

## Dynamic Architecture Simulator Modeling

Ruiqi Ren  
Daoyu Zhuang

## **Abstract**

Dynamical solutions for the development of any software become a hot point nowadays. This paper is trying to modeling an extendable dynamic architecture by using Java language and eclipse rich client platforms in this special domain. The core of this model is to modeling a simple architecture, and implements a simulator based on it, which perform more like a hardware self-adaptor. To reach this goal, the modeling is done with the support of the technology of dynamic software architecture evolution. The product of this research also uses the technique of GEF to develop a GUI for this model, so that it could be demonstrated more understandable and for the model itself, it could be more flexible and easier for others to do the future work.

# Contents

<b>1. INTRODUCTION</b> .....	<b>4</b>
1.1. BACKGROUND .....	4
1.2. ORIENTATION .....	4
1.3. PROBLEMS IN GENERAL AND METHODS .....	5
<b>2. THEORY AND TECHNOLOGY</b> .....	<b>6</b>
2.1. SOFTWARE ARCHITECTURE .....	6
2.2. DYNAMIC SOFTWARE EVOLUTION .....	7
2.3. CORE OF DYNAMIC ARCHITECTURE ADAPTOR SOLUTION .....	7
2.4. BRIEF INSTRUCT TO GEF .....	8
2.5. MORE DETAILS .....	8
<b>3. IMPLEMENTATION</b> .....	<b>10</b>
3.1. STATUS AND PROBLEMS .....	10
3.2. DESIGNS ON DASIM ARCHITECTURE: .....	11
3.2.1. <i>Design of the model</i> .....	11
3.2.2. <i>Design of Events</i> .....	12
3.2.3. <i>Design of GUI</i> .....	12
3.3. RESULTS OF THE DEVELOPMENT ON DASIM ARCHITECTURE .....	15
3.3.1. <i>Standard DASim Model Events</i> .....	15
3.3.2. <i>GEF GUI</i> .....	17
<b>4. EVALUATION</b> .....	<b>19</b>
4.1. GUI .....	19
4.2. FUNCTIONALITY .....	20
4.3. SCENARIOS .....	21
<b>5. CONCLUSION AND FUTURE WORK</b> .....	<b>23</b>
5.1. SPECIFY SIMULATOR .....	23
5.2. EXTEND BASIC EVENTS .....	23
5.3. EXTEND BASIC OPERATIONS .....	23
5.4. NETWORK DYNAMISM CONTROL .....	24
5.5. OTHER SPECIFICATIONS .....	24
<b>6. APPENDIX</b> .....	<b>25</b>
6.1. USER MANUAL .....	25
6.2. REFERENCE .....	29

# **1. Introduction**

## **1.1. Background**

Nowadays, our life more and more depends on computer and software, one trend today in computing is autonomous software system. The new area brings new requirements to software developers. That is run-time adaptability. Systems today require more flexibility and availability in different levels. A strong and integrated system must be able to adapt to changes in the environment, such as self-healing and adaptation. Customers will never be pleased with constantly maintaining or updating their software every time it needs to be update or changed.

Developing self-adaptable applications is currently a one-time achievement. What should be done now is to change the current situation, to shorten lead times, reduce costs, and improve quality.

## **1.2. Orientation**

Through over the description above, we can easily see that the technique infrastructure for customers in self-adaptation area is not sound yet. These techniques are not mature. Still there are many places can be improved. We would like to make an attempt to contribute to this area. Software reuse could be a nice path to get into the domain. Object oriented Framework is a proven reuse technology and projects that apply frameworks demonstrate the improvements we are looking for. The goal for this thesis project is to develop a tool, which will assist developers to develop attach adaptation to software architecture with framework support.

The aim is to reuse technology and reduce costs for others to develop their own application with dynamic adaptor designed in their architecture. It can save a lot of extra efforts if a development tool is available while developing a product, comparing with the developers developing their own dynamic adaptation from scratch.

As we said in the abstract, the final aim is to develop an integrated IDE tool, a workspace studio, which should aim at developing self-adaptation systems for their own software architecture. Use this tool to generate a framework for their architecture. The framework will detect any kind of evidence of environment changes and even generate and catch the change information from the architecture itself. The benefit from this approach is that software developers do no more consider implementing a new complex self-adaptation system of their own in every detail. Instead, they can just simply invoke the methods from the framework in their own software architecture and that will be enough to deploy any kind of self-adaptable strategy.

Simulator for example, is a very basic concept in this thesis. The simulator here is like a hardware adaptor. When bound to software architecture, it detects situations like changes to the execution environment, a system crash, and updated information. It collects necessary internal and external factors, such as parameters and analyzes them; then it takes the most proper strategy to deal with the events.

The tool, the DASim Studio (Dynamic Architecture Simulator Studio), which will be the final product of this work, will be intended for software developers to easily and conveniently develop their own simulator instance for their software architecture. Exactly, each simulator the developers generate for their own product is a framework wrapping their software architecture.

In our thesis, we are not going to do any job about the IDE. Instead, we are going to develop a simulator model, which will works like a real simulator that could be

generated by an IDE tool. This model could be used by others to develop the IDE tool that should be the future work.

### **1.3. Problems in General and Methods**

The main problem in theory is to find a breach in the references and try to ascertain a particular work that we will try to improve. How to conclude a proper technique and look for a doable method to improve it could also be an important problem.

The basic approach is to understand those technique references which have been found. Analyze those techniques, and look to some related techniques which have not been completed yet or not sound enough. Make a design to improve it. The second thing then is to search all what we need to implement the design, command those tools in the area we needed. The third step is to complete the product and try to make it better. At last, analyze the product in serial aspects and give a impersonal comments on it.

Problems in particular in implementation and design will be introduced more in detail in next section. The basic conceptions will be a start and we will approach the core of our model step by step.

## 2. Theory and Technology

First of all, let us start from basic concepts of software architecture. That is the most essential base of all what we are going to study, to design an architecture tool that solves the problem. A typical architecture for software could be composed by component, connector and configuration. We will soon introduce this abstract in more detail. Then we will expend this concept into dynamic area. What factors inside the architecture so that we could classify it as a dynamic architecture? The most important original thought of the design (consist of Initiation, Evolution, and Control) will gradually appear and you will find out how our thoughts get into the object in practical.

### 2.1. Software Architecture

#### Static Software Architecture

In the descriptions of static software architecture styles, there are three architectural entities/artifacts, **component**, **connector** and **configuration**. These are three entities recognized as the basic elements that build up the structure of any software architecture. It is easy to find several similar definitions in other architectural articles and books. Below, we give definitions of these artifacts.

#### Component

*“A **component** is a state bearer where computations can be performed. All interactions with a component take place at a typed interface.” [1]*

The definition here however, is in a quite formal style and maybe a little hard to understand. Generally, a component is the most basic element composing software architecture. A component usually is not only one instance. Most of time a component consists of quite a lot of objects. It is more like a kind of ‘function’ which does a part of work inside the architecture. However, these ‘functions’ can even be coded with different languages if it is necessary. It could be only one class, but most of the time, it will contain lots of classes. At runtime a component will be instantiated to a group of objects that map to the static class structure. Each Component has its own interface, which is the only way communicating with other components inside the architecture. Those interfaces are methods for other components or connectors to invoke. There is no specialization with the dynamic software architecture. The basic element still is component.

#### Connector

A *connector* is a specialization of a component that responsible for the interactions between components. End-points of a connector connect to components’ interfaces. As we said before, two components may even using different languages. In this case, there always exists a special connector capture the interaction data flowing between them and does the translation work. If we consider connectors as specialized components, it is no doubt that it has the same name in dynamic architecture as well.

#### Configuration

A configuration is a combination of connected component and connector instances. In fact, we can consider each configuration is a shape of software architecture. It is a network which made up by components connected with connectors. Beside the structural information, a configuration can contain a series of constraints constraining the configuration. Such as the rules for instance, describe how components can be connected and combined. Dynamic architecture also maps to the general understanding of a configuration.

