FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# How to Execute Parts of BPMN

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Wirtschaftsinformatik**

eingereicht von

**Christian Gutschier**
Matrikelnummer 0826184

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Roman Popp

Wien, 20.03.2014

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# How to Execute Parts of BPMN

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Business Informatics**

by

**Christian Gutschier**
Registration Number 0826184

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl
Assistance: Univ.Ass. Dipl.-Ing. Dr.techn. Roman Popp

Vienna, 20.03.2014

(Signature of Author) (Signature of Advisor)

# Erklärung zur Verfassung der Arbeit

Christian Gutschier, Schüttelstrasse 13/14, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Wien, 20.03.2014) (Unterschrift Verfasser)

# Acknowledgements

First, I want to thank my fiancee Angelina Schaupal, who was a great help and support during my study. I also want to thank my parents, sister and brother, who always have been supporting me. Furthermore, I want to thanks Ralph Hoch, David Raneburger and my cousin Thomas Huber, who have helped with words and deeds during this master's thesis. I would also like to thank my supervisor Prof. Hermann Kaindl and my assistant Dr. Roman Popp for the great support and helpful feedback. Finally, I want to thank all my fellow students for the great time while studying.

# Abstract

The introduction of version 2.0 of Business Process Model and Notation (BPMN) promised direct executability of compliant business processes. In the previous versions of BPMN (1.x), a transformation of BPMN models into another, executable format (e.g., Business Process Execution Language, BPEL) was required to make these models executable. The reason why direct execution is now possible with BPMN 2.0 is that attributes necessary for execution were added to the XML-representation. This master's thesis investigated how BPMN 2.0 models (based on the BPMN 2.0 standard) can be executed directly. Furthermore, the standard should support portability between different tools.

Direct execution has been investigated based on a simple business process example, which was modeled with BPMN 2.0. The focus is on Tasks that can connect to software (e.g., Web Services or Java) and do not required human interaction (e.g., User Tasks). During the investigation, a pitfall was found related to the number of parameters (which a Service Task accepts) and solutions are proposed for circumventing this pitfall. Due to this pitfall it is not possible to use BPMN 2.0 as a general orchestration language (such as BPEL). Based on the simple BPMN 2.0 model, also the portability between tools was investigated. To select suitable tools, the following three criteria were defined. First, the tools have to be freely available (e.g., open source or trial version). Second, the tools have to support BPMN 2.0 modeling. Third, the tools have to support direct execution of BPMN 2.0 models.

All selected tools in this master's thesis were able to directly execute the business process example. However, adjustments in the XML-representation of this example were needed when it was not created with the designer of the same tool. For this reason, portability between the tested tools was not given without adjustments.

# Kurzfassung

Mit der Einführung der Business Process Model and Notation (BPMN) Version 2.0 wird die direkte Ausführbarkeit von Business Prozessen versprochen, da bei den vorherigen Versionen von BPMN 1.x eine Transformation der erstellten BPMN Modelle in eine andere Ausführungssprache (z.B. Business Process Execution Language, kurz BPEL) erforderlich war. Für die direkte Ausführung wurden in der XML-Repräsentation von BPMN die notwendigen Attribute hinzugefügt. Im Rahmen dieser Diplomarbeit wurde untersucht, wie BPMN 2.0 Modelle (basierend auf dem BPMN 2.0 Standard) direkt ausgeführt werden können. Außerdem wird untersucht, ob die Portabilität zwischen Tools, wie sie durch den Standard unterstützt werden sollte, gegeben ist.

Die direkte Ausführbarkeit wird anhand eines einfachen Business Prozesses, welches in BPMN 2.0 modelliert wurde, untersucht. Der Fokus liegt dabei auf Tasks, welche mit einer Software (Web Service oder Java) verbunden und automatisch ausgeführt werden können (d.h. kein User Task). Dabei wurde ein sogenannter „Pitfall“ in Bezug auf die Parameter-Anzahl eines Service Tasks gefunden und Lösungen zur Umgehung vorgeschlagen. Durch den „Pitfall“ ist eine Verwendung von BPMN 2.0 als allgemeine Orchestrierungssprache (wie z.B. BPEL) nicht möglich. Anhand dieses einfachen BPMN 2.0 Models wurde ebenfalls die Portabilität zwischen verschiedenen Tools untersucht. Um die Auswahl der Tools einzuschränken, wurden folgende drei Kriterien definiert. Erstens, diese Tools müssen frei verfügbar (d.h. Open Source, Trial-Version, etc.) sein. Zweitens, die Tools müssen die Modellierung von BPMN 2.0 konformen Modellen unterstützen und drittens, die Tools müssen die direkte Ausführung von BPMN 2.0 konformen Modellen unterstützen.

Im Rahmen dieser Arbeit wurde festgestellt, dass die direkte Ausführung in den getesteten Tools möglich ist. Allerdings mussten in jedem Tool vor der Ausführung Anpassungen an dem BPMN 2.0 Model durchführt werden, wenn das Modell nicht mit dem dazugehörigen Designer modelliert wurde. Aus diesem Grund ist die Portabilität zwischen den getesteten Tools ohne Anpassungen nicht gegeben.

# Abbreviations

| | |
|---|---|
| BAM | Business Activity Monitoring |
| BPD | Business Process Diagram |
| BPE | Business Process Engine |
| BPEL | Business Process Execution Language |
| BPDM | Business Process Definition Meta-model |
| BPML | Business Process Modeling Language |
| BPMN | Business Process Model and Notation |
| BPMS | Business Process Management System/Suite |
| EPC | Event-driven Process Chain |
| MEP | Message Exchange Pattern |
| UML | Unified Modeling Language |
| WfMS | Workflow Managment System |
| WSBPEL | Web Services Business Process Execution Language |
| WSDL | Web Services Description Language |
| XPDL | XML Process Definition Language |

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation and Research problem

Today it is necessary for managing business processes that they are well specified and documented. Previously, simple business processes mostly have been described in textual form in natural language, or a flowchart was created as a mean of presentation. However, descriptions in natural language are inherently ambiguous and it is not possible to represent more complex business processes with the flowchart, because important aspects cannot be considered (like branching rules, data flows, etc.). To address these problems specific notations for business process have been developed. The notation defines which graphical symbols represent certain processes and declare how to link these symbols to each other [2].

A notation is important, because it makes it possible to define a uniform language for certain processes (e.g., business processes). The language helps to draw uniform models and also allows people who have learned the language to understand models which were created by another person more easily. This leads to a communicative value of a model and so processes can be analyzed better due to the standardized representation of the models [2].

Business Process Model and Notation (BPMN) published by the Object Management Group (OMG) [1] is a standard for the graphical representation of workflows and business processes. Due to the clear and simple graphical representation of processes in BPMN, it can be used for various business processes (e.g., baking bread in a bakery or complex production of a product with shipping) within an organization, but also between organizations. BPMN was the first specification language that has successfully merged the two worlds of the operating department and the information technology department. The first versions of BPMN (BPMN 1.x) put great importance to the graphical representation of business processes [3]. However, for the execution of business processes the created BPMN models had to be transformed into an other executable format, e.g., Business Process Execution Language (BPEL).

[1] http://www.omg.org/

Today processes are often executed by business process management systems (BPMS). Here, the execution of the processes occurs by a process engine on the basis of formal process descriptions, e.g., in the form of business process models. A challenge here is that the created models have to fully specify all details required for automated execution by the machine [2]. This also allows simulating models. With the knowledge gained from simulations, business processes can be improved inexpensively and money and time can be saved. Many commercial and non-commercial tools very often provide the functionality to execute BPMN models. This has been solved in different ways in the tools.[2]

Direct execution was not supported by BPMN 1.x, but is supported by BPMN 2.0. This is at this stage the latest available version. In this version, a BPMN model is stored in a standardized XML-based format, which facilitates the portability between different tools (e.g., tools for modeling, simulation or execution of process models).

BPMN 2.0 promises also direct execution, because in addition to storing the model in a standardized format, execution semantics have been introduced. Important attributes were added to the notation of BPMN, which are required for the direct execution of BPMN models. This has the advantage that no transformation of BPMN models into another executable format (e.g., BPEL) is needed anymore.

The *research question* to be answered in this master's thesis is, *whether the same BPMN 2.0 model is directly executable by different tools and if not, which additional modifications are required for direct execution.*

In the BPMN 2.0 standard, so-called extensions are allowed, but the standard does not specify exactly how these extensions must be implemented. This entails that the extensions are implemented differently in the tools and this is a problem for the portability. Therefore, models using extensions are out of the scope for this master's thesis. In addition, the master's thesis only investigates direct execution of BPMN 2.0 models. Therefore, parts of BPMN which require human interaction are not examined in the master's thesis. In this investigation, the focus lies only on Tasks which can be connected to software (Web Service or Java), the other kinds of Tasks are out of scope.

## 1.2 Approach

The major aim of the master's thesis is to find out, how BPMN 2.0 models can be automatically executed. Another aim is to investigate the portability of the different tools. This means that a created model should be executable in any tool which supports the direct execution of BPMN 2.0 models without modifications. For this purpose, only tools were used for the investigation, which support the modeling and execution of BPMN 2.0 models. These tools have been checked for their suitability. This has been done by comparing information from Websites, books and papers about the various tools. Based on this comparison, the eligible tools were selected. Then a simple business process example was created. The created model was tried to import and to execute in the different tools. This simple business process example is "Issuing Invoice" with two Tasks, Start-Event, End-Event, three Data Inputs and a Data Output. In the first Task, an

[2] directly or through transformation into an execution language e.g., BPEL

invoice is created with two Data Inputs (amount and address). In the second Task, the created invoice is sent to a customer. If necessary, the BPMN 2.0 model or the implementation of the tool (if possible) were adapted.

