



Diplomarbeit

Automated Generation of JIAC AgentBeans from BPMN Diagrams

Fachbereich *Agententechnologien in betrieblichen Anwendungen
und der Telekommunikation (AOT)*

Prof. Dr.-Ing. habil. Sahin Albayrak
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

Vorgelegt von: **Petrus Setiawan Tan**

Betreuer: Dipl.-Inform. Tobias Küster

Petrus Setiawan Tan
Matrikelnummer: 213933
Stettiner Str. 9
10243 Berlin

Die selbstständige und eigenhändige Anfertigung dieser Diplomarbeit versichere ich an Eides statt.

Ort, Datum

Petrus Setiawan Tan

Abstract

Zusammenfassung

Acknowledgement

Inhaltsverzeichnis

1	Introduction	1
1.1	Motivation	1
1.1.1	Model Driven Engineering (MDE)	1
1.1.2	MDE in Multi-agent systems	2
1.2	Goals	2
2	Background	3
2.1	JIAC(Java-based Intelligent Agent Componentware)	3
2.2	BPMN(Business Process Modelling Notation)	4
2.3	VSDT(Visual Service Design Tool)	7
2.3.1	BPMN Editor	7
2.3.2	Transformation Framework	7
2.3.3	Existing Transformation to JiacV	8
2.4	JET(Java Emitter Templates)	9
2.4.1	JET-Templates	9
2.4.2	2 different JET-Versions	10
3	State of the art	13
3.1	BPMN to BPEL	13
3.2	BPMN to Agents	13
3.2.1	JADE (Java Agent DEvelopment Framework)	13
3.2.2	WADE (Workflows and Agents Development Environment) . .	13
3.2.3	Wade-Workflow	13
4	Mapping BPMN to Jiac AgentBeans	15

5	Implementation of the transformation	21
5.1	Transformation Stucture	21
5.2	AgentBean Model	22
5.3	JET-Transformation	23
5.4	Merging generated and manually edited code	23
6	Examples	25
6.1	Simple Flow	25
6.2	Time Event Handler	25
7	Conclusion	27
	Literaturverzeichnis	30

Abbildungsverzeichnis

2.1	Jiac Basic concepts and their structural relationships [4]	4
2.2	A simple Business Process Diagramm	4
2.3	BPMN Event types. From left to right: Start Event, Intermediate Event, End Event	5
2.4	Examples of event subtypes. From left to right: Multiple, Link, Message, Rule, Timer	5
2.5	Empty Pool with 2 Lanes	6
2.6	VSDT - Editor View	7
2.7	Essential classes of the transformation framework, including the BPEL case.[14]	8
2.8	JET transformation steps.	9
5.1	Transformation stages	21
5.2	AgentBean - Metamodel	22
6.1	Process with event handler	25

1. Introduction

In this chapter, we will start by introducing the motivation and the goals of this work.

1.1 Motivation

A common problem in the software engineering is the communication gaps between the business people as a client and the IT world. Communicating through models, normally consists of graphical sketches, especially using those which are commonly used in the client's domain such as the BPMN, clearly is a solution in bridging these gaps.

The main motivation of this work is to present a solution that will simplify the software development process and help bringing the concepts of software agents and multi agent systems, a topic which has been researched for decades, to gain more acceptance in the industry by using the benefits of Model Driven Engineering(MDE).

1.1.1 Model Driven Engineering (MDE)

Over the past few years more and more software developers have been adopting the principle of *Model Driven Engineering*(MDE) where they no longer focus on writing programs but on creating a set of models which define the software. By modelling the software, the developer creates documents that provides an abstract view of the software system, independently from the platform or a specific programming language, making it understandable for non experts i.e. the stakeholders as well as applicable in different platforms.

A significant number of the so called CASE (Computer Aided Software Engineering) tools have been developed to support this methodology. Beside providing support in creating and editing the models, most of these CASE tools are also equipped with transformation features that allows us to transform the model into text or even executable Programs, thus increasing efficiency in the software development process.

We can say that the real benefits of MDE lies in the transformation. By providing a mapping between the model and the code, we can create standardized programs, accelerate development time and minimize faults in writing the code.

1.1.2 MDE in Multi-agent systems

Back in 2007, an MDE-approach has been made in order to bridge the gap between the industry and the multi-agent systems. As a result, a CASE-tool called the ***Visual Service Design Tool (VSDT)*** was developed by Tobias Küster in scope of his Diploma Thesis, which provides the transformation of BPMN (Business Process Modelling Notation) to BPEL (*Business Process Execution Language*) and JIAC (Java-based Intelligent Agent Componentware) framework. This tool allows agents to be designed using a business process diagram, a model which has already been manifested in the industry, thus most people from the industry will be able to design software agents. In the scope of this work, a plugin to VSDT will be developed to enrich it's transformation feature with a code generator that will transform BPMN models into executeable Java Code, or JIAC AgentBeans to be more specific.