## 2.2. Dynamic Software Evolution

Dynamic Software Evolution is a technology that to modify an executing application to satisfy the changed requirement. This technology supported by means of dynamic software architecture, an expensive technique that require lots of extra design and implementation development and run-time resources. There are three software properties can be referred here: **dynamic quality footprints, adaptability** and **dynamic feature sets**.

A system, of which the environment is static but some requirements like security may need to be varied over time. This can be referred as a dynamic “quality footprint”. The firewall of your system is a good example. You need to vary the secure level according to different situations, but the environment maybe has not changed.

Adaptability is required when the application is affected by changes in the environment. If some emergency happen to the system such as resources cannot be found, a system which has adaptability should do something to recover the system or to reconfigure the system to survive. In order to continuously comply with functional and quality requirements the adaptable system should to be able to detect related changes and adapt its structure.

In some applications, they need to alternate the architecture, so the application itself needs to extend its functionality, i.e., it needs to grow up. This is referred to as dynamic feature sets. An example of dynamic feature sets is systems that provide plug-in capabilities. During application execution a dynamic loader binds functionality available in dynamic libraries to certain function-place-holders defined in standardized interfaces.

There are two categories of evolution, **expected Scenario** and **un-expected Scenario**. The first category represents all changes that the developer either knows about during the design period or can anticipate during development. Since that, a solution can be found and could be implemented during the developing process. Because we are able to know changes in advance, dynamic support can be built in to the application. The second category represents future changes that can not be anticipated in advance. Such kind of change requires provisioning which should be provided by either the platform or something inside the application prepared by the developer. Another dimension of evolution is time. Even if we can expect a change, it can be practically impossible to find when the change will occur.

## 2.3. Core of Dynamic Architecture Adaptor Solution

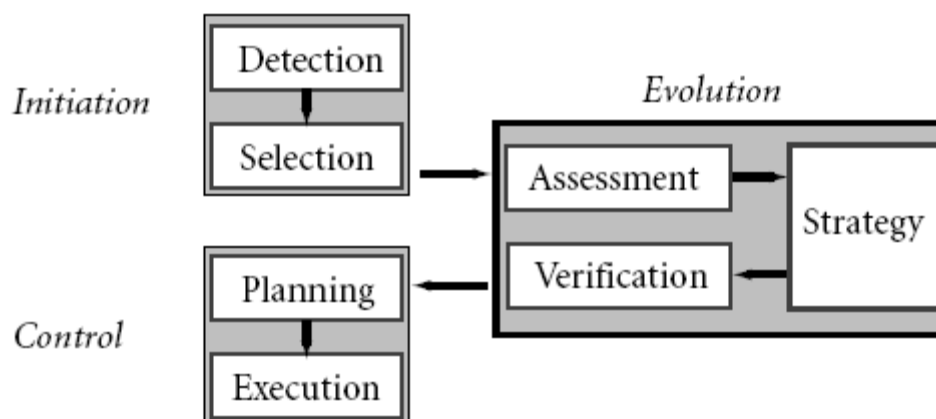


Figure 2.1 Core of Dynamic Architecture

Here is the core solution for our DASim, the design of our simulator, which is shown in Figure 1 [1], contains three components: *Initiation*, *Evolution*, and *Control*. The initiation component continuously detecting events happening inside or outside the architecture, which is the job of the subcomponent: *Detection*. Then, it collects indispensable data from the event. Subcomponent *Selection* takes them as parameters. The Evolution component use some algorithms to analyze the data which initiation component offered, try to assess the situation the architecture is faced with, then invoke the proper control component to execute the changes of architecture itself.

## 2.4. Brief instruct to GEF

GEF [3] stands for Graphical Editing Framework. It allows users develop graphical representations for existing models. And one important thing for GEF is Draw2D, which stand for 2D drawing framework based on SWT [2] from eclipse.org. [5]

The editing possibilities of GEF allow you build graphical editors for nearly every model. With these editors, you can change element properties or change the structure of your model in different ways at the same time. These modifications to the model can be handled in a graphical editor using very common functions like drag and drop, copy and paste, and actions invoked from menus or toolbars.

Draw2D [4] provides the **lightweight graphical system** that GEF depends on for its display. It is packaged in Eclipse as a separate plug-in. As we said earlier, Draw2D is a self-contained graphics library and can be used independently of GEF or even of Eclipse. It is very convenient to use Draw2D's draw methods to draw the connections, and class diagrams.

EditParts are the controllers specify how model elements are mapped to visual figures and how these figures behave in different situations. And you need one edit part for each model. When it was created, it is not yet visible or active. When GEF inform it is active, it can be seen. And if you do not need it anymore, just use deactivate.

Each EditPart should create a figure, which is returned in the create figure method.

## 2.5. More Details

### **Lightweight graphical system:**

A lightweight system is a graphics system that is hosted inside a single heavyweight control. The graphics objects in the lightweight system, known as figures in Draw2D, are treated as if they are normal windows. They can have focus and selection, get mouse events, have their own coordinate system, and have a cursor. They each get a graphics context for rendering. The advantage of lightweight systems is that they are much more flexible than the native windowing system, which is generally composed of rectangular components. They allow you to create and manipulate arbitrarily shaped graphics objects. Because they simulate a heavyweight graphics system within a single heavyweight window, they allow you to create a graphically complex display without consuming a lot of system resources.

### **org.eclipse.ui.part.EditorPart:**

Some source package refers to the EditorPart in the artifact.

### **EditPolicies:**

The **requests** (which are specified later) are forwarded to EditPolicies. They are the parts in GEF that bring the editing functionality into EditParts. And it defines what can be done with an EditPart.

### **Commands:**

A command is the part that actually modifies your model. It simplifies the way of

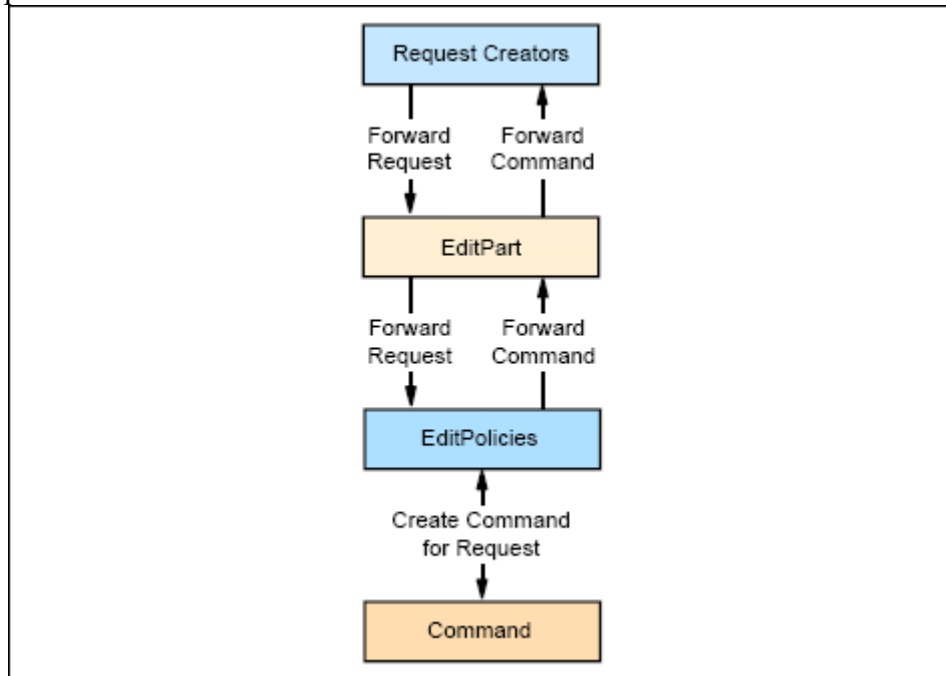


modifying your model by support:

1. Execution limitations
2. Undo and redo.
3. Combining and chaining.

**Requests:**

Requests are the object that uses to transform the information between all components. They contain the information that might be necessary for executing the request later.



**Figure 2.2 Communication Objects Used in the GEF.**

**GraphicalViewers:**

It provides a seamless integration of EditParts into the Eclipse workbench. You don't need an editor for the viewer, and a GraphicalViewer can be used anywhere an SWT control is available.