## 1.3 Overview of the master's thesis

The remainder of this master's thesis is organized in the following manner.

Chapter 2 provides general information about BPMN. After that, the reader gets a historical overview of the development of BPMN in recent years, and also the current situation of BPMN is presented. Afterwards important notations and diagrams are explained which are offered by BPMN 2.0.

Chapter 3 analyzes and discusses already available non-direct execution approaches for BPMN.

Chapter 4 explains the principle of a Business Process Engine. Furthermore, the BPMN model example is defined and created. Based on this model, an explanation is given how Business Process Engines execute a BPMN model. However, the created model has no integrated services (like Web Services). In the next step, it is shown how the BPMN model needs to be changed so that a service can be integrated.
Here it is shown that the BPMN 2.0 Standard has a restriction for Service Tasks. This entails that only a small number of available services can be involved. Different approaches to solving the problem of restricting Service Tasks are presented. These approaches are BPMN 2.0 standard compliant. Then the various Business Process Engines of selected tools are examined. Finally, a comparison of the various tools is given with respect to execution.

In Chapter 5 the conclusion is presented.

CHAPTER 2

# Background on BPMN

Business Process Model and Notation (BPMN) is a graphical specification language which provides symbols to model business processes, workflows and business activities. BPMN was introduced to the public in 2004 by BPMI (Business Process Management Initiative). BPMI is an organization whose representatives consist mainly of software companies which had taken on the task of defining standards in the scope of business process modeling [2].

Today BPMN is an essential part of business informatics and mostly used by professionals and computer science experts. The BPMN standard can be used for a direct execution (possible in the current Version BPMN 2.0) of a BPMN-model (which show the business processes) or for the transformation of a BPMN-model into an execution language for business processes (e.g., Business Process Execution Language (BPEL)) so that the business processes may be executed. However, this transformation does not always work perfectly and it is possible that in some cases there is a semantic difference. More detailed information about this can be found in Chapter 3.

Nowadays it is possible to find BPMN in many big organizations as an integral part of business modeling, because it offers a standard for modeling and it is used internationally. Another reason was that BPMN was first to merge the operating department and the IT department. A benefit for organizations is the ability to simulate business operations. The obtained information from the simulation is very important for organizations because this processes can be improved based on it.

One aim of BPMN is to be easily understandable and comprehensible for all parties. This includes business analysts, technical developers and even end users. All of them should be able to read and understand the created model easily [6].

An advantage of BPMN is that it offers different types of diagrams, which are explained in detail in chapter 2.3.2. This section gives a view at business processes from different perspectives. Each type of diagram represents certain aspects of a business process better than another type and so the participants get a better overview of the business process [6].

Another aim of BPMN is the executability of the models which represent the business processes. In the last version of BPMN at the time of this writing (BPMN 2.0) it was attempted to implement executability. Details are described below.

## 2.1 Historic development – BPMN 1.x

As mentioned above, BPMN was presented by BPMI in 2004, but the development has begun in the year 2001 under the direction of Stephan A. White from IBM. The aim was to create a graphical notation, so that process descriptions of BPML (Business Process Modeling Language) can be represented visually. With BPML which is similar to BPEL, it is possible to specify process models which are executed by Business Process Management Systems (BPMS). However, other approaches (like BPEL), which have been established in this area, have subsequently replaced BPML [2].

It was important for the developers, that both business and also technical models can be represented with the graphical notation. BPMN can be read and understood by IT experts but also from business experts. So there should be a common language for differently qualified people.

This was, of course, not an easy task. It is possible to use the same notation for both models (business and technical). In practice, a significant difference between them was found. For example understanding the process flow is more needed for a business model and therefore we don't need to model too many details. In contrast in a technical model a lot of details are extremely important for the subsequent executability [2].

*"In June of 2005, the Business Process Management Initiative (BPMI.org) and the Object Management Group™ (OMG™) announced the merger of their Business Process Management (BPM) activities to provide thought leadership and industry standards for this vital and growing industry."* [1]

OMG is a society that developed standards for object-oriented programming since 1989. Therefore, the OMG already had a good reputation in standardization in the software sector. A well-known standard modeling language of OMG is, for example, the Unified Modeling Language (UML).[2]

In 2006, the first official BPMN version 1.0 was introduced and presented by OMG as a standard. Two years later (2008) a new version (1.1) was released. The main difference to version 1.0 was in their presentation of the model. This was an important criterion, which had to be taken into account, because many tools supporting BPMN included the version 1.0. A more detailed overview of the change from version 1.0 to version 1.1 is available in [10]. In 2009, the next version (1.2) was released. Only small corrections and clarifications were made and so most of the content of version 1.1 was unchanged [2].

However, the direct execution of business process model was not possible including version BPMN 1.2, because the focus was laid only on the modeling. So that a BPMN model could be executed, the model had first be transformed to another execution language for business processes (describe in chapter 3).

[1] `http://www.bpmi.org/`

[2] `http://de.wikipedia.org/wiki/Object_Management_Group`

## 2.2 Current status – BPMN 2.0

Only the next and current version (2.0) of BPMN which was fully developed in 2011, brought again important changes and interesting innovations. In this version bugs and problems of the previous version have been fixed, so that the new one is more stable. Until today it appeared that after the release of version 2.0 no changes are planned or expected during the next years. So some manufactures began to develop modeling tools which integrate the new version of BPMN [2].

During the development of the new version it was important, that the notation elements of version 1.2 have not changed essentially. The graphical notations are extended only by some constructs and some model types. So an already created model of version 1.2 is compatible with version 2.0. The following overview (taken from [2, p. 11]; translated from German) provides a brief survey of the most important extensions of the graphical notation which was added to the current version of BPMN:

- *"New Event types: parallel multiple Events and escalation*
- *Parallel Event-Based Gateway to the start of processes*
- *Intermediate Events tacked at the Activities, so that they indeed start an exception flow, but they don't stop their Activities*
- *Event-Sub-processes which are performed only when a specific Event occurs and run in parallel to the surrounding process or cancel it*
- *Extended possibilities for modeling data processes*
- *Extensions in modeling of collaborations*
- *Identification of the different types of tasks by symbols*
- *New representation of the call from Activities defined elsewhere*
- *Identification of different multiple Activities, depending on whether they are performed in parallel or sequentially"*

Furthermore two new types of diagram have been introduced into BPMN 2.0:

- Choreography diagram
- Conversation diagram

These are visible changes. However, there are also hidden changes which have a much greater importance than the mentioned visible changes. In BPMN 2.0, a formal definition has been implemented in the form of a meta-model, also called Business Process Definition Meta-model (BPDM). Due to this new development, a consistent language was achieved. Furthermore, in the meta-model additional language constructs exist which cannot be displayed in graphical models, since they are only required by the process engine for the execution of the model (some additional information for the process engine) and unimportant for understanding the process. Another reason, why these constructs should not be in the diagram, is that without them a better

and quicker overview of the business process can be given, so that everyone can understand the diagram [2].

Due to the meta-model it was also possible that models are uniformly designed (modeled), because only certain constructs are allowed by the specification of the meta-model. In addition, a standardized exchange format for BPMN models in the version 2.0 was developed/integrated, which allows models to be transferred from one tool to another. This was more difficult and a lot of effort with earlier versions of BPMN [2].

A further major milestone in the development of BPMN 2.0 was the introduction of execution semantics. BPMN 2.0 models can already be executed directly and do not to have to be converted into an executable format (like BPEL) first. However, this only works partially and only for very detailed and simple BPMN models [2].

As mentioned above, today BPMN is a standard, and so more and more tool manufactures integrated BPMN into their software in recent years. But BPMN has not always been extensively integrated into the vendor tool as needed. Sometimes only a part of BPMN was integrated and the vendor still claimed support for BPMN. The reason was that there were no requirements for the software which had to be satisfied in order to conform with the BPMN requirements. Therefore, it was important to introduce certain criteria during the last years. Due to the several kinds of tool for BPMN, different BPMN conformities have been defined. These different BPMN conformities can help to choose the right software tool. The two most important are: [2]

- Process modeling conformity (only modeling tool)
- Process execution conformity

## 2.3 Explanation of selected BPMN notation and different diagram types

In this section first the different main core elements of a Business Process Diagram (BPD) are briefly described. After that an overview and a short description of the different diagram types in BPMN 2.0 is presented.

