10

A Graphical Development Method for Multiagent Simulators

Keinosuke Matsumoto, Tomoaki Maruo, Masatoshi Murakami, Naoki Mori
Osaka Prefecture University
Japan

1. Introduction

A multiagent system (Weiss, 2000); (Russell & Norving, 1995) is proposed as an approach to social phenomena and complex systems in recent years. Agents are connected with networks, and they cooperate and negotiate with each other to solve problems by exchanging information. In addition, many multiagent simulators (MAS) are proposed, and some frameworks for developing MAS are also developed. These frameworks make the amount of work reduce. But it is necessary to build models that are required to develop simulators from scratch. It becomes a burden to developers. The models correspond to a model of MVC (Model-View-Controller) pattern. These models would be specialized in the framework and lack in reusability. Problems of these simulator frameworks are shown in the following:

- Implementing models takes time and cost.
- Managing models is very difficult.
- The reusability of models is low.

To solve these problems, this paper proposes a graphical model editor that can build models diagrammatically and a simulator development method using the editor. The method is compared with a conventional method from the viewpoint of usability and workload. The proposed method is applied to some examples and the results show that our method is effective in construction of multiagent simulators.

2. Multiagent Simulators

A multiagent system consists of the following components:

- Agent

An agent is an actual piece of operating software. It senses an environment and acquires information. It acts according to the information.

- Environment

An environment is a field where agents act. It also includes objects.

- Object

An object is a non-autonomous entity arranged in an environment. It does not influence the environment and agents.

In addition to them, a multiagent simulator has schedules for simulations:
• Schedule
A schedule prescribes behaviour of agents, state transitions of an environment, etc. It is invisible for simulations.

In a multiagent simulator, an agent works with other agents or an environment, and gives some influences. The MAS may show an unexpected aspect as a result. Such emergent phenomena are useful for analyzing complex systems.

3. Frameworks for Multiagent Simulators

A multiagent simulator is software for actually simulating behavior of agents on a computer. It is also called an agent base simulator. There are mainly two kinds of methods in building a multiagent simulator. One, you could develop it using existing programming languages from scratch. The other, you could also develop it implementing only required components (multiagent models) by the aid of simulator frameworks. The former, while flexibility is very high, the quality of the software depends on a developer's skill. Because all parts are implemented from scratch, the burden of development is very heavy. The latter, components common to simulations are already implemented, and the burden of development is cut down greatly. In addition to this, the latest simulator frameworks have various functions, and they become very convenient to analyze. This paper focuses on a development method that uses simulator frameworks. Typical existing simulator frameworks are listed in the following:

- Swarm
- Mason
- KK-MAS
- TeamBots
- Repast
- StarLogo
- Ascape
- Breve

Among these frameworks, Repast (North et al., 2005) and Mason (Luke et al., 2003) are made to be target frameworks in the following. These two frameworks take many concepts from Swarm (Swarm Development Group, 2004), and their flexibility and functionality are very high.

4. Graphical Model Editor

In order to describe multiagent models by diagrams, a dedicated editor is needed for drawing the diagrams. This section proposes a graphical model editor.

4.1 Definition of Drawing

Data of agents, environments, objects, and schedules are required to define multiagent models. It is also necessary to determine which diagrams should be used to express these data. These data can be divided into static and dynamic ones. We use class diagrams for static data, and flowcharts for dynamic data respectively. Some examples of model diagrams are shown in Fig. 1.