**RootEditParts:**

It is a special kind of an EditPart. Its task is to provide a suitable and homogeneous environment for the real EditParts that represents your model. It is an interface between a GraphicalViewer and your model EditParts.

The things list above is all used in our project. Each part is in a separate package.

### 3. Implementation

This section we will focus on our product in very detail and will also demonstrate the results to help you understand more clearly of what we have done. Next chapter we will analyze the product in very aspect, give some comment to let you have a proper idea of how useful that our program is in many useful areas.

#### 3.1. Status and Problems

After studying in area of dynamic technology for a week, we both think it is the time for us to try to develop something now. We plan to assume some related questions, try to find the answer or find a method to solve it. Then think of new problems and find new solutions. We regard each cycle as an iteration course. We plan to make a design of solution for the problems we will assume at first during each iteration course. Each iteration course will continue for 1 ~ 3 weeks. Problems we assume will be list in this chapter, and both design plan for each iteration course and results will be available in next chapter: Implementation. Statuses described here are just a brief recapitulation. More detail descriptions are in the subchapter *Results* under chapter *Implementation*.

##### **Status and Problems for Iteration I:**

###### **Status:**

We now decided to use Java to develop our model, it will not be a wisdom to make a complete model design and try to figure it out, that will be too complex and we will really run a high risk of the project since we can never predict what problem we will meet and how long it will stumble us. Here we got the idea of generating a simplest model, as simple as we can. Make sure that we can finish it in a short time. Then we can extend it step by step in the future days.

###### **Problems:**

- How simply the model should be so that it will be easy to make a plan for next step and we can finish it in a few days?
- How can we find an aspect to get to design and implement the first model?

##### **Status and Problems for Iteration II:**

###### **Status:**

The model which we have now is quite simply, a program load instructions from a txt script file and do no more operations than write corresponding result into another xml file. It even does not look like any kind of software or tool, but it is quite flexible and easy to design the extending plan.

###### **Problems:**

- How can we extend the first model so that it will work a little bit like a self-adaptable frame binding to the existing simplified architecture?

##### **Status and Problems for Iteration III:**

###### **Status:**

Now we have got a simplified dynamic model. From this step, we should design carefully in each step, to make sure our development is on the right track. To be extendable for more kinds of events, the component, Event Generator will no doubt being focused on in this iteration course. And it is not easy to make a design in detail to fulfill the core design of our simulator in last chapter. Since we have a basic structured dynamic model, we should now think about the event flow for future development.

###### **Problems:**

- How to extend the Event Generator component so that it could be extendable

for more kinds of events?

- How to design a reasonable event flow for further development and design?

#### **Status and Problems for Iteration IV:**

##### **Status:**

To make it more extendable, the model has been modified a lot during last iteration course. The event flow has been changed and no more events generated directly by the Event Generator. We also designed to develop an Event Detector component model inside the Event Generator to integrate this component according to the design of Initiation component in our core design, and we change the name of this component in our model from Event Generator to Event Generator/Detector. However, we met some problem about how to implement the message sending and detection system. We also got mad on the events design and implementation. Our supervisor request us to standardize the events in our model, complete the development of the original self-adaptable model and start to learn GEF/EMF so that we can use this technology to generate a GUI for our model.

##### **Problems:**

- How can we find our own path to implement the detection of the simulator?
- What kind of events is necessary for our model?
- How to standardize the events system in our model?
- What can we know from GEF/EMF?

### **3.2. Designs on DASim Architecture:**

#### **3.2.1. Design of the model**

As a beginning of the implementation, we try to simplify the simulator model design so that we will have a good start. We just want to throw the normal architecture and components away. Our supervisor suggested that we can start from loading design language from a script file which contains basic component operations such as add component and delete component. And export the outputs to another script.

Then we extends the first model, try to make it more work like an adaptation simulator. We design to implement a special component, which works like an event generator. We consider just implement the simplest event generator first. This event generator only sends a message to simulator once every 5 sec. During this step, our simulator will do only add component operation, we design to create a new file which will be a log file recording all operation the simulator did.

Third part, we continue further extending the DASim Model. We divide tasks into three parts: Events Generator/Detector (EG&D), Simulator and Normal Component.

Events flow now will be changed like this: Events Generator/Detector will have a series of interfaces for the developer to invoke. The developer who is going to use Simulator Studio to develop their architecture will invoke those methods in their codes when generate instances of their components. So the developer can make their own decision of when, where and what kind of events should be detected. Events Generator will load the events list from its configuration file. Then it will either detect system events or generate man-made events and send specific signals for each event to Simulator.

Events Generator/Detector could be regarded as a part of the Simulator. In the model that we are not going to develop during this phase, it will play the most important role of Simulator.

At last, we have two main tasks to over our projects. First step, we are going to select events which are necessary for our simulator model, and standardize some events for

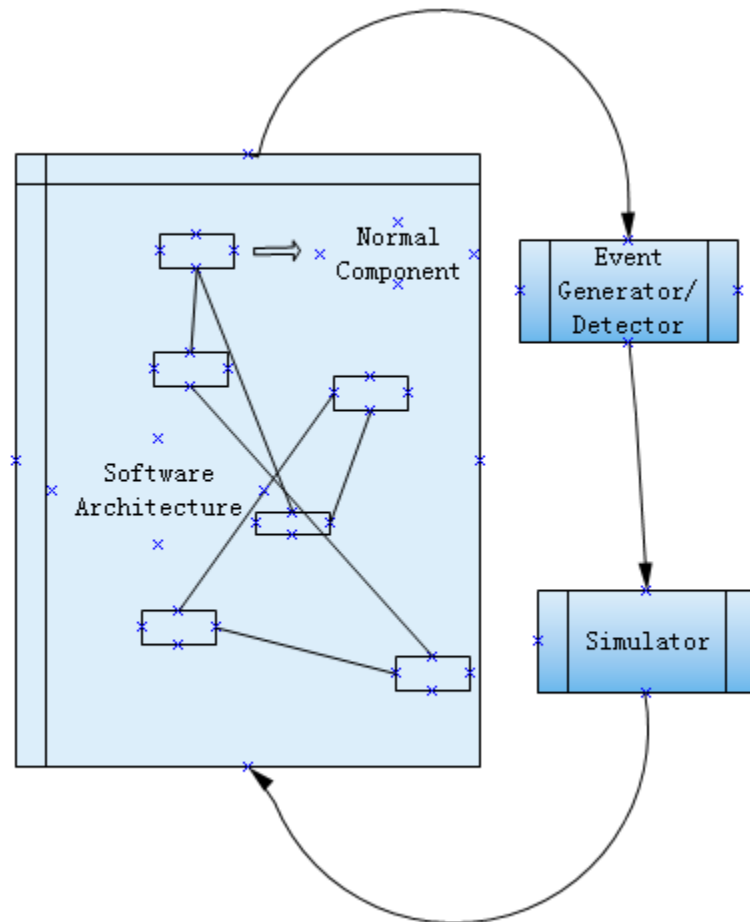
our model. Second step, start to get familiar with the technology and think about how to use it to generate a GUI for our model.

### 3.2.2. Design of Events

After read some dynamic architecture technology articles, we divided the events in dynamic architecture into three different categories: Internal, External and self-Evolution. Internal events are the events happen inside the software architecture while external events on the other hand, are events appearing outside the architecture. Self-Evolution should belong to category External. Since this kind of scenario is quite popular and typical, we decide to separate it from category External and distribute it to a new category alone. We design to standardize one event for each category as typical examples. We also design to generate some User Defined Events as examples for the model. These designs have been extended in detail in subchapter *Standard DAsim Model Events* in the section *Results*.

### 3.2.3. Design of GUI

When the standardization has been finished, we find it is clearer on how to implement the message sending and detect system, since each kind of event has been specified. So we plan to complete implementing the detect components for the Event Generator/Detector as the plan described in last iteration course.



