

Generation of platform independent GUIs by means of state charts

Project Work

Submitted by:
Luis Bustamante
salinas.bustamante@tuhh.de
Information and Media Technologies
Matriculation Number:
30152

Supervised by:

Prof. Dr. Ralf Möller
STS - TUHH

M.Sc. Miguel GARCIA
STS - TUHH

Hamburg, Germany
2006-06-09

I declare that:
this work has been prepared by myself,
all literally or content-related quotations from other sources are clearly pointed out,
and no other sources or aids than the ones that are declared are used.

Hamburg, 09.06.2006

Abstract

Software development process most of the time is focused on the business logic disregarding a methodology for developing the Graphical User Interface, formal ways to design them has taken importance during the last years and a technique that has gained popularity is using statecharts for modeling the behavior.

The generation of GUI supported by IDE it is not something new, there are many tools in the market with reach visual editors, but a weak on this tools is the GUI generation in a platform independent way.

Both statements will be addressed in this work by taking an independent platform GUI and an executable statechart generator.

Index

1	Introduction.....	6
1.1	Problem Definition	6
1.2	Objective of the Project Work	6
1.3	Some words about software design.....	7
2	Unimod Description	8
2.1	Overview	8
2.2	Switch Technology.....	8
2.3	Unimod Model	8
2.4	Working with different state machines.....	9
2.5	Working with Sub states.....	10
2.6	Using Unimod context	11
2.7	Representation of an event	13
2.8	Generating a Stand Alone GUI application using Unimod.....	14
2.8.1	Working with Unimod on the web	17
2.8.2	Unimod Remarks	18
3	Implementing applications with Unimod and Java	19
3.1	Navigation Model for a Wizard	19
3.2	Statechart and MVC	21
3.3	Unimod and Model Driven Development	22
3.4	Creating the application	22
3.5	Look and feel	23
3.6	State chart modeling.....	23
3.6.1	Statechart Diagram.....	24
3.6.2	Implementing the application with Unimod	24
3.7	Output.....	26
3.8	Observations after implementation.....	26
4	Generating Platform Independent GUI	27
4.1	Overview	27
4.2	AUIML	27
4.2.1	Overview	27
4.2.2	How does it work?	27
4.2.3	AUIML Remarks.....	29
4.3	MVC Scenario in Unimod and AUIML	30
4.3.1	Ideal behavior	30
4.3.2	Intention of modeling the controller with a statechart and AUIML	30
4.3.3	Mapping between the statechart and AUIML	30
4.4	Binding the GUI editor and the GUI logic (View and Controller)	31
4.4.1	Overview	31
4.4.2	Joining both worlds.....	31
4.4.3	Coupling Details.....	32
4.4.4	Problems.....	33
4.4.5	Implementing a Wizard with AUIML - Unimod	33
4.4.6	Unimod Model	38

4.4.7	Missing formalism	39
4.5	Problems during framework interaction.	40
4.5.1	Obervations after implementation	40
5	Conclusions	42
5.1	Considerations.....	42
5.2	Related Work	43
5.3	Further Work.....	44
6	References	45
7	Appendix	46
7.1	File generated by Unimod:	46
7.2	File generated by AUIML	48

1 Introduction

1.1 Problem Definition

Currently on the market there are development environments that help to build complex interfaces in relatively short time frames. The user needs to get acquainted with the tool and in matter of short time a fully functional GUI is ready to use without the need to invest time coding.

These tools miss one or both of the following features:

- platform independence of the modeled GUI
- the ability to simulate the GUI (and thus refine it) before deployment

Considering another point of view, it would be beneficial to empower the client side(GUI) with some intelligence, it is effective because sensitive user data would be protected, the context used would be validated on the client side without requiring a round trip.

1.2 Objective of the Project Work

The intention of this Project work is the modeling of GUI interactions by means of Statecharts. In order to achieve this, executable models are needed, thus the open source tool Unimod shall be used.

A common flaw when developing a GUI stem from the disregard the developer has when building a GUI.

A GUI consists of 2 parts:

- The elements that conform the look and feeling.
- The behavior of such elements.

The GUI can be modeled using statecharts to represent the behavior and interactions between the elements, the statechart can be serialized to a platform independent format as XML and later on deserialized to a platform specific code to generate the GUI.

These platform independent representations could be used as base for later generation technologies like Struts, Java Server Faces and so on.

Coupling Unimod with a GUI yields the following advantages:

- Neutral representation.
- Separation of concerns.

1.3 Some words about software design

The software design consists of 2 elements

- User interface: Interacts with the user and enables access to the model.
- Model: Represents the business logic and does not know how the user interacts with the user interface.

Model driven development focuses on creating a model which is capable to run independently of the platform, it abstract the business problem and delivers a model that it can be transformed to code.

Advantages of developing based on the model

- Platform independence
- Validation of models.
- Formal description
- Standardization.

The software developing process is strongly supported by methodologies and tools to ease the task, on the model side OMG propose standards to enable interoperability between technologies, but on the user interface side the support is still underway.

In the last years statechart modeling has gained importance among the community to model user interactions, several extensions have been proposed trying to enrich the formality and semantics of the original notation created by Harel; stochastic Statecharts, Interaction object graphs, Queuing statecharts, objectcharts and some others.

2 Unimod Description

2.1 Overview

The tool creates a java based statecharts framework which is attached to the eclipse development environment as a plug in and allows the modeler to design statechart diagrams and execute them.

The tool relies on SWITCH technology for implementing the state machine framework.

2.2 Switch Technology

It is based on projecting of systems and programs as a set of connected finite automata. Each automate is described by a connections scheme and a state transition graph. There is a formal and isomorphic way to generate a skeleton code for systems functionality implementation on the base of these two descriptions. This method doesn't depend on the platform, operating system or programming language.

In the case of UniMod, the state chart is interpreted, it represents the state chart as a xml file, which allows to modify the code and load it during runtime; if the code were generated the program would need to be stopped and load a .class.

2.3 Unimod Model

The Unimod model structure comprises 3 parts:

- Definition of the state machine(s).
- Definition of the object(s) that generates the events.
- Definition of the object(s) affected by the state machine. (Controlled Object).

An event generator is the entity that fires the execution within the statemachine, therefore they must be linked on the main model.

The state machine defines the transitions and constrains between the states. The transitions are fired whenever the entity called event provider issues an event that meets the constraint. To perform an action within the statechart the entity called Controlled object is invoked to generate the expected behavior, it might execute a task given a parameter, and might return a value.

2.4 Working with different state machines

Unimod offers the possibility to work with different state machines interacting among them, each one of them performing independent tasks within the same model.

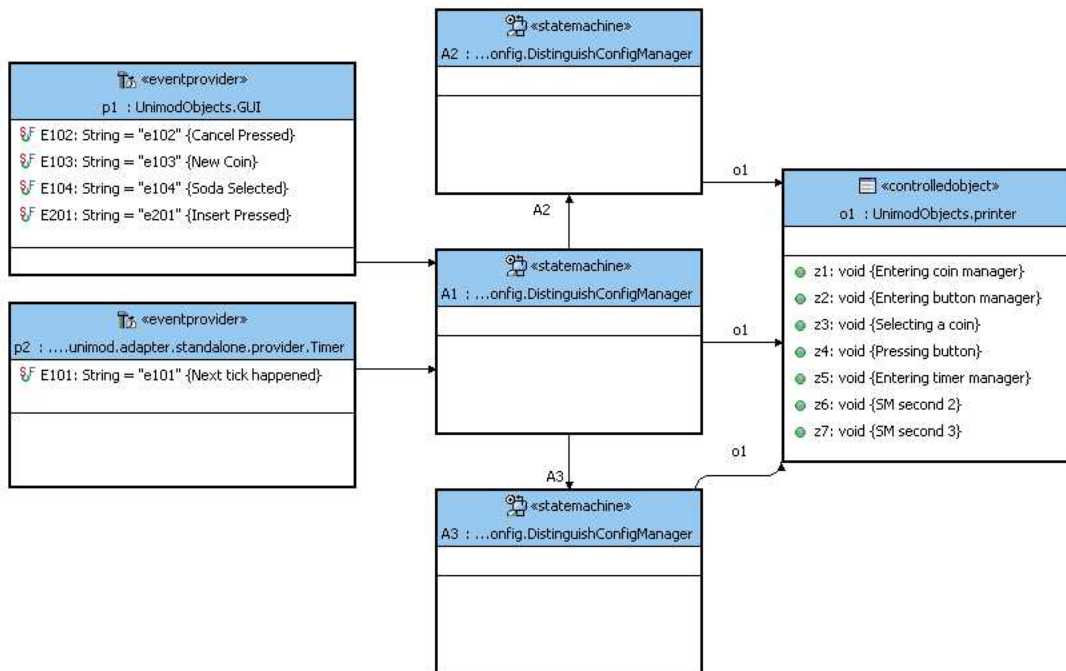
The model must be structured with one main state machine and one or more underlying state machines. The main state machine is connected to the event provider and is on charge of event delegation, the dependent state machines are connected to the main one.

An action execution is represented on the diagram within a controlled object, contrary to the event providers, the underlying state machine must be bound to a controlled object.

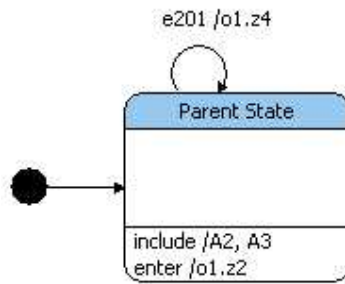
On the bottom part of each state there are two options, include and enter.

Include allows the modeler to create the link to several state machines, they will inherit the events from the state who is invoking them.

Enter executes an action when the process flows enter the state.



“Parent State” invokes the state machines A2 and A3. After completion it will execute the action registered under enter.



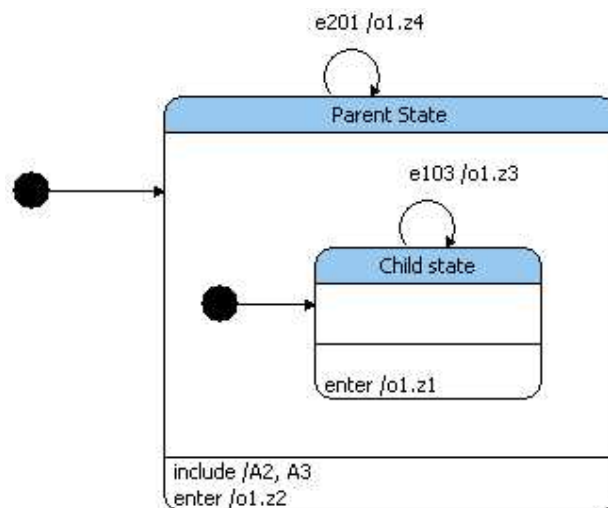
In this case the state “Parent State” is invoking the state machines A2 and A3 whenever the state is entered. After executing the state machines defined it will call the action registered under enter.

2.5 Working with Sub states

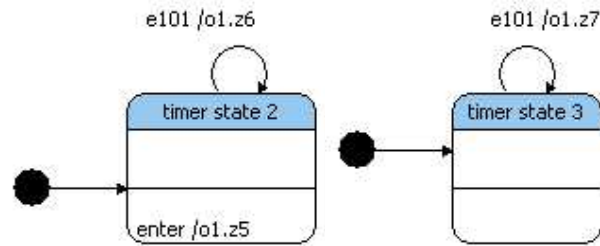
A state might have sub states; this feature enables different levels of abstraction to meet different levels of detail.

Especially helpful when trying to have an overview of the model, it is not necessary to understand the model in all the abstraction layers, give more flexibility and the possibility to encapsulate related process flow in the same state.

The sub states inherit events received by the parent, an event will be transferred to the states within the parent's scope no matter the depth. At the end of the execution the process flow is turn over to the parent.



