Project definition - Group G

Project 1: Code generation

Jonas Hansen
s052905

Peter Wind
s052425

Michael Sørensen
s052683

March 27, 2009

Technical University of Denmark

Course: Advanced topics in software engineering (02265)

Contents

1 Problem Statement

During the course Software Engineering 2 at DTU a CASE tool for defining, modeling and combining components was developed. An instance of the model could be simulated with the CASE tool and the behavior of the system could thereby be analyzed. However the tool was not capable of generating code which could be used outside of the tool. It is exactly the purpose of this project to fulfill this lack by providing the functionality of code generation directly from a model. The generated code could be in any programming language but Java has been selected for this project since it is the language of choice when using Eclipse. Besides providing code generation the result of this project should also include a small framework for executing the generated code and interacting with this execution.

1.1 Terms

To getter a better understanding of the contends of the project some key term will be described briefly in this section.

- A typical *model* provided from the CASE tool of Software Engineering 2 contains a number of *component definitions*.
- A *component definition* consists of a *state machine*, a number of *attributes* and a number of *ports*.
- The internal *state machine* of a component definition defines the behavior of a component and how it reacts on events.
- Instances of these components are connected in a *deployment* using their *ports*.
- The connections between ports are made through *busses* that provide buffer capabilities.

1.2 Example

As mentioned the purpose of this project is to generate code from a model. To illustrate how a deployment is transformed into generated code an example model is used troughout the following sections. The model represents a simple Wiper system (eg. for a car) and is a modified version of an example model used in the course Software Engineering 2. The model is not realistic but is composed of all the details which are relevant when testing the code generator which should be developed.

The example model can be seen in Figure 1.

The model is a deployment consisting of two wiper components and a controller software component. The software component c is connected to two wipers through a bus. The simple behavior of the model is as follows: A message is placed on the sw port of the controller. This message is then propagated to each of the wipers. The wipers have a behavior in which they can either be turned off or turned on to a specific level. When a wiper is turned on to a specific level it can only have its level increased, not decreased.

The internal behaviour of the controller and the wipers is illustrated in Figure 2 and Figure 3.

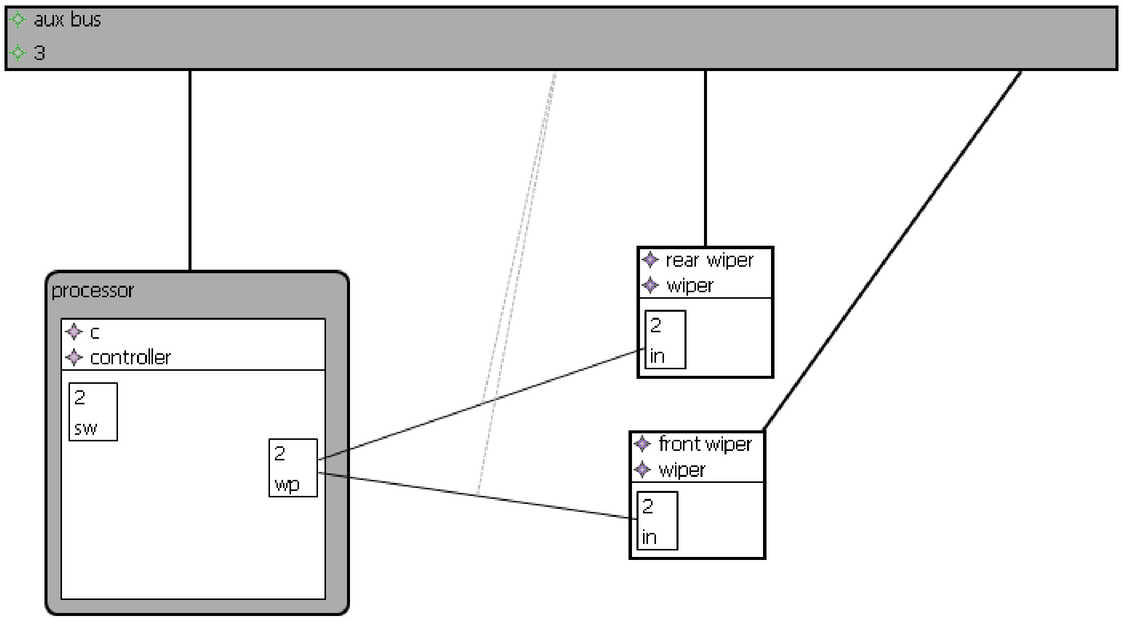


Figure 1: WiperSystem - Example model

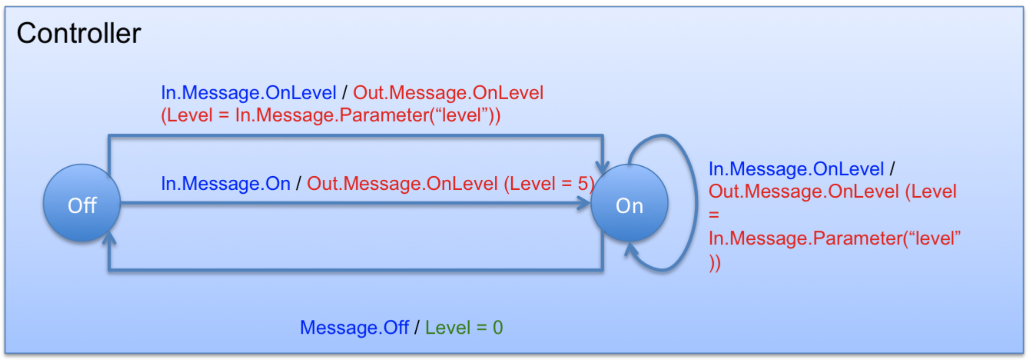

Controller
In.Message.OnLevel / Out.Message.OnLevel (Level = In.Message.Parameter("level"))
In.Message.On / Out.Message.OnLevel (Level = 5)
Off
On
In.Message.OnLevel / Out.Message.OnLevel (Level = In.Message.Parameter("level"))
Message.Off / Level = 0