**Figure 3.1 Events Flow of the Design**

**Events Flow:**

Figure 3.1 shows the events flow in general. When execute our program of the model, it should first instance the Simulator model together with the Event Generator/Detector, and initialize them. Then load the architecture description language in the input file, the Original Architecture File. Execute the architecture description language program and generate a virtual software architecture based on the codes in the Original Architecture File in the memory. The components in the software architecture should contain the invoking of the interface method of the Event Generator/Connector in their initialize functions. When the architecture is executing, it should generates valid events signal to the Event Generator/Detector. Before that, when the Event Generator/Detector was initializing, it will load the Events-operation Table from the Configuration File. According to the table, the Event Generator/Detector will send related messages to the Simulator. Finally, the Simulator will perform the corresponding architecture modification operations and add change records to the end of the Log File.

**Architecture Description Language:**

We design the description language ourselves. Only simple design has been used, which contains the most basic operations, to be more extendable and easy to see the results. The result would be listed under the Section *Results*. During the design phase, we think that it will take the style like ‘add component()’, ‘delete component()’, ‘add connector()’, or something like that.

**Original Architecture File:**

This file we designed to use the form of XML. It should contain the architecture description language program which performs the original architecture. Although this file uses the architecture description language which we will design, it should only show the entire architecture with the briefest way. That means it should only contain the necessary steps to generate architecture without any kind of re-code. The operation like “Add component A; Delete component A;” should not be allow in this file.

**Modified Architecture File:**

This file also has been designed to use the form of XNL, and contains the architecture description language program which performs the modified architecture. Like the Original Architecture File, it also should only show the entire modified architecture with the briefest way.

**Configuration File:**

This file is the input file of the Event Generator/Detector. It has been designed to using some form of description language to structure an Event-operation Table. This table describes a design of events and their corresponding operations where the events contains the events that the software architecture want the simulator to detect and the operations are those measures to take when those events happens.

**Log File:**

This file just uses the simple form of TXT files. This file should contain the records of which all the operations relate to modify the input original architecture has been done. The record should be added to the end of the file once a Valid Event occurs. It should contain basic information at least: the time of when the event happens, the description of the events, all corresponding operations which have been done.

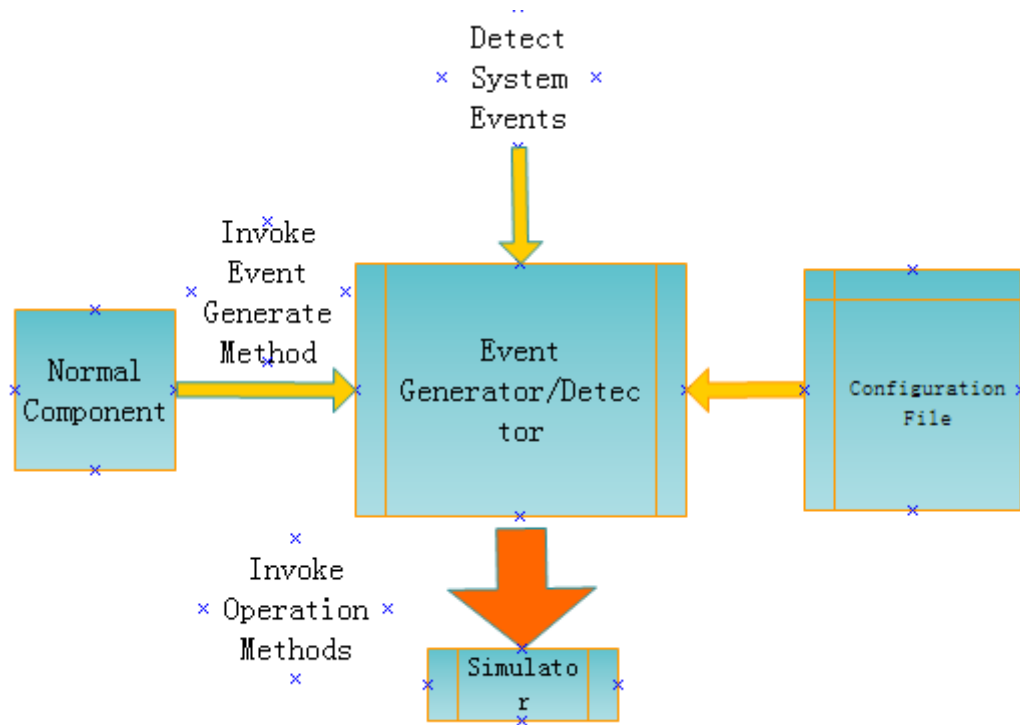
**Normal Component:**

This abstract could be mapped to the definition of Component which has been mentioned under Chapter *Theory and Technology* in Section *Software Architecture*. However, in our design, we decide to simplify it in our model. Since we are using java to program, we plan to design a class to interpret Normal Component, which should contain two attributes: name and ID. We will extend them with invoking events methods which have been listed in the configuration file of Events Generator/Detector before,

when they had been instanced.

**Connector:**

This abstract could be mapped to the definition of Connector which has been mentioned under Chapter *Theory and Technology* in Section *Software Architecture*. However, in our design, we decide to simplify it in our model as well. We plan to design only one kind of connector which only contains the connector ID, the two components which the connector links to. That means, it is just a class which symbolize the connector, but actually there is no more operation between to components to be performed by the connector.



**Figure 3.2 Relations between the Main Components in Design**

**Event Generator/Detector:**

A special component, but to separate it from the component of the input software architecture, we use the word Normal Component to distinguish the components from actually two architectures. The abstract of this special component could be mapped to the abstract of Initiation Component which mentioned under Chapter *Theory and Technology* in Section *Core of Dynamic Architecture Adaptor Solution*. First, we decide to simplify it to only perform as the subcomponent Detection. Second, we should extend it to perform some ways similar with an Event Generator. Figure 3 shows the work flow of this special component.

**Simulator:**

This abstract could be mapped to the abstract of *the design of our simulator* which has been mentioned under Chapter *Theory and Technology* in Section *Core of Dynamic Architecture Adaptor Solution*. As we said in the last paragraph, we have just separated the Initiation Component from it. And we also design to simplify the functions which should be performed by the Evaluation Component and Control Component. In the design of our model, the simulator should perform the corresponding operation when it got messages of valid events from the Event Generator/Detector. Simulator itself should contain the implementation of every basic operations like the implementation of add component and delete connectors.

**Interfaces of Events Generator/Detector:**

We design the interfaces of Events Generator/Detector could be only one method which contains one parameter: NameOfEvent.

When this method has been invoked, EG&D will send a series of signals which specify every operations will be executed when the specific event happens according to the configuration loaded from Configuration File.

**Ex:**

If the name of the class which specify Events Generator/Detector is “EGDetector” and the name of its interface is “happen”, then code in Normal Component could be:

```

...
EGDetector detector = new EGDetector();
detector.happen(“OnMouseLeftClick”);
detector.happen(“SystemCrash”);
...

```

Then, in the EG&D, if the configuration is like this:

Name Of Event	Corresponding System Events	Operations
...	...	...
“OnMouseLeftClick”	OnMouseLeftClick	<pre> If(Find(“A”)){ /*Find component “A” */     Delete (“A”); /*If found, delete “A”*/ } Else{     Add(“A”); /*If not, add “A”*/ } </pre>
“SystemCrash”	None	<pre> Recover(){ // suppose default architecture has Add(“A” + id); //only component “A” ++id; // and component “B”, then Add(“B” + id); // restructure a new AddCon(“A”+(id-1), “B”+id); //architecture. ++id; } </pre>
...	...	...

If the Correspond System Events value is not “None”, then the EG&D should detect those system events from the local System and send signals to Simulator when those system events happens.

### 3.3. Results of the Development on DASim Architecture

#### 3.3.1. Standard DASim Model Events

According to the scenario of dynamic architecture that we are going to solve, here defines the most basic standard events, customers can add new events by combine the basic standard events below. The events below are only for the DASim Model we have developed.

All Standard Events should be implemented inside the special component: the Event Generator/Detector. Each Event should relate to a serious of specify protected methods, which can only be invoked by the original Event Generator/Detector itself or its children class.

