Required Functions of Component Builder

The four tools of CB have following functions in order to support modeling. One of the functions is to draw Node and Connection. Node is the object consisting of two elements that are size and location. Node is used as Class or State.

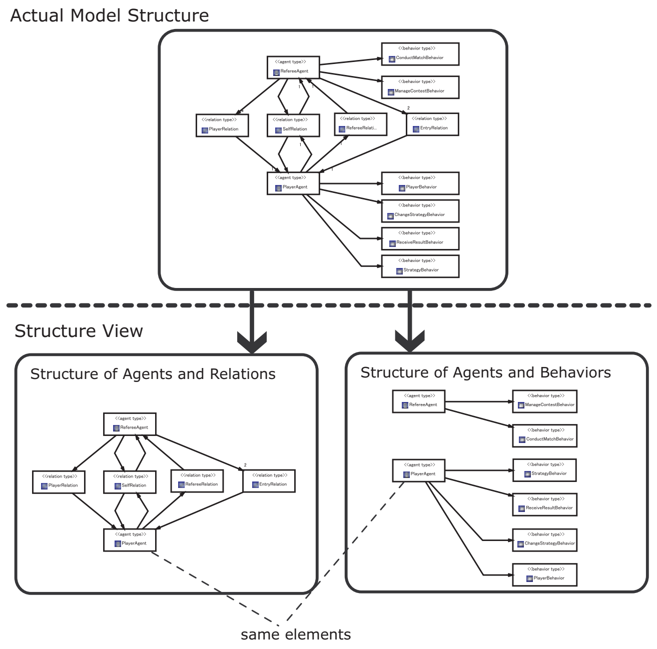

Actual Model Structure
Structure View
Structure of Agents and Relations
Structure of Agents and Behaviors
same elements

Fig. 12. An Example of Diagrams to Describe Several Different Views

Connection is the line drawn between source Node and target one. Connection is used as Association, Generalization and Transition (Figure 11).

- Functions about Node (used as Class, State, Activity, etc.)
  - create new Node
  - delete Node
  - move Node
  - resize Node
- Functions about Connection (used as Association, Generalization, Transition, etc.)
  - create new Connection
  - delete Connection
  - change source or target of Connection
  - change line form of Connection
- Undo
  - undo and redo as memory area permits

In Conceptual Modeling Phase and Simulation Design Phase, there is a demand to draw diagrams in order to describe several different views. The diagrams can refer to common elements (Figure 12). Class Diagram Editor has the functions fulfilling the demand.

CB has the functions to generate the following program codes.

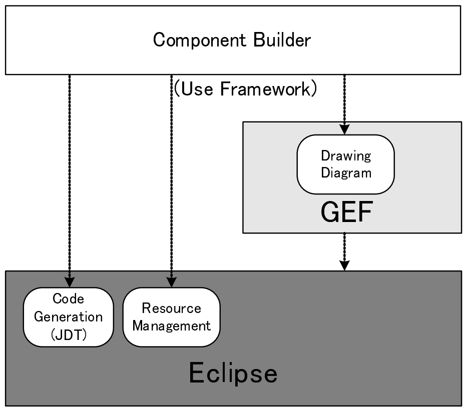


Fig. 13. Architectures of Component Builder

- The program codes that define Types is generated by Class Diagram (at one program file for each model)
- The program codes that define state transitions of Behavior is generated by Statechart Diagram (at one program file for each Behavior)
- The program codes that define Actions of Behavior is generated by Action description of Statechart Diagram (at one program file for each Behavior)
- The program codes that define initial settings of World (at one program file for each World)

For the purpose of smooth simulation development, the functions of creating file easily, deleting and moving are important. Moreover, the following functions are needed to support model sharing and collaborative development.

- Sharing models over the networks
- Importation and Exportation of model

CB is considered to use CVS (Concurrent Versioning System) which enables us to share model over the networks.

CB has the function to make documents with diagrams.

- Printing diagram
- Exportation of picture files from diagrams

4.3 Architectures of Component Builder

For implementing the required functions efficiently, Figure 13 shows an architecture we designed.

Eclipse is an open platform for tool integration built by an open community of tool providers [Eclipse Project, 2004]. It provides many functions to implement editors on it. Using Eclipse, we don't have to implement following functions.