1.2 Goals

The main goal of this work is to develop an eclipse plugin as an extension to VSDT to enrich its transformation features with a new transformation from BPMN to Java Code or JIAC AgentBeans to be more specific. Because it is nearly impossible to put all implementation details into the model, and to anticipate the possibility, that the generated code will be edited manually, considerations has to be made, such that conflicts should not occur when the transformation is called to a code that has been edited manually.

2. Background

In this chapter, we will discuss the backgrounds of the transformation that should be developed. We will start with JIAC (the target of the transformation), BPMN (the model that is being used), followed by VSDT (the existing modeling and transformation framework to JIAC that should be extended), and JET (the technology that will help us to implement the transformation).

2.1 JIAC(Java-based Intelligent Agent Component-ware)

JIAC is a Java-based agent architecture and framework that was developed to simplify the development of software agents. The framework supports the entire software development process of a software agent system by providing features such as FIPA compliant communication, Believe-Desire-Intention (BDI) reasoning, strong migration, web-service connectivity and many others. It also provides high security (Common Criteria EAL3, certified by the Federal Office for Information Security of Germany, BSI) and advanced accounting mechanisms, making it suitable for the use in industrial and commercial applications.

JADL

With its core component JADL(*Jiac Agent Description Language*), Jiac also provides an agent programming language. With JADL, the agents *plan elements*, *rules*, *ontologies* and *services* can be described. JADL is based on three predicate logic which allows the values *true*, *false* and *unknown*, which makes it suitable for open world problems in unknown environments.

In Figure 2.1, we can see the typical structure of a JIAC Application, which consists of AgentNodes, Agents, AgentBeans. An *AgentNode* is a Java VM where the runtime infrastructure for agents, such as discovery services, white and yellow pages services, as well as communication infrastructure, are provided. A JIAC application might consist of multiple AgentNodes (distributed application). With the so-called

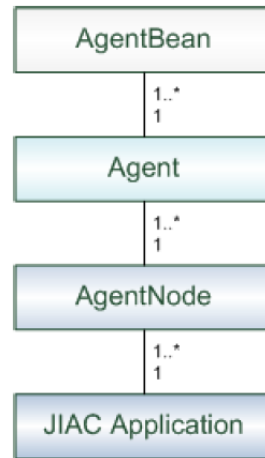


Figure 2.1: Jiac Basic concepts and their structural relationships [4]

AgentNodeBeans, one can extend the AgentNode with additional components.

Each AgentNode may run several *Agents*. Agents provide services to other agents and comprise lifecycle, execution cycle and a memory. An agent can use infrastructure services in order to find other agents, to communicate to them and to use their services. Skills and abilities of the agent can be extended by so-called AgentBeans. *AgentBean* is the mean to implement the functionality of an Agent, in other words, the abilities of an Agent are defined in the AgentBeans attached to it. They are plugged into agents and provide services (so-called Actions) to other agents.

2.2 BPMN(Business Process Modelling Notation)

BPMN [5] is a standard Notation for modelling business processes, initially published by the BPMI which is later adopted by the OMG(Object Management Group). A business process diagramm (as seen in Figure 2.2) can be compared to UML's activity diagram.