<b>ID</b>	<b>Name</b>	<b>Type</b>	<b>Description</b>
0001	MouseClicked	External	Response to the System Mouse Right Click Event.
<p>In this event, the Event Generator/Detector should detect the System Mouse Right Click Events during the detection duration. To implement this event, one suggestion could be like this:</p> <p>Implement a method which initialize the detection process and start to detect Right Click Mouse Event.</p> <p>Implement a corresponding Method will only be invoked when a System Mouse Right Click Event happens, catch the x , y of the mouse coordinates and the time value of when this event happened.</p> <p>Implement a method to stop the detection.</p>			

<b>ID</b>	<b>Name</b>	<b>Type</b>	<b>Description</b>
0002	Periodical Checking	External	Call back once every X sec/min/hour/day
<p>In this event, the Event Generator/Detector should calculate the time changing, customer can use this event to do some operations in a periodical way. To implement this event, one suggestion could be like this:</p> <p>Implement a method which initialize the calculating process and start to count the time, this method should contain a parameter of the frequency.</p> <p>Implement a corresponding Method will be invoked once every X. The X is depending on the parameter of frequency.</p> <p>Implement a method to stop the time counting.</p>			

<b>ID</b>	<b>Name</b>	<b>Type</b>	<b>Description</b>
0003	Component Existing Detection	Internal	Response to the scenario if found a special component is existing or not.
<p>In this event, the Event Generator/Detector should detect the basic change of configuration. That is, detect if a specify component is still existing or not. To implement this event, two suggestions could be like this:</p> <ol style="list-style-type: none"> <li>1. Implement two message sender methods, which should be coded in the initial function and finalizer function of each normal component respectively. When a normal component has been created or destroyed, this method would be invoked and send the Event Generator/Detect a message that a specific component is existing or has been destroyed.</li> <li>2. Implement the similar way of the Standard Event ID 0002, check if a specify component is existing or not in a periodical way.</li> </ol> <p>Whichever approach has been taken, we should implement an initial method which contains parameter: name or ID of a component, a bool value to determine which state is going to be detected, to find if a component is existing or to detect if a component has been destroyed.</p>			

<b>ID</b>	<b>Name</b>	<b>Type</b>	<b>Description</b>
0004	Version Detection	Adaption	Check from a specific location where the latest version of architecture has been placed.
<p>In this event, the Event Generator/Detector should search a specific place (e.g. e:\version.txt) to check if the version number of current running architecture is the same as the latest one in a periodical way. To implement this event, one suggestion could be like this:</p> <p>Implement a method to check the latest version file, with a parameter of frequency.</p> <p>Implement a method which will be invoked once every X , where X is determined by the frequency parameter.</p>			

### **User Defined Event**

A User Defined Event could be a combination of the Basic Standard Events. Normally, the user (customer) generates their own Event Generator/Detector from our Model. Add their own public event method which will invoke the protected standard events methods.



## User Defined Events in Our DASim Model:

ID	Name	Type	Description
0101	Add	Self Defined	Add a new component which will be connected to the latest created component once every 5 sec.
<pre> StandardEvent(ID_0002). Initial(5, AddCom); ... AddCom() {     Int oldID, newID;     oldID = Architecture.getMaxComponentID();     newID = oldID + 1;     add(newID)     conn(oldID, newID); } </pre>			

ID	Name	Type	Description
0102	Delete	Self Defined	Delete a component once Right Mouse Click Event happens.
<pre> StandardEvent(ID_0001). Initial(DelComInit); ... Private int ID; DelComInit() {     ID = Math.random() * 10;     StandardEvent(ID_0003). Initial (ID, true, DelCom) //Initial(ComponentID, isExisting, //callbackFunction) }  DelCom(){Del(ID); } </pre>			

We also implemented more User Defined Events, like:

**Update;** (Automatically)  
**Recover;**  
**Change Configuration;**

### 3.3.2. GEF GUI

#### Editor class:

It is created by extending org.eclipse.ui.part.EditorPart. It is the main class of the editor and is responsible for receiving the input, creating and configuring the viewer, handling the input and saving the input.

#### EditDomain:

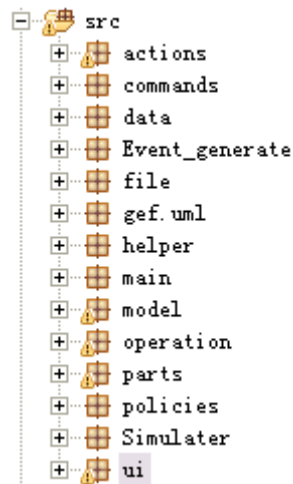
It is an interface that logically bundles an editor, viewers, and tools. It defines the real editor application. It provides a **commandstack**, which keep track of all executed commands.

#### Actions:

The actions are used to define what to do and how to do when the menu are chose. By define it you can use the framework eclipse gives.

#### Adapting to the **properties view**:

Editparts are responsible for delivering **IPropertySource** adapters for the properties view. GEF provides a solution to cover modifications occurred in the properties view into the commandstack. So if you choose a component in the figure, GEF will get its properties and shows them in the property window.



**Figure 3.3 Component Tree of the Code in GEF**

In the **model** package, stores all information about model, include component, connection, and content (which can be seen as dynamic architecture.)

In the **parts** package, stores all information of the edit part, which specify how to draw the figure for the component, for the connection, and for other things, and the operations on the model can also specified here.

In the **policies** package, stores all information about if a command is active, what to do and how to do. And to use these policies, you need to install it into the part.

In the **UI** package, it specifies when load architecture from file, what to draw on the window. It gets figure form all parts, and shows it to user.

In the **command** package, it stores all the information about command, for example, the delete or adds component, or new a connection.

In the **actions** package, it specifies when click the menu, what to solve. Here I specify some dialogs to show the information and other things.

## 4. Evaluation

Now the project has almost been finished. In this chapter, we will try to give you an intuitional idea of the product we developed and recommend some domain which our program could be useful. We will begin from demonstrating the GUI of the program which may attract a lot and bring your mind on the track immediately. Then we will briefly introduce the functionalities by providing a basic work flow. List some scenarios and suggest some areas based on them.