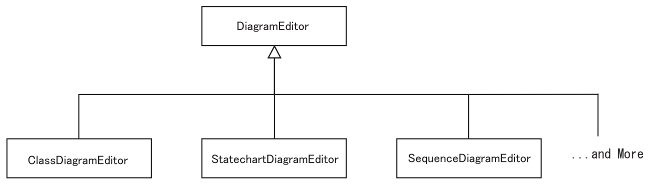


Fig. 14. Diagram Editor Framework

– File Management
– Importation and Exportation of models
– Sharing models with CVS

Diagram Editing We also use the framework "Graphical Editor Framework (GEF)" for implementation of CB. GEF allows developers to take an existing application model and easily create a rich graphical editor [Eclipse Project, 2004]. GEF enables the graphical editor to manipulate Nodes and Connections. We only define concrete Node and Connection. Therefore, we don't have to implement basic functions like printing or undoing/redoing.

UML editors have some common parts among the tools. They have the mechanism common to create or to delete elements. Even if drawing elements like Class, State and Activity, differ from diagrams. We make a framework to abstract the mechanism, "Diagram Editor Framework" (Figure 14). Each UML editor is based on the framework. Therefore, we can implement multiple editors efficiently.

Code Generation CB generates understandable program codes. The program codes must be understandable to debug a simulation model. CB uses two following frameworks in order to generate program codes on Eclipse Platform. These frameworks make it easy to edit Java program codes.

– JDT(Java Developer Tool) [Eclipse Project, 2004]
– JET(Java Emitter Templates) [Eclipse Project, 2004]

Using JDT, we can treat java programs as a syntax tree, format program codes, and use a template of program comment. CB uses it for generating program codes to represent state-transitions of Behavior, actions of Behavior and initial settings of World. The reason for using JDT is that we can edit programs more freely than JET, and it is suitable for generating complex program codes.

JET is a generic template engine that can be used to generate SQL, XML, Java source code and other output from templates like JSP [Sun, 2004]. CB uses it for generating program codes to represent type definitions with Class Diagram. The reason for using JET is that we can edit programs more easily than JDT, and it is suitable for generating fixed program codes.

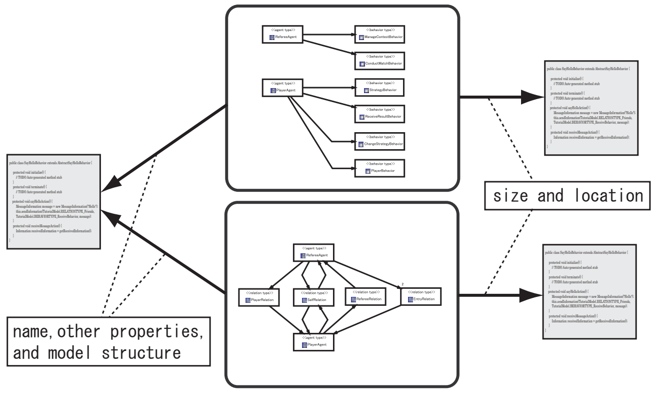


Fig. 15. A Mechanism of Saving

Save We need to save two following information with CB.

- coordinates information (location and size)
- model information (model features and model structures)

Model information is saved to files corresponding to each diagram, and model information is saved to one file for each model on CB (Figure 15). It is because we often make each diagram refer to same element.

We accept XML as file format for two reasons. First, It makes easy to use CVS. Second, we can convert the models into documents using technologies like XSLT.

Export Picture File CB exports diagrams to picture file. The file is formatted with SVG [W3C, 2004] that is based on XML. This format makes it possible not only to convert it into EPS or JPEG but also to edit the XML file with other applications.

4.4 Each Tool of Component Builder

In this section, we describe the three tools of CB that are important parts especially in MDA.

Model Designer Model Designer is a modeling tool to define types in a simulation model with Class Diagram (Figure 16). With this tool, a modeler can draw diagrams and generate program codes represented by the type structure.

Behavior Designer Behavior Designer is a modeling tool to define state machines of Behaviors with Statechart Diagram (Figure 17). With this tool, a modeler can draw diagrams and generate program codes to represent rules of the Behavior.