### 2.3.1 Explanation of the different core elements of a BPD

One of the main goals of the developers of BPMN has been that the creation of models can be made without great effort, but anyway very complex business processes can be modeled. Therefore, they tried to ensure that the number of different elements is as small as possible. The elements are divided into various categories. Due to the small number of category assessors, the observer is able to identify very quickly the basic elements of the diagrams. We distinguish between the following basic graphical categories of elements in BPMN [8]:

- Flow Objects [2.3.1.1]
- Connecting Objects [2.3.1.2]
- Swimlanes (Pools and Lanes) [2.3.1.3]

- Artifacts [2.3.1.4]
- Data [2.3.1.5]

#### 2.3.1.1 Flow Objects

These elements are responsible for the definition of the behavior of a business process and, therefore, they are the most important graphical elements in BPMN. They are divided into the following three types:

- Activity
- Event
- Gateway

**Activity**
Activity elements are used to model different work steps in a BPMN-Model. An Activity is modeled as a rectangle with rounded corners as shown in Figure 2.1 and can fundamentally distinguish between an atomic and a non-atomic Activity. An Activity can be a Task (atomic), Sub-Process (non-atomic) or Call Activity. A Task is work which has be done in the business process. A Sub-Process can be modeled in either collapsed or expanded state and can be contained all BPMN elements. The Sub-Process is used that several work steps (Tasks) put together in an abstract Task (if the Tasks pursue the same goal). The reason is that so a complex diagram (with many Tasks) can be better understood. A task and a Sub-Process differ by a "+" symbol in the rounded rectangle. The Task and Sub-Process are the most commonly used types of representing Activities in BPMN. Figure 2.1 shows the different presentations of Task and Sub-Process [6] [1]:

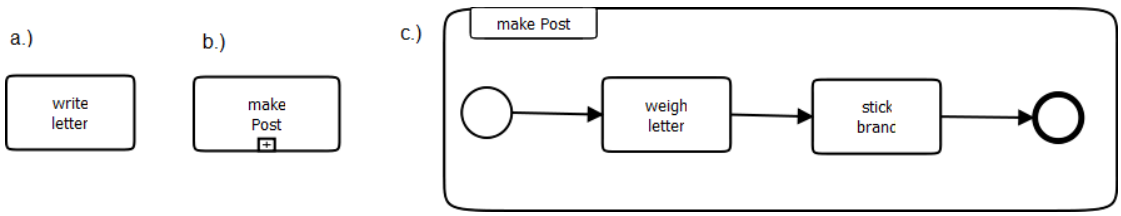


Figure 2.1: a.) atomic Activity (Task), b.) collapsed Sub-Process, c.) expanded Sub-Process

The Call Activity is used, if a Task, Process or Sub-Process will be used again in the diagram. All these elements of the activity can be distinguished into other types. A good overview of different types of Task, Sub-Process and Call Activity can be found in [1].

**Event**
Events occur in the course of a process and there are also different types. In BPMN 2.0, we distinguish between three different types of Events (start Events, intermediate Events and final

Events). The three basic types of Events do not only differ in their name, but also their graphics and their semantics.

The Start Event has the form of a circle with a thin line. This Event is always at the beginning of a process or a section and has no incoming edge. The opposite is the End Event. It has to be always at the end of the process or section. The final Event is also modeled as a circle like the Start Event, but the circle is represented with a thicker line. Furthermore, it must not have any outgoing edge, only incoming edges are allowed. An Intermediate Event may occur during the process execution and can be used between the other two types of Events in a process. It is also shown as a circle, but it is plotted with a double thin line [6]. Figure 2.2 shows the different representation of the above mentioned basic Events.


Figure 2.2: Representation of basic types of Events taken from tool Yaoqiang BPMN Editor 2.2

In most cases, intermediate events have an incoming and an outgoing edge (Sequence Flow) in contrast to the other two types of events. However, there exist also certain events which only have one outgoing edge (for example interrupt and non-interrupt intermediate events). Such types of intermediate events occur mostly within an Activity or a Sub-Process, and therefore they are placed at the border of the Activity or Sub-Process [2]. An example of an interrupt intermediate event within an Activity shows Figure 2.3. Here the Activity "Try to solve Exercise" has two outgoing edges (Sequence Flows), but only one can be executed. If the exercise has been completely solved within the given time, the student gets the full number of points. If not, the student gets only a reduced number of points.

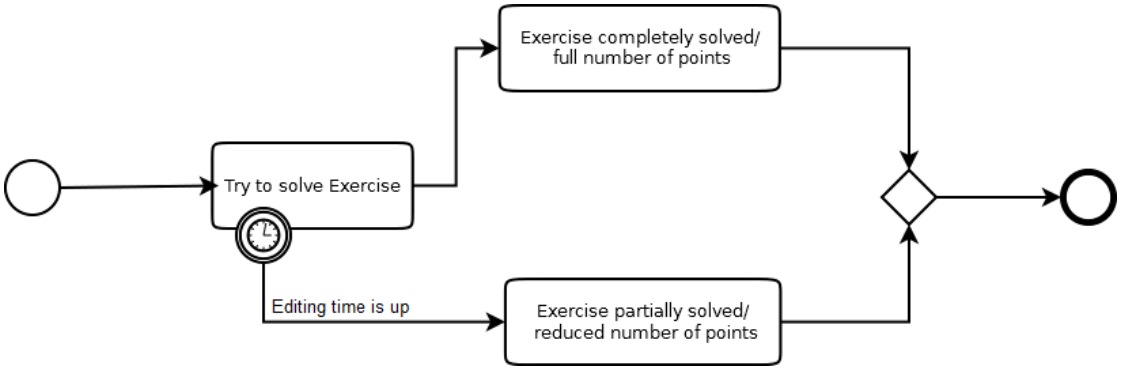


Figure 2.3: Example with an interrupt intermediate event

All three types of basic Events can be represented with various symbols. Each symbol has a different meaning, for example if a letter-symbol (shows in Figure 2.4) is modeled in the middle of the Event, it is possible to expect or send here a message. A good and detailed overview of the various symbols representing Events can be found in [6].


Figure 2.4: Different basic Events with the letter-symbol taken from tool Yaoqiang BPMN Editor 2.2

**Gateway**
BPMN introduced elements called "Gateways". There are several types of gateway with different semantics. A gateway is used for splitting (fork) and/or bringing together (join) the control flow (Sequence Flow) from a business process. Gateways have the form of a rhombus, and conform to a so-called decision point. Gateways can have usually one or more incoming edges and multiple outgoing edges. In the most cases, the outgoing edges of the Gateway are connected with a condition. Depending on which condition is satisfied, one or more edges are activated or on e or more paths are pursued.

Figure 2.5 shows a simple example for a Gateway. In this section of a process example, it is important that the letter is correctly sent. After weighing, the letter can go abroad or not. If so, the letter needs a priority sticker. If the letter does not go abroad, the letter does not need a priority sticker (nothing has to be done). In this example the Gateway is an "Exclusive Gateway", because the decision at the Gateway is an XOR decision. This means that due to a predefined condition only one of the outgoing paths (edges) will be pursued. Therefore it has to be ensured, that the conditions are mutually exclusive. As seen in Figure 2.5, we have two Gateways constituted, namely a fork- and a join-Gateway. The fork Gateway has the condition ("Does the letter go into foreign?") and splits the Sequence Flow based on this decision. This condition can only be "true" or "false", and therefore the conditions are mutually exclusive and only one of the two outgoing paths is pursued. A join Gateway merges two or more Sequence Flows back together. It is also important that only one Sequence Flow can be activate on the joining Exclusive Gateway. This can only be guaranteed if the Sequence Flow was split with an Exclusive Gateway. Figure 2.5 shows that the Sequence Flow is first split by an Exclusive Gateway and then merged together. This type of Exclusive Gateway is also called Data-Based Exclusive Gateway and a further type of Exclusive Gateway is called Event-Based, which is describe below [6]. More detailed information and examples of the various possibilities which can occur in an Exclusive Gateway can be found on the homepage of BPMN[3]. The representation of an "Exclusive Gateway" can has an "X"-symbol in the middle, but this is not required, because the Exclusive Gateway is the default Gateway in BPMN. This means that a blank Gateway is seen as an exclusive one.

[3] http://www.bpmn.org/

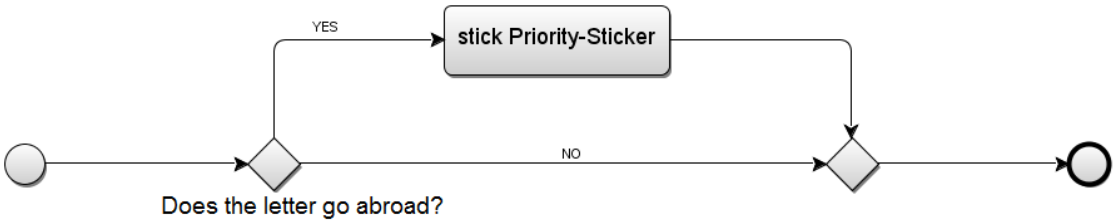


Figure 2.5: A simple example with a Gateway

This example shows that the business process is done sequential, but in the most business processes, it happens often that the work is not to done sequentially, but to be done in parallel. For this reason BPMN has introduced the following four types of Gateway, which differ not only graphically but also semantically.

- Inclusive Gateway
- Parallel Gateway
- Complex Gateway
- Event-Based Gateway

The “Inclusive Gateway” is drawn with a circle-symbol in the middle of the Gateway representation. In contrast to the Exclusive Gateway the Inclusive Gateway can have multiple outgoing edges activated. In a split Inclusive Gateway all outgoing edges are associated with a condition which does not have to eliminate them mutually and, therefore, it is possible to follow several paths. So the semantics is here interpreted as “OR”. It is also possible to define a default path which is executed only if no other conditions of the outgoing paths are met. The “join Inclusive Gateway” has the analogous behavior as the “join Parallel Gateway”. The task that comes after the “join Inclusive Gateway” can be executed only if all paths which have been activated at the “fork Inclusive Gateway”, are also activate at the “join Inclusive Gateway”.