Figure 2: Controller - Internal behaviour

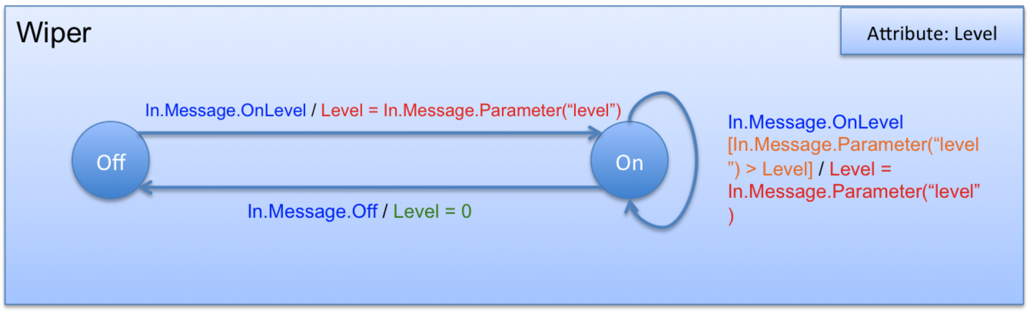

Wiper
Attribute: Level
In.Message.OnLevel / Level = In.Message.Parameter("level")
Off
On
In.Message.Off / Level = 0
In.Message.OnLevel [In.Message.Parameter("level ") > Level] / Level = In.Message.Parameter("level")

Figure 3: Wiper - Internal behaviour

2 Project Definition

The section will give the definition of the project. In section 2.1 two use cases will be covered two get a better understanding on how the system will be used and then. In section 2.2 the steps need to complete the project will be described.

2.1 Use cases

The two main use cases of this project will be covered in this section. These are the generation of code and the use of the framework.

2.1.1 Code generation

Use case description: Code generation is done from a simple CASE-tool model.
Precondition: A deployment exist with three component definitions (similar to the one described in section 1.2): Two wipers for the front and wiper, and a controller.
Postcondition: Three classes with the correct attributes and methods have been generated for frontwiper, rearwiper and controller. Furthermore a class for the deployment have been generated with its attributes and methods.

2.1.2 Framework

Use case description: The user interacts with a simple framework by clicking on widgets or buttons.
First the user pushes the button to see some attributes of the controller. Afterwards the user pushes the button to turn on the front-wiper. **Precondition:** Code generation have been already been done for a simple deployment consisting of a front-wiper, rear-wiper and controller, as described in the previous use case.
Postcondition: The user will be presented with attributes for the controller. For the instance of the front-wiper the automata is now in a state where the wiper is turned on, which also is inform to the user.

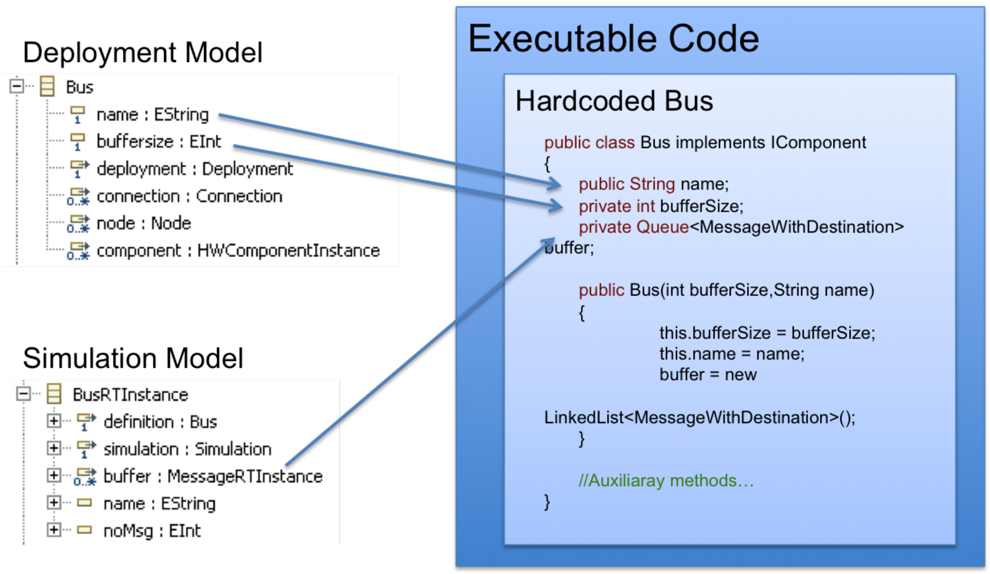
2.2 Fulfillment of requirements

The main purpose of this project is to be able to generate code from a given CASE-tool model. To be able to make a system which generates code from a model, it is important to actually manually write an example of how this generated code could look like. This has been done and can bee seen in Appendix A. This section will in contrast to the example in the appendix, look at how the different concepts of the model in general can be transformed into executable code. As it can be seen in the following sections, some parts of the model can be hardcoded without the need of any dynamic code generation, while other parts has to be generated completely.