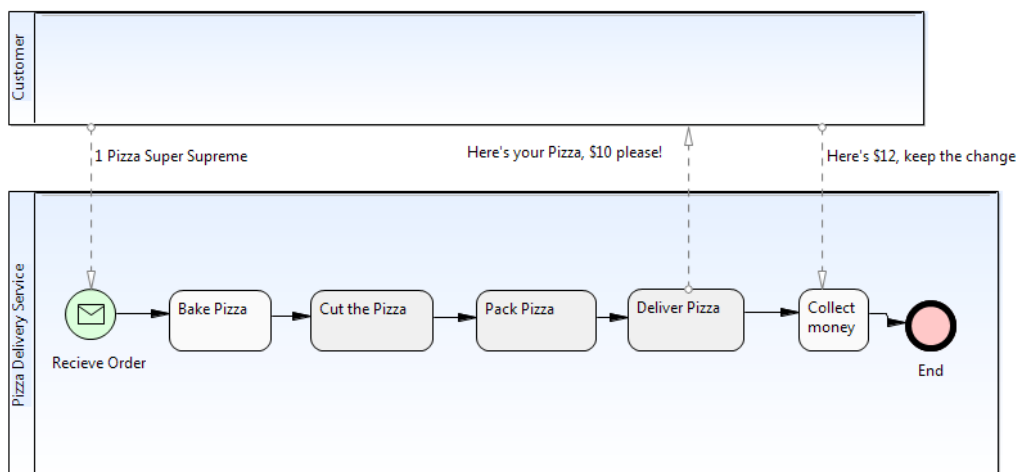


Figure 2.2: A simple Business Process Diagramm

BPMN was made to provide a notation that is understandable by all business users, creating a bridge for the gap between the business process design and the process implementation. Besides providing an easy to read graphical notations, each BPMN elements are also equipped with additional attributes (so called properties), which are hidden from the diagram and provides the informations needed for automated code generation. With the mapping of BPMN to Agents, we hope to be able to increase the spreading of the multi agent systems in the business world.

BPMN elements can be categorized in five basic groups of elements[5] :

- Flow Objects
- Data
- Connecting Objects
- Swimlanes
- Artifacts

Flow Objects includes Events, Activities and Gateways. These elements are the most important in BPMN, and they are held in a lane.

An *Event* describes something that happens during the course of a process. It is divided into Start Event, Intermediate Event and End Event (see Figure 2.3).



Figure 2.3: BPMN Event types. From left to right: Start Event, Intermediate Event, End Event

BPMN Events are further divided into subtypes according to the type of the event's trigger(for start and intermediate events) or result(for end events). The event figure(see Figure 2.2) are drawn with different icon in the middle, according to the trigger.



Figure 2.4: Examples of event subtypes. From left to right: Multiple, Link, Message, Rule, Timer

An *Activity* describes something that is done during a process. It is divided into *Tasks* (Atomic Activities) and *Sub Processes* (Composite Activities).

Gateways are used to define all kinds of splitting and merging behavior. It's semantics depends on the dimension of it's incoming and outgoing Sequence Flows.

Data is represented by the four elements:

- Data Objects
- Data Inputs
- Data Outputs
- Data Stores

Data Object describes information that is needed by an activity or what they produce. It can represent a singular object or a collection of objects. *Data Inputs* and *Data Outputs* describe the same information for Processes. *Data Stores* describe the location where information, that persists beyond the scope of a process are stored.

With *ConnectingObjects*, *FlowingObjects* can be connected to each other or with other informations. There are 3 different kinds of Connecting Objects:

- *Sequence Flows* - represent Flow Control, used for connecting FlowObjects within a Pool in the order of execution.
- *Message Flows* - represent Messages being exchanged exclusively between Pools.
- *Associations* - mainly used for documentation, in example between Flow Objects and a Text annotation.

Swimlanes are divided into *Pools* and *Lanes*. Each Pool represents one Participant in the business process, while Lanes are used to partition a Pool, in example to model different Departments of an Institution. Figure 3 shows an empty Pool with 2 Lanes.

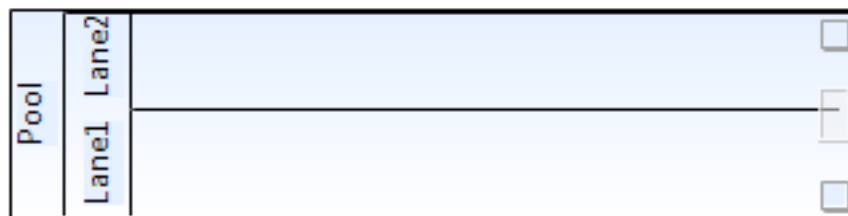


Figure 2.5: Empty Pool with 2 Lanes

Artifacts are additional Information about the Process. It is mainly used for documentation purpose. The Current set of Artifacts includes:

- Text Annotation

- Group

To provide a rough overview on how Agent technology can support the implementation of Business Processes let us take a look on this mapping example of BPMN elements to Agents. A Pool in a Business Process Diagram can represent an Agent, which are able to communicate with other agents (another pool) through messages (represented with the BPMN MessageFlows). Agents can react to Events. A detailed mapping needed for implementing the transformation will be discussed in chapter 4

2.3 VSDT(Visual Service Design Tool)

The VSDT is a CASE Tool, developed to support the idea of Process Oriented Agent Engineering, where Agents are designed by defining use cases and processes described with the BPMN. It's features include the BPMN editor, process structure view, model validation, import of existing web services, transformations to BPEL, and JIAC and many more.