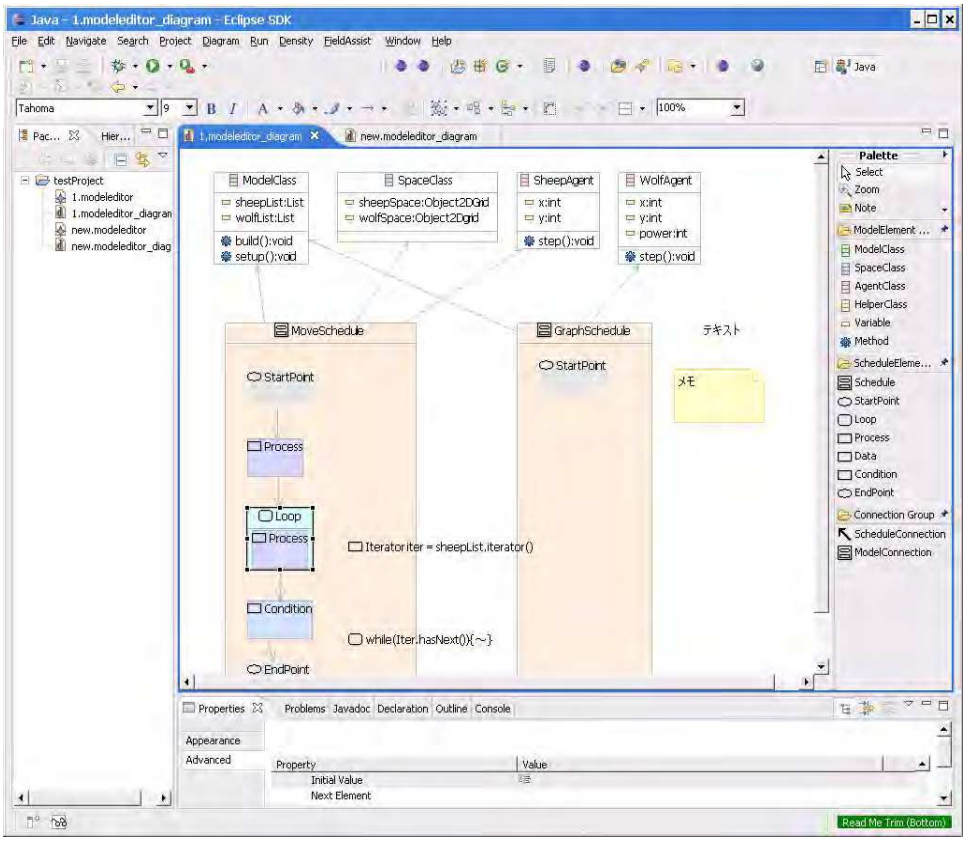


Fig. 1. Examples of model diagrams.

4.2 Definition of Model Structure

In defining model structure, we use a meta-model. A meta-model is a meta-level model for defining specific models. It gives definitions using the data about agents, environments, objects, and schedules. Adding the definitions of model structure as constraints of the model editor, an editing that is contrary to the constraints is prohibited.

The outline of the meta-model is shown in Fig. 2. In this figure, classes under the class object correspond to class diagrams, and classes under the schedule area to flowcharts. Some kinds of class objects and schedule objects are prepared, and these are used as nodes on the editor. The following two things are realized by using this meta-model in Eclipse: If you try to draw an entity that does not meet drawing constraints of the meta-model, the model editor would not allow us to do. When the model is stored, model structure is preserved in the same structure of the meta-model. The meta-model restricts users to use the editor semantically wrong. In addition, the editor use a format called XMI (XML Metadata Interchange) (OMG, 2008) when the model is stored. There is an advantage that this model can be used with other tools in future. This format can raise the reusability of the model.

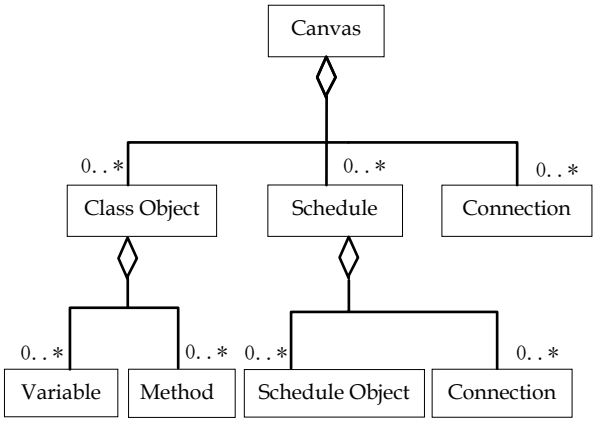


Fig. 2. A definition of the model by a meta-model.

4.3 Definition of Mapping

Drawing information and model structure are defined separately. These data must be connected to each other. We define a mapping from a part of the meta-model to each drawing node. Without this mapping, a situation could happen that you cannot save it even if you can draw a node. The outline of mapping is shown in Fig. 3. Mapping from an attribute of the meta-model to a label of drawing objects enables us to deliver model information.