### 4.1. GUI

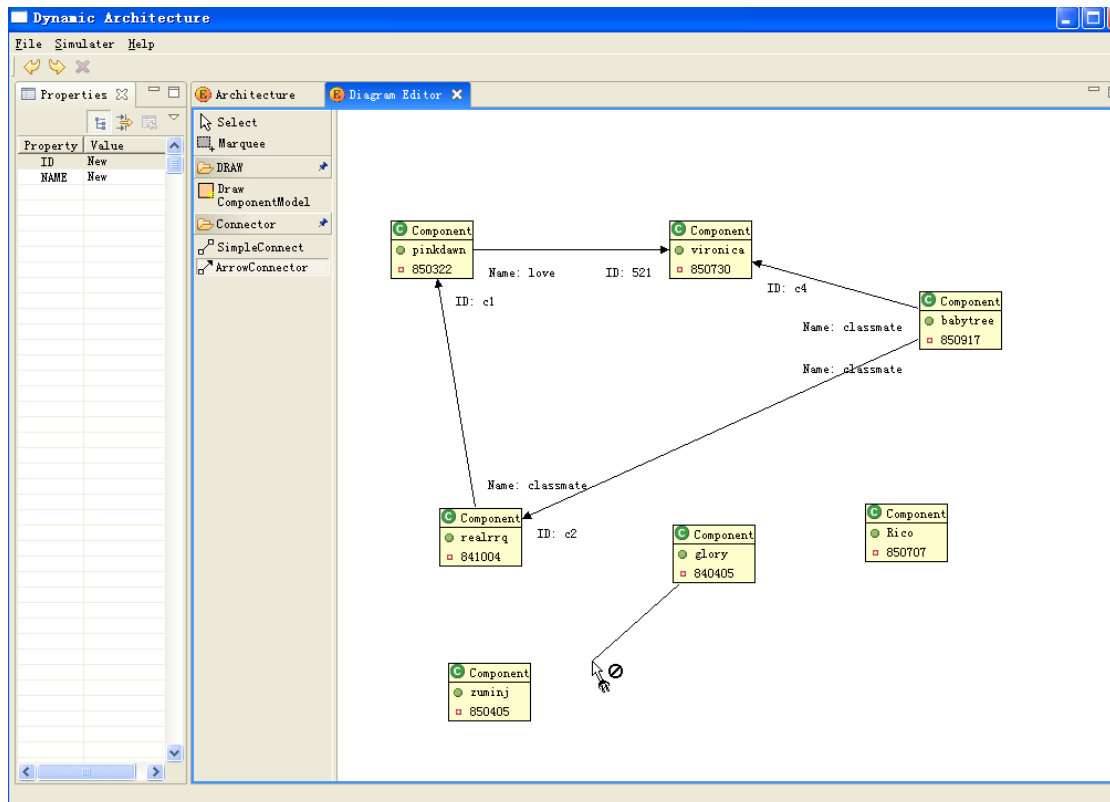


Figure 4.1 Screenshot of the Final Program

By utilizing the GEF bonding to eclipse IDE, the GUI finally has been developed and wraps the DASim Model, which is shown in figure 4.1. To implement the GUI through GEF need some plug-in installed in eclipse and this program is an eclipse program as well.

The program could be divided into four main parts: **Menu**, **Properties**, **Toolbar** and **Demonstration Area**. The Menu contains File (new, open, save, save as, exit), Simulator (Event Generate, Simulate), and Help (about only). The Properties shows the properties of current selected component, which you can modify here. Toolbar contains two basic selecting methods: single-selection, and square-selection. ComponentModel is located in draw folder, and two kinds of connectors, SimpleConnect and ArrowConnector are available in the Connector folder. All the objects above in Toolbar are used to draw the architecture manually and visibly. The Demonstration Area is no doubt the place for drawing new architectures through toolbar or to demonstrate an existing architecture. Each component demonstrated here has a title with the head of 'c', which label this object is a normal component. And it also will list the ID and the name

of that component. Although connectors are shown as arrows and lines, then also has their own name and ID.

## **4.2. Functionality**

### **Basic workflow**

The program can draw static software architecture, demonstrate it, load existing architecture, and show the result when there is a 'change' occurred in the architecture. The 'change' here specially means the effect from simulation process.

To start to use this tool, first, one should File→New to create a new blank architecture or File→Load to load an existing architecture. The architecture file should be in the form of XML. If one wants to design a new architecture or modify an existing architecture, he/she can use the operations in the toolbar to draw components, link them with different connectors, and change their ID, name or other properties in the Properties panel. The new architecture could be saved to another XML.

To demonstrate a simulation process, one first needs to either generate a new architecture or load an existing architecture. Through Simulator→Event Generate to active a dialog. Here one should load a script file related to events and corresponding actions. This file should be created first, this tools does not support to edit any script file. However, in this dialog, you can modify some existing script and save them to another script file. If no problem found in the script, one can then click on OK button. Events and related operations have then been remembered by the program. The final step to do is to active the simulation process. The Program will first want you to make sure about the location of the script file, and then want you to input an address and name for record the logs of any operation taken by the simulator. If everything is ready, the Event Generator/Detector will start to run together with the simulator and the architecture diagram shown in Demonstrate Area will update once every 3 seconds, and any operations active by any events will be totally recorded in the log file which you have just input the address for.

### **How to use the program to create components and simulate events?**

#### **Create components:**

After you new a dynamic architecture, you can see a new component in the toolbar. You can click it and add it. The new added components are have the name "new" and ID "new", so after you create it, you need to change the name and ID, or you will get something wrong when you save the architecture. You can also add the connectors by click at the links choose the source and target components. The new added connection also has the name "new" and ID "new", you need also change it.

After you added all the components and connections, you can choose to save it to a XML file. The XML file can save all the in formations of the architecture.

#### **Simulate events:**

Then as you have the architecture of the system, you can now have some events to the system. The name of the systems is decided by the user. The only thing user need to care is how to act when an event happen.

The user click on the "simulator → Event generate", then he can choose to open a script file. If he does not have one, he can just open a blank file, and write every line in the format like "Event Name, Operations". After he chooses the certain file, modify it, save it, and then he has a script file which is fit to his architecture.

After the Event generate, the next step to simulate is to just simulate to see how the system act when the events happen. User chooses simulate, and choose the script file and the path to put the log file. Then the simulate starts. When something happen that can affect the architecture, then the simulator will show it dynamically.

### 4.3. Scenarios

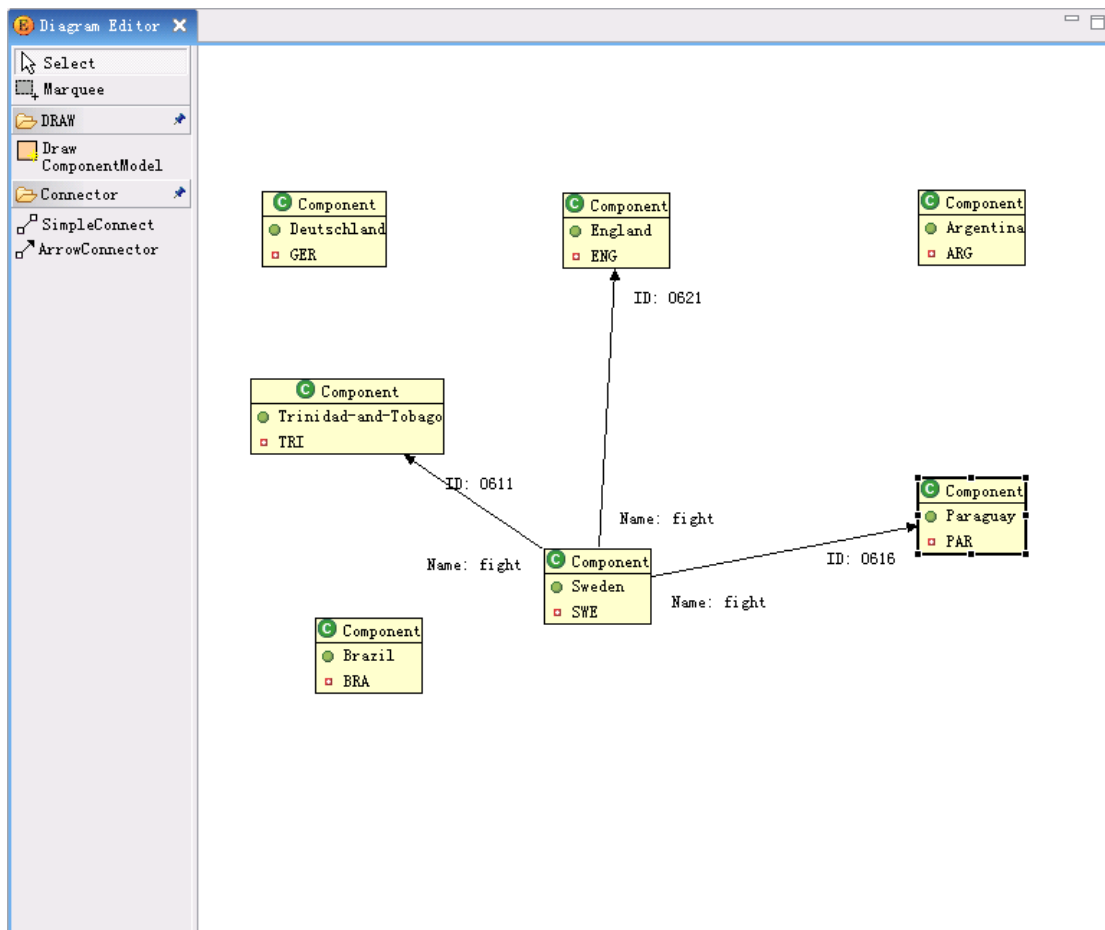
In both of the program and the script file, we design some scenarios to demonstrate as examples of where this kind of technology will probably be used to solve the problems in real world. The list following is just want demonstrated in the example model, anyone also can use this model to demonstrate some other scenarios.