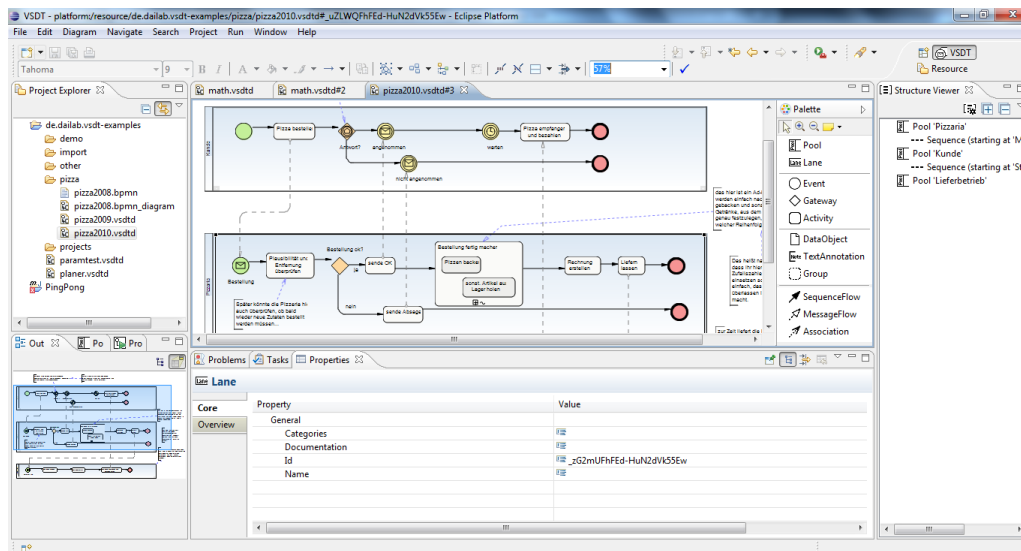


Figure 2.6: VSDT - Editor View

2.3.1 BPMN Editor

The BPMN Editor in VSDT was created using eclipse's GMF (Graphical Modeling Framework)

2.3.2 Transformation Framework

Since it's initial development, VSDT's transformation framework is designed to be extensible and reusable. This allows the development of a new transformation to be easier. For this purpose the transformation process is subdivided into several stages:

1. *Validation*: Validate the input model.
2. *Normalisation*: Prepare the input model for transformation.

3. *Structure Mapping*: Convert the input model to a block-like structure.
4. *Element Mapping*: Perform the actual mapping, create target model.
5. *Clean Up*: Remove redundancies, improve readability, etc.

Due to the fact that the validation, normalisation and structure mapping are mostly independent from the target language, the standard mapping provided for these stages are reusable, which makes it possible to implement a new transformation by specifying the element mapping only. Figure 2.7 shows the UML Class Diagram of the transformation framework with the example transformation to BPEL.

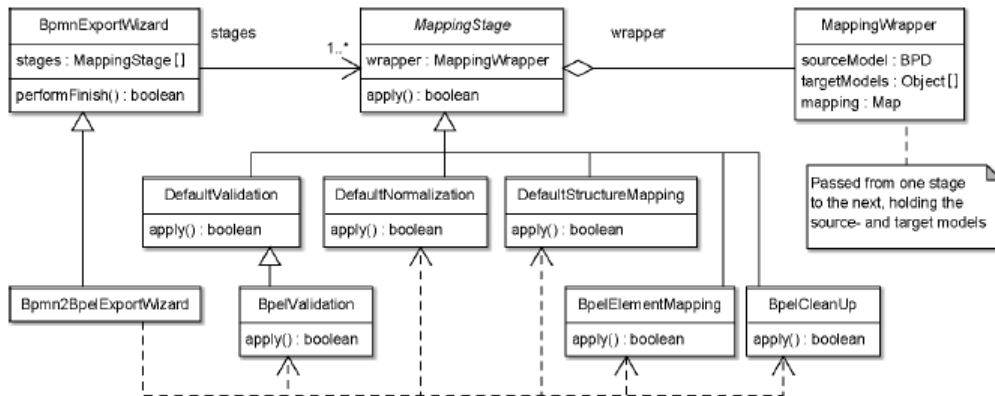


Figure 2.7: Essential classes of the transformation framework, including the BPEL case.[14]

2.3.3 Existing Transformation to Jiacy

At the moment, the VSDT is already equipped with a transformation of BPMN to Jiacy, which generates the JADL(Jiacy Agent Description Language). The transformation to Jiacy AgentBeans is in no way a replacement to the existing transformation. Instead both transformations shall complement each other as the products of both transformations have their own advantages, some of which we can find in the following table:

Advantages of:	
JADL	AgentBeans
can be deployed to a running jiacy application	Written fully in Java, therefore developer friendly. Java is more powerful than JADL. Better performance because no parser are involved.