2.2.1 Transformation of a Bus

In the provided model, a bus is a buffer which a connection from one port to another can be attached to. When a message is sent trough a connection attached to a bus, the message is not delivered right away but is instead stored in the queue of the bus. In each step of a simulation the topmost message in the queue is delivered to its target.

As the busses used in a deployment always are very alike (they only differ in name and buffersize), it is sufficient to hard code the bus and its concepts, and then create one or more instances in the generated deployment class. How the concepts of the bus is translated into executable code can be seen in Figure 4. From the figure it can be seen that the hard coded Bus contains a queue which accepts items of type MessageWithDefinition. This is simply a Pair consisting of a MessageDefinition and a Destination Port. From the figure it is also clear that many references from the model has been left out in the hard coded bus. This is intentional as many of the references are not needed to create a working executable code.



```
public class Bus implements IComponent
{
    public String name;
    private int bufferSize;
    private Queue<MessageWithDestination>
buffer;

    public Bus(int bufferSize,String name)
    {
            this.bufferSize = bufferSize;
            this.name = name;
            buffer = new
LinkedList<MessageWithDestination>();
    }

    //Auxiliaray methods...
}
```

Figure 4: Transforming a bus

2.2.2 Transformation of Messages

In a deployment, messages can be sent among components and busses. In the model these messages are defined using `MessageDefinitions`. A `MessageDefinition` has a name and a list of parameters. To generate executable code based on the MessageDefinitions used in a deployment at least two solutions could have been choosen:

- Generate a class for each `MessageDefinition` in the model with a class name of the value of the attribute "name".
- Make a single hard coded class `MessageDefinition` with a string variable "name" and a HashMap for the parameters. Generate a class `MessageDefinitions` which contains methods for creating instances of each message definition in the model as well as static string for all message definition names.

The last solution was chosen, because it fulfills all requirements in a very simple manor and makes it easy to reference the `MessageDefinitions` in the rest of the generated code.

Based on the chosen solution, Figure 5 provides a simplified overview on how messages in the model has been transformed into executable code.

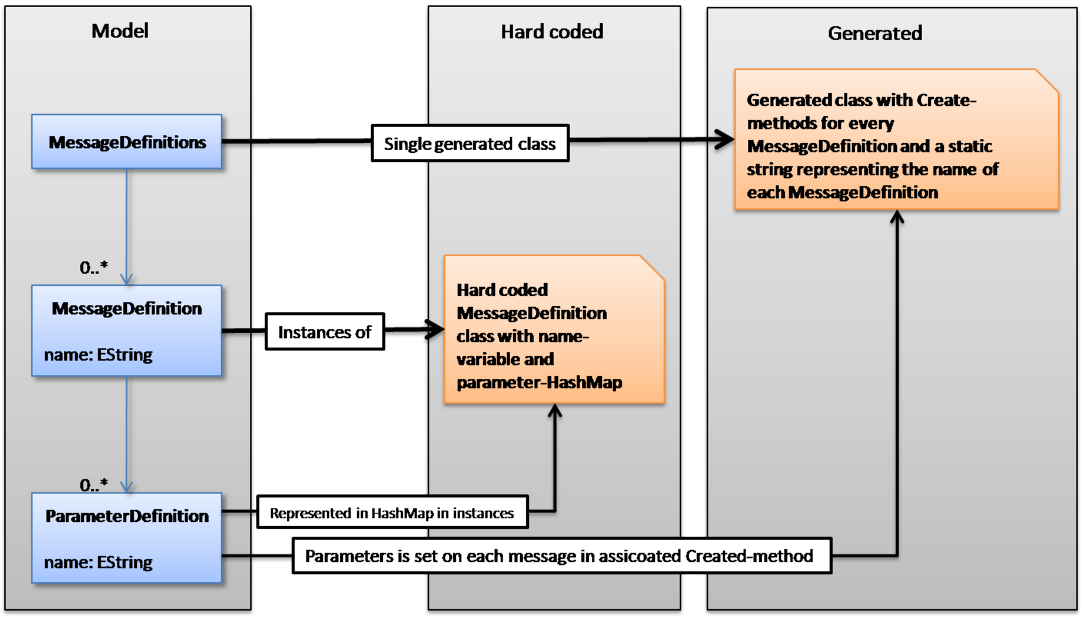

Model
Hard coded
Generated
MessageDefinitions
Single generated class
Generated class with Create-methods for every MessageDefinition and a static string representing the name of each MessageDefinition
0..*
MessageDefinition
name: EString
Instances of
Hard coded MessageDefinition class with name-variable and parameter-HashMap
0..*
ParameterDefinition
name: EString
Represented in HashMap in instances
Parameters is set on each message in assicoated Created-method

Figure 5: Transforming messages into code

2.2.3 Transformation of a Port

Because all ports in the model inherit the same behavior and basic structure, the ports will all be instances of a hard coded class. This class is almost identical to how a port is modeled: It has two lists of supported ingoing and outgoing message definitions, a buffer with message definitions and a list of connections which represents its outgoing connections. The instantiation of ports happens inside the constructers of the component definitions.