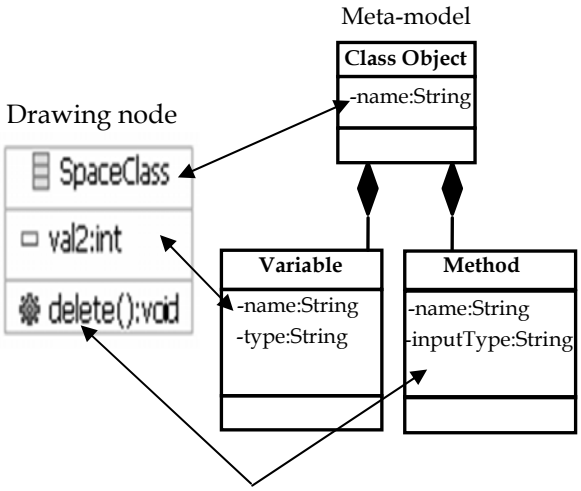


Fig. 3. Mapping of a drawing node and a model structure definition.

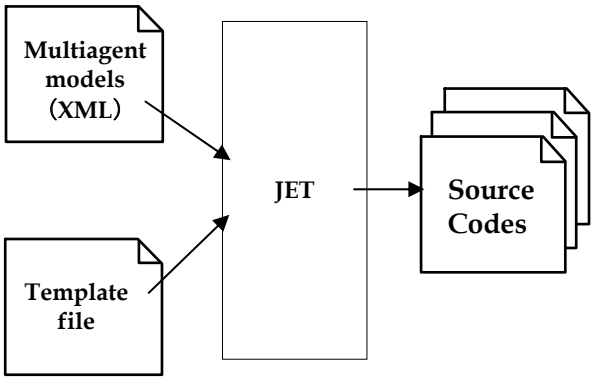


Fig. 4. Technologies for transforming models into codes.

It is necessary to modify the mapping if a definition of drawing, such as a form or colour of an object, is modified. But there is no necessity of changing the meta-model, and vice versa. Since the mapping is only tying up two items, so modification is very easy. Thereby, we can flexibly modify definitions of drawing and the meta-model.

5. Development Method by Model-Code Transformation

This section describes a simulator development method using the proposed graphical model editor. A transformation process of the method is shown in Fig. 4.

5.1 Transforming Multiagent Models into Source Codes

JET (Java Emitter Templates) (Marz & Aniszczyk, 2006) is used for transforming multiagent models into source codes. JET is one of Eclipse project results and it is a code generation technique for improvement in productivity. A code-generator is an important component of Model Driven Development (MDD). The goal of MDD is to develop a software system using abstract models (such as UML models or EMF/ECORE models), and then refine and transform these models into source codes. Although it is possible to create abstract models, and manually transform them into codes, the real power of MDD comes from automating these processes. Such transformations accelerate the MDD processes, and result in better code quality. In JET, codes are outputted using templates. Models can be applicable to various frameworks by changing templates. JET is useful to raise the reusability of models. It needs an XML file storing the model information and a template file for transforming the model into codes. The template file is described for every target language using XPath (XML Path Language).

The model describes XML forms as a tree structure. A template corresponds to a medium which creates source codes by reading information of the model. The tags in the template are simplified tags of XPath, and it is possible to take out the information of input models by designating these tags directly. Source codes corresponding to each platform are generated on the basis of this information.

5.2 Templates

A template is created for Java language because one of the target simulator frameworks, Repast, corresponds to Java. The flexibility of the template depends on the contents in the template. Existing many sample programs are referred to make template's schema as general as possible.