- **Normal Dynamic Modification in Software architecture.** In the basic events named 'Mouse\_click', the action 'add-component' has been taken. This demonstration is related to the normal modification of a software architecture which is needs to update, to recover or any kind of operations needs to be taken. And it is better if the software can do them themselves. In this example it just shows the most basic operation that would be taken to modify any architecture.

- **Emergency Dynamic Reply in Software architecture.** There is an event call 'SystemCrash' in the example script file. This event is related to the emergency issue happened to the software and how they will response to it and recover or reduce lost from the accident. Obviously if a system can handle these problems themselves before any one realize the faults, it will more easily to fix the problem, more possibilities to recover the system. And no doubt there will be more customers who use and support this system. In real world this is still a big issue because of the difficulties to detect those emergencies before they cause any problem and the hardness of recover from all kinds of bad situations. Only some kind of accidents can be stop and recovered perfectly.

- **Automatic Update in Software Architecture.** This issue is quite common in software architecture which has been modeled as event 'Version\_Detection' in the example script file. It is a quite important point for the life cycle of any software architecture. To adapt the changing environment, any software architecture should be as flexible as it could and always update itself to suit the new situations. Since that everyone will simply think into dynamic update area and will be glade to see the software could update itself automatically. Fortunately, this area is usually not a trouble for most of the architecture, though there exists quite a lot of approaches to update the software automatically.

As you can see, our product is light and useful. It can be used in some many areas. These areas are so hot and they seem just stride their first step and still have a huge potential. Our product has three main strong points. The first has just been mentioned, and it will be more clearly here. That is, it has a latent big market, which is growing very fast and has a large number of potential customers. Second, our product is quite an essential model. It is too basic so that it is necessary for many more deeply techniques. So can be extended and used in so many areas, all the scenarios above for example, and could be more. The last but not least, our product has a quite friendly user interface. It is easy to learn to use and it is very convenient for you to use. After you read through the User Manual in Chapter 6, you can not be more agreed with us. Here let us show you a screenshot.



**Figure 4.2 Demo: Flights between Counties**

You may be confused on this picture when compared it with **Figure 4.1**. But if you look at this figure carefully you will see the differences. This picture shows the flights between countries: Sweden, England, Deutschland and so on. Each country could be regarded as a component and the flight could be regarded as a kind of connector. If you have read the manual, you will just need 10min to create a model like this one, or even a more complex one. So you see how convenient and easy to use our product. And the result is also vivid as well.

However, we have to accept the fact that our product is not so perfect. To the root, it is only and just a model also it is really extendable. Next chapter we will match the product to the original requirements and give some suggestions of where our model could be improved or extended in the future days based on our experience.

## **5. Conclusion and Future Work**

In general, the final program matches the original design very well. Corresponding to section 3.1, all the problems have been fulfilled and each iteration is successful. The simplified DASim has been really instanced. It is easy and convenient to use. Also it has a nice user interface. User can program some basic events and operations according to the standard description language we designed. The program can detect events or even generate some events based on the script file and then active the related operations and change the configuration of the original architecture. Though someplace in detail has been implemented because of the complexity issue. The GUI bonding to the model is quite efficient and the two parts cooperated smoothly.

As mentioned before, this research is a start of the Simulator Studio. The Simulator Studio could be developed in the future days is an IDE platform, which aims at develop dynamic adaptation system for other software. To do this, it could start from extending the existing java model. Do a research of all kinds of dynamic events, complete the event list. Enlarge the complexity of the design of simulator, extend it to the original design of **Figure 2.1**, that is, to contain Initiation, Evolution, and Control components. Here gives some possible aspects which could be improved or implemented in the nearly future days.

### **5.1. Specify Simulator**

At the moment, the program is lack of an important functionality: The simulator it self is not editable. Any architecture does not want a static simulator. What they really want is to generate a dynamic adaptor suitable for their product. In our program, the IDE is bounding with the simulator, and that is where the problem is. The next urgent job could be to separate the IDE tools with the simulator. Customers only use the IDE tools to generate a special simulator object for themselves. Before the generation, there should be any instance of the simulator or event generator/detector, and the simulator is the product from the IDE, where the IDE contains all basic events and corresponding operations implementation code inside. If that had been done, then users could specify their new events and operations by combine the basic events and operations existing already, and generate their own dynamic adaptor which is suitable for their architecture.

### **5.2. Extend Basic Events**

The basic events in our program are very limited and it should be extend a lot in the future days. Before this an investigation and study of system messages and operations should be done first. Then design and classify different system events into categories since some events detections are not allowed at the same time. And implement the invoking of those events in the default simulator definition inside the IDE. These events could be: Mouse events, Time events, Key events, Process and Thread events, System Warning and Error events and so on.

### **5.3. Extend Basic Operations**

Although we call it as extending basic operations, actually the official product in the future should not contain any basic operations. Instead, the IDE tools should provide several methods that the only job they will do is to invoke different kind of program in different situations. Here, these methods should focus on invoking, that is, to do the preparation stuff so that the simulator could invoke any kind of program with out any

problem. Because these operations are really depend on the architecture itself. Two architectures usually have kinds of abilities which are far from each other and have no relationship between them and those operations no doubts have no issue with system, the IDE or the simulator themselves. This could be a hard task for future. However, it could start from some kinds of architectures and generate the invoking methods, then move to next step.

#### **5.4. Network Dynamism Control**

Our Model is quite simple that there is nothing related to internet or any network solution has been considered and every operations or events are based on local host. However, most of the impactive and important architectures are all based on networks. The IDE tools have to support dynamical control of architectures with network. Otherwise the market of the tools would be quite small and has no benefits.

To support network architecture, some issues should be thought of. The first question comes to your mind maybe is the communication. During these years lots of communication technologies are based on network. Either to choose one or to support multiple technologies could be a problem in future days since the tools is aiming at develop product which based on architecture itself. However, if lucky enough, with the integrating of the development of Internet, there also exists a possibility that the development of Internet will solve this technique problem.

Second problem could be the new basic events. It also should be extended to obtain special events based on network technology, e.g. network crowded, no connection, no response... Also, today some official third parts offers some important public network services and the simulator should be able to recognize those message from them any provide some basic events of those third parts service such as SOAP message.

#### **5.5. Other Specifications**

As a integrated software product, some basic functionalities should be developed for the IDE, those issues includes programming language, nation language, security issue, usability, interactive with GUI and so on. In these issues, supporting multiple programming languages could be a difficult issue. However, this issue is still hard to say at the moment. That should depend on how close the simulator needs to bind to the architecture. When it has been developed to the level of multiple program language consideration, this question could be further discussed.

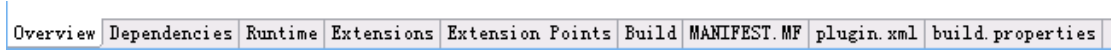


## 6. Appendix

### 6.1. User Manual

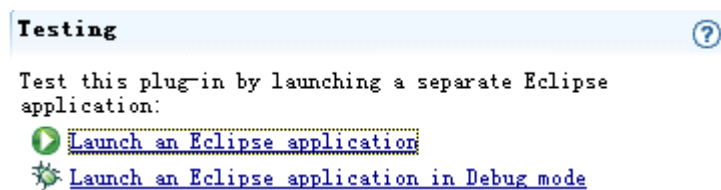
#### Run the Project

First, open the “plugin.xml” in Eclipse, choose the “overview” from the bottom of the page.



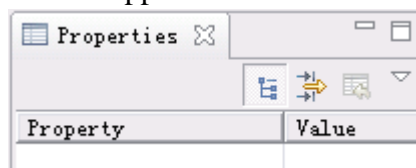
**Figure 6.1 Step1: overview**

Then choose the “Launch an Eclipse application” from the “Testing”.