2.4 JET(Java Emitter Templates)

JET[12] is a code generating framework, developed as a part of the Eclipse Modelling Project [7]. Using the so-called templates, one can transform a model into various type of text, from a simple plain text up to text containing html, xml or java code.

The transformation process is done in two steps(see figure 2.4). First, the JET-Builder will translate the template file into a Java class holding a generate method. Then we can create an instance of the template class and call it's generate method to get the result String which we can process further for example writing it into a file.

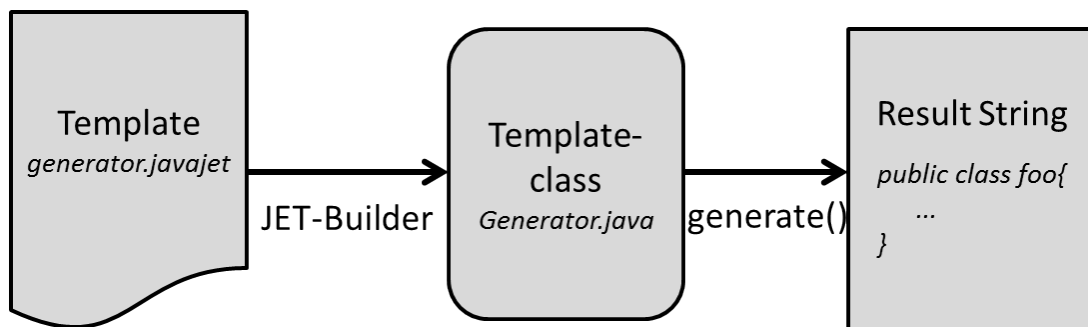


Figure 2.8: JET transformation steps.

2.4.1 JET-Templates

JET-Templates uses a JSP-like syntax which makes it easy to write and understand. The following listing shows a simple example of a JET-Template that generates an XML:

```

1 <% @ jet package="generator" imports="java.util.*" class="StudentListGenerator" %>
2 <?xml version="1.0" encoding="UTF-8"?>
3 <% List<String> elementList = (List<String>) argument; %>
4 <class>
5   <% for (Iterator i = elementList.iterator(); i.hasNext(); ) { %>
6     <student><%=i.next() %></student>
7   <% } %>
8 </class>
  
```

Listing 2.1: a simple JET-Template

A Jet-Template starts with the so called **jet-directive**. It contains informations for the JET-Builder, for example the name of the translated Java class(also called the template class), the package in which the template class should be placed into and a list of classes that should be imported by the template class.

The JET-Builder will then translate this template into the Java Class `generator.StudentListGenerator`:

```

1 package generator;
2
3 import java.util.*;
4
5 public class StudentListGenerator
6 {
7     protected static String nl;
8     public static synchronized StudentListGenerator create(String lineSeparator)
9     {
10         nl = lineSeparator;
11         StudentListGenerator result = new StudentListGenerator();
12         nl = null;
13         return result;
14     }
15
16     public final String NL = nl == null ? (System.getProperties().getProperty("line.separator")) : nl;
17     protected final String TEXT_1 = " " + NL + "<?xml version='1.0' encoding='UTF-8'?">";
18     protected final String TEXT_2 = NL + "<class>" + NL + "\t ";
19     protected final String TEXT_3 = NL + "    <student>";
20     protected final String TEXT_4 = "</student>";
21     protected final String TEXT_5 = NL + "<class>";
22
23     public String generate(Object argument)
24     {
25         final StringBuffer stringBuffer = new StringBuffer();
26         stringBuffer.append(TEXT_1);
27         List<String> elementList = (List<String>) argument;
28         stringBuffer.append(TEXT_2);
29         for (Iterator i = elementList.iterator(); i.hasNext(); ) {
30             stringBuffer.append(TEXT_3);
31             stringBuffer.append(i.next());
32             stringBuffer.append(TEXT_4);
33         }
34         stringBuffer.append(TEXT_5);
35         return stringBuffer.toString();
36     }
37 }

```

Listing 2.2: the translated Java-Class

A set of JET templates is called transformation. It is possible to build this transformation with a main template which acts as a visitor and runs through the model, and this main template will then use other templates which handle a specific element of the model. For example, in UML to Java transformation you can have special templates that handles the package, class, variables and methods.

2.4.2 2 different JET-Versions

There are currently 2 different JET-Versions in the Eclipse Modelling Project. The older Version allows us to generate text with an Object as argument. In the template, one can type cast the argument variable into the class of our model.