The state machines that complete the model with 3 state machines and sub states looks as follows:



And the output looks as follows:

```

Setting engine [com.evelopers.unimod.runtime.ModelEngine@fcfa52], called from EG
20:25:05,247 INFO [Run] Start event [e101] processing. In state [/A1:Top]
20:25:05,257 INFO [Run] Transition to go found [s1#Parent State##true]
20:25:05,257 INFO [Run] Start on-enter action [o1.z2] execution
---- Thicking on State Machine 2
20:25:05,378 INFO [Run] Transition to go found [s1#timer state 3##true]
20:25:05,378 DEBUG [Run] Try transition [timer state 3#timer state 3#e101#true]
20:25:05,378 INFO [Run] Transition to go found [timer state 3#timer state 3#e101#true]
20:25:05,388 INFO [Run] Start output action [o1.z7] execution
---- Thicking on State Machine 3
20:25:05,388 INFO [Run] Finish output action [o1.z7] execution
20:25:05,388 INFO [Run] Finish event [e101] processing. In state [/A1:Parent State/A3:timer state 3]
20:25:05,388 INFO [Run] Finish event [e101] processing. In state [/A1:Child state]
20:25:06,239 INFO [Run] Start event [e101] processing. In state [/A1:Child state]
20:25:06,239 INFO [Run] Start event [e101] processing. In state [/A1:Parent State/A2:timer state 2]
20:25:06,239 DEBUG [Run] Try transition [timer state 2#timer state 2#e101#true]
20:25:06,239 INFO [Run] Transition to go found [timer state 2#timer state 2#e101#true]
20:25:06,239 INFO [Run] Start output action [o1.z6] execution
---- Thicking on State Machine 2
20:25:06,239 INFO [Run] Finish output action [o1.z6] execution
20:25:06,249 INFO [Run] Start on-enter action [o1.z5] execution
---- Entering time manager
20:25:06,249 INFO [Run] Finish on-enter action [o1.z5] execution
20:25:06,249 INFO [Run] Finish event [e101] processing. In state [/A1:Parent State/A2:timer state 2]
20:25:06,249 INFO [Run] Start event [e101] processing. In state [/A1:Parent State/A3:timer state 3]
20:25:06,249 DEBUG [Run] Try transition [timer state 3#timer state 3#e101#true]
20:25:06,249 INFO [Run] Transition to go found [timer state 3#timer state 3#e101#true]
20:25:06,249 INFO [Run] Start output action [o1.z7] execution
---- Thicking on State Machine 3
20:25:18,557 INFO [Run] Start output action [o1.z3] execution
---- Selecting a coin
20:25:21,461 INFO [Run] Start output action [o1.z4] execution
---- Pressing button

```

2.6 Using Unimod context

In order to communicate among the controlled objects a third entity is needed, a mechanism that is aware of the values being handled throughout the statechart.

Unimod implements a context that stores the common values needed by each one of the controlled objects. For the purpose of activating the context there must be a class instantiating the Unimod application context during the initialization phase, the latter will be in charge of retrieving and storing the parameters generated by the events providers.

Example:

The selected value of a combo box

To make use of information that might be issued by the component, it is needed a way to keep the information throughout the process flow.

The combo knows about the information contained in it, it should use the application context to read and write data; the latter is accessible by any controlled object.

When an event is created, parameters can be attached to the source and they will be stored in the context.

```
Event e170 = new Event(GUI.E170, new Parameter(Country, new String(strCountry)));
```

Let's assume the code line above was generated by the following GUI:

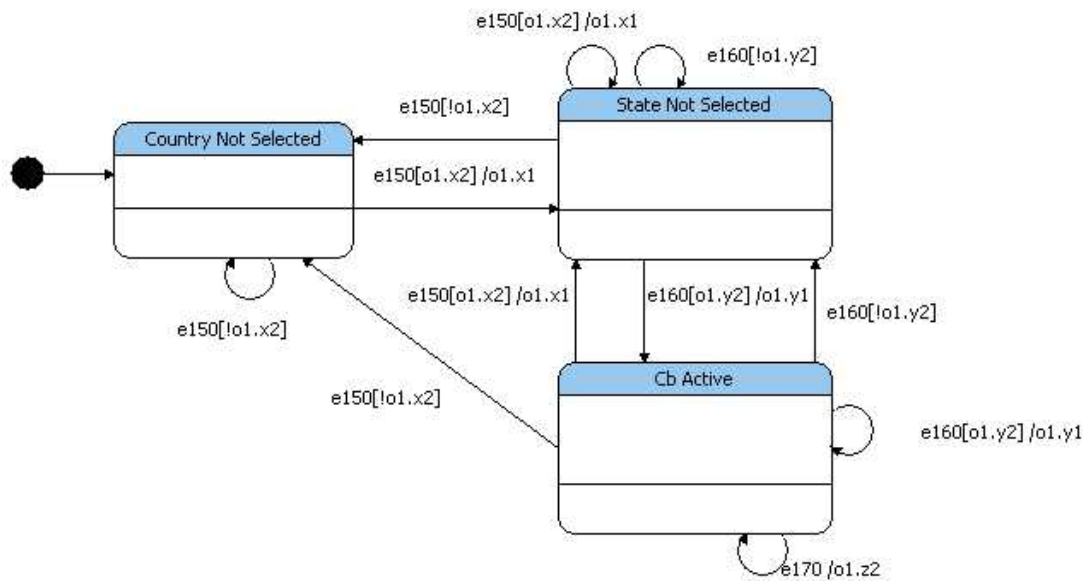


Whenever the country combo box selected item changes, the state combo box updates the information.



The changes on the GUI are supported by the state chart below; whenever one of the combos issues an event, the statechart invokes a controlled object (o1) that reads the parameter value from the context and the state chart transitions to the right state.

Unimod model.



Action	Description	Event	Description
o1.x1	Load states combo.	e150	Country combo value changed.
o1.x2	Validates country.	e160	State combo value changed.
o1.y1	Enable a check box.	e170	Checkbox clicked.
o1.y2	Validates state.		
o1.z2	Display a text.		

2.7 Representation of an event

An event in Unimod is represented as a String as follows:

```

/**
 * @unimod.event.descr Ping
 */
public static final String E101 = "e101";

```

Unimod handles the events through an event manager, the latter is contained in the Unimod engine,

An event is invoked as follows:

```

String event = eventProvider.E102
engine.getEventManager().handle(new Event(event), StateMachineContextImpl.create());

```

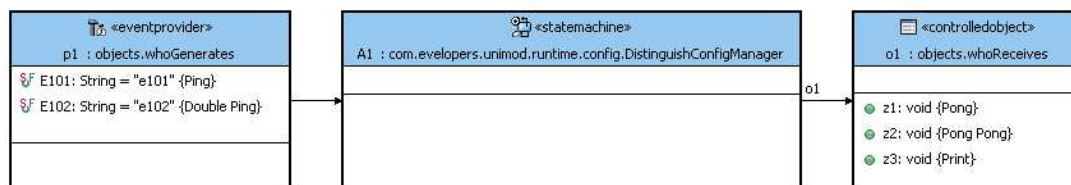
A parameter is composed by a String and by an object value, invoking an event with a parameter looks as follows:

```
String event = eventProvider.E102;
Parameter parameter = new Parameter("newParameter", value);

engine.getEventManager().handle(new Event(event, new Parameter[] {parameter}),
StateMachineContextImpl.create());
```

2.8 Generating a Stand Alone GUI application using Unimod

The connectivity diagram needs to be created as usual; the event provider and the controlled object must have the desired events and actions respectively.



The GUI is in charge of handling user's interactions by listening to the events generated on the components available. A java Listener must be implemented to fulfill the task, i.e. action, mouse, list selection, etc.

The Unimod engine will act as the intermediate between the GUI and the model. This should be set up during the initialization phase on the events providers, i.e.:

```
public void init(ModelEngine engine) throws CommonException
{
    GUI.getGUI().init(engine);
    try{
        // invoke in Swing event thread
        SwingUtilities.invokeLater (
            new Runnable(){
                public void run(){
                    GUI.getGUI().setVisible(true);
                }
            });
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

During the event provider disposal, the GUI must be as well disposed.

After enabling Unimod engine the event manager can be called to process events generated. The events can be accessed from anywhere in the application because they are public and static members within the event provider class.

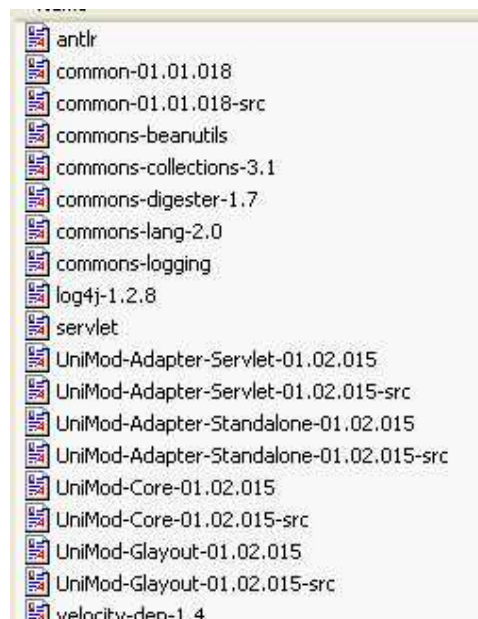
The idea is to create a .jar file that can be called from the command line and which contains a manifest file pointing to the main class that uses an external GUI as front end and implements the logic behind by using Unimod.

The tool ant is used to simplify the process.

A directory structure is needed:

Name	Size	Type
lib		File Folder
resources		File Folder
src		File Folder
build	1 KB	PROPERTIES File
build	4 KB	XML Document

Lib contains the .jar related files that contains the Unimod classes.



The folder resources contain the files generated by UniMod:

- Model file with extension **.Unimod**
- State Machine file with extension **.xml**
- Event processor with extension **.java**

The folder src contains the code of the events generators, the controlled objects and the GUI itself.

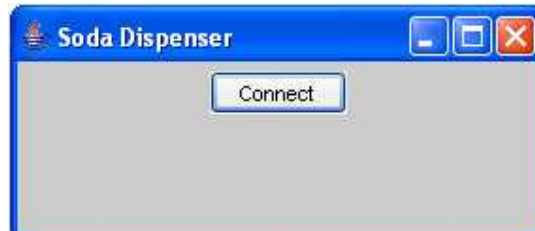
The important part of the build.xml is resumed here

```
<jar destfile="${assemble}/independentApp.jar"
    basedir="${build.classes}"
    includes="**/*.class, **/*.properties"
    excludes="**/client/**/*.class">
    <manifest>
        <attribute name="Class-Path" value="${build.jar.adapter}"/>
        <attribute name="Main-Class" value="com.evelopers.unimod.adapter.standalone.Run" />
    </manifest>
</jar>
```

When executing the standalone application, the command must make reference to the xml file where the state machine is described.

It can be executed as follows:

```
java -jar independentApp.jar statemachine.xml
```



Interaction with the application gives as result:

```
C:\_Standalone>java -jar server.jar a1.xml
Setting engine [com.evelopers.unimod.runtime.ModelEngine@1b60
G

C:\_Standalone>java -jar server.jar a1.xml
Setting engine [com.evelopers.unimod.runtime.ModelEngine@1b60
G
[info] Start event [e102] processing. In state [/A1:Top]
[info] Transition to go found [s1#s2##true]
[info] Transition to go found [s5#s4##true]
[debug] Try transition [s4#s4#e102#true]
[info] Transition to go found [s4#s4#e102#true]
[info] Start output action [o1.z3] execution
Printing cause an event was generated
[info] Finish output action [o1.z3] execution
[info] Finish event [e102] processing. In state [/A1:s4]
```


2.8.1 Working with Unimod on the web

Unimod is capable to run on the web as well, this is possibly through the use of a .jar provided in the distribution called **UniMod-Adapter-Servlet**. This .jar extends the java class `HttpServlet` and interprets a model created by Unimod.

The parameters used for setting up the environment on the web application are read from a **web.xml** descriptor. Example:

```
<web-app>
  <display-name>Messenger</display-name>
  <servlet>
    <servlet-name>HttpServletAdapter</servlet-name>
    <servlet-class>com.evelopers.unimod.adapter.servlet.HttpServletAdapter</servlet-class>
    <init-param>
      <param-name>MODEL_URL</param-name>
      <param-value>A1.xml</param-value>
    </init-param>
    <init-param>
      <param-name>LOGGER_NAME</param-name>
      <param-value>CONSOLE</param-value>
    </init-param>
    <servlet-mapping>
      <servlet-name>HttpServletAdapter</servlet-name>
      <url-pattern>/controller</url-pattern>
    </servlet-mapping>
  </web-app>
```

Unimod overrides the methods `init`, `service` and `destroy`; the state machine engine is instantiated during the servlet's `init` method, the listeners and handlers are bounded as well during the same method call.

As in the case of the stand alone application, there is a context which stores the information handled throughout the process flow.

There is a slight difference between the stand alone model and the web model, the entity that will create the events is not bounded to the Unimod on the connectivity diagram. A modified servlet is the responsible for generating the events; the events are defined within the service method.

An html form must contain the name of the event that will be processed by the Unimod state chart, and it is passed to the servlet to be created. I.e.:

Type your name:

```
<form action="/controller" method="POST">
  <input name="NAME" type="TEXT"/>
  <input name="evt" value="e2" type="HIDDEN"/>
  <input name="button" value="Submit" type="SUBMIT"/>
```

The data needs to be stored in the Unimod context and retrieved using a .jsp. I.e.:

In Unimod:

```
context.getEventContext().setParameter(Parameters.EventContext.USERS, users);
```

In .jsp

```
<%  
    List users = (List)request.getAttribute(Parameters.EventContext.USERS);  
    for (int i = 0; i < users.size(); i++) {  
        out.println("First Name: " + (String)users.get(i) + "<br/>");  
    }  
%>
```

The class `StateMachineContextImpl` provided by Unimod maps the context used within the model to the session and request objects handled by the servlet.

An instance of a controlled object is needed to display the response results; this object must extend the class `BasePresentationManager` provided by Unimod to retrieve the request / response objects and forward them as a normal servlet would do. I.e.