A part of the created template is shown in the following:

```
<?xml version="1.0" encoding="utf-8"?>
public class <c:get select="$element/@elementName" /> {
<c:iterate select="$element/variables" var="vari">
   private <c:get select="$vari/@type" />
    <c:get select="$vari/@elementName" />;
  public void set<c:get select="$vari/@elementName" /> (
      <c:get select="$vari/@type" /> <c:get
      select="$vari/@elementName" />) {
        this.<c:get select="$vari/@elementName" /> = <c:get
        select="$vari/@elementName" />;}
   public <c:get select="$vari/@type" /> get<c:get
      select="$vari/@elementName" />() {
      return <c:get select="$vari/@ elementName" />;}
</c:iterate>
<c:iterate select="$element/methods" var="meth">
   public <c:get select="$meth/@outputType" /> <c:get
      select="$meth/@elementName" />(<c:get
      select="$meth/@inputType" />){}
</c:iterate>
}
```

6. Experimental Results and Evaluations

To evaluate the proposed method, the method is applied to some multiagent simulators, and it is compared with a conventional method from the viewpoint of usability and workload.

6.1 Model Editor

The model editor enables us to edit models graphically and to add information in detail using a property sheet. Code based model generation could be replaced with diagram based model generation by the editor. The model editor is built using the framework of Eclipse, it can be customized, and easily changed. It also increases the reusability of the models created by the model editor.

On the flowchart expression, there is a problem that it can not deal with multiplex loops. As a result, coding by hands still remains. Expressions used in the editor are similar to programming languages such as class diagrams and flowcharts, so that some knowledge is needed for understanding them. To solve these problems, it is necessary to improve the contents of drawing and to consider more intelligible expressions.

6.2 Automatic Code Generation

Some sample simulators were developed using the model editor. The developed sample simulators are taken from sample programs of Repast (http://repast.sourceforge.net/repast_3/examples/index.html) and reference (Yamakage & Hattori, 2002). We examined how much codes were generated automatically by this proposed method. To be more precise, we compared the amount of codes automatically generated by the model editor and model-code transformation with the amount of codes that are manually added. These situations are shown in Fig. 5. The developed sample simulators are shown in Table 1.

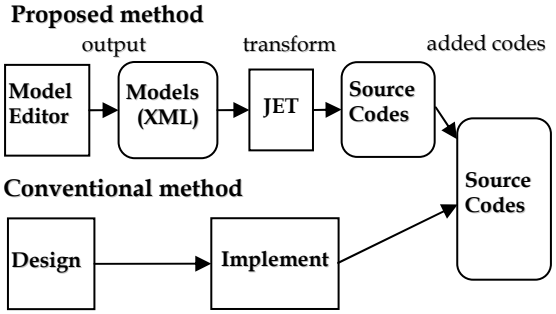


Fig. 5. Comparison of the proposed method and the conventional method.

	Sample name	Total codes (lines) (A)	Automatically generated codes (lines) (B)	Automatic code generation rate (C)=(B)*100/(A)
1	Heat Bug	585	261	44.62%
2	Sugar Scape	490	233	47.55%
3	Regression Office	580	239	41.21%
4	Rabbit Population	364	192	52.75%
5	Open map	296	164	55.41%
6	Neural from file	186	74	39.78%
7	Neural Office	647	199	30.76%
8	Mousetrap	282	128	45.39%
9	Game of life	555	196	35.32%
10	JinGirNew	359	176	49.03%
11	Jiggle Toy	310	116	37.42%
12	Jain	332	142	42.77%
13	Hypercycles	712	201	28.23%
14	Hexa Bug	426	193	45.31%
15	Gis Bug	184	85	46.20%
16	Genetic Office	519	203	39.11%
17	Enn	566	224	39.58%
18	Asynchagents	435	178	40.92%
19	Lotka-Volterra	478	247	51.67%
20	Carry Drop	418	186	44.50%
average		436.2	181.9	42.88%

Table 1. Automatic code generation rate using the model editor.

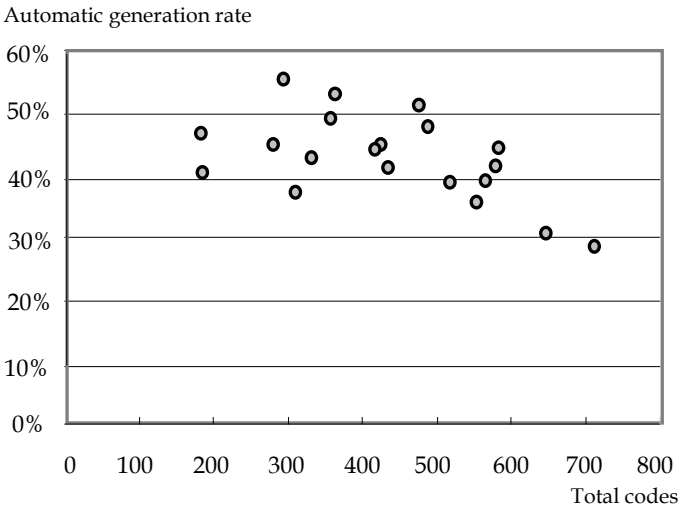


Fig. 6. Automatic code generation rate.