The Java Class translated from the JET-Template will have the generate method:

```

1 public String generate(Object argument){
2     ...
3 }

```

Listing 2.3: generate method translated from the JET-Template

This JET-Version is effective if the model we want to generate the text from is a Java object. We get a String as a result which we can write into a File using the Java-IO or even Eclipse API.

In the updated version of JET, also called JET2, some workspace and java related “tag-libraries“ are provided, enabling us to do the transformation without using the Java and Eclipse API. Unfortunately, in this Version the model has to be an xml-File. To directly transform the BPMN directly from its xml-Representation will be harder and the template will be confusing, therefore a decision has been made to implement the mapping using the existing transformation framework where an intermediate model class of the AgentBean will be created, and then transform this intermediate model into Java code using the older version of the JET Transformation. More details on the implementation will be discussed in chapter 5.

3. State of the art

3.1 BPMN to BPEL

3.2 BPMN to Agents

A similar approach (designing agents behaviour with processes and transforming it into Java Code) has been developed by the Telecom Italia with their JADE-extension called WADE (Workflows and Agents Development Environment). While Jade was developed to simplify the implementation of Software Agents, WADE extended JADE with a workflow engine, making it possible to create Agents that executes tasks defined as workflows.

3.2.1 JADE (Java Agent DEvelopment Framework)

JADE is an application framework and a middleware written in Java, which support the development of software agents. The framework provides distributed runtime environments, agent and behaviour abstractions as well as communications between agents and discovery mechanisms. We can say that it's role is very similar to JIAC.

3.2.2 WADE (Workflows and Agents Development Environment)

WADE is an extension to JADE, which enrich the application framework with a workflow engine. The WorkflowEngineAgent extends the JADE basic Agent class with an ability to execute workflows represented in a WADE specific formalism.

A WADE Workflow is actually a Java Class, thus it can be edited and managed as Java classes and can contain pieces of code which is needed to implement the process. With WOLF, a development environment that comes with WADE, developers can edit the Workflow graphically as well as textually. The code view and the graphical view of the workflow are kept in sync.


3.2.3 Wade-Workflow


Despite having all the advantages of a Java code, the WADE workflow is rather simple and not so expressive as the BPMN. Similarity to the agent technology such as events and communication flow are also missing in the workflow.

4. Mapping BPMN to Jiac AgentBeans

The element mapping from BPMN to Jiac AgentBeans is created based on the existing mapping to JiacV - JADL script. In comparison to a JADL script, an AgentBean is written completely in Java, enabling more possibilities in mapping concepts such as intermediate Event handling. This chapter will provide a tabular overview of the mapping.

Business Process Diagram

Business Process Diagram	
Pool 	<p>A pool is currently mapped to an Agent Bean extending the AbstractMethodExposingBean. The name of the bean is derived from the pool name and the name of the business process diagram it is contained in.</p> <pre>public class Pool_Process extends AbstractMethodExposingBean{ //... }</pre>

<p>Process</p> 	<p>The process content in a pool is mapped to a Workflow Method in the generated AgentBean. Depending on the type of the StartEvent, an Action might be exposed, enabling this method to be invoked as a service. This method will contain a Script which is generated from the flowObjects contained in the process.</p> <pre> public class Pool_GetTotalPrice extends AbstractMethodExposingBean{ public final static String ACTION_GETTOTALPRICE = "pool.Pool_GetTotalPrice#getTotalPrice"; @Expose(name = ACTION_GETTOTALPRICE, scope = ActionScope.GLOBAL) public double getTotalPrice(double nettPrice, double taxRate){ //...Workflow content calculateTax(); calculateTotalPrice(); return result; } //... } </pre>
<p>Lane</p>	<p>Lanes will not be mapped in this mapping</p>

Events

For intermediate Events, an EventHandlerThread will be generated and started. If the intermediate Event is attached to an activity, the activity will also be started as a Thread.

Start Events	
Timer	The generated process will be started in the execute() method of the JIAC AgentBean
Message	If the implementation is a Service, an action will be created, and the process can be started through a service call (invoke). If the implementation is a Message Channel, a Space Observer will be attached to the agents memory, and the process will start as soon as a message is recieved.
Rule	Rule start events are not regarded in the current mapping
Link	
Multiple	If a pool has multiple start events, each start event will be mapped according to it's respective trigger
Intermediate Events	
Rule	not regarded in the current mapping
Timer	A TimerEventHandler will be created and started.
Message	A MessageEventHandler will be created and started.
Link	...
Multiple	each event will be mapped according to it's respective trigger
Cancel	...
Compensate	...
Error	If attached to an activity, the method generated from the activity will be called in a try-catch block
End Events	
Message	If implementation is Message Channel, a message with the payload will be written to the channel
Link	
Multiple	
Cancel	
Compensate	
Error	The generated service will throw an exception
Terminate	

Activities

Basically, an Activity-method will be created for each activity, and then a script calling this method will be added into the Workflow-method. This way, each activity can have their own scope of variables.