```
include(context, "/LoggedIn.jsp");
```

A servlet complaint container is needed; the folder structure is the same as any other servlets application.

The web.xml descriptor must specify the servlet class **com.evelopers.unimod.adapter.servlet.HttpServletAdapter** and the parameter **MODEL_URL** which contains the .xml that describes the state chart.

The .jars used for creating the Unimod model in eclipse need to be placed in the lib folder.

2.8.2 Unimod Remarks

The visual editor visible area is consumed really fast because descriptive name restrictions need a lot of space, on the other hand two letter descriptions as Unimod (i.e. x1, z2) suggests, on a later stage, is hard to read because the modeler does not know the meaning of all of them.

Unimod's visual editor has a restriction when creating a function that returns a value, the only data types allowed are integer and Boolean. The data types can be changed manually, but it is a disadvantage given that we want to avoid manual coding.

A disadvantage about Unimod's model is the lack of concurrent execution support, when an event is generated, the underlying state machines execution is sequential and not concurrent, and it is not supported by Unimod.

It does not support transient states, do not offer history and do not specify how to prioritize transitions.

3 Implementing applications with Unimod and Java

3.1 *Navigation Model for a Wizard*

There are some cases where the completion of a task is composed of several steps; the final objective is reached only if the whole series of interactions are performed. A Wizard is a typical example containing such behavior.

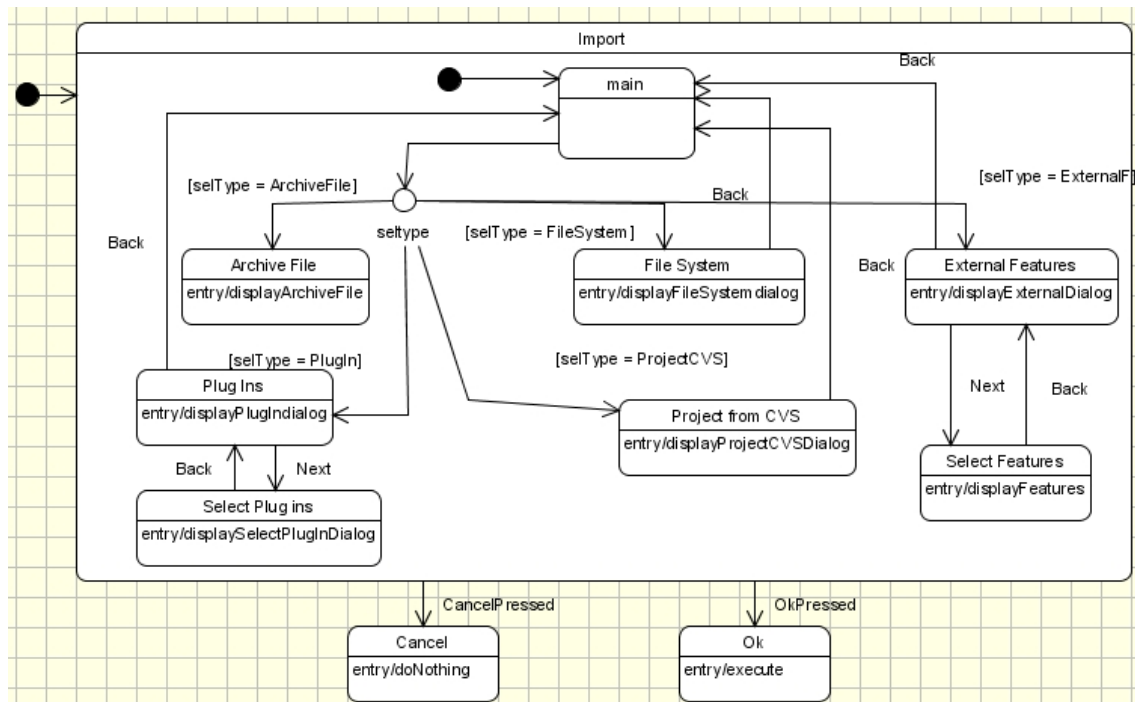
It is mandatory to have a consistent and adequate way of modeling the steps involved and determine the resultant state after a gesture from the user.

A GUI consists of components that might be bound to each other and that might be able to issue a certain behavior, as for example, the action of showing a pop up menu, to enable a component, to load a component with data and so on.

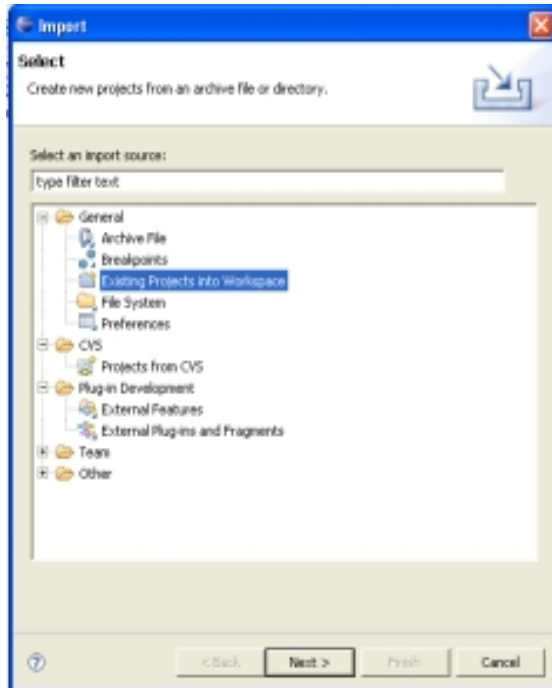
Eclipse implements this functionality throughout the framework; a specific implementation will be taken and modeled as an example of a navigation model. This is the wizard for importing a new project to the workspace.

Characteristics:

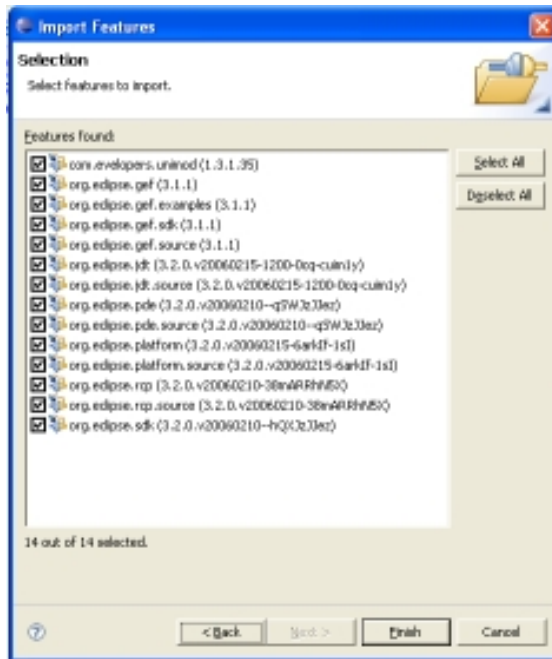
- Offers the possibility to add an existing project.
- There are different types of projects that can be imported.
- Some of them have common characteristics and they require only one screen.



The GUI looks as follows:

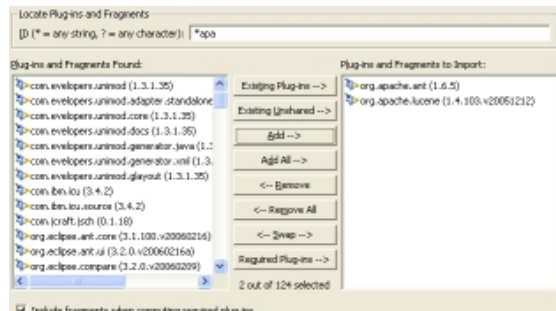


- The wizard request to select the type of project to be imported.
- This is the main state represented on the statechart.
- After making the selection and pressing the button „next“ the statechart transitions accordingly.



- The screen to be displayed is responsibility of the state where the statechart is transiting to.
- If the button cancel is pressed at any time, the statechart exits the flow.
- If the user presses the button „back“ the statechart transitions to the previous state.

- The objective of the statechart shown is to model the navigation, but it is important to notice that each one of the GUI have an independent statechart.
- The way the buttons, texts and other controls interact, is modeled by separate.



3.2 Statechart and MVC

To build a robust presentation layer it is recommended to model the user interface behavior by using statecharts and then implement the statechart design using the mediator pattern and bound them by implementing the MVC pattern[10], this kind of coupling is a powerful architecture for GUIs and it is suitable to event driven systems.

By implementing the MVC it is necessary to comply with the requirement that the controller must lead the process flow of the system. The controller listens to incoming events, processes them and decides about the following action to be executed.

The model encapsulates the domain logic and keeps its independence from the view and controller, but at anytime when the model is modified it must notify the view in order to update the presentation.

Advantages of having a MVC

- The Model, View and Controller pattern allow building the dependency between the components during runtime. This brings flexibility because the classes do not need to be recompiled.
- Decouples the processing of data and its visual representation and the View (GUI) and the controller (state chart) can be developed separately.

Disadvantages of having a MVC

- The design of a complex system requires more effort as with another approach.

3.3 Unimod and Model Driven Development

Unimod encourages model driven development because provides means to build the application logic from a visual representation to executable code insulating it from technology evolution.

The end product can be executed and will provide feedback about the accuracy of the design, this is helpful to discover flaws at early stage of the development process and correct them before noticing them at later stages of the software development where modifications might imply major changes.

3.4 Creating the application

To exemplify how a statechart created with Unimod behaves under the role of controller and interacts with the view and the model, a java based application simulating a Soda dispenser is shown.

The Soda dispenser is implemented using a swing generated GUI as view and the statechart as controller.

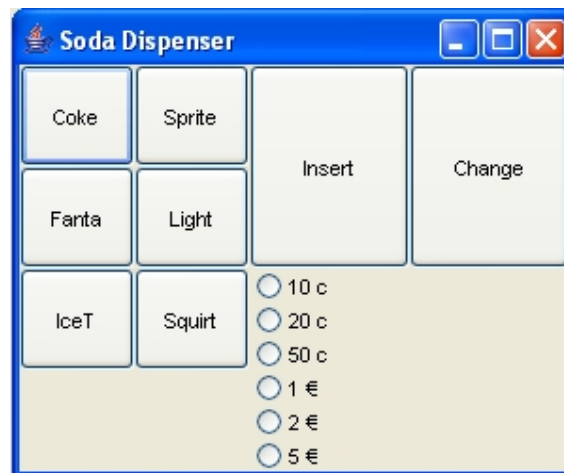
Considerations

- Different coin denomination ranging from 10 cents to 5 euros is accepted.
- The user can cancel the operation at any time.
- There are 6 different beverages, each one with a different price.
- The user will get a notification if the amount of money for the desired beverage is not enough.

- The user does not get the change after selecting a beverage, in case the user wants to select a second one, the GUI waits until the button change is pressed.
- There is a cooling unit that keeps the beverages under an ideal temperature, every n seconds the thermometer is checked and adjust the temperature accordingly if needed.

3.5 Look and feel

- To simulate different coin denominations, some radio buttons specifying the coin values will be used.
- After selecting the desired coin denomination, the button insert is pressed to simulate coin insertion; the user sees the output on the screen.
- The cooling unit output is just visible on the console because it is not relevant for the GUI implementation, but is useful for illustrating statecharts behavior.
- Every component on the GUI notifies the controller via a Unimod event.

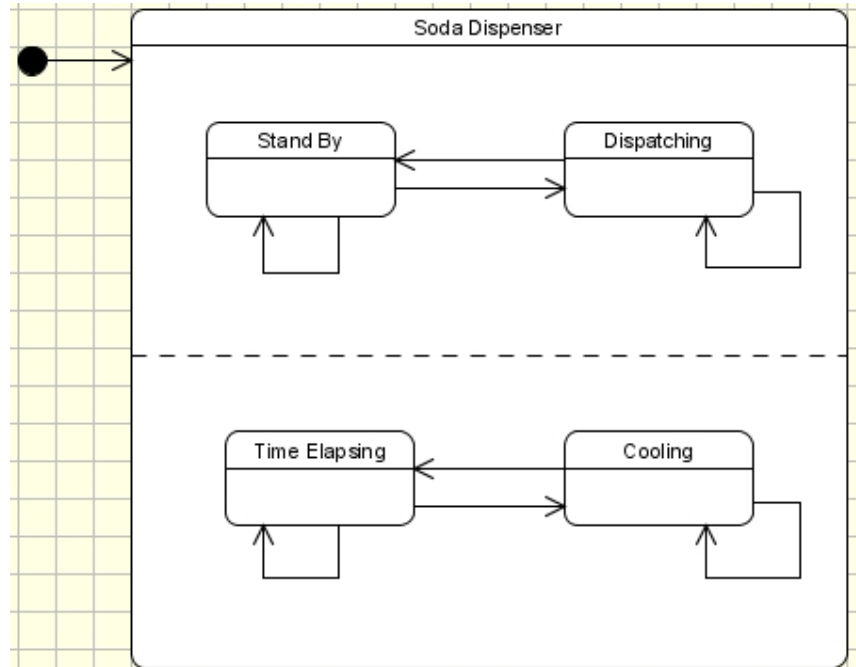


3.6 State chart modeling

The simulation of the GUI needs concurrent states for managing interactions on the GUI and the cooling unit working on the background. The statechart representation in Unimod cannot be implemented as Harel's original specifications; the reason is that it does not support direct concurrent statemachines, thereby the diagrams need to be split into individual diagrams and later called by a main state machine as explained in chapter 2.

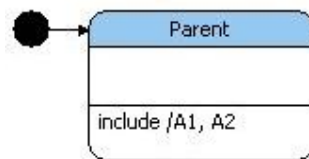
3.6.1 Statechart Diagram

This is the statechart supporting the View component, given that it is the simulation of a machine that is running non stop, there is no end state.

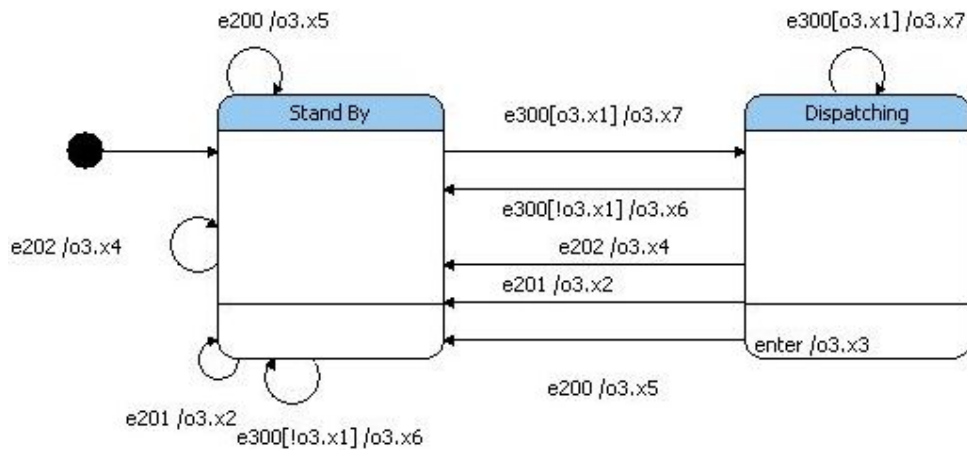


3.6.2 Implementing the application with Unimod

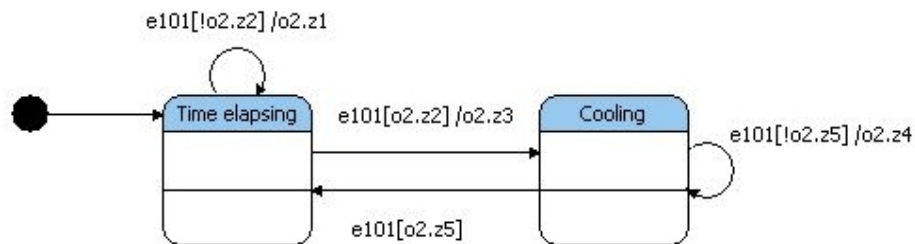
Unimod statechart diagram is divided in three parts:



The parent includes state machines A1 and A2, they are executed sequentially and the events generated are forwarded to both state machines.



A1 Diagram, the statechart supporting the GUI has only two states stand by and dispatching.



State Machine A2 diagram, the cooling unit is either cooling the dispenser, or waiting.

List of actions and events involved

Action	Description	Event	Description
o3.x1	Enough money	e200	Coin denomination selected
o3.x2	Sum	e201	Coin inserted.
o3.x3	Dispense	e202	Change button pressed.
o3.x4	Give change.	e300	Soda selected.
o3.x5	Coin created		
o3.x6	Message		
o3.x7	Soda Name		
o2.z1	Increase timer.		
o2.z2	Read time.		
o2.z3	Create random temperature		
o2.z4	Decrease Temperature		
o2.z5	Read temperature		

3.7 Output

Some of the most representative output generated looks as follows:

```
23:27:52,718 INFO [Run] Start event [e101] processing. In state [/A3:Top]
23:27:52,728 INFO [Run] Transition to go found [s1#Parent##true]
23:27:52,758 INFO [Run] Start event [e101] processing. In state [/A3:Parent/A1:Top]
23:27:52,758 INFO [Run] Transition to go found [s1#Stand By##true]
23:56:32,431 INFO [Run] Start output action [o3.x5] execution
---- Coin Created: 20
23:56:32,431 INFO [Run] Finish output action [o3.x5] execution
---- Adding coin: 50
---- So far 50
23:56:35,946 INFO [Run] Finish output action [o3.x2] execution
23:56:35,946 INFO [Run] Finish event [e201] processing. In state [/A3:Parent/A1:Stand By]
23:56:35,966 INFO [Run] Start event [e201] processing. In state [/A3:Parent/A2:Time elapsing]
-- Creating parameter Soda... Soda
---- Creating parameter price... 30
23:56:38,800 INFO [Run] Start event [e300] processing. In state [/A3:Parent]
23:56:38,800 INFO [Run] Start event [e300] processing. In state [/A3:Parent/A1:Stand By]
23:56:38,800 DEBUG [Run] Try transition [Stand By#Dispatching#e300#o3.x1]
23:56:38,800 INFO [Run] Finish input action [o3.x1] calculation. Its value is [true]
23:56:38,800 INFO [Run] Transition to go found [Stand By#Dispatching#e300#o3.x1]
23:56:38,810 INFO [Run] Start output action [o3.x7] execution
23:56:39,311 INFO [Run] Start output action [o2.z4] execution
---- Decreasing Temp: 8
23:56:39,311 INFO [Run] Finish output action [o2.z4] execution
23:56:40,312 INFO [Run] Start output action [o2.z4] execution
---- Decreasing Temp: 7
23:56:40,312 INFO [Run] Finish output action [o2.z4] execution
-- Performing: 120 - 25
23:56:40,993 INFO [Run] Finish on-enter action [o3.x3] execution
```

3.8 Observations after implementation

- Statechart logic it is relatively easy to implement.
- Generating manually the GUI consumes time.
- Coupling the implementation between the control and view is not a problem.
- It could run platform independent if the java virtual machine and the appropriate Unimod libraries are present.

4 Generating Platform Independent GUI

4.1 Overview

The GUI behavior can be modeled by using a statechart, but there are still missing links to create an interface based on a graphical representation.

When the statechart has been created (controller), we have not more than a model that is able to react to a given an event. This controller and the GUI (View) cooperate to accomplish a given task, and separate the concerns of displaying and handling interactions for robustness.

There are tools that allow GUI modeling, which simplify the process of building a visual interface and create a platform independent representation of it by serializing the description to an xml file. During this project the tool AUIML was used to generate the GUI.

4.2 AUIML

4.2.1 Overview

It is a tool developed by IBM, it stands for Abstract User Interface Markup Language Toolkit. It runs under Eclipse 3.01 or IBM Software Development Platform 6.01.

The particularity of this application is the capability to render a GUI in different environments, either could be rendered in SWING or in WEB, the same binaries can be used for both purposes just changing the deployment procedure.

4.2.2 How does it work?

The user creates a GUI under a drag and drop environment, and AUIML creates the representation in a XML dialect and provides a main event listener that the user has to invoke sending as parameter the events generated during runtime in order to create / handle flow between the components.

After the GUI has been designed, the functionality is completed by:

- Creating a main class that will invoke the UserTaskManager, this class receives as parameters the name of the AUIML model, the component being observed (a panel), one or more DataBeans and the context.
- Adding the bean class with the respective setter and getter methods that will hold the data of the form, this class inherits from an already predefined AUIML class named DataBean (data object capable to supply data to an AUIML task)

- Adding the event listener, it inherits from an already predefined AUIML class named TaskActionListener(Receive action events).

Example 1:

```
UserTaskManager utm = new UserTaskManager(fileName, panelName, new DataBean[] {db},
getUserContext());
```

```
//Adds a TaskActionListener which is notified when commit processing is complete.
utm.addCommitListener(db);
```

```
//Adds a TaskActionListener which is notified when the user cancels out of a task.
utm.addCancelListener(db);
```

```
//Add an ActionListener for Buttons
utm.addTaskActionListener(db);
```

```
//Set's reference to UserTaskManager
db.setUtm(utm);
//Show the panel.
utm.invoke();
```

Example 2:

```
public int getTotalRows(){
    return m_dTotalRows;
}

public void setTotalRows(int i) throws IllegalUserDataException{
    m_dTotalRows = i;
}
```

Example 3:

```
public void actionPerformed(TaskActionEvent e) {
    String command = e.getActionCommand();

    if ( "CANCEL".equals(command) || "COMPLETE".equals(command) ) {
        SamplesLogger.getLogger().info("User requested to close the application");
        return;
    }
}
```

User task manager is used to interpret the AUIML document and to render the application correctly, it performs the following tasks:

- Locates and parses the AUIML document.
- Locates and loads the locale.
- Maps the components to the right DataBean object.

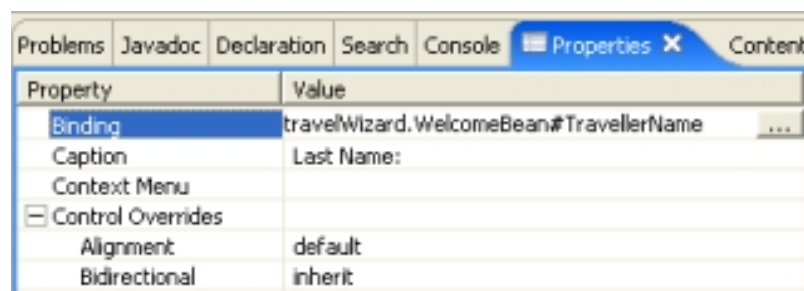
The DataBean is just a java bean that holds the data in the user interface, the way this is done is specified by the developer.

Utm is the class that keeps the context between the GUI and the interactions within the AUIML model; it is part of the AUIML core code.

To bind the involved elements (GUI, eventListener and DataBean) AUIML provides additional features to some of the components that send parameters. The sequence to set it up looks as follows:

- Any editable component is selected and in the properties panel is specified the class that receives the value and the parameter name, under the key binding.(See figure below).

com.table.TableBean# TotalRows



- The class specified in the last step must contain the code that match the expected parameter within the actionPerformed method.