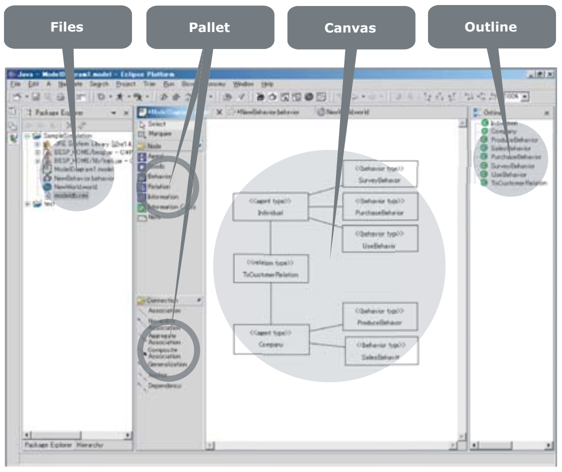


Fig. 16. Model Designer

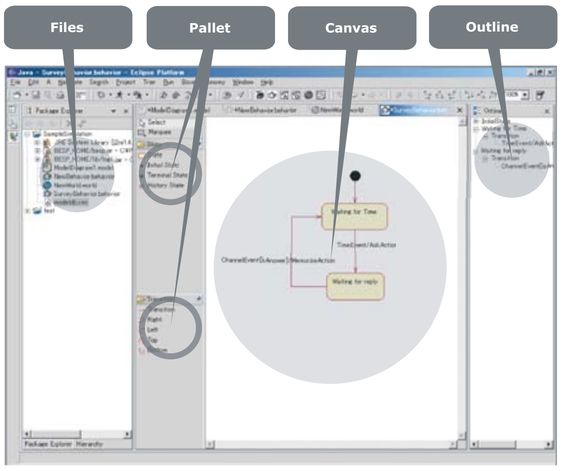


Fig. 17. Behavior Designer

Action Designer Action Designer is a modeling tool to describe details of actions defined in Behaviors (Figure 18). With this tool, a modeler can draw diagram and generate program codes of action procedures.

An action is a set of some procedures. As a model becomes larger, they become more complicated. We propose drawing diagram of actions with hierarchical structure to understand them. Action Designer supports to describe procedures hierarchically.

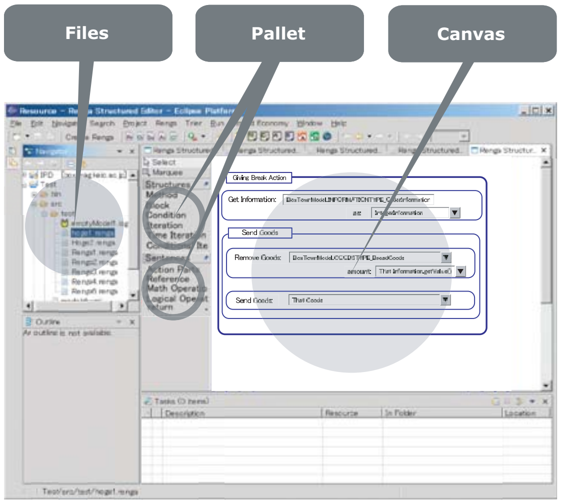


Fig. 18. Action Designer

Table 1. Examples of Action Parts

The Name of Action Parts	**The Explanation of Action Parts**
send information	send a information to relative agent(s)
add relation	add a relation to the agent which have the action
extract agents with random	extract some agents from collection of agents randomly

In an action, we have to describe procedures to edit elements defined with BEFM. When we develop some simulation models, there are many duplicated procedures in them. We call the procedures "Action Parts". Action Designer supports to use Action Parts appeared frequently. Table 1 is examples of provided Action Parts.

5 Evaluation

Now we would like to evaluate the proposed tools. The example is evolutionary simulation of strategy in the Iterated Prisoners' Dilemma. "The Prisoner's Dilemma" is an elegant embodiment of the problem of achieving mutual cooperation, and therefore provides the basis for the analysis [Axelrod, 1997].

The Iterated Prisoner's Dilemma model consists of 6899 lines of program. We can show that 23 UML Diagrams and 2 World settings generate 6184 lines on the proposed tools (Table 2). It follows from this that we can design model and simulation with little programming by using the proposed tools.