Yet another type of Gateway is the “Parallel Gateway”. Here is a “+”-symbol drawn in the middle of the Gateway symbol. There are two important differences as compared with the Exclusive Gateway explained above. The first difference is, that no further conditions in a Parallel Gateway are needed, which split the control flow. All outgoing paths from the “fork Parallel Gateway” are pursued concurrently. The second difference is, that a task that comes after a “join Parallel Gateway”, will be executed as soon as all the converging paths of the Gateway were completed [6]. Figure 2.6 shows an example for both differences. It shows a part of the process when a goal is scored in a hockey game. If a goal is scored, the process will started. At the first Gateway (fork Parallel Gateway) the Sequence Flow is split into two Sequence Flows. This means that task “increase the goal-counter on scoreboard” and task “enter the goal into the system” are performed at the same time in the process. At the second Gateway (join Parallel Gateway) the two sequence flows are reunited to a sequence flow. This means that task “confirm the goal by speaker” can only be executed if both task “increase the goal-counter on scoreboard” and task “enter the goal into the system” are completed.

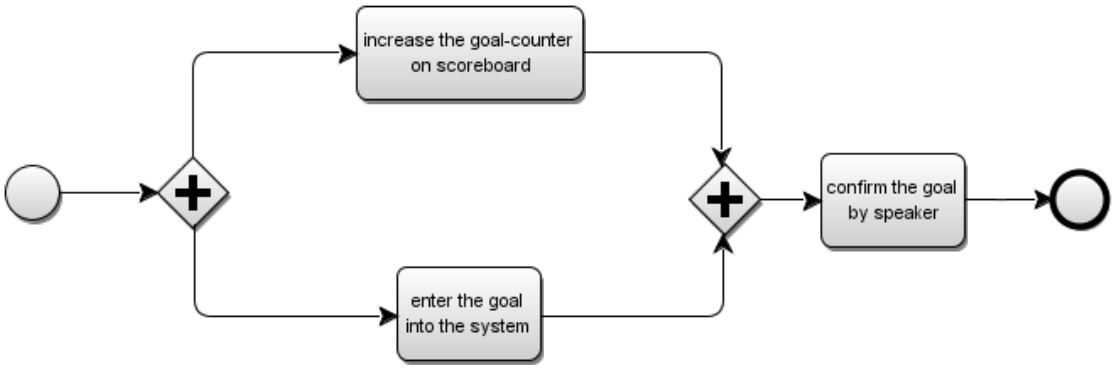


Figure 2.6: A simple Parallel Gateway

Another type of Gateway is the “Complex Gateway”. In this Gateway different semantics can be defined, which would not be possible with any one of the other three types of Gateways. The exact semantics can be defined in an Expression. An example to use a Complex Gateway is, if a Gateway has three incoming flows, but the following task only needs two of three flows to complete its work. Then the gateway waits for only two of them to switch to the task (they two flows deliver required information). This can only be modeled with a Complex Gateway. Therefore, it is important that the semantics is described as precisely as possible. [6].

The last type of Gateway is the Event-Based Gateway. *“The **Event-Based Gateway** represents a branching point in the **Process** where the alternative paths that follow the **Gateway** are based on **Events** that occur, rather than the evaluation of Expressions using **Process** data (as with an **Exclusive** or **Inclusive Gateway**). A specific **Event**, usually the receipt of a **Message**, determines the path that will be taken. Basically, the decision is made by another Participant, based on data that is not visible to **Process**, thus, requiring the use of the **Event-Based Gateway**.”* [1, p. 297]. For example, a company has made and sent a customer an offer and is waiting for the reply proceed in the process. Depending on the decision of the customer to accept or refuse the offer another branch is to be pursued in the Process.

As in the Events listed above, the representation of the different Gateways distinguish through the symbol in the middle of Gateways. Figure 2.7 summarizes the graphical representations of the different types of Gateways as described above:

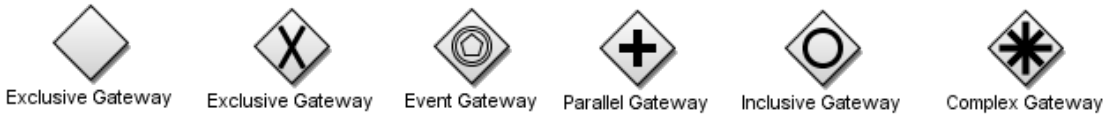


Figure 2.7: The graphical representations of different Gateways taken from the tool Yaoqiang BPMN Editor 2.2

A BPMN model can also include branches without using gateway. It all depends on what style of modeling or modeling convention is defined in the organization. Figure 2.8 shows an example. This Figure shows two BPMN model. In both model an amount is first checked and

then it is decided if the amount is transferred (if the amount is over 100) or the amount is paid in cash (if the amount is under 100). On the left side there is the model drawn with a Gateway and on the right side the model is drawn without a Gateway. In the left model the Gateway is an Exclusive Gateway, which means either the upper or the lower path is pursued. In the right model there is no Gateway, but the behavior is the same as in the left model. This is guaranteed by the two Sequence Flows. The upper Sequence Flow with the small route is a "Conditional Sequence Flow". This means it is only activated as soon as the condition on it is satisfied. The lower Sequence Flow with the oblique stroke is a "Default Sequence Flow" and it is activated if no other outgoing Sequence Flow is activated. But these drawn models (if they are bigger than this simple example) are often very difficult to understand as opposed to a model that uses Gateway. In addition not all types of gateway can be replaced through the various types of Sequence Flows.

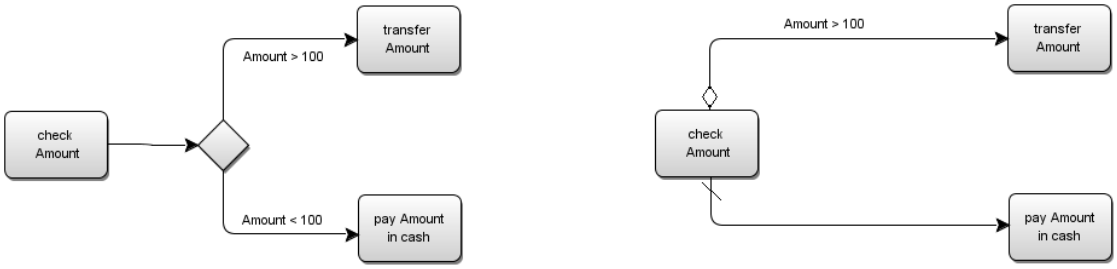


Figure 2.8: Branches with and without Gateway taken from the tool Yaoqiang BPMN Editor 2.2

#### 2.3.1.2 Connecting Objects

Connecting Objects are responsible for connecting Flow Object elements described above. They represent in a BPMN model the different types of edges. The following three types of Connecting Objects are distinguished:

- Sequence Flow
- Message Flow
- Association

**Sequence Flow**
The Sequence Flow is the most used form of Connecting Objects. It is used to define the order in which the Flow Objects of a business process are executed. Sequence Flows are only allowed to connect Flow Objects inside of a Pool, but not to another Pool. Graphically a Sequence Flow is represented as a directed edge and shows in which order the Flow Objects are executed. This means that, for example, the Activity with an incoming Sequence Flow may only be performed, if the preceding Activity has been carried out (the Activity where the Sequence Flow comes from). Due to this modeled order, a well-structured and easy-to-read model is obtained [6].

A simple example of such a dependency is shown in Figure 2.9. First task 1 must be finished, before the execution of task 2 can start.

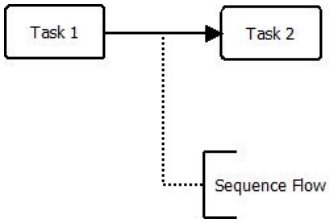


Figure 2.9: Sequence Flow

Sequence Flows are distinguished in three different types. First, there is the “Normal Sequence Flow”, this is a simple directed edge, as seen it in Figure 2.8 (The left model is drawn with a normal Sequence Flow). Another type of Sequence Flow is the “Conditional Sequence Flow” (Figure 2.8 in the right model the upper Sequence Flow) which has a small rhomb at the beginning of the directional edge and is taken only under certain conditions (as an alternative to Gateways). The last one is the “Default Sequence Flow” (Figure 2.8 in the right model the lower Sequence Flow) which has an oblique stroke at the beginning of the directional edge. It is only taken if no other outgoing Sequence Flow can pass through.

**Message Flow**

Message Flows always connect Lanes, Pools or Flow Objects together and they are supposed to show, how messages flow between various parties which are involved in the process. Message Flows are only allowed between different Pools, but not within a Pool [2]. An example of a Message Flow is shown in Figure 2.10. Here, the customer sends an order to the bakery with a message using a Message Flow. Once the order has been produced, the customer gets a message back via Message Flow. In this message, the customer is informed that the order has been delivered.