```
if ( "RegenerateButton".equals(command) )  
    utm.storeElement("TotalRows");
```

4.2.3 AUIML Remarks

There are some nice features about the tool, it provides automate validation for string, numerical and date/time data types, and it makes use of locales to bring flexibility having different display options on the texts.

It is important to notice that it does not run by itself, some other component are needed in order to run the tool.

- EMF 2.02 (Modeling Framework)
- GEF 3.01 (Graphical Editing Framework)
- VE 1.0.2.2 (Visual Editor)

4.3 MVC Scenario in Unimod and AUIML

4.3.1 Ideal behavior

- An event is generated by the user by clicking on the wizard form generated by AUIML and it is forwarded to the controller automatically.
- The state chart examines the source of the event and sends a value to the java bean (Model).
- The java bean broadcast to the GUI (view) the change of value.
- The GUI detects the notification and updates its content accordingly.

4.3.2 Intention of modeling the controller with a statechart and AUIML

- Define how and when the user can interact with the GUI by catching the events.
- Associate actions on the GUI with actions on the model, the user is allowed to execute x but not y.
- Notify visually the user when the model changes.
- The changes done on the GUI have relatively no impact on the model.

4.3.3 Mapping between the statechart and AUIML

The mapping between the model and the code is represented as follows:

The transition, guard, and action in the statechart is represented by

o1.x2	Event
[condition == true]	Guard
o1.z4	Action

An action represented on the model is being coded in Unimod as a controlled object, within this object the guards are checked for validity using Unimod's context, the actions are represented as well within a controlled object.

Action and guard representation.

```
/**
 * @unimod.action.descr Check date validity
 */
public boolean x1(StateMachineContext context) {
    boolean blnValid = false;
    if(context.getEventContext().getParameter("myDate", Date().before("2004/06/06"));
        blnValid = true;
    return blnValid;
}

Action
/**
 * @unimod.action.descr Output
 */
public void z1(StateMachineContext context) {
    //execute any desired behavior.
}
```

The events issued on the model are not mapped with the code on the GUI. This could be accomplished by a listener on the AUIML side that detects when any modification on the model and as a response would fire an event on the GUI.

4.4 Binding the GUI editor and the GUI logic (View and Controller)

4.4.1 Overview

So far we have seen that it is possible to have a platform independent serialized version of two entities that might cooperate together, the GUI representation and the logic representation.

We have on one side the visual interface which knows when an event has been issued and on the other side a state chart with states and transitions which needs to be notified by an event in order to modify its state.

4.4.2 Joining both worlds

AUIML and Unimod are not compatible at the first glance because the first one runs under eclipse 3.0 and the second under eclipse 3.1, each one with their respective plug ins.

To make use of both frameworks one of them must play the role of main controller and the other must run on the background.

In this case AUIML must be the controller because the implementation details are not published and it would be harder to begin with that framework, on the other hand Unimod's implementation detail are published. The latter will be instantiated and will

listen to events generated by AUIML. This coupling is achieved by creating an event listener on the GUI side that forwards the notification to the state machine framework to be interpreted; then a Unimod event is instantiated and finally Unimod's event manager is notified about the event.

The Unimod specification dictates that there must be a provider and a controlled object that feeds the statechart with events and issue a response respectively.

In this implementation we would like to have the GUI as event provider and as a controlled object, but this would imply having both frameworks interacting with each other and running seamlessly, unfortunately this is not true due compatibility problems.

The GUI development with AUIML comprises 3 parts:

- Design of the GUI.
- Main file invoking AUIML and Unimod instances.
- Java beans binding to the components.

4.4.3 Coupling Details

There are two ways to model the GUI behavior with AUIML, using predefined events generated by the framework or by implementing manually the methods.

On the first alternative, there are some predefined events and listeners for modeling the action response, these are limited and might work for simple designs, but for fare more complex designs it is not sufficient. Some of the predefined methods include selected, enabled, disabled, etc.

On the second alternative is where the statechart can play the role of controller, we must link a basic event attached to any control, as a button, with a manually coded method that in exchange will invoke the statechart and perform the validations.

To accomplish such task, a Unimod instance is created during AUIML start up, it must have the reference to the .xml file describing the statechart, the events providers and controlled objects.

After creating the instance and thereby having access to the statechart engine, it is possible to request the managing of an event by invoking Unimod's event manager. At this point an event containing the values in the GUI component must be created to complete the process.

On the following step, the event is processed accordingly and the process flow returns to AUIML. The latter receives the response and determines the following action. Finally both frameworks, Unimod and AUIML, are again in stand by mode.

An interesting feature from AUIML is an internal statemachine for handling navigational issues. The internal state machine forwards or rewinds the flow to the proper screen whenever a next or last step is detected. That saves some work because the controls do not need to be refreshed or loaded before being shown.

4.4.4 Problems

- Both frameworks are running concurrently but in different instances.
- When AUIML issues the request Unimod needs to handle the call sequentially as one single process, but there are two processes which are not running synchronously.
- AUIML needs to stop the process flow to guarantee the correct output.

4.4.5 Implementing a Wizard with AUIML - Unimod

The wizzard implements a reservation system; it gathers information from the customer as name, last name, age, desired travel dates and the desired reservation type, plane, hotel or both.

Restrictions

Contains

- | | |
|--------------|------------------|
| • Last Name | • Address |
| • First Name | • Country |
| • Age | • Departure Date |
| • Email | • Return Date |

Main screen looks as follows:

Travel Reservations Wizard - Welcome

Welcome to the Wizard.

Last Name: First Name:

Age: Email:

Address:

Zip: Country:

Specify your travel dates:

Depart: May 21, 2006

Return: May 28, 2006

Specify your travel plans:

☒ Reserve a plane ticket

☐ Reserve a hotel room

☐ Reserve both a plane ticket and a hotel room

< Back Next > Finish Cancel

The data must be typed before continuing any further.

Typing right information and selecting reserve a plane ticket:

Graphical Response:

Travel Reservations Wizard - Reserve a Plane Ticket

Depart: Rochester, MN

Arrive: Raleigh, NC

☐ First

- Austin, TX
- Raleigh, NC
- Rochester, MN
- San Jose, CA
- Tampa, FL

< Back Next > Finish Cancel

Console:

```
[info] Finish input action [o1.y2] calculation. Its value is [true]
[info] Start input action [o1.y5] calculation
[info] Finish input action [o1.y5] calculation. Its value is [true]
[info] Transition to go found [Welcome#Plane Ticket#e201#o1.y1 && o1.y2 && o1.y5]
[info] Start output action [o1.x0] execution
[info] Finish output action [o1.x0] execution
[info] Start on-enter action [o1.z2] execution
Arriving to the reservation system
[info] Finish on-enter action [o1.z2] execution
[info] Finish event [e201] processing. In state [/A1:Plane Ticket]
```

The statechart needs to be activated during AUIML initialization, to accomplish such task it is necessary to import the Unimod class **com.evelopers.unimod.adapter.standalone.run** and send as parameter the .xml file generated previously by Unimod, the code invoking Unimod's engine looks as follows:

```
Run.execute(str);
```

```
ModelEngine engine = helper.createStandAloneModelEngine(model, true);
engine.getEventProcessor().addEventProcessorListener(logger);
engine.getEventProcessor().addExceptionHandler(eh);
engine.start();
```

During preparations for rendering the form, AUIML initializes the respective java beans for the forms and register the listeners.

Some lines output by the console during initialization of the statechart and AUIML.

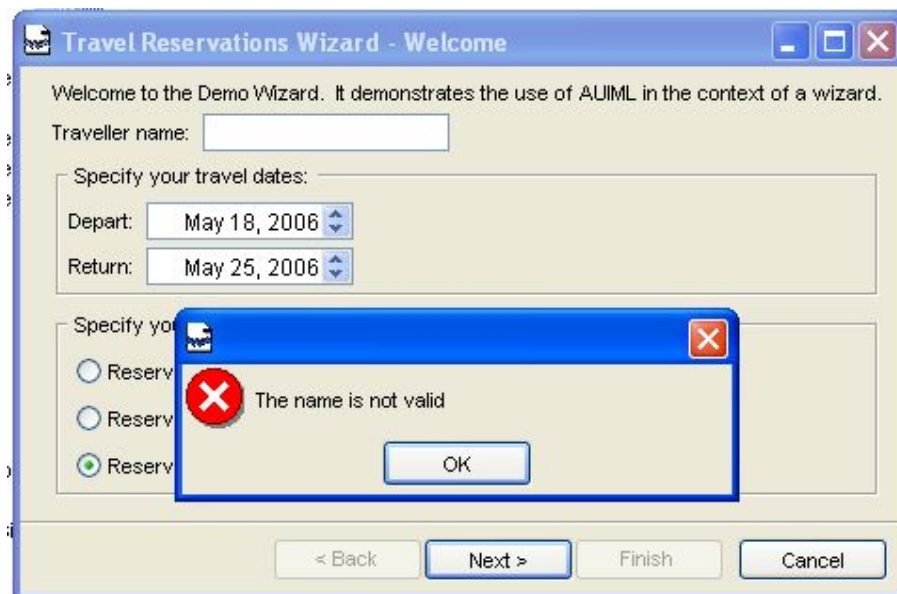
```
Setting engine [com.evelopers.unimod.runtime.ModelEngine@39443f]
INFO: SrLookAndFeelManager: Setting look and feel to [The Microsoft Windows Look and Feel -
com.sun.java.swing.plaf.windows.WindowsLookAndFeel].
INFO: Running under JDK version 1.4.2_06-b03
```

After the main screen was displayed and the user tries to advance to the next screen, the method bound to that specific control must throw the suitable event.

```
Event e201 =new Event(GUI.E201,
                    new Parameter[]
                        {
                            new Parameter("name", getTravellerName()),
                            new Parameter("startDate", getDepartDate()),
                            new Parameter("endDate", getReturnDate()),
                            new Parameter("planeTicket", new Boolean(true))
                        }
                    );

GUI.engine.getEventManager().handle(e201, GUI.context);
```

Graphical response



Console Response

```
[info] Start event [e201] processing. In state [/A1:Top]
[info] Transition to go found [s1#Welcome##true]
[info] Start on-enter action [o1.z1] execution
[info] Finish on-enter action [o1.z1] execution
[debug] Try transition [Welcome#Plane Ticket#e201#o1.y1 && o1.y2 && o1.y5]
[info] Start input action [o1.y1] calculation
the name is :
[info] Finish input action [o1.y1] calculation. Its value is [false]
[info] Start input action [o1.y2] calculation
[info] Finish input action [o1.y2] calculation. Its value is [true]
[info] Start input action [o1.y5] calculation
[info] Finish input action [o1.y5] calculation. Its value is [true]
[debug] Try transition [Welcome#Welcome#e201#o1.y1]
[info] Transition to go found [Welcome#Welcome#e201#o1.y1]
[info] Start output action [o1.x1] execution
[info] Finish output action [o1.x1] execution
[info] Start on-enter action [o1.z1] execution
[info] Finish on-enter action [o1.z1] execution
[info] Finish event [e201] processing. In state [/A1:Welcome]
```

At this point flow control is delegated to the statechart to handle the event.

Be aware to keep two things in mind.

- Both frameworks are running separately.
- After the execution of the invoked method AUIML commits the action.

In this case the wizard is being implemented by the AUIML component for building wizards.

To guarantee that the statechart will finish the task before delegating back the control to AUIML the thread needs to be stopped, this would simulate a sequential execution, and at the end of the process Unimod must notify to continue the old process together with the result output by the state chart.

Unimod's source code needs to be modified for such purpose, within the class **AbstractEventProcessor.java** from the package **com.evelopers.unimod.runtime** the completion of the event manger is fired in line number 132