Table 2. Difference of Artifacts

	Without CB	With CB
World	2318 lines program	2 settings
Type definition	245 lines program	7 diagrams
Behavior and Action	3621 lines program	28 diagrams
Information	715 lines program	715 lines program
Total	6899 lines program	715lines program, 28 diagrams, and 2settings

There is no need to program in order to design model and simulation, if the model includes not complex Information but simple Information such as string or number. Modelers have to program the structure of Information, only if their models include complex Information. This case is the subject of future investigation.

6 Conclusion

In this paper, we proposed the paradigm and tools for development of agent-based social simulations. The tools are open to the public on http://www.boxed-economy.org/. Creating the foundation for the social simulations study is an oversized project for our members to complete. We would like to realize it by collaborating with many researchers in various fields. Please contact us, if you are interested in our challenge.

7 Acknowledgements

This research was supported by Boxed Economy Project. We wish to thank Yoshihide Chubachi, Yoshiaki Matsuzawa and Ryunosuke Tsuya for their encouragement and guidance throughout this research. However, any mistakes that remain are our own.

References

[Axelrod, 1997] R. M. Axelrod (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration* (Princeton University Press).

[Boxed Economy Project, 2003] Boxed Economy Project. *Designers Guide to Social Simulations.* Fujita Institute of Future Management Research 2 edn. (2003). in Japanese.

[Collier, 2003] N. Collier (2003). Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research* , http://repast.sourceforge.net/.

[Frankel, 2003] D. S. Frankel (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing* (Wiley Publishing).

[Iba et al., 2002] T. Iba, Y. Chubachi, Y. Takabe, K. Kaiho, and Y. Takefuji (2002). Boxed Economy Foundation Model in *The AAAI-02 Workshop on Multi-Agent Modeling and Simulation of Economic Systems* pp. 78–83.

[Iba et al., 2002] T. Iba, Y. Takabe, Y. Chubachi, J. Tanaka, K. Kamihashi, R. Tsuya, S. Kitano, M. Hirokane, and Y. Matsuzawa (2002). Boxed Economy Foundation Model: Toward Simulation Platform for Agent-Based Economic Simulations in *Exploring New Frontiers on Artificial Intelligence* pp. 227–236 (Springer-Verlag).

[Iba et al., 2002] T. Iba, Y. Chubachi, Y. Matsuzawa, K. Asaka, and K. Kaiho (2002). Resolving the Existing Problems by Boxed Economy Simulation Platform in *Agent-based Approaches in Economic and Social Complex Systems* pp. 59–68 (IOS Press).

[Iba et al., 2004] T. Iba, Y. Matsuzawa, and N. Aoyama (2004). From conceptual models to simulation models: Model driven development of agent-based simulations.

[Kleppe et al., 2003] A. Kleppe, J. Warmer, and W. Bast (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise* (Addison-Wesley).

[Mellor and Balcer, 2002] S. J. Mellor and M. J. Balcer (2002). *Executable UML: A Foundation for Model-Driven Architecture* (Addison-Wesley).

[Minar et al., 1996] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system:a toolkit for building multi-agent simulations. http://www.santafe. edu/ projects/ swarm/ overview/ overview.html (1996).

[Parker, 2001] M. T. Parker (2001). What is ascape and why should you care? *Journal of Artificial Societies and Social Simulation 4*(1), http://www.soc.surrey.ac. uk/ JASSS/4/1/5.html.

[Sun, 2004] Sun. Javaserver pages. http://java.sun.com/products/jsp/ (2004).

[W3C, 2004] W3C. Scalable Vector Graphics. http://www.w3.org/Graphics/SVG/ (2004).

[Eclipse Project, 2004] Eclipse Project. Eclipse. http://www.eclipse.org/ (2004).

[Eclipse Project, 2004] Eclipse Project. Graphical Editing Framework . http://www.eclipse.org/gef/ (2004).

[Eclipse Project, 2004] Eclipse Project. Java development tools . http://www.eclipse.org/jdt/ (2004).

[Eclipse Project, 2004] Eclipse Project. Java emitter templates . http://www.eclipse.org/emf/ (2004).

[Object Management Group, 2000] Object Management Group (2000). *OMG Unified Modeling Language Specification* (Object Management Group).