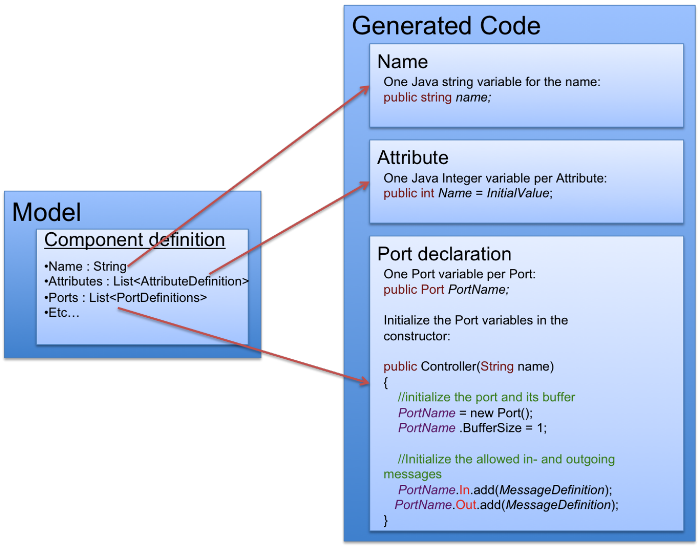
2.2.4 Transformation of Components

A model of a deployment consists of a number of components which are instances of one or more component definitions. For each component definition used in a deployment model, a new java class needs to be created. It is then the task of the generated deployment to instantiate the generated classes. As all component definitions can consist of the same parts a general scheme for the code generation can be developed which should be sufficient to generate any type of component definition. The scheme needs to handle the generation of the following parts of a component:

- Name
- Attributes
- Port declarations
- Automata

How code could be generated for the name, attributes and port declarations, can be seen in Figure 6. How code could be generated for the automata requires some elaboration as the automata contains a number of different concepts for which code needs to be generated.

The Automata contains a number of states which are connected using transitions. A transition contains a trigger and a number of actions. The trigger is composed of a trigger message associated with a specific port as well as a trigger condition. Both elements of the trigger are optional. The actions of a transition can be assignment operations and/or output messages. Each of the concepts mentioned needs attention in the code generation and Figure 7 tries to give an overview on how the different concepts could be transformed into executable code.



One Java string variable for the name:

```
public string name;
```

One Java Integer variable per Attribute:

```
public int Name = InitialValue;
```

One Port variable per Port:

```
public Port PortName;
```

Initialize the Port variables in the constructor:

```
public Controller(String name)
{
   //initialize the port and its buffer
   PortName = new Port();
   PortName .BufferSize = 1;

   //Initialize the allowed in- and outgoing
messages
   PortName.In.add(MessageDefinition);
  PortName.Out.add(MessageDefinition);
}
```

Figure 6: Transforming name, attributes and port declarations into code

2.2.5 Transformation of a deployment

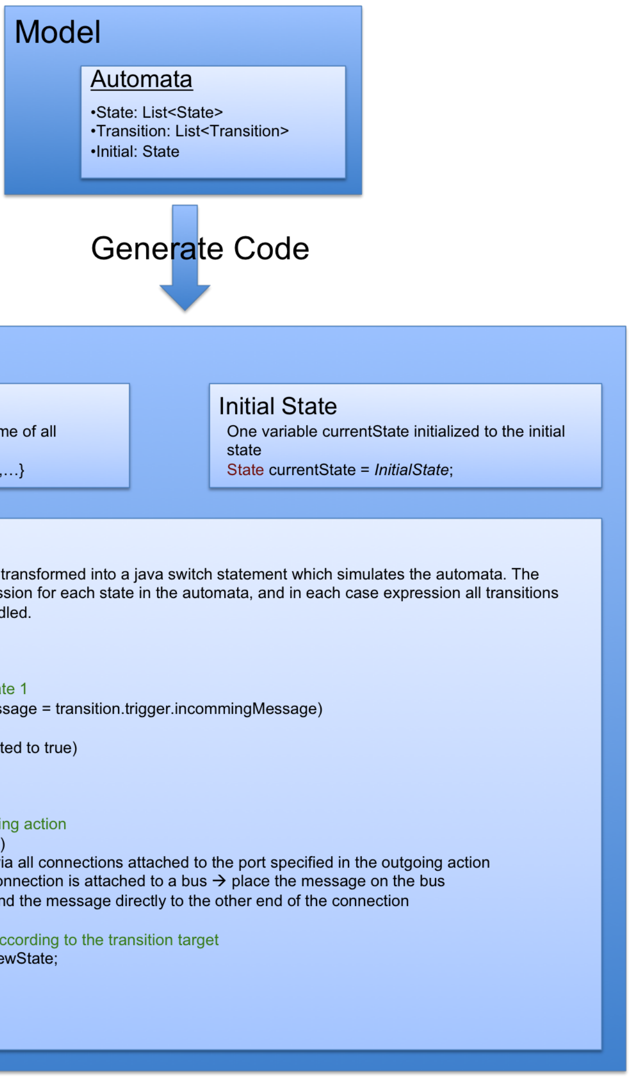
In a deployment, instances of component definitions, busses, ports etc are inserted to model a runtime environment. Because the deployment uses all component definitions it should in fact be the last thing which should be generated. The deployment will be a completely generated class where the name of the class will be the name of the deployment. A deployment-class will be constructed by a declaration section where all components and busses are initialized. This will be straight forward by using the generated constructers on the generated component definitions. The class will contain three important methods:

addBusses() Adds all the generated busses to a list on the deployment class. The busses will exist as public variables too.

connectPorts() This method will do all the necessary connecting of the components through their ports and busses. All ports have been generated as public variables on each component and is accessible that way.

addComponents() Adds all the generated components to a list on the deployment class. The components will exist as public variables too.

The transformation from a model of a deployment into executable code is also is illustrated in Figure 8 and Figure 9.