```
/* Event Processing Finished */  
fireEventProcessingFinished(context, event, path, config);
```

Immediately we can notify AUIML to continue working with its flow.

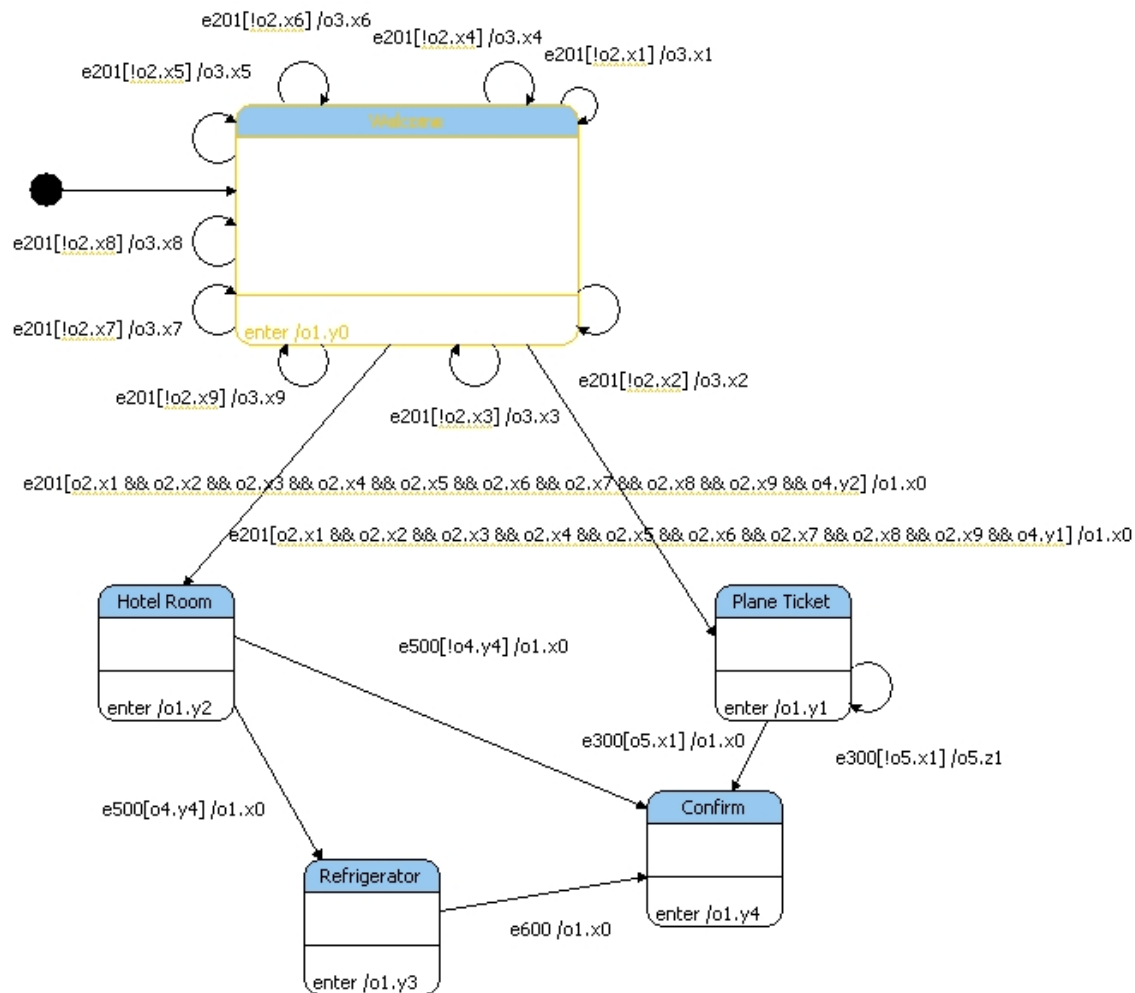
Besides the signal to resume the process a parameter specifying the output of the state chart is required to complete the task and allow AUIML to respond accordingly. This result is generated by a controlled object and stored into the common context.

```
/**  
 * @unimod.action.descr Throw Missing name  
 */  
public void x1(StateMachineContext context) {  
    context.getEventContext().setParameter("state", new Integer(0));  
}
```

After the control is back AUIML can process the result and act by transitioning to another screen or by showing an error. This is done by throwing an exception:

```
switch(m_iError)  
{  
    case 0:  
        throw new IllegalArgumentException("msg.MissingName");  
    case 1:  
        throw new IllegalArgumentException("msg.DepartAfterReturn");  
}
```

4.4.6 Unimod Model



Note how on the welcome state the same event can happen several times, as Harel model dictates, a priority must be specified in order to denote the state must be executed.

Unimod, as well as other tools as Rhapsody, execute a random transition if the event is satisfied by more than one transition.

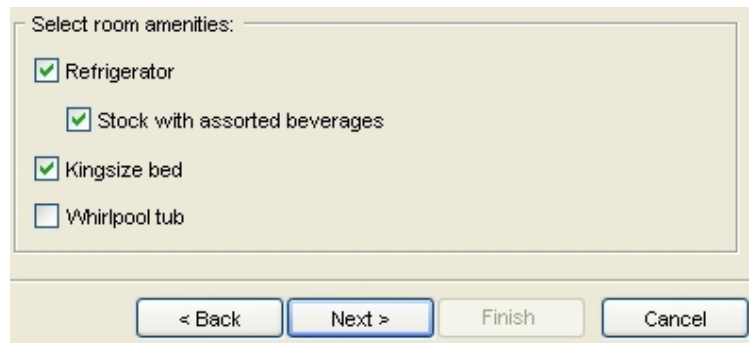
The event – action notation is as follows

Action	Description	Event	Description
o2.x1-x9	Main data validation	e201	Next button pressed
o1.y0-y4	Message indicating new state.	e300	Next button on Plane ticket pressed.
o3.x0-x9	Message errors	e500	Next button on Hotel pressed.
o4.y1-y4	Navigator validation.	e600	Next button on Refrigerator pressed.
o1.z0-z1	Data plane validation		

4.4.7 Missing formalism

Statechart's specification includes the history element which is helpful during the execution of consecutive states.

A history indicator may have any number of incoming transitions from outside states. It may have at the most one outgoing unlabeled transition; this identifies the default "previous state" if the region has never been entered.



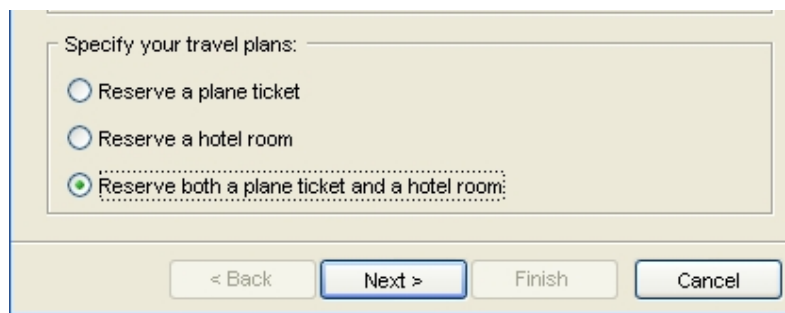
Select room amenities:

- ☒ Refrigerator
- ☒ Stock with assorted beverages
- ☒ Kingsize bed
- ☐ Whirlpool tub

< Back Next > Finish Cancel

At this point there is the possibility to continue with the flow or go backwards.

If a transition to the history indicator fires it indicates that the object resumes the last state it had within the complex region; any necessary entry actions are performed.



Specify your travel plans:

- ☐ Reserve a plane ticket
- ☐ Reserve a hotel room
- ☒ Reserve both a plane ticket and a hotel room:

< Back Next > Finish Cancel

This element is missing within Unimod's implementation; the only way to simulate this behavior would be by throwing an event on the back button specifying the desired screen.

In the Unimod – AUIML integration luckily the GUI generator contains a control for dealing with this specific problem, the navigation issues are handled automatically by the GUI.

4.5 Problems during framework interaction.

It would be easier to stop the flow execution right at the point where the state chart detected a constraint that is not fulfilled. This would be the optimal solution, because what we want is to model the behavior with a state chart and not to depend on the code implementation.

This would imply to throw an exception inside the controlled object's method as follows:

```
/**
 * @unimod.action.descr Throw missing name
 */
public void x0(StateMachineContext context) {
    throw new IllegalArgumentException(DemoWizardRes.format("msg.MissingName"));
}
```

Or by invoking a function within AUIML code to generate the exception and stop the process.