This table shows automatic code generation rates by using the model editor. Fig. 6 depicts the results graphically. It turns out that every sample automatically generates about 40% - 50% of the total codes. The average rate of the automatic code generation attains about 43%. The automatic code generation rates tend to decrease with the increase of total codes. This tendency is caused by the model editor's ability that cannot deal with complex logics. The larger the total codes are, the more codes you need to add by hand. It is necessary to improve the model editor as a future subject.

6.3 Workloads

This section investigates how much the proposed method reduces workload. Unless the workload of drawing models by the editor is less than that of developing models from scratch, it does not make the burden of development reduce. These two workloads were measured. We compare the automatic generated codes to the number of nodes placed on the model editor. The amount of lines per one node is computed in Table 2. The numeric value (E) of Table 2 expresses how many lines are generated from one node on average. The average value of (E) is 2.46.

The workload of arranging one node (two clicks and some little things) seems to be less than that of writing by hand the codes of 2.46 lines (about 62 characters, (G) = (E)*(F)). It is difficult to compare these correctly, and we assume that the workload of arranging one node and that of writing one line are equivalent for the simplicity. Under this assumption, Fig. 7 depicts how much workloads are reduced. First, in the area of automatic code generation,

$$\text{Nodes} : \text{Codes} = 1 : 2.46 \tag{1}$$

because the workload of arranging one node and that of writing one line are assumed to be equivalent, the proposed method can be suppressed in 41% (1*100/2.46) of the amount of workload of the conventional method. There is no difference in the area of the manual generation, since these parts are generated by hand for both the proposed method and the conventional one. The average automatic code generation rate (C) in the Table 1 is about 43%. The workload of the nodes part corresponds to 18% (=43%*41%) as a whole. As a result, 25% of the total workload is reduced as shown in the Fig. 7. The validity of the proposed method is also shown from the viewpoint of workloads.

	Sample name	Total noses (D)	Lines per node (E) =(B)/ (D)	Average characters per line (F)	Characters per node (G)=(E) * (F)
1	Heat Bug	99	2.64	24.1	63.62
2	Sugar Scape	72	3.24	23.7	76.58
3	Regression Office	88	2.72	25.9	70.42
4	Rabbit Population	67	2.87	26.8	76.81
5	Open map	61	2.69	23.2	62.46
6	Neural from file	22	3.36	31.4	105.47
7	Neural Office	110	1.81	24.5	44.32
8	Mousetrap	63	2.03	22.7	46.19
9	Game of life	92	2.13	22.6	48.12
10	JinGirNew	64	2.75	25.8	70.83
11	Jiggle Toy	68	1.71	24.6	41.93
12	Jain	57	2.49	25.2	62.76
13	Hypercycles	110	1.83	24.0	43.77
14	Hexa Bug	76	2.54	23.6	60.00
15	Gis Bug	33	2.58	25.7	66.09
16	Genetic Office	78	2.60	29.1	75.82
17	Enn	100	2.24	23.6	52.85
18	Asynchagents	71	2.51	28.3	70.83
19	Lotka-Volterra	103	2.40	22.8	54.74
20	Carry Drop	90	2.07	23.9	49.43
average		76.2	2.46	25.1	62.15

Table 2. The amount of codes per one node.