•State: List<State>
•Transition: List<Transition>
•Initial: State

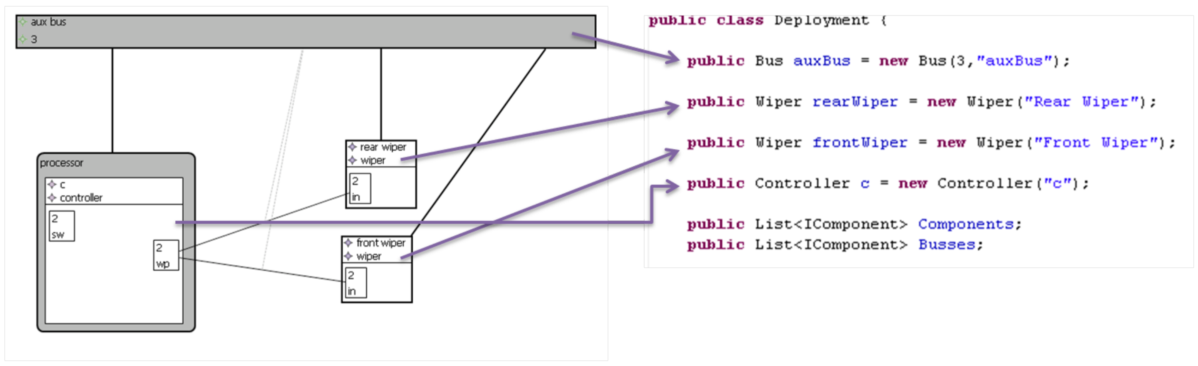
Create an enum containing the name of all states
private enum States {*state1*,*state2*,…}

One variable currentState initialized to the initial state
State currentState = *InitialState*;

The states and the transitions are transformed into a java switch statement which simulates the automata. The Automata consits of a case expression for each state in the automata, and in each case expression all transitions from the state in question are handled.

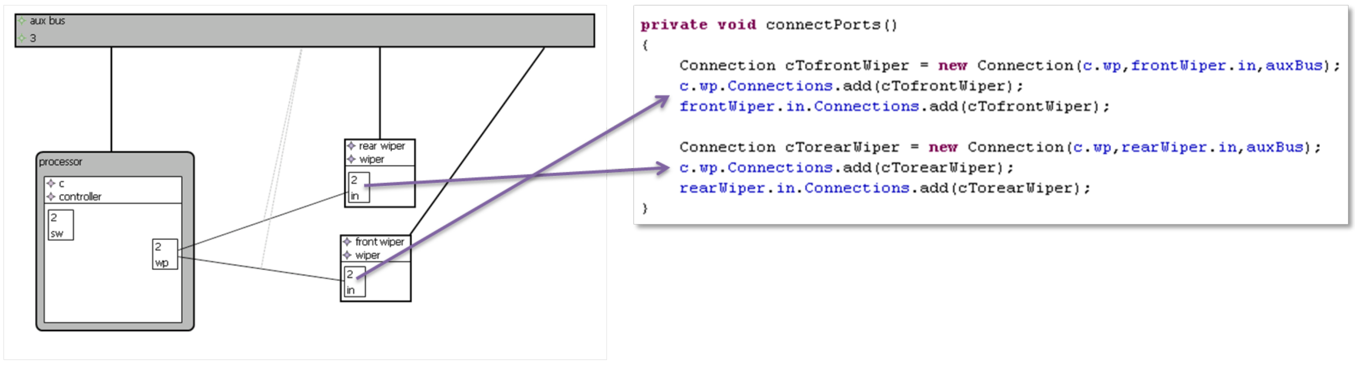
```
switch(currentState) {
   case state1:
       //for each transition from state 1
       if(transition.trigger.port. Message = transition.trigger.incommingMessage)
       {
               if(condition is evaluated to true)
               {
                   SetAttributes();

                   //For each outgoing action
                   CreateMessage()
                   Send Message via all connections attached to the port specified in the outgoing action
                             If the connection is attached to a bus → place the message on the bus
                             else send the message directly to the other end of the connection

                   //Change state according to the transition target
                   currentState = newState;
               }
       }
   case state2: ……
```

Figure 7: Transforming a automata into code



```
public class Deployment {
    public Bus auxBus = new Bus(3,"auxBus");
    public Wiper rearWiper = new Wiper("Rear Wiper");
    public Wiper frontWiper = new Wiper("Front Wiper");
    public Controller c = new Controller("c");
    public List<IComponent> Components;
    public List<IComponent> Busses;
```

Figure 8: Transforming component instances of a deployment



```
private void connectPorts()
{
    Connection cTofrontWiper = new Connection(c.wp,frontWiper.in,auxBus);
    c.wp.Connections.add(cTofrontWiper);
    frontWiper.in.Connections.add(cTofrontWiper);

    Connection cTorearWiper = new Connection(c.wp,rearWiper.in,auxBus);
    c.wp.Connections.add(cTorearWiper);
    rearWiper.in.Connections.add(cTorearWiper);
}
```

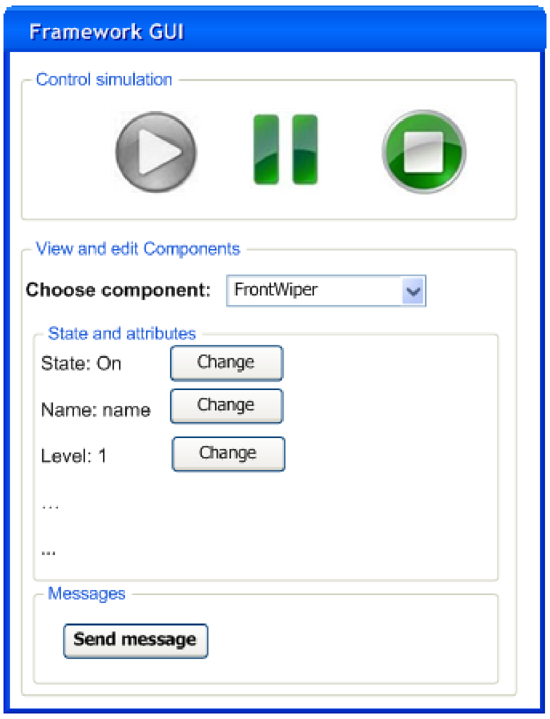
Figure 9: Transforming connections of a deployment