Activity	
Standard Loop	...
Multi Instance Loop	...
With Event Handler	the method generated will be started parallel to an EventHandlerThread. When the activity is completed, the EventHandlerThread will be terminated. In contrary, if the Event is triggerred before the activity is completed, the Thread performing the activity will be interrupted.
Task	
Manual	manual task will be ignored
Recieve	...
Send	...
Service	service tasks will be mapped into a service call(invoker)
Script	the given script will be added into the activity method. The script has to be written in java or the resulting code will have syntax error(not checked)
Reference	...
User	...
Subprocess	
Embedded	
Reference	
Independent	
Transaction	

Gateways

Gateway	
XOR Data(Loop)	
XOR Data(Block)	
XOR Event	
AND	Each branch will be wrapped in a paralel execution
OR	
COMPLEX	

Other Elements

Connections	
Sequence Flow	
Message Flow	
Association	
Artifact	
Data Object	
Text Annotation	text annotations can be added as a comment block in the code
Group	
Custom Artifact	

5. Implementation of the transformation

In this chapter, some details of the transformation implementation will be presented.

5.1 Transformation Structure

As mentioned before in section 2.3, the transformation process in the VSDT is divided into 5 stages. We've also mentioned that the default validation and structure mapping provided by the transformation framework are reusable. For the implementation of the new transformation, the framework's `DefaultBpmnValidator` and `BPMN2StrucBPMNTransformation` are being reused, as we can see in Figure 5.1.

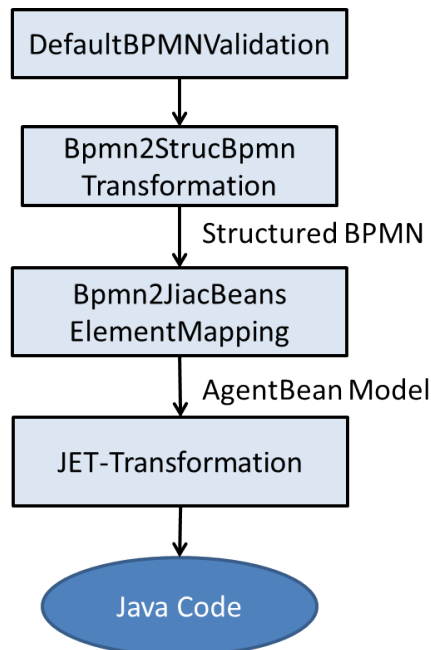


Figure 5.1: Transformation stages

The Element mapping stage is implemented on the basis of the existing `Bpmn2JiacV-ElementMapping`. It takes a structured Bpmn as a model and transforms each pool

Each script implements the method `public String toJavaCode()` which returns the java code representation of the script. In the following listing we can see the implementation of the method in the `IfThenElse` class:

```

1  public String toJavaCode() {
2      String code = "";
3      code += "if("+condition+"){\n";
4      if(thenBranch!=null){
5          BufferedReader reader = new BufferedReader(new StringReader(thenBranch.
6              toJavaCode()));
7          try{
8              String line = reader.readLine();
9              while(line!=null){
10                 if(!line.equals("")) code += "\t"+line+"\n";
11                 line = reader.readLine();
12             }
13         }catch(IOException e){
14             code += "\t//Error occured while reading if branch\n";
15         }
16     }
17     code+="}\n";
18     if (elseBranch!=null){
19         code+="else{\n";
20         BufferedReader reader = new BufferedReader(new StringReader(elseBranch.
21             toJavaCode()));
22         try{
23             String line = reader.readLine();
24             while(line!=null){
25                 if(!line.equals("")) code += "\t"+line+"\n";
26                 line = reader.readLine();
27             }
28         }catch(IOException e){
29             code += "\t//Error occured while reading else branch\n";
30         }
31     }
32     code+="}\n";
33     return code;
34 }

```

Listing 5.1: `toJavaCode()` implementation in the `IfThenElse` class

You might notice, that this method is also responsible for the text formatting because this method will be used by the JET-Transformation and the result will then be written in a *.java File. Therefore, as you can see in line 7-11 and 21-25, a tab are added in front of each line in the then and else branch.