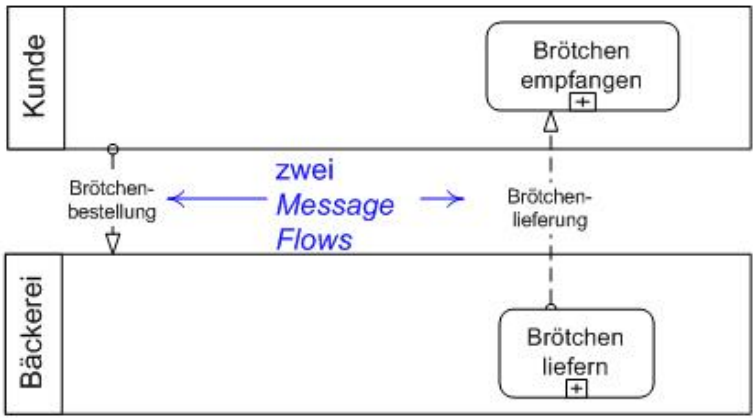


Figure 2.10: Message Flow (copied from [4])

**Association**
Associations connect Artifacts (see 2.3.1.4) with Flow Objects and Connecting Object. They are simple links without any semantics in the model, like comments. An association is represented by a dotted edge [8].

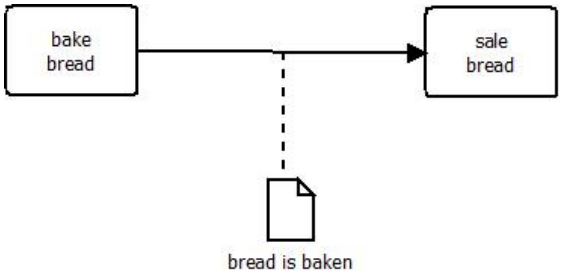


Figure 2.11: Association example

#### 2.3.1.3 Swimlanes (Pools and Lanes)

Swimlanes are available in the latest version (2.0) of BPMN and they are important to recognize who is responsible for which tasks in the model. They are used to group different notations of BPMN. Swimlanes can be in Pools or Lanes. A Pool usually constitutes of an involved party of a process, like a person or a department. In addition all the other elements of BPMN cab occur in a Pool. It is necessary to make sure that a Sequence Flow only connects other object flows within a Pool, while the other connecting flows can refers to other Pools [6]. The graphical representation of a Pool is shown in Figure 2.10. The customers as well as the bakery are represented as a Pool, and a message is flowing back and forth by Message Flows.

However, a Pool, if it has to represent a department or a company, for example, is not always sufficient to obtain a good structure. In most cases, sub-groupings are required, like in a company which has various workers. Therefore, there is the possibility of BPMN to divide the Pool into so called Lanes, so it is easier to understand the whole process and to know quickly who was responsible for what [6]. Figure 2.12 shown an example of a Pool landscape.

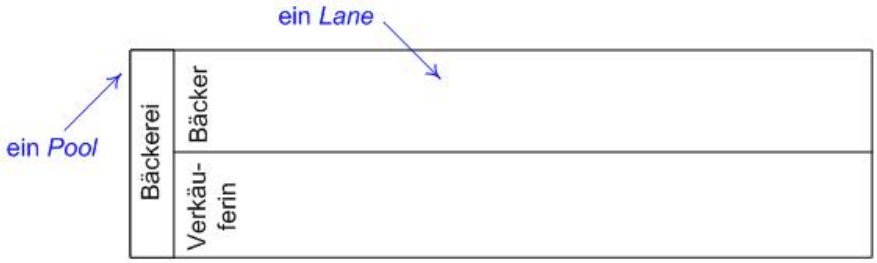


Figure 2.12: A Pool landscape (copied from [5])

[4] http://commons.wikimedia.org/wiki/File:Bpmn-7.png
[5] http://commons.wikimedia.org/wiki/File:Bpmn-6.png

#### 2.3.1.4 Artifacts

Artifacts are used to display additional information in a process model or to get a better overview, which elements belong together. Here a distinction is made between "annotation" and "group". Annotations are comments which are connected with the elements of a business process. Groups summarize the elements of a process for a better overview, but they are not a Sub-process. Figure 2.13 shows these artifacts in an example [8].

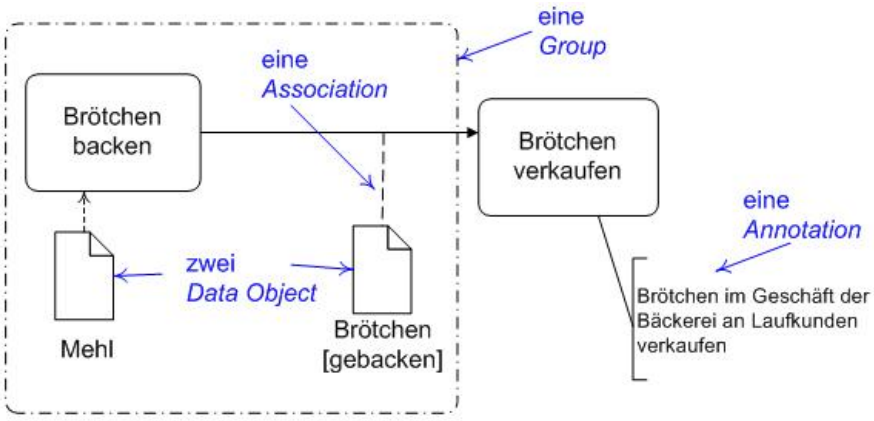


Figure 2.13: Artifacts (copied from [6])

#### 2.3.1.5 Data

The last category explains the data elements and distinguishes four different kinds:

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

Data Objects have or provide information. This information may be data which are required for the execution of the process. *"Data Objects can represent a singular object or a collection of objects."* [1, p. 30].[7] Data Input and Data Output provides the same information for the execution of activities and processes as Data Objects. Data Inputs are specifically declared data items which are used as inputs for Tasks and Processes. Via Data Associations it is possible to map Item Definitions (e.g., Data Objects and Properties) to a given Data Input. The same applies to Data Outputs with the difference that these items are produced by Tasks or Processes.

*"A Data Store provides a mechanism for Activities to retrieve or update stored information that will persist beyond the scope of the Process. The same Data Store can be visualized, through*

[6] http://commons.wikimedia.org/wiki/File:Bpmn-8.png
[7] http://manual.altova.com/de/umodel/umodelenterprise/index.html

*a Data Store Reference, in one or more places in the Process"* [1, p. 208]. Figure 2.14 shows the different graphical representation of data elements.

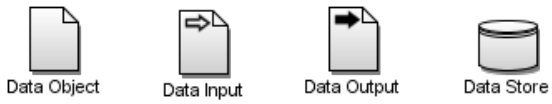


Figure 2.14: Different data notations taken from the tool Yaoqiang BPMN Editor 2.2

### 2.3.2 Explanation of the different diagram types

While the previous version 1.2 of BPMN could distinguish only two types of diagram, two new types of diagrams ("Choreography" and "Conversation") were added in the current version 2.0. So, it is possible now to distinguish between the following four different types of diagram:

- Process diagram [2.3.1.1]
- Choreography diagram [2.3.2.2]
- Collaboration diagram [2.3.2.3]
- Conversation diagram [2.3.2.4]

As mentioned above, this distinction is important, because it is possible to represent business processes from different perspectives. In these different types of diagram, the above-mentioned main core elements can also be used. However, the various diagrams have also other additional elements which are explained briefly below at the associated type of diagram.

#### 2.3.2.1 Process diagram

Process diagrams are used to model processes which e.g. a company has to perform to do its work. Here, the individual steps (tasks) of the processes are defined and displayed [6]. This should lead to a visual and a good structuring of processes and it should provide a quick overview of the task and the business process.

Figure 2.15 shown a simple Process diagram. The nodes (e.g., "Write Job Posting") corresponds to a single Activity or Event and the directed connected edges (e.g., between "Report Job Opening" and "Write Job Posting") are a sequence of Activities. In such given sequence also from a control flow is spoken. In addition, there is also another node which can either divide or join this control flow. More detailed information on the individual notations can be found in the previous section [2.3.1].

Furthermore it is possible to distinguish between two types of Process diagrams. There is on the one hand the internal (private) process which maps the internal processes of an organization in a certain sequence [8]. Figure 2.16 shown an example.

On the other hand the public (abstract) process which represents the interactions between an internal and one or more other processes or users. Only the Activities and the Connecting Object elements (control flow) of the process are shown, which are important for the communication with the partners. The other elements are not shown [8]. Figure 2.17 shown a public process.