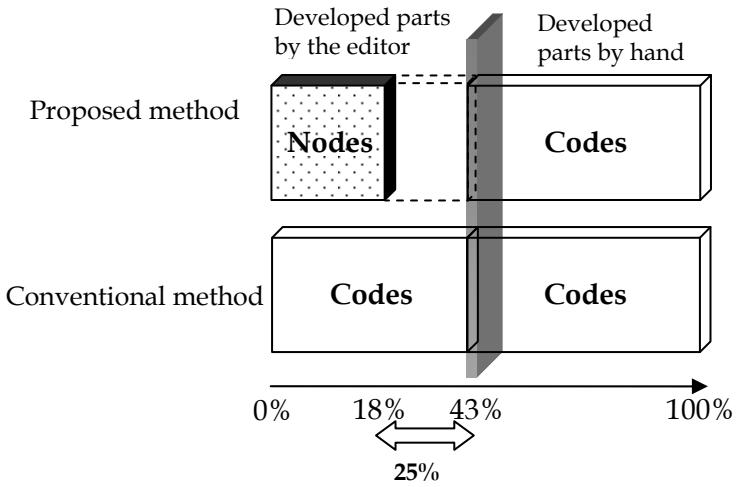


Fig. 7. Difference of the two methods.

7. Conclusion

This paper proposed a model editor that can create graphically multiagent models, and a simulator development method using the editor to build multiagent simulators. Development and management of the models became very easy. In addition, reusability of the models also became very high, and simulator platforms could be changed flexibly. The proposed method was applied to some multiagent simulators and the results show that our method is effective in developing multiagent models and simulators.

As future problems, we must improve the model diagrams to increase automatic code generation rates, and prepare other templates for various frameworks.

8. Acknowledgment

This work was partially supported by JSPS KAKENHI 21560430.

9. References

Luke, S.; Balan, G. C.; Panait, L.; Cioffi-Revilla, C. & Paus, S. (2003). MASON: A Java Multi-Agent Simulation Library, *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*, pp. 49-64, Chicago, USA, October 2003.

Marz, N. & Aniszczyk, C. (2006). Create more -- better -- code in Eclipse with JET, *IBM Developer Works Article*.

North, M.J.; Howe, T.R.; Collier, N.T. & Vos, J.R. (2005). The Repast Symphony Runtime System, *Proceedings of Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, ANL/DIS-06-1, ISBN 0-9679168-6-0, pp. 159-166, Chicago, USA, October 2005.

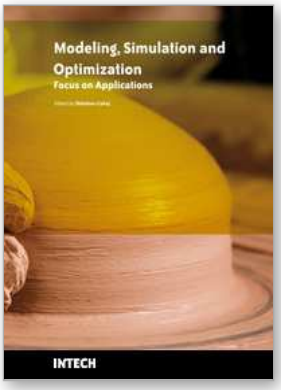
OMG (2008). XMI, See http://www.omg.org/technology/documents/formal/xmi.htm.

Russell, S.J. & Norving, P. (1995). *Artificial intelligence: A Modern Approach*, Prentice-Hall, ISBN 0-13-103805-2, Englewood Cliffs.

Swarm Development Group (2004). Swarm 2.2, See http://wiki.swarm.org.

Weiss, G. (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, ISBN 9780262731317, Cambridge.

Yamakage, S. & Hattori, S. (2002). *Artificial society in computers – multiagent simulation model and complex systems* – (in Japanese), Kyoritsu Shuppan, ISBN4-320-09735-1, Tokyo.



Modeling Simulation and Optimization - Focus on Applications

Edited by Shkelzen Cakaj

ISBN 978-953-307-055-1
Hard cover, 312 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

The book presents a collection of chapters dealing with a wide selection of topics concerning different applications of modeling. It includes modeling, simulation and optimization applications in the areas of medical care systems, genetics, business, ethics and linguistics, applying very sophisticated methods. Algorithms, 3-D modeling, virtual reality, multi objective optimization, finite element methods, multi agent model simulation, system dynamics simulation, hierarchical Petri Net model and two level formalism modeling are tools and methods employed in these papers.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Keinosuke Matsumoto, Tomoaki Maruo, Masatoshi Murakami and Naoki Mori (2010). A Graphical Development Method for Multiagent Simulators, Modeling Simulation and Optimization - Focus on Applications, Shkelzen Cakaj (Ed.), ISBN: 978-953-307-055-1, InTech, Available from: http://www.intechopen.com/books/modeling-simulation-and-optimization-focus-on-applications/a-graphical-development-method-for-multiagent-simulators

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821