The role of MDE in the Implementation

The benefits of Model Driven Engineering are also found during the implementation of the AgentBean model. As we can see in Figure 5.2, it is created graphically using eclipse's Ecore Tools - Ecore Diagramm. With the help of the EMF Generator each element of the model can be easily generated into JavaCode including a Factory class that can be used to instantiate an object of each generated class. This way, we only have to implement the method `toJavaCode()` for each newly added script, everything else are generated automatically.

5.3 JET-Transformation

5.4 Merging generated and manually edited code

One of the challenge in generating java code is how to handle code that has been manually edited.

6. Examples

In this chapter we can see a selection of BPMN-examples and how the resulting Code of the transformation look like.

6.1 Simple Flow

6.2 Time Event Handler

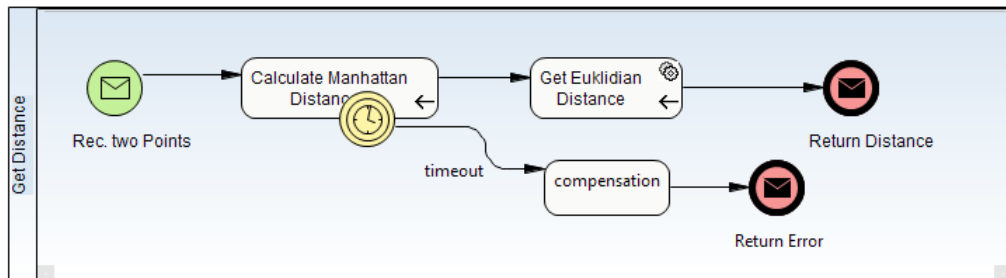


Figure 6.1: Process with event handler

7. Conclusion

Literaturverzeichnis

- [1] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade: A software framework for developing multi-agent applications. lessons learned. *Inf. Softw. Technol.*, 50:10–21, January 2008.
- [2] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the Practical Applications of Intelligent Agents*, 1999.
- [3] Giovanni Caire, Danilo Gotta, and Massimo Banzi. Wade: a software platform to develop mission critical applications exploiting agents and workflows. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, AAMAS '08, pages 29–36, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
- [4] CC ACT, DAI-Labor, TU-Berlin. *JIAC - Java Intelligent Agent Componentware*, 06 2010. Version 5.1.0 Manual.
- [5] Object Management Group. Bpmn specification v2.0. <http://www.omg.org/spec/BPMN/2.0/>.
- [6] Benjamin Hirsch, Thomas Konnerth, and Axel Heer. Merging agents and services the jiac agent platform. In Amal El Fallah Seghrouchni, Jrgen Dix, Mehdi Dastani, and Rafael H. Bordini, editors, *Multi-Agent Programming*., pages 159–185. Springer US, 2009. 10.1007/978-0-387-89299-3_5.
- [7] Eclipse Modelling Project Homepage. <http://www.eclipse.org/modeling/>.
- [8] JADE Homepage. <http://jade.tilab.com/index.html>.
- [9] JIAC Homepage. <http://jiac.de/?id=35>.
- [10] VSDT Homepage. <http://jiac.de/?id=30>.
- [11] WADE Homepage. <http://jade.tilab.com/wade/index.html>.
- [12] JET. <http://www.eclipse.org/modeling/m2t/?project=jet#jet>.
- [13] Thomas Konnerth, Benjamin Hirsch, and Sahin Albayrak. JADL — an agent description language for smart agents. In Mateo Baldoni and Ulle Endriss, editors, *Declarative Agent Languages and Technologies IV*, volume 4327 of *LNCS*, pages 141–155. Springer Verlag, 2006.

- [14] Tobias Küster. Development of a visual service design tool providing a mapping from BPMN to JIAC. Master's thesis, Technical University of Berlin, Faculty of Electrical Engineering and Computer Science, 2007.
- [15] Tobias Küster and Axel Heßler. Towards transformations from BPMN to heterogeneous systems. In Massima Mecella and Jian Yang, editors, *BPM2008 Workshop Proceedings*, 2008.
- [16] Tobias Küster, Marco Lützenberger, Axel Heßler, and Benjamin Hirsch. Integrating process modelling into multi-agent system engineering. In Michael Huhns, Ryszard Kowalczyk, Zakaria Maamar, Rainer Unland, and Bao Vo, editors, *Proceedings of the 5th Workshop of Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE) 2010*, 2010.
- [17] Giovanni Caire (Telecom Italia S.p.A). *WADE User Guide*. Copyright ©2010, Telecom Italia, July 2010.