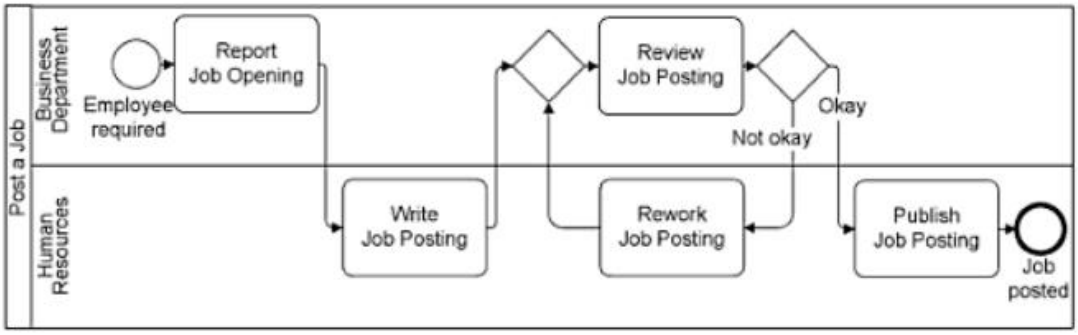


Figure 2.15: A simple BPMN Process diagram model (copied from [2, p. 16])

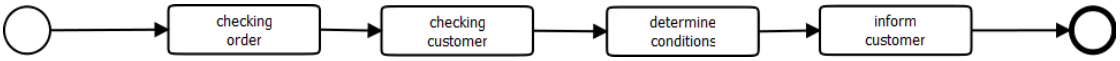


Figure 2.16: A simple internal process

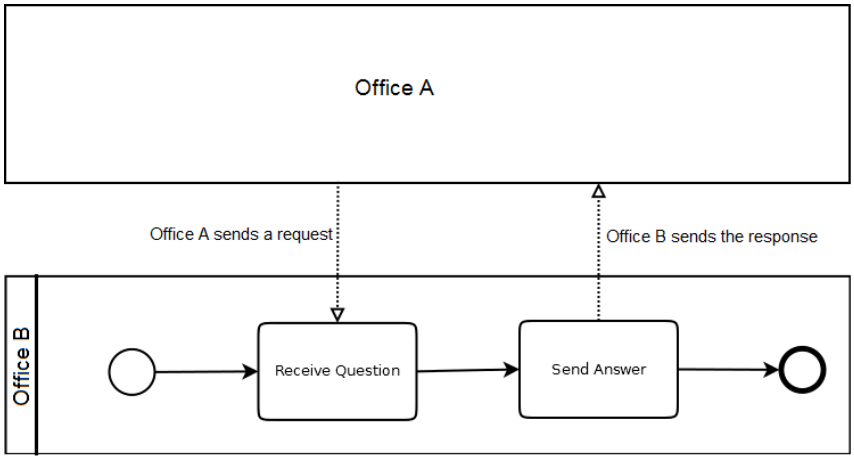


Figure 2.17: A simple abstract process

#### 2.3.2.2 Choreography diagram

The Choreography diagram was introduced in the current version of BPMN 2.0. In contrast to the Process diagram, here the priorities lie on the exchange of messages (communication traffic) which is performed by the participants of the process. These diagrams model so called “Message Exchange Pattern” (MEP). The most popular patterns are “sending messages”, “send a message synchronously”, “wait for a message”. The advantage is, that as accurate and detailed the model is, it can later serve as interface specification between the participants. Mostly the receiver, the sender and the message are usually shown graphically in a Choreography diagram [6].

Figure 2.18 shows an example. As seen in this Figure, most of the elements are already known and are described in subsection [2.3.1]. In the most cases, they also have the same

semantics. One element is new, it is called “Choreography task”. It is introduced, so that it is possible to model a MEP. A task exactly models a MEP between two participants. They have the same shape as an Activity (rounded rectangle) but in contrast to an Activity they consist of three components:

- The central field, which describes the task (as well as the Activity).
- A field above the central field, which specifies one communication partner and,
- a field below the central field, which specifies the other communication partner.

The different color of the communication partners, as shown in Figure 2.18, helps to distinguish between the recipient (gray background) and the sender (white background).

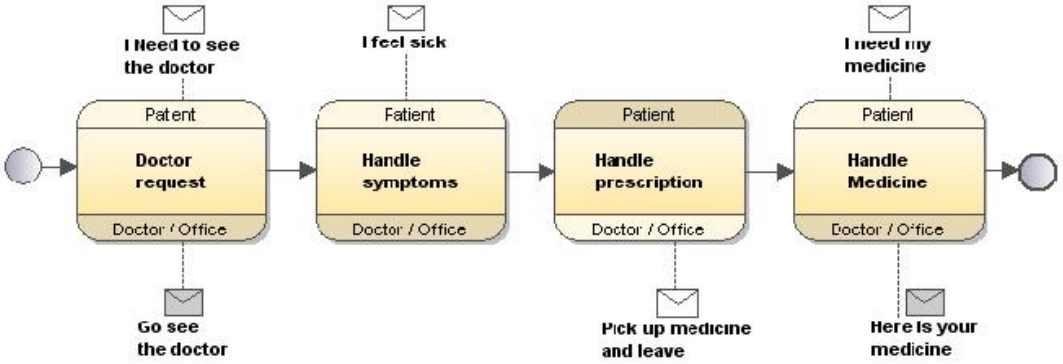


Figure 2.18: BPMN Choreography diagram (copied from [8])

A Choreography task can (like an Activity-task) be provided with different symbols below the task name. A more detailed description of the different Choreography tasks can be looked up in [6]. As already mentioned and shown in Figure 2.18, also other main core elements can be used in a Choreography diagram. In [6] a more detailed description can be found. There also an explanation is given, which elements can be used and which cannot be used.

#### 2.3.2.3 Collaboration diagram

In the previous two sections two types of diagrams which provide two different perspectives of a process are described. While the focus of the Process diagram lies on the individual task sequences in a business process, the focus of the Choreography diagram lies on the exchange of messages between different process participants (e.g., people, companies, software systems, etc.). However, it would be useful sometimes to see both views at once (either detailed or abstract). For this purpose the Collaboration diagrams were introduced in BPMN 2.0 [6].

Collaboration diagrams show the interaction of at least two or more different involved process-parties. It can show on the one hand the exchange of messages like the Choreography diagrams

[8] http://manual.altova.com/de/umodel/umodelenterprise/index.html?umchoreography_diagram.htm

and on the other hand the sequence of process, on the basis of a Process diagram. This means that Collaboration diagrams can combine and represent elements of the two kinds of diagram (Choreography diagram and Process diagram) [6]. Figure 2.21 shown an example. The advantage is that it is possible to integrate both views in this type of diagram. As a result designers, engineers and managers get a better overview of the business process.

Process participants are represented as a Pool, shown in Figure 2.21. Furthermore, the two different perspectives can be represented by modelers. The various processes can be modeled as black box (customer in Figure 2.21), public (newspaper publisher in Figure 2.21) or abstract [6].

In conclusion, Collaboration diagrams are important to get a good overview of the different perspectives of the business process. Depending on which processes are important or what is important for the modeler the model can be presented more or less detail [6].

#### 2.3.2.4 Conversation diagram

The Conversation diagram is used to represent an overview of the communication between the process-participants. Here an attempt is made to summarize the messages according to certain characteristics to abstract them. It is possible to get a very rough and quick overview of the communication behavior/dependencies of the involved parties. The messages are then centralized to a form called "Conversations". In this case it is not necessary to model every Message Flow, but to summarize related Message Flows to a global Message Flow [6]. The next two figures (Figure 2.19 and Figure 2.20) are intended to represent an example, how is looks like to abstract multiple Message Flows between two involved parties into a global Message Flow.

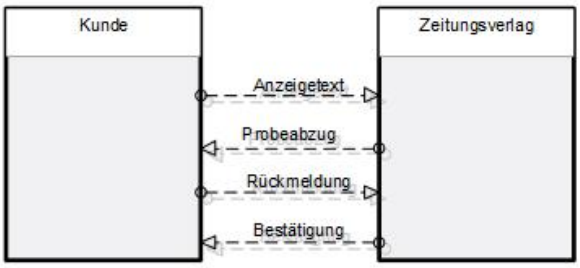


Figure 2.19: Normal Message-Flow (copied from [6, p. 38])

As seen in Figure 2.19, multiple Message Flows are sent back and forth. These Message Flows have one and the same challenge which is "giving up an announcement". And so it is possible to summarize the Message Flows to a global Message Flow, if it is interesting to show the dependency between the involved parties and not in which way the involved parties communicate actual. This is the purpose of a Conversation diagram.

In a Conversation diagram two new elements are introduced, the "Conversation Node" and "Conversation-Link". While the conversation node (represented by a hexagon as shown in Figure 2.20) summarizes Message Flows which are responsible for the same tasks, the conversation link connects the nodes with the involved process parties. This provides a quick overview of the communication dependencies of involved parties. The advantage here is that if there is a change

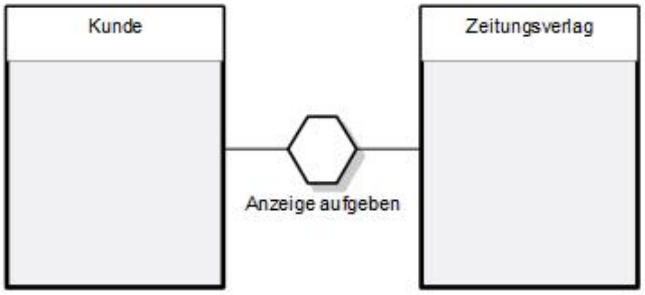


Figure 2.20: Conversation diagram (copied from [6, p. 38])

in the model, it is easy to see where dependencies are, and where changes are necessary. In a more detailed model this step would be much more complicated [6].