2.2.6 Implementation of an execution framework

A simple framework with a grahical user interface (GUI) needs to be developed.
Using this framework the user will be able to interact with the execution of the generated code. Examples of this interaction is:

- Pause or stop the execution
- View state and attributes of components.
- Edit state and attributes of components.
- Send a message to a specific component.

A sketch of a how the GUI could look like is presented in figure 10. Pushing the buttons will prompt the user for suitable input.



Choose component: FrontWiper

State: On

Name: name

Level: 1

...

...

Figure 10: Sketch of the GUI

2.2.7 Technology used to transform the model into code

The technology used in this project is Java Emitter Templates (JET). JET is a template engine, which can be used to generate Java code and other output from templates. The source of the generation can be any Java object, e.g. an EMF model, and using a JSP-like syntax it provides a simple way of generating code.
As described in section 2.2 some of the generated code needs to be hard coded while are acquired dynamically from the model. The JSP-like syntax of JET makes it easy to combine these two ways of generating code within the same file.

A Example transformation

This section contains the example transformation of the model mentioned in section TODO. The code has been written manually and should only be seen as an example of how a model could be transformed into executable code.

Listing 1: Bus.java

```java
package bus;

import java.util.LinkedList;
import java.util.Queue;

import component.IComponent;

import messages.MessageDefinition;
import port.Port;


public class Bus implements IComponent
{
  private Queue<MessageWithDestination> buffer;
  private int bufferSize;
  public String name;

  public Bus(int bufferSize,String name)
  {
    this.bufferSize = bufferSize;
    this.name = name;
    buffer = new LinkedList<MessageWithDestination>();
  }

  public void addToBus(MessageDefinition md,Port destination)
  {
    if(buffer.size() < bufferSize)
    {
      buffer.add(new MessageWithDestination(md,destination));
      System.out.println("The message " + md.Name + " has been added to bus " +
          this.name);
    }
    else
    {
      System.out.println("Message " + md.Name + " has been thrown away as the
          buffer of the bus " + this.name + " is full");
    }
  }

  @Override
  public void step()
  {
    if(buffer.size() > 0)
    {
      MessageWithDestination m = buffer.poll();
      m.destination.addToBuffer(m.message);
      System.out.println("The message " + m.message.Name + " has been added from
          the bus " + this.name + " to the port " + m.destination.toString());
    }

  }

  private class MessageWithDestination
  {
    public MessageDefinition message;
    public Port destination;

    public MessageWithDestination(MessageDefinition message, Port destination)
    {
      this.message = message;
      this.destination = destination;
    }
  }
}
```

Listing 2: Connection.java

```java
package connection;

import port.Port;
import bus.Bus;

public class Connection
{
  public Port source;
  public Port target;
  public Bus bus;

  public Connection(Port source, Port target)
  {
    this(source,target,null);
  }

  public Connection(Port source, Port target, Bus bus)
  {
    this.source = source;
    this.target = target;
    this.bus = bus;
  }

  public Port getOtherEnd(Port p)
  {
    if(source == p) return target;
    else if(target == p) return source;
    else return null;
  }
}
```