```
public static void throwException(String strError){
    throw new IllegalArgumentException(strError);
}
```

Ideally AUIML would stop at this point and the state chart would wait for incoming events.

The disadvantage of having two frameworks which were developed independently obstructs simple solutions and turns them into problematic ones.

The code shown above is not possible to implement without considering modifying Unimod's code strongly, because whenever an exception is detected in the execution of a method within a controlled object, an exception is thrown and it is handled within Unimod's context. At the end AUIML is not aware that an exception occurred.

4.5.1 Observations after implementation

- Generated Code: The amount of automatic generated code is considerable good on the View side, on the controller side the amount of code generated is acceptable, but there is room for improvement.
- Simplicity of the model: The model is fair easy to understand on the controller side and the viewer side.
- Maintainability: Considering that the coupling is not trivial because of the different nature of the implemented frameworks, at the time any substantial modification to the design would lead to strong changes.

- Simulation: The application cannot be simulated because of Unimod features.
- The event is generated by AUIML and the event is forwarded by hard coding the notification.
- The controller communicates to the java beans by using its own context as intermediary and yielding back the control to AUIML, and then the java bean reads the value.
- The model notifies the view, and the update takes place.

5 Conclusions

We have seen that it is possible to use model driven code generation technologies expressed in a graphical way to generate a platform independent GUI with its behavior modeled by a state chart.

Two different frameworks were used and both generated a platform independent format, an .xml file in the case of Unimod and a .xml dialect in the case of AUIML, which has the advantage of interpretation over compilation of a .class file. Modifications can be done during runtime without compromising the stability of the system

At this time, tools providing a framework for developing a GUI and its behavior based on a state chart model are still on the first generations and no direct link is possible, thereby the binding needs to be done manually.

Efforts being done are not heading towards a common goal, there are scattered sets of frameworks which focus in one direction and which are difficult to extend due the restrictions imposed by their creators.

In despite of Unimod's efforts to create a statechart framework, it is still in an early stage of development; some of Harel's specifications on statecharts are not implemented, as it is the case of the history and transient states.

User dependent functionalities on the client side allows the user to have full control over user sensitive data, in that way the decision of which information is disclosed and which is denied is up to the client.

The experiments described in this report are focused on integrating two independent frameworks; full integration would not be possible because AUIML is not an open source tool.

5.1 Considerations

As stated on [7] statechart implementations are still not standardized and the executable behavior might vary depending on the type of statechart applied.

Subtle semantic and syntactic differences exist in the implementations of UML and rhapsody statecharts, which are:

- Notation: An element can be represented with alternative notations. Orthogonality in Rhapsody is not represented in Unimod in the same way.
- Well Formed ness: A model might be well formed in a specific formalism, but it might not be well formed when porting the description to another

implementation. For example the statechart element history is not implemented within the Unimod statemachine framework.

- Executable behavior: Critical issue, a model will output different results depending where the statechart is implemented.

Any of the three formalisms complies with MDD, but the modeler need to be aware of how the models will be interpreted in other environments, as well as the tool developers must be aware of the differences and possible problems which might arise.

5.2 Related Work

Statecharts and activity diagrams are the essence of executable models, and they are taking more importance because the advantages that at any point the models can be checked by means of simulation in a graphical way.

Unimod provides a graphical environment where behavioral models can be executed and they can be debugged, but there are still missing features as others commercial tools to be able to compare the executed results.

There are some interesting extensions to the statechart notation which focus in enriching and address missing issues to accomplish a seamlessly implementation to the software systems.

AUIML works under a scheme that binds the process logic and the GUI representation in where to notify any change in visible way an exception must be thrown. This was a problem described in the last chapter, which could be solved with the implementation of a statechart notation that handles exception handling.

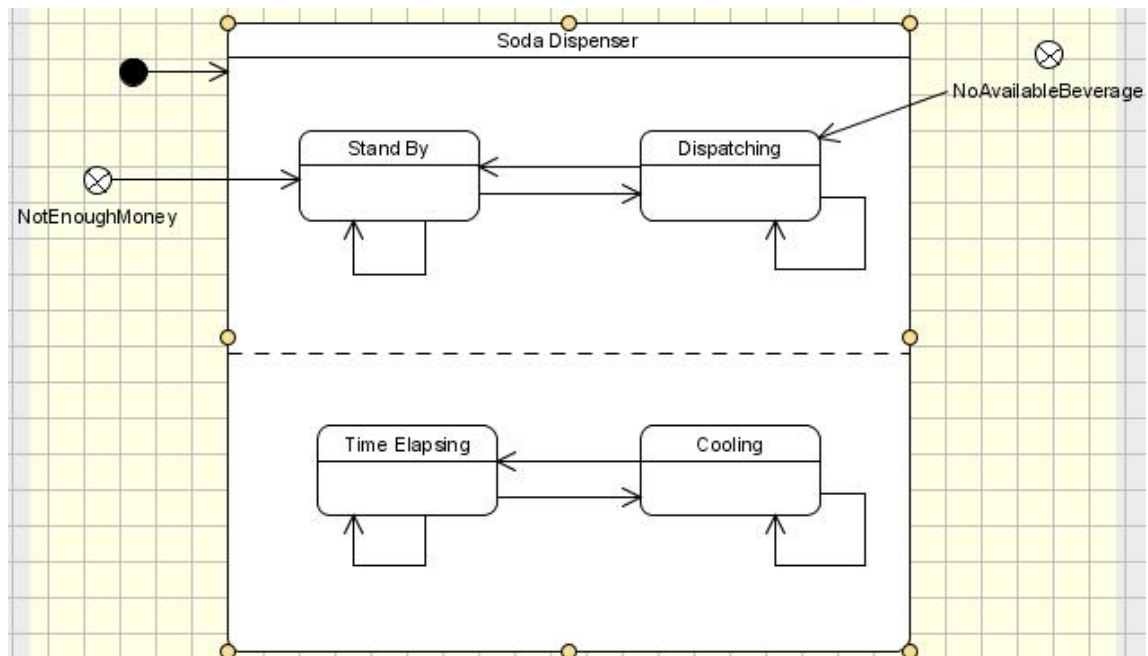
This problematic is addressed in a proposal extension made by Anderson, David in his paper Extending UML for UI presented at the Tupis workshop at UML2000.

Basically the exception is modeled within the statechart as another type of event that modifies the state of the GUI. The difference stems on the source of the event, the model is the responsible for providing the notification and not the GUI.

The event must be caught by a state that handles exceptions and then it must transitions to a stable state based on the interpretation of the error.

The exception symbol resembles the history state on the original statechart specification but with a X instead of an H.

Example:



Whenever the user selects a beverage and the right of amount of money is not sufficient the model must throw the exception, in a similar case, when a beverage is not available any more an exception is thrown.

5.3 Further Work

There is a middle stage that needs to be addressed before going any further, no open source framework following the model driven guidelines has fulfilled the gap between the GUI and statechart design. The mapping is still an open issue and it is an area for further research.

Current integration between the GUI event generator and the statechart event listener takes time and knowledge of the frameworks being integrated.

As a following step, the full integration of an open source GUI generator framework with an open source statechart framework must be done in order to ease the study of more complex scenarios.

6 References

- ¹ Horrocks, I. 1999 *Constructing the User Interface with Statecharts*. 1st. Addison-Wesley Longman Publishing Co., Inc.
- 2 Stefano Ceri, Peter Dolog, Maristella Matera, and Wolfgang Nejdl: Adding Client-Side Adaptation to the Conceptual Design of e-Learning Web Applications. *Journal of Web Engineering*, 4(1):21-37, March 2005
- ³ Statechart XML: Statemachine notation for control abstraction, <http://www.w3.org/TR/scxml/>, W3C Working Draft 2006
- 4 Harrel David, Gery Eran, Executable Object modeling with Statecharts, IEEE Computer Society, Pages 246-257, Berlin, 1996
- 5 Anderson, David, Extending UML for UI, position paper for the TUPIS2000 workshop at UML2000, 2000
- 6 Dobrowolski Janusz, Kolodziej Jerzy, A method of building executable platform independent application models, Statesoft, 2002
- 7 Crane, Michelle, Dingel Juergen, UML vs. Classical vs. Rhapsody Statecharts: Not all models are created equal, School of Computing, Queen's University, 2005
- 8 O'Byrne, Brian, Interaction Design and Implementation with ViewControl – using Statechart modeling and runtime verification, Statesoft Ltd., 2004
- 9 Anderson, David, O'Byrne Brian, Lean Interaction Design and Implementation: Using statecharts with feature driven development, 2003
- 10 Anderson David, Web MVC Browsers, Transactions and exceptions, Whitepaper, uidesign, 2000
- 11 David Harel and Hillel Kugler, The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML), *Lecture Notes in Computer Science*, Volume 3147, Jan 2004, Pages 325 - 354.
- 12 Carr David, Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces, Department of Computer Science, Lulea University, Sweden, 1999
- 13 Anderson David, A statemachine engine for Web MVC, uidesign, Whitepaper, 2000
- 14 Niemann Scott, Executable systems Design with UML 2.0, I-Logix, Whitepaper, 2004
- ¹⁵ Douglass, Bruce, Real time UML, Deveolping efficient objects for embedded systems. USA, 1998

7 Appendix

7.1 File generated by Unimod:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE model PUBLIC "-//eVelopers
Corp.//DTD State machine model V1.0//EN"
"http://www.evelopers.com/dtd/unimod/statemachine.dtd">
<model name="Modell">
  <controlledObject class="UnimodObjects.printer" name="o1"/>
  <controlledObject class="UnimodObjects.welcomeData" name="o2"/>
  <controlledObject class="UnimodObjects.welcomeErrors" name="o3"/>
  <controlledObject class="UnimodObjects.navigator" name="o4"/>
  <controlledObject class="UnimodObjects.PlaneData" name="o5"/>
  <eventProvider class="UnimodObjects.GUI" name="p1">
    <association clientRole="p1" targetRef="A1"/>
  </eventProvider>
  <rootStateMachine>
    <stateMachineRef name="A1"/>
  </rootStateMachine>
  <stateMachine name="A1">
    <configStore
      class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/
    >
    <association clientRole="A1" supplierRole="o4" targetRef="o4"/>
    <association clientRole="A1" supplierRole="o2" targetRef="o2"/>
    <association clientRole="A1" supplierRole="o3" targetRef="o3"/>
    <association clientRole="A1" supplierRole="o5" targetRef="o5"/>
    <association clientRole="A1" supplierRole="o1" targetRef="o1"/>
    <state name="Top" type="NORMAL">
      <state name="s1" type="INITIAL"/>
      <state name="Welcome" type="NORMAL"/>
      <state name="Refrigerator" type="NORMAL">
        <outputAction ident="o1.y3"/>
      </state>
      <state name="Plane Ticket" type="NORMAL">
        <outputAction ident="o1.y1"/>
      </state>
      <state name="Hotel Room" type="NORMAL">
        <outputAction ident="o1.y2"/>
      </state>
      <state name="Confirm" type="NORMAL">
        <outputAction ident="o1.y4"/>
      </state>
    </state>
    <transition sourceRef="s1" targetRef="Welcome"/>
    <transition event="e201" guard="!o2.x1" sourceRef="Welcome"
      targetRef="Welcome">
      <outputAction ident="o3.x1"/>
    </transition>
    <transition event="e201" guard="!o2.x4" sourceRef="Welcome"
      targetRef="Welcome">
      <outputAction ident="o3.x4"/>
    </transition>
    <transition event="e201" guard="!o2.x5" sourceRef="Welcome"
      targetRef="Welcome">
      <outputAction ident="o3.x5"/>
    </transition>
```

```

<transition event="e201" guard="!o2.x6" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x6"/>
</transition>
<transition event="e201" guard="!o2.x8" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x8"/>
</transition>
<transition event="e201" guard="!o2.x7" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x7"/>
</transition>
<transition event="e201" guard="!o2.x9" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x9"/>
</transition>
<transition event="e201" guard="!o2.x2" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x2"/>
</transition>
<transition event="e201" guard="!o2.x3" sourceRef="Welcome"
  targetRef="Welcome">
  <outputAction ident="o3.x3"/>
</transition>
<transition event="e201" guard="o2.x1 && o2.x2 && o2.x3
  && o2.x4 && o2.x5 && o2.x6 && o2.x7
  && o2.x8 && o2.x9 && o4.y2"
  sourceRef="Welcome" targetRef="Hotel Room">
  <outputAction ident="o1.x0"/>
</transition>
<transition event="e201" guard="o2.x1 && o2.x2 && o2.x3
  && o2.x4 && o2.x5 && o2.x6 && o2.x7
  && o2.x8 && o2.x9 && o4.y1"
  sourceRef="Welcome" targetRef="Plane Ticket">
  <outputAction ident="o1.x0"/>
</transition>
<transition event="e600" sourceRef="Refrigerator" targetRef="Confirm">
  <outputAction ident="o1.x0"/>
</transition>
<transition event="e300" guard="o5.x1" sourceRef="Plane Ticket"
  targetRef="Confirm">
  <outputAction ident="o1.x0"/>
</transition>
<transition event="e300" guard="!o5.x1" sourceRef="Plane Ticket"
  targetRef="Plane Ticket">
  <outputAction ident="o5.z1"/>
</transition>
<transition event="e500" guard="!o4.y4" sourceRef="Hotel Room"
  targetRef="Confirm">
  <outputAction ident="o1.x0"/>
</transition>
<transition event="e500" guard="o4.y4" sourceRef="Hotel Room"
  targetRef="Refrigerator">
  <outputAction ident="o1.x0"/>
</transition>
</stateMachine>
</model>