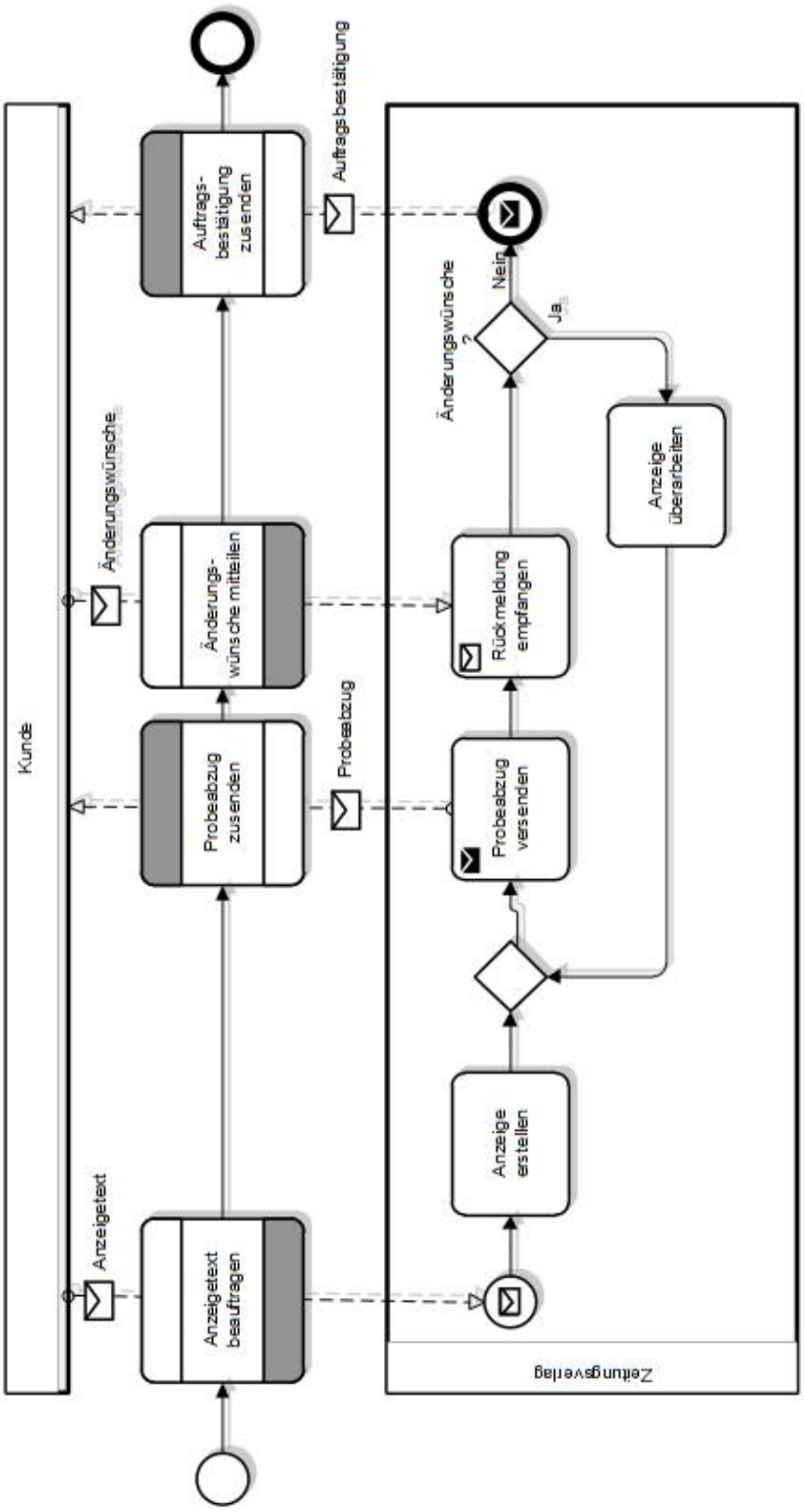


Figure 2.21: Collaboration diagram (copied from [6, p. 37])

CHAPTER 3

# Non-direct execution of BPMN 2.0 models

As in chapter 2 mentioned, the created BPMN models are not directly executable before the introduction of BPMN 2.0. Non directly executable means that the BPMN model first had to be transformed into an other executable format. For this reason, the selected approaches of non direct executable format for the feasibility of executing BPMN models are described in this chapter based on literature. These include the following approaches:

- Business Process Execution Language (BPEL) [3.1]
- XML Process Definition Language (XPDL) [3.2]

## 3.1 Execution via Business Process Execution Language (BPEL)

BPEL or Web-service BPEL (WS-BPEL) is an XML-based language. It is an imperative, domain-specific programming language that is used to define business processes and to make them executable. In contrast to BPMN, which is based on graphs, BPEL is a block-oriented language. It specifies the control flow through structured activities, it tries to integrate a rigid structure into a process [4] [15].

BPEL combines Web Services to new and more powerful services, which is also called orchestration. At the beginning BPEL was used rather for the execution of business processes (backend processes which could be executed without human interaction). Later BPEL was extended with two other approaches (BPEL4People and WS-Human Task) so that manual steps in the execution are allowed. However, BPEL is not really suitable for front-end processes [4] [15]. BPEL was mostly used (before the introduction of BPMN 2.0) for the execution of created BPMN models.

BPEL was introduced by major IT companies (e.g. IBM) in 2002. The current version (2.0) was introduced in 2008. The main goals for the development of BPEL were according to [4] [8]:

- Web Service as a basis
- XML for description
- collective approaches to orchestration and choreography
- control flow
- data processing
- identification (based on users)
- lifecycle
- transaction model (for specification, so that it is possible to intercept errors)
- modularization (for processes Web Services should be used)
- composition

A BPEL process specification and its main components are described in an XML document. The BPEL process consists of basic activities (atomic activities) and structured activities (determine the order in which the basic activities should be executed). Figure 3.1 shows a graphical representation of the different types of activities [15].

A brief description of the key elements is given in Table 3.1. For a more detailed description [15] or the BPEL specification[1] are recommended:

In the following Figure 3.2 a general structure of a BPEL process is shown.

In recent years, BPEL has become well established and it was a standard for technical and executable processes in the area of the Web Service technology. Many of the major manufacturers integrated it in their BPMS products (e.g., IBM, Oracle, Microsoft, Intalio, etc.). An advantage of BPEL is that BPEL process definitions are much more accurate and detailed as the BPMN process definitions (in the older version of BPMN) and so they can have elements which can be related for example to data manipulation, Web Service binding and other implementation aspects [10]. But a disadvantage of BPEL is that the specification of a graphical representation of business processes has been neglected and so BPMN has established itself as a standard for this area. Another disadvantage of BPEL is that neither the graphical representation of a BPEL model nor the BPEL code from a BPEL model can easily be maintained or modified without background knowledge.

It was possible to transform BPMN models of many BPMS providers to BPEL processes. This transformation tries first to locate certain patterns in a BPMN model, which are transformed into BPEL constructs. Thereafter, the patterns found are chained together or nested, and form the BPEL process [4] [14].

The following example should help to get a better understanding about the transformation from BPMN to BPEL. Figure 3.3 shows a graphical representation of a simple BPMN model. Here, task A is executed until condition (c = 0) is satisfied. This is also called as a simple nested loop. Once the condition is satisfied, task B is executed, and the process ends thereafter.

[1] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

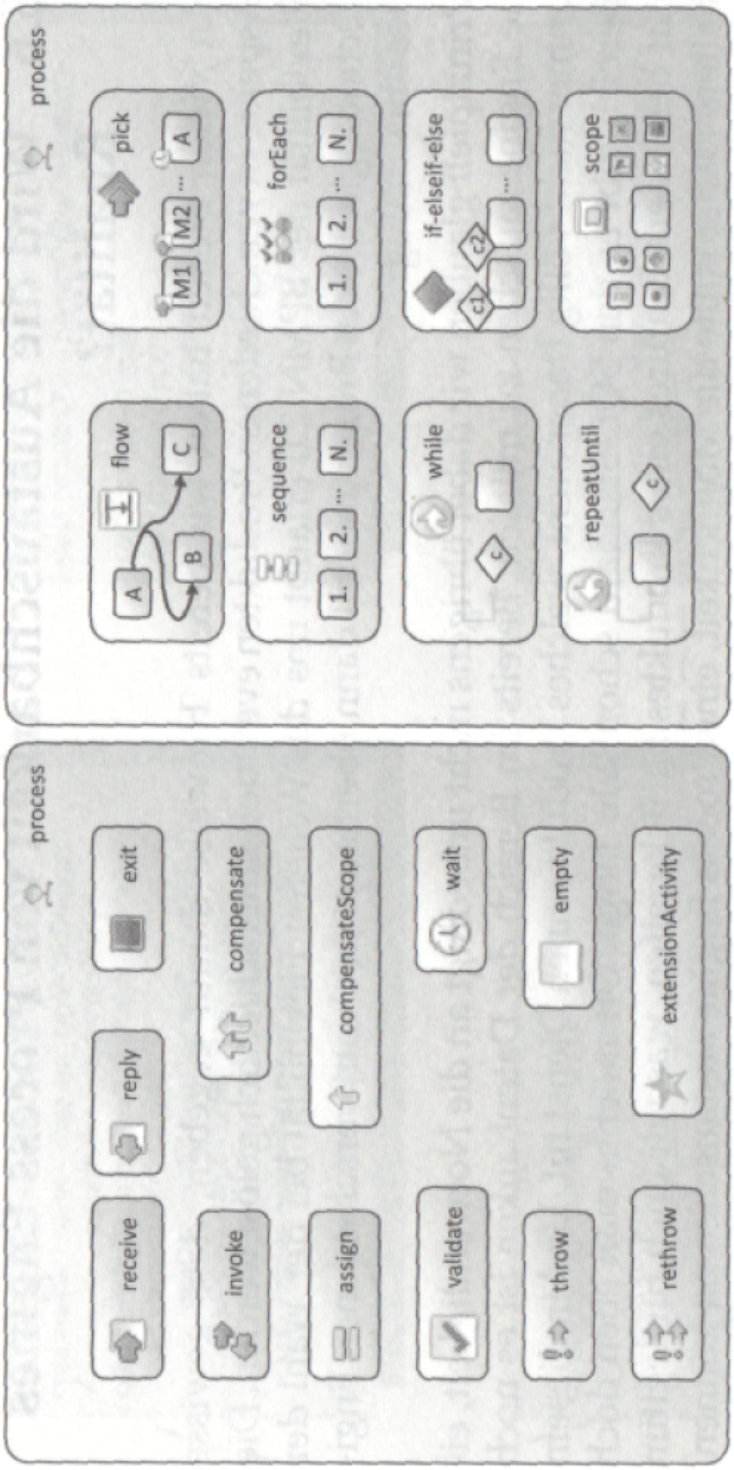


Figure 3.1: Graphical Representation of the basic and structured activities (copied from [4, p. 238])