Listing 3: Controller.java

```java
package component;

import messages.MessageDefinition;
import messages.MessageDefinitions;
import port.Port;
import connection.Connection;

public class Controller implements IComponent
{

  public int level = 0;

  public String name;
  public Port sw;
  public Port rs;
  public Port wp;

  public boolean isHardware = false;

  enum States
  {
    off,on,auto
  }

  States currentState;

  public Controller(String name)
  {
    currentState = States.off;

    this.name = name;

    sw = new Port();
    sw.BufferSize = 1;
    sw.In.add(MessageDefinitions.CreateON());
    sw.In.add(MessageDefinitions.CreateOFF());
    sw.In.add(MessageDefinitions.CreateONLEVEL());

    wp = new Port();
    wp.BufferSize = 1;
    wp.Out.add(MessageDefinitions.CreateOFF());
    wp.Out.add(MessageDefinitions.CreateONLEVEL());
  }

  public void step()
  {
    switch(currentState) {

    case off:
      if(sw.ContainsMessageOfType(MessageDefinitions.ON))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.ON);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
          outgoingMessage = MessageDefinitions.CreateONLEVEL();
          //Set outgoing variables
          outgoingMessage.Parameters.put("level", 5);
          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The controller turned ON from OFF");
          break;
        }
      }
      if(sw.ContainsMessageOfType(MessageDefinitions.OFF))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.OFF);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
          outgoingMessage = MessageDefinitions.CreateOFF();
          //Set outgoing variables

          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.off;
          System.out.println("The controller turned OFF from OFF");
          break;
        }
      }
      // This method is perfect :) Should be used for auto-gen
      if(sw.ContainsMessageOfType(MessageDefinitions.ONLEVEL))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.ONLEVEL);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
          outgoingMessage = MessageDefinitions.CreateONLEVEL();
          //Set outgoing variables
          outgoingMessage.Parameters.put("level", incomingMessage.Parameters.get(
              "level"));
          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The controller turned ONLEVEL from OFF");
          break;
        }
      }
    case on:
      if(sw.ContainsMessageOfType(MessageDefinitions.OFF))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.OFF);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
          outgoingMessage = MessageDefinitions.CreateOFF();
          //Set outgoing variables

          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.off;
          System.out.println("The controller turned OFF from ON");
          break;
        }
      }
      if(sw.ContainsMessageOfType(MessageDefinitions.ONLEVEL))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.ONLEVEL);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
```

```java
          outgoingMessage = MessageDefinitions.CreateONLEVEL();
          //Set outgoing variables
          outgoingMessage.Parameters.put("level", incomingMessage.Parameters.get(
              "level"));
          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The controller changed level");
          break;
        }
      }
      if(sw.ContainsMessageOfType(MessageDefinitions.ON))
      {
        MessageDefinition incomingMessage = sw.GetBufferMessageByType(
            MessageDefinitions.ON);
        if(true) // <-- Condition!
        {
          // Set attributes!

          //For each outgoing (shall be hard coded):
          MessageDefinition outgoingMessage;
          // Block: Creaste outgoing
          outgoingMessage = MessageDefinitions.CreateONLEVEL();
          //Set outgoing variables
          outgoingMessage.Parameters.put("level", 5);
          //Send messages to all receivers
          for(Connection c : wp.Connections)
          {
            Port otherPort = c.getOtherEnd(wp);
            if(otherPort != null)
            {
              if(c.bus != null)
              {
                c.bus.addToBus(outgoingMessage,otherPort);
              }
              else
              {
                otherPort.addToBuffer(outgoingMessage);
              }
            }
          }
          //Blok end
          //Clean up etc
          sw.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The controller turned ON from ON");
          break;
        }
      }
    default:
      System.out.println("Controller did nothing in this step");
      break;
    }
  }

}
```

Listing 4: Deployment.java

```java
package deployment;

import java.util.ArrayList;
import java.util.List;

import component.*;

import bus.Bus;
import connection.Connection;


public class Deployment {

  public List<IComponent> Components;
  public List<IComponent> Busses;
  public Wiper frontWiper = new Wiper("Front Wiper");
  public Wiper rearWiper = new Wiper("Rear Wiper");
  public Controller c = new Controller("c");
  public Bus auxBus = new Bus(3,"auxBus");

  public Deployment()
  {
    Components = new ArrayList<IComponent>();
    Busses = new ArrayList<IComponent>();
    addComponents();
    addBusses();
    connectPorts();
  }


  private void addBusses()
  {
    Busses.add(auxBus);
  }


  private void connectPorts()
  {
    Connection cTofrontWiper = new Connection(c.wp,frontWiper.in,auxBus);
    c.wp.Connections.add(cTofrontWiper);
    frontWiper.in.Connections.add(cTofrontWiper);

    Connection cTorearWiper = new Connection(c.wp,rearWiper.in,auxBus);
    c.wp.Connections.add(cTorearWiper);
    rearWiper.in.Connections.add(cTorearWiper);
  }

  private void addComponents() {
    Components.add(frontWiper);
    Components.add(rearWiper);
    Components.add(c);
  }

}
```

Listing 5: IComponent.java

```
package component;

public interface IComponent {

  public void step();

}
```

Listing 6: MessageDefinition.java

```
package messages;

import java.util.Map;

public class MessageDefinition {

  public String Name;
  public Map<String, Integer> Parameters;

  public MessageDefinition(String Name, Map<String, Integer> Parameters)
  {
    this.Name = Name;
    this.Parameters = Parameters;
  }

  public void setParameterValue(String name, int value) {
    if(Parameters.containsKey(name))
    {
```

```java
      Parameters.put(name, value);
    }
  }

}
```

Listing 7: MessageDefinitions.java

```java
package messages;

import java.util.HashMap;
import java.util.Map;

public class MessageDefinitions {

  public static final String ON = "on";
  public static final String OFF = "off";
  public static final String ONLEVEL = "onLevel";

  private static MessageDefinitions INSTANCE;

  public static MessageDefinition CreateONLEVEL()
  {
    Map<String,Integer> params = new HashMap<String, Integer>();
    params.put("level", 0);
    MessageDefinition message = new MessageDefinition(ONLEVEL,params);
    return message;
  }

  public static MessageDefinition CreateON()
  {
    Map<String,Integer> params = new HashMap<String, Integer>();
    MessageDefinition message = new MessageDefinition(ON,params);
    return message;
  }

  public static MessageDefinition CreateOFF()
  {
    Map<String,Integer> params = new HashMap<String, Integer>();
    MessageDefinition message = new MessageDefinition(OFF,params);
    return message;
  }

}
```

Listing 8: Port.java

```java
package port;

import java.util.ArrayList;
import java.util.List;

import component.IComponent;

import messages.MessageDefinition;


import connection.Connection;


public class Port {

  public IComponent Component;
  public int BufferSize;
  public List<MessageDefinition> In;
  public List<MessageDefinition> Out;
  public List<Connection> Connections;
  private List<MessageDefinition> Buffer;


  public Port()
  {
    Connections = new ArrayList<Connection>();
    In = new ArrayList<MessageDefinition>();
    Out = new ArrayList<MessageDefinition>();
    Buffer = new ArrayList<MessageDefinition>();
  }

  public void addToBuffer(MessageDefinition md)
  {
    if(Buffer.size() < BufferSize)
    {
      Buffer.add(md);
    }
    else
    {
      System.out.println("Message " + md.Name + " has been thrown away as the
          buffer of the port is full");
    }
  }

  public MessageDefinition GetBufferMessageByType(String type)
  {
    for(MessageDefinition m : Buffer)
    {
      if(m.Name.equals(type))
        return m;
    }
    return null;
  }

  public boolean ContainsMessageOfType(String type)
  {
    return GetBufferMessageByType(type) != null;
  }

  public void RemoveBufferMessageByType(MessageDefinition type)
  {
    Buffer.remove(type);
  }
}
```