```

7.2 File generated by AUIML

```
public class DemoWizard extends java.util.ListResourceBundle {
    public Object[][] getContents() { return contents; }
    static final Object[][] contents = {
        {"@@GenerateBeans", "0"},
        {"@@GenerateHandlers", "0"},
        {"@@GenerateHelp", "1"},
        {"@@Package", "wizard"},
        {"@AdvTravelWizard", "STYLE:SERIAL;"},
        {"@AdvTravelWizard.Intro", "STYLE:CONTAINER;"},
        {"@AdvTravelWizard.Reservations", "STYLE:CONTAINER;"},
        {"@AdvTravelWizard.Summary", "STYLE:CONTAINER;"},
        {"@BeveragesPanel", "ROWS:7;COLUMNS:2;"},
        {"@BeveragesPanel.COLA", "CELL:1,3;"},
        {"@BeveragesPanel.DIETCOLA", "CELL:1,4;"},
        {"@BeveragesPanel.MILK", "CELL:1,5;"},
        {"@BeveragesPanel.ORANJUICE", "CELL:1,2;"},
        {"@BeveragesPanel.SPRWATER", "CELL:1,1;"},
        {"@HotelPanel", "ROWS:4;COLUMNS:2;"},
        {"@HotelPanel.AmenitiesCheckboxGroup",
"ROWS:4;HELPGEN:FALSE;CELL:0,3;COLSPAN:0;COLUMNS:2;"},
        {"@HotelPanel.City", "CELL:1,0;CONTROL-TYPE:COMBOBOX;"},
        {"@HotelPanel.City.CAPTION", "CELL:0,0;"},
        {"@HotelPanel.HotelName", "CELL:0,2;COLSPAN:0;CONTROL-TYPE:LIST;"},
        {"@HotelPanel.HotelName.CAPTION", "CELL:0,1;"},
        {"@HotelPanel.KingBedCheckbox",
"ROWS:1;HELPGEN:FALSE;CELL:1,2;COLUMNS:1;"},
        {"@HotelPanel.KingBedCheckbox.CAPTION", "CELL:0,2;"},
        {"@HotelPanel.RefrigeratorCheckbox",
"ROWS:2;HELPGEN:FALSE;CELL:0,1;COLUMNS:1;"},
        {"@HotelPanel.RefrigeratorCheckbox.CAPTION", "CELL:0,0;"},
        {"@HotelPanel.StockRefrigeratorCheckbox", "INSETLEFT:20;CELL:0,0;"},
        {"@HotelPanel.WhirlpoolTubCheckbox",
"ROWS:1;HELPGEN:FALSE;CELL:1,3;COLUMNS:1;"},
        {"@HotelPanel.WhirlpoolTubCheckbox.CAPTION", "CELL:0,3;"},
        {"@InProgressPanel", "ROWS:4;REFRESHINTERVAL:5;COLUMNS:2;"},
        {"@InProgressPanel.Group1",
"ANCHOR:CENTER;ROWS:1;FILL:NONE;HELPGEN:FALSE;CELL:0,3;COLSPAN:0;COLUMNS:1;"},
        {"@InProgressPanel.HotelImage", "CELL:1,1;"},
        {"@InProgressPanel.HotelImage.CAPTION", "CELL:1,2;"},
        {"@InProgressPanel.InProgressOrCompleteLabel",
"CELL:0,0;COLSPAN:0;CONTROL-TYPE:LABEL;"},
        {"@InProgressPanel.OkButton", "CONSOLE-CONTRIBUTION:TRUE;CELL:0,0;"},
        {"@InProgressPanel.PlaneImage", "CELL:0,1;"},
        {"@InProgressPanel.PlaneImage.CAPTION", "CELL:0,2;"},
        {"@PlanePanel", "ROWS:4;COLUMNS:2;"},
        {"@PlanePanel.ArriveCity", "CELL:1,1;CONTROL-TYPE:COMBOBOX;"},
        {"@PlanePanel.ArriveCity.CAPTION", "CELL:0,1;"},
        {"@PlanePanel.DepartCity", "CELL:1,0;CONTROL-TYPE:COMBOBOX;"},
        {"@PlanePanel.DepartCity.CAPTION", "CELL:0,0;"},
        {"@PlanePanel.FirstClassCheckbox",
"INSETLEFT:10;CELL:0,2;COLSPAN:0;"},
        {"@SummaryPanel", "ROWS:4;COLUMNS:1;"},
        {"@SummaryPanel.ArriveCity", "CELL:1,1;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.ArriveCity.CAPTION", "CELL:0,1;"},
        {"@SummaryPanel.DepartCity", "CELL:1,0;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.DepartCity.CAPTION", "CELL:0,0;"},
        {"@SummaryPanel.DepartDate", "CELL:1,1;"},
        {"@SummaryPanel.DepartDate.CAPTION", "CELL:0,1;"},
        {"@SummaryPanel.Group1", "ROWS:6;HELPGEN:FALSE;CELL:0,0;COLUMNS:2;"},
```



```

        {"@SummaryPanel.HotelGroup",
"ROWS:5;HELPGEN:FALSE;CELL:0,2;COLSPAN:0;COLUMNS:2;"},
        {"@SummaryPanel.HotelName", "CELL:1,0;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.HotelName.CAPTION", "CELL:0,0;"},
        {"@SummaryPanel.KingBed", "CELL:1,2;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.KingBed.CAPTION", "CELL:0,2;"},
        {"@SummaryPanel.PlaneGroup",
"ROWS:3;HELPGEN:FALSE;CELL:0,1;COLSPAN:0;COLUMNS:2;"},
        {"@SummaryPanel.Refrigerator", "CELL:1,1;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.Refrigerator.CAPTION", "CELL:0,1;"},
        {"@SummaryPanel.ReserveHotel", "CELL:1,4;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.ReserveHotel.CAPTION", "CELL:0,4;"},
        {"@SummaryPanel.ReservePlane", "CELL:1,3;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.ReservePlane.CAPTION", "CELL:0,3;"},
        {"@SummaryPanel.ReturnDate", "CELL:1,2;"},
        {"@SummaryPanel.ReturnDate.CAPTION", "CELL:0,2;"},
        {"@SummaryPanel.TravellerName", "CELL:1,0;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.TravellerName.CAPTION", "CELL:0,0;"},
        {"@SummaryPanel.WhirlpoolTub", "CELL:1,3;CONTROL-TYPE:LABEL;"},
        {"@SummaryPanel.WhirlpoolTub.CAPTION", "CELL:0,3;"},
        {"@TravelWizard", "STYLE:SERIAL;"},
        {"@WelcomePanel", "ROWS:8;COLUMNS:5;"},
        {"@WelcomePanel.address", "CELL:1,3;CONTROL-TYPE:TEXTFIELD;OPTIMUM-
LENGTH:20;"},
        {"@WelcomePanel.address.CAPTION", "CELL:0,3;"},
        {"@WelcomePanel.age", "CELL:1,2;CONTROL-TYPE:TEXTFIELD;"},
        {"@WelcomePanel.age.CAPTION", "CELL:0,2;"},
        {"@WelcomePanel.BothRadioButton",
"ROWS:1;CELL:1,2;HELPGEN:FALSE;COLUMNS:1;"},
        {"@WelcomePanel.BothRadioButton.CAPTION", "CELL:0,2;"},
        {"@WelcomePanel.country", "CELL:3,4;CONTROL-TYPE:TEXTFIELD;"},
        {"@WelcomePanel.country.CAPTION", "CELL:2,4;"},
        {"@WelcomePanel.DepartDate", "CELL:1,0;CONTROL-TYPE:SPINNER;"},
        {"@WelcomePanel.DepartDate.CAPTION", "CELL:0,0;"},
        {"@WelcomePanel.email", "CELL:3,2;CONTROL-TYPE:TEXTFIELD;"},
        {"@WelcomePanel.email.CAPTION", "CELL:2,2;"},
        {"@WelcomePanel.firstName", "CELL:3,1;CONTROL-TYPE:TEXTFIELD;"},
        {"@WelcomePanel.firstName.CAPTION", "CELL:2,1;"},
        {"@WelcomePanel.Group1",
"ROWS:3;CELL:0,5;COLSPAN:0;HELPGEN:FALSE;COLUMNS:2;"},
        {"@WelcomePanel.HotelRadioButton",
"ROWS:1;CELL:1,1;HELPGEN:FALSE;COLUMNS:1;"},
        {"@WelcomePanel.HotelRadioButton.CAPTION", "CELL:0,1;"},
        {"@WelcomePanel.Label1", "CELL:0,0;COLSPAN:0;CONTROL-TYPE:LABEL;"},
        {"@WelcomePanel.PlaneRadioButton",
"ROWS:1;CELL:1,0;HELPGEN:FALSE;COLUMNS:1;"},
        {"@WelcomePanel.PlaneRadioButton.CAPTION", "CELL:0,0;"},
        {"@WelcomePanel.ReturnDate", "CELL:1,1;CONTROL-TYPE:SPINNER;"},
        {"@WelcomePanel.ReturnDate.CAPTION", "CELL:0,1;"},
        {"@WelcomePanel.TravellerName", "CELL:1,1;CONTROL-
TYPE:TEXTFIELD;OPTIMUM-LENGTH:15;"},
        {"@WelcomePanel.TravellerName.CAPTION", "CELL:0,1;"},
        {"@WelcomePanel.TravelPlansButtonGroup",
"ROWS:3;CELL:0,6;COLSPAN:0;HELPGEN:FALSE;COLUMNS:2;"},
        {"@WelcomePanel.zip", "CELL:1,4;CONTROL-TYPE:TEXTFIELD;"},
        {"@WelcomePanel.zip.CAPTION", "CELL:0,4;"},
        {"AdvTravelWizard.IMAGE", "auimlLogo.gif"},
        {"AdvTravelWizard.Intro.TEXT", "Introduction"},
        {"AdvTravelWizard.Reservations.TEXT", "Reservations"},
        {"AdvTravelWizard.Summary.TEXT", "Summary"},
        {"AdvTravelWizard.TEXT", "Travel Reservations Wizard"},
        {"BeveragesPanel.COLA.TEXT", "Cola"},
        {"BeveragesPanel.DIETCOLA.TEXT", "Diet Cola"},

```

```

        {"BeveragesPanel.MILK.TEXT", "Milk"},
        {"BeveragesPanel.ORANJUICE.TEXT", "Orange Juice"},
        {"BeveragesPanel.SPRWATER.TEXT", "Spring Water"},
        {"BeveragesPanel.TEXT", "Beverage Selection"},
        {"HotelPanel.AmenitiesCheckboxGroup.TEXT", "Select room amenities:"},
        {"HotelPanel.City.TEXT", "Select a city:"},
        {"HotelPanel.HotelName.TEXT", "Select a hotel:"},
        {"HotelPanel.KingBedCheckbox.TEXT", "Kingsize bed"},
        {"HotelPanel.RefrigeratorCheckbox.TEXT", "Refrigerator"},
        {"HotelPanel.StockRefrigeratorCheckbox.TEXT", "Stock with assorted
beverages"},
        {"HotelPanel.TEXT", "Reserve a Hotel Room"},
        {"HotelPanel.WhirlpoolTubCheckbox.TEXT", "Whirlpool tub"},
        {"InProgressPanel.HotelImage.TEXT", "Hotel room"},
        {"InProgressPanel.HotelImage.VALUE", " hotel.jpg"},
        {"InProgressPanel.OkButton.TEXT", "OK"},
        {"InProgressPanel.PlaneImage.TEXT", "Plane ticket"},
        {"InProgressPanel.PlaneImage.VALUE", " plane.jpg"},
        {"InProgressPanel.TEXT", "In Progress"},
        {"msg.CancelWizard", "Are you sure you want to cancel?"},
        {"msg.DepartAfterReturn", "The specified departure date must not be
later than the specified return date."},
        {"msg.DepartAndReturnCitySame", "The departure city and arrival city
can not both be {0}."},
        {"msg.missingFirstName", "The first name is not valid"},
        {"msg.MissingName", "The name is not valid"},
        {"msg.NoHotelIfNoOvernight", "You can not reserve a hotel room if the
specified return date is the same as the specified departure date."},
        {"PlanePanel.ArriveCity.TEXT", "Arrive:"},
        {"PlanePanel.DepartCity.TEXT", "Depart:"},
        {"PlanePanel.FirstClassCheckbox.TEXT", "First class seat"},
        {"PlanePanel.TEXT", "Reserve a Plane Ticket"},
        {"str.No", "No"},
        {"str.ReservationsComplete", "Travel reservations complete."},
        {"str.ReservationsInProgress", "Travel reservations in progress..."},
        {"str.Yes", "Yes"},
        {"str.YesFirstClass", "Yes, first class"},
        {"str.YesStocked", "Yes, stocked with beverages"},
        {"SummaryPanel.ArriveCity.TEXT", "Arrive:"},
        {"SummaryPanel.DepartCity.TEXT", "Depart:"},
        {"SummaryPanel.DepartDate.TEXT", "Depart:"},
        {"SummaryPanel.HotelGroup.TEXT", "Hotel room details:"},
        {"SummaryPanel.HotelName.TEXT", "Hotel name:"},
        {"SummaryPanel.KingBed.TEXT", "Kingsize bed:"},
        {"SummaryPanel.PlaneGroup.TEXT", "Plane ticket details:"},
        {"SummaryPanel.Refrigerator.TEXT", "Refrigerator:"},
        {"SummaryPanel.ReserveHotel.TEXT", "Reserve hotel room:"},
        {"SummaryPanel.ReservePlane.TEXT", "Reserve plane ticket:"},
        {"SummaryPanel.ReturnDate.TEXT", "Return:"},
        {"SummaryPanel.TEXT", "Summary"},
        {"SummaryPanel.TravellerName.TEXT", "Traveller name:"},
        {"SummaryPanel.WhirlpoolTub.TEXT", "Whirlpool tub:"},
        {"TravelWizard.IMAGE", " auimlLogo.gif"},
        {"TravelWizard.TEXT", "Travel Reservations Wizard"},
        {"WelcomePanel.address.TEXT", "Address"},
        {"WelcomePanel.age.TEXT", "Age"},
        {"WelcomePanel.BothRadioButton.TEXT", "Reserve both a plane ticket
and a hotel room"},
        {"WelcomePanel.country.TEXT", "Country"},
        {"WelcomePanel.DepartDate.TEXT", "Depart:"},
        {"WelcomePanel.email.TEXT", "Email"},
        {"WelcomePanel.firstName.TEXT", "First Name :"},
        {"WelcomePanel.Group1.TEXT", "Specify your travel dates:"},

```

```

        {"WelcomePanel.HotelRadioButton.TEXT", "Reserve a hotel room"},
        {"WelcomePanel.Label1.VALUE", "Welcome to the Wizard. "},
        {"WelcomePanel.PlaneRadioButton.TEXT", "Reserve a plane ticket"},
        {"WelcomePanel.ReturnDate.TEXT", "Return:"},
        {"WelcomePanel.TEXT", "Welcome"},
        {"WelcomePanel.TravellerName.TEXT", "Last Name:"},
        {"WelcomePanel.TravelPlansButtonGroup.TEXT", "Specify your travel
plans:"},
        {"WelcomePanel.zip.TEXT", "Zip"},
    };
}

```