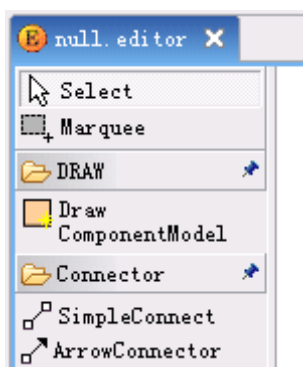
**Figure 6.2 Step2: Launch an Eclipse application**

Click it, and you will run the program. When it has been in running, there will be only a properties window in the application.



**Figure 6.3 Properties**

#### New Architecture



**Figure 6.4  
New Architecture**

And then you can choose “File → new” to open a new architecture. Then you will get the editor window.

## New a Component and Connection

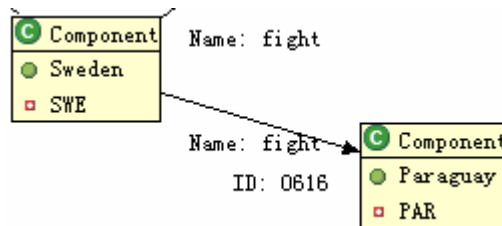


**Figure 6.5**  
**New Component**

You can create a component by click at the “Draw ComponentModel”. You choose the place where you want to put the component; you will get a component like this:

You can change the name and ID of the component in the properties window. After your modification, press Enter will commit the changes.

After you have created at least two components, you can create a connection now to connect 2 components. Click at the “connector”, and choose the ‘from’ component and ‘end’ component, you will get the connector.



**Figure 6.6** New a Connector

Now we have talked about add component and add connectors, in the “**Editing the architecture**” part we will talk about how to delete it.

### File Operation

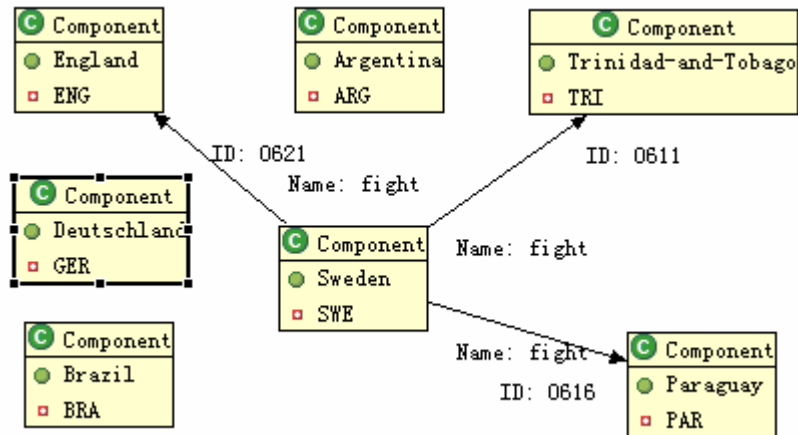
You can save the architectures you created to an XML file, which contains all the information your architectures have, and you can rebuild architecture by load the XML file. To save the architecture, click “Save as”, choose the file path, and then click ok. To open an existed architecture, click “Open”, choose the file path, and then you can open an architecture you created before. But the XML file you choose to open must have the right format, created by the project or by command lines the project provided.

Here we give an example:

```
<Component>
  <name>Brazil</name>
  <ID>BRA</ID>
</Component>
- <Connector>
  <name>fight</name>
  <ID>0616</ID>
  <FromComponent>SWE</FromComponent>
  <EndComponent>PAR</EndComponent>
</Connector>
```

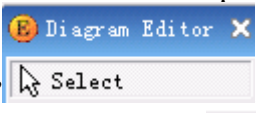

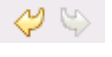
Choose the close to close the current architecture that you are now edited. But take attention that, if your modification on the architecture is not saved before you close it, there will be no warning, and you will lose the data. So make sure save it before close it.

### Editing the architecture

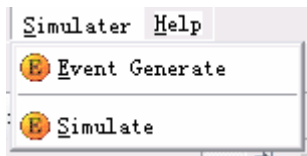


**Figure 6.7 Demonstration of Existing Architecture**

Here is example architecture. If you want to delete the component or connector, first

click on the “Select” in the “Diagram Editor” , and then click on the connectors or components you want to delete, then click the  in the left top, below the menu bar. If you want to redo or do the change, you can click the  in the left top, below the menu bar.

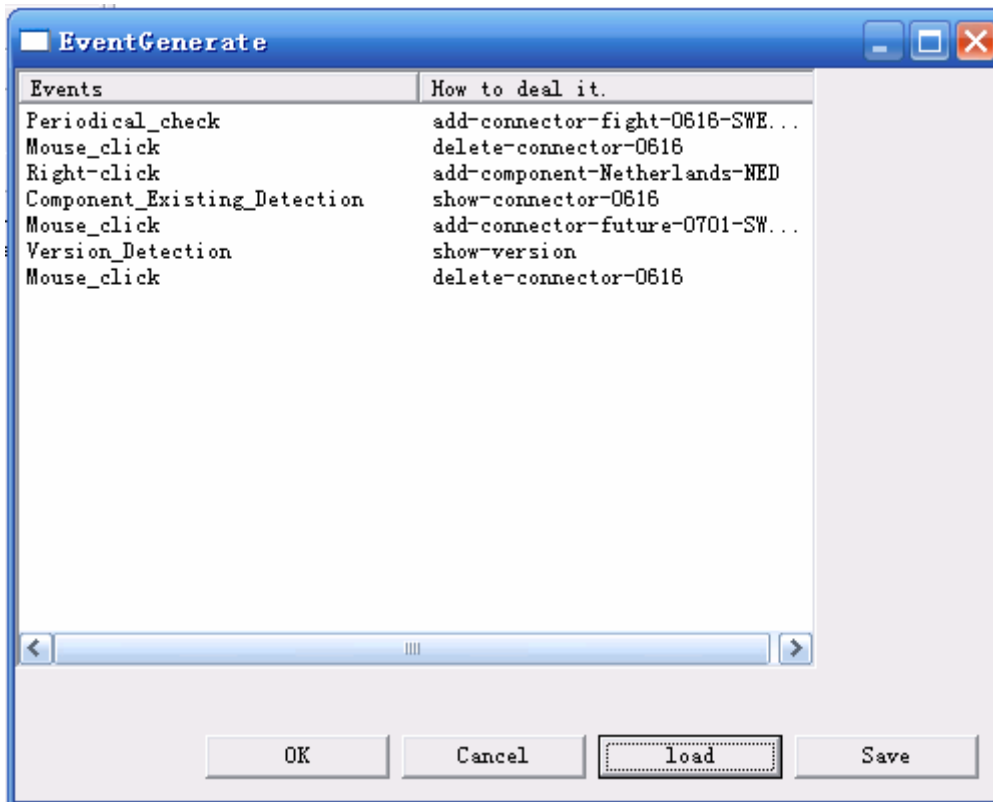
**Simulator**



**Figure 6.8 Step 1: Menu Simulator**

You can choose from the menu “Simulator” to simulate the changes to the architecture.

And you click on the “Event Generate” to generate some changes to the environment, such as “system crashed” or “system busy” or “overload” or some other things. After you click on it, the dialog will jump out.



**Figure 6.9 Step2: Load Events**

You click on the load to load a script file which is pre-written, such as what showed. And by click on the labels, you can edit each “Events” and “How to deal it” to achieve the goals of simulate of architecture changes. Click “OK” after you changed the script file. And then clicks save to save the changes you make to the script file.

After you do all the things, you can click the “Simulate” in the menu, and then you can see what will happen to the architecture if some events happened, by view the architecture dynamically.

## 6.2. Reference

- [1] -- <<A Classification of Dynamic Software Architectures>> : Jesper Andersson
- [2] -- Standard Widget Toolkit: [http://en.wikipedia.org/wiki/SWT](http://en.wikipedia.org/wiki SWT)
- [3] -- Graphical Editing Framework: <http://www.eclipse.org/gef/>
- [4] -- 2D drawing framework: <http://www.eclipse.org/gef/>
- [5] -- <http://www.eclipse.org/>



**Matematiska och systemtekniska institutionen**  
SE-351 95 Växjö

Tel. +46 (0)470 70 80 00, fax +46 (0)470 840 04  
<http://www.vxu.se/msi/>