Listing 9: Simulator.java

```java
package simulator;

import component.IComponent;

import messages.MessageDefinition;
import messages.MessageDefinitions;

import deployment.Deployment;


public class Simulator {

  public Deployment deployment;
  public static int step = 0;

  public Simulator()
  {
    deployment = new Deployment();
  }

  public void Step()
  {
    System.out.println("--------STEP " + step++ + "--------");
    for(IComponent component : deployment.Components)
    {
      component.step();
    }
    for(IComponent bus : deployment.Busses)
    {
      bus.step();
    }
  }

  public static void main(String[] args)
  {
    Simulator sim = new Simulator();

    sim.Step();

    sim.deployment.c.sw.addToBuffer(MessageDefinitions.CreateON());
    sim.Step();
    sim.Step();
    sim.Step();

    MessageDefinition onLevelMsg = MessageDefinitions.CreateONLEVEL();
    onLevelMsg.Parameters.put("level", 10);
    sim.deployment.c.sw.addToBuffer(onLevelMsg);

    sim.Step();
    sim.Step();
    sim.Step();

    MessageDefinition onLevelMsg2 = MessageDefinitions.CreateONLEVEL();
    onLevelMsg.Parameters.put("level", 8);
    sim.deployment.c.sw.addToBuffer(onLevelMsg2);

    sim.Step();
    sim.Step();
    sim.Step();

    MessageDefinition onLevelMsg3 = MessageDefinitions.CreateOFF();
    sim.deployment.c.sw.addToBuffer(onLevelMsg3);

    sim.Step();
    sim.Step();
    sim.Step();
    sim.Step();
    sim.Step();

  }

}
```

Listing 10: Wiper.java

```java
package component;

import messages.MessageDefinition;
import messages.MessageDefinitions;
import port.Port;

public class Wiper implements IComponent
{
  public int level = 0;

  public String name;

  public Port in;

  public boolean isHardware = true;

  private enum States
  {
    off,on
  }

  private States currentState;

  public Wiper(String name)
  {
      currentState = States.off;

      this.name = name;
      in = new Port();
      in.BufferSize = 2;
      in.In.add(MessageDefinitions.CreateOFF());
      in.In.add(MessageDefinitions.CreateONLEVEL());
  }

  public void step()
  {
    switch(currentState) {

    case off:
      if(in.ContainsMessageOfType(MessageDefinitions.OFF))
      {
        MessageDefinition incomingMessage = in.GetBufferMessageByType(
            MessageDefinitions.OFF);
        if(true) // <-- Condition!
        {
          // Set attributes!
          //For each outgoing (shall be hard coded):
          // Block: Creaste outgoing
          //Set outgoing variables
          //Send messages to all receivers
          //Blok end
          //Clean up etc
          in.RemoveBufferMessageByType(incomingMessage);
          currentState = States.off;
          System.out.println("The " + this.name + " turned OFF from OFF");
          break;
        }
      }
      if(in.ContainsMessageOfType(MessageDefinitions.ONLEVEL))
      {
        MessageDefinition incomingMessage = in.GetBufferMessageByType(
            MessageDefinitions.ONLEVEL);
        if(true) // <-- Condition!
        {
          // Set attributes!
          level = incomingMessage.Parameters.get("level");
          //For each outgoing (shall be hard coded):
          // Block: Creaste outgoing
          //Set outgoing variables
          //Send messages to all receivers
          //Blok end
          //Clean up etc
          in.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The " + this.name + " turned ON from OFF");
          break;
        }
      }
    case on:
      if(in.ContainsMessageOfType(MessageDefinitions.OFF))
      {
        MessageDefinition incomingMessage = in.GetBufferMessageByType(
            MessageDefinitions.OFF);
        if(true) // <-- Condition!
        {
          // Set attributes!
          //For each outgoing (shall be hard coded):
          // Block: Creaste outgoing
          //Set outgoing variables
          //Send messages to all receivers
          //Blok end
          //Clean up etc
          in.RemoveBufferMessageByType(incomingMessage);
          currentState = States.off;
          System.out.println("The " + this.name + " turned OFF from ON");
          break;
        }
      }
      if(in.ContainsMessageOfType(MessageDefinitions.ONLEVEL))
      {
        MessageDefinition incomingMessage = in.GetBufferMessageByType(
            MessageDefinitions.ONLEVEL);
        if(level < incomingMessage.Parameters.get("level")) // <-- Condition!
        {
          // Set attributes!
          level = incomingMessage.Parameters.get("level");
          //For each outgoing (shall be hard coded):
          // Block: Creaste outgoing
          //Set outgoing variables
          //Send messages to all receivers
          //Blok end
          //Clean up etc
          in.RemoveBufferMessageByType(incomingMessage);
          currentState = States.on;
          System.out.println("The " + this.name + " turned ON from ON");
          break;
        }
        //DEBUG ONLY
        else
        {
          System.out.println("Level to low");
        }
      }
    default:
      System.out.println("The " + this.name + " did nothing in this step");
      break;
    }
  }
}
```