| Basic activities | |
|---|---|
| **Name** | **Description** |
| <receive> | Reception of a message (providing a Web interface for external partners) |
| <reply> | Sending a message (providing a Web interface for external partners) |
| <exit> | Terminates the process instance |
| <invoke> | Calling Web Service |
| <compensate> | Starts the compensation of the immediately surrounding scope |
| <assign> | Manipulates and stores variable values |
| <compensateScope> | Compensates a certain scope |
| <validate> | Validates one or more variables against their schema definition |
| <wait> | Process or process branch is stopped for a certain period of time |
| <throw> | Cast an error |
| <empty> | Activity simply does nothing |
| <rethrow> | Possibility to treat an already casted error later |
| <exentsionActivity> | Allows the developer to define a new custom activity |
| **Structured activities** | |
| **Name** | **Description** |
| <flow> | All contained activities are performed in parallel |
| <sequence> | All contained activities are performed in sequence in a certain order |
| <while> | All contained activities are executed as long as the condition is met. The difference to the command <repeatUntil> is that the condition is checked before passing the loop. |
| <repeatUntil> | All contained activities are executed as long as the condition is met. The difference to the command <while> is that the condition is checked at the end of passing the loop. |
| <pick> | Not exactly waiting for one message, but several, which can be treated differently. |
| <forEach> | All contained activities are carried out for a specified number of passes. The number is set before the first execution and not changeable. Unlike the commands <while> and <repeatUntil> the activities can be executed simultaneously. |
| **Other important components** | |
| **Name** | **Description** |
| <scope> | To group activities logically and for example perform them transactional |
| <variable> | Specifies the data for a message exchange. Once a process receives a message, the message content is written to the corresponding variable. |
| <partnerlink> | An optional service which invokes the process or reverses it |

Table 3.1: Description of the BPEL key elements

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process id="PROCESS_EXAMPLE" xmlns:bpel="http://..." xmlns:xsd="http://..." targetNamespace="http://..."
    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
                 location="http://localhost:9898/ProReUseWebServices/services/BillService?wsdl"
                 namespace="http://ict.tuwien.ac.at/proreuse/BillService" />
    <bpel:variables>...</bpel:variables>
    <bpel:sequence name="ABC">
        <bpel:while>
            <bpel:invoke operation="A".../>
        </bpel:while>
        <bpel:invoke operation="B"..../>
    </bpel:sequence>
</bpel:process>
```

Figure 3.2: Simple general structure of a BPEL process

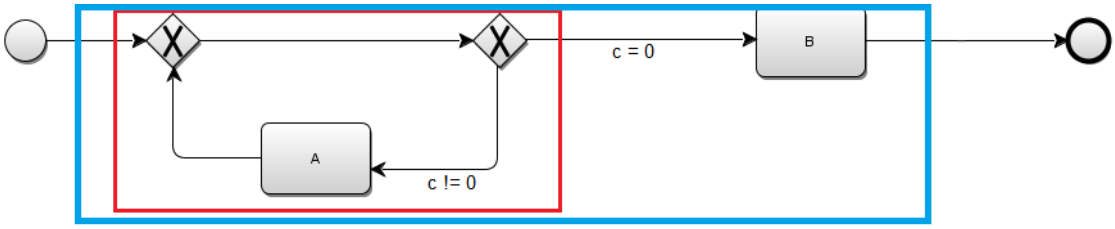


Figure 3.3: Example process in BPMN notation

In Figure 3.4 the related XML-code of this BPMN-model is shown simplified, in which the “startEvent”, “endEvent”, “task” and “exclusiveGateway” represent the nodes from the BPMN-model and the “sequenceFlow” represents the links between the nodes. Also in Figure 3.3 there are two sequence flows which have a condition. This means that in the XML-representation the sequence flow is expanded to include the “conditionExpression” element, shown in Figure 3.4.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions>
    <process id="PROCESS_EXAMPLE">
        <startEvent id="_2" />

        <task id="_3" name="A"/>
        <task id="_14" name="B"/>


        <exclusiveGateway  id="_5"/>
        <exclusiveGateway  id="_6"/>

        <sequenceFlow id="_7" sourceRef="_2" targetRef="_5"/>
        <sequenceFlow id="_8" sourceRef="_5" targetRef="_6"/>
        <sequenceFlow id="_13" sourceRef="_3" targetRef="_5"/>
        <sequenceFlow id="_16" sourceRef="_14" targetRef="_4"/>

        <sequenceFlow id="_12" sourceRef="_6" targetRef="_3">
            <conditionExpression><![CDATA[C != 0]]></conditionExpression>
        </sequenceFlow>
        <sequenceFlow id="_15" sourceRef="_6" targetRef="_14">
            <conditionExpression><![CDATA[C = 0]]></conditionExpression>
        </sequenceFlow>

        <endEvent id="_4"/>
    </process>

</definitions>
```

Figure 3.4: Example process from Figure 3.3 in BPMN 2.0 XML-representation is shown

The transformation from BPMN to BPEL (as mentioned above) tries to break down the model in blocks. Each block represents a pattern found. This is done until no more patterns can be found. After that the blocks are joined together. Figure 3.3 includes two different types of patterns. The first pattern at the bottom level in Figure 3.3 is marked with the red rectangle (it is the previously mentioned nested loop). The second pattern is one level above and is marked with a blue rectangle in Figure 3.3. The transformation of the found pattern from BPMN-model

to BPEL-XML-code is shown in Figure 3.5, where the transformation of the first pattern (nested loop) is shown in the red rectangle and the second pattern is shown in the blue rectangle.

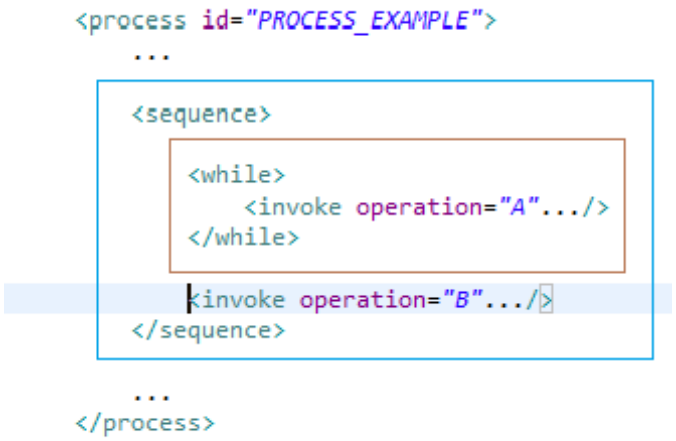


Figure 3.5: Example process from Figure 3.4 as block-oriented BPEL code

As can be seen in the two XML-representations in Figure 3.4 and Figure 3.5, there are not only differences between the element/tag-name. Also the structures of these XML-representations are different. In the XML-representation of BPMN it does not matter, if a Sequence, a Task or a Gateway is at the beginning, at the end or even in the middle (they are only must be between the "process"-tag) of the XML-representation (Figure 3.4). In the XML-representation of BPEL (Figure 3.5), there is a well-structured order between the elements. Further transformation patterns of BPMN to BPEL for control structures can be found in Figure 3.6.

A good explanation for the transformation from BPMN to BPEL can be found in [10]. In this paper an algorithm is presented which can translate BPD components to a block-structured BPEL process. Here, three different approaches are suggested for this transformation [10].

- For well-structured BPD components which can be directly mapped to BPEL structured activities.
- For non well-structured, but acyclic BPD components, which can be mapped to control link-based BPEL code.
- Using event-action-rules, if a component is neither non well-structured nor can be translated to using control links.

Basically it can be said that a transformation from BPMN to BPEL is possible in most cases. But during the transformation also problems can occur, because the two languages are conceptually very different. For example, many mistakes are created by the need to decompose the graphical BPMN models first into blocks, as BPEL works block-structured. The problem is, that there could be links within the block to another block (which is also known as the "goto" statement in programming languages). However, this is not allowed in blocks of BPEL, and so a mapping is in some case very difficult [4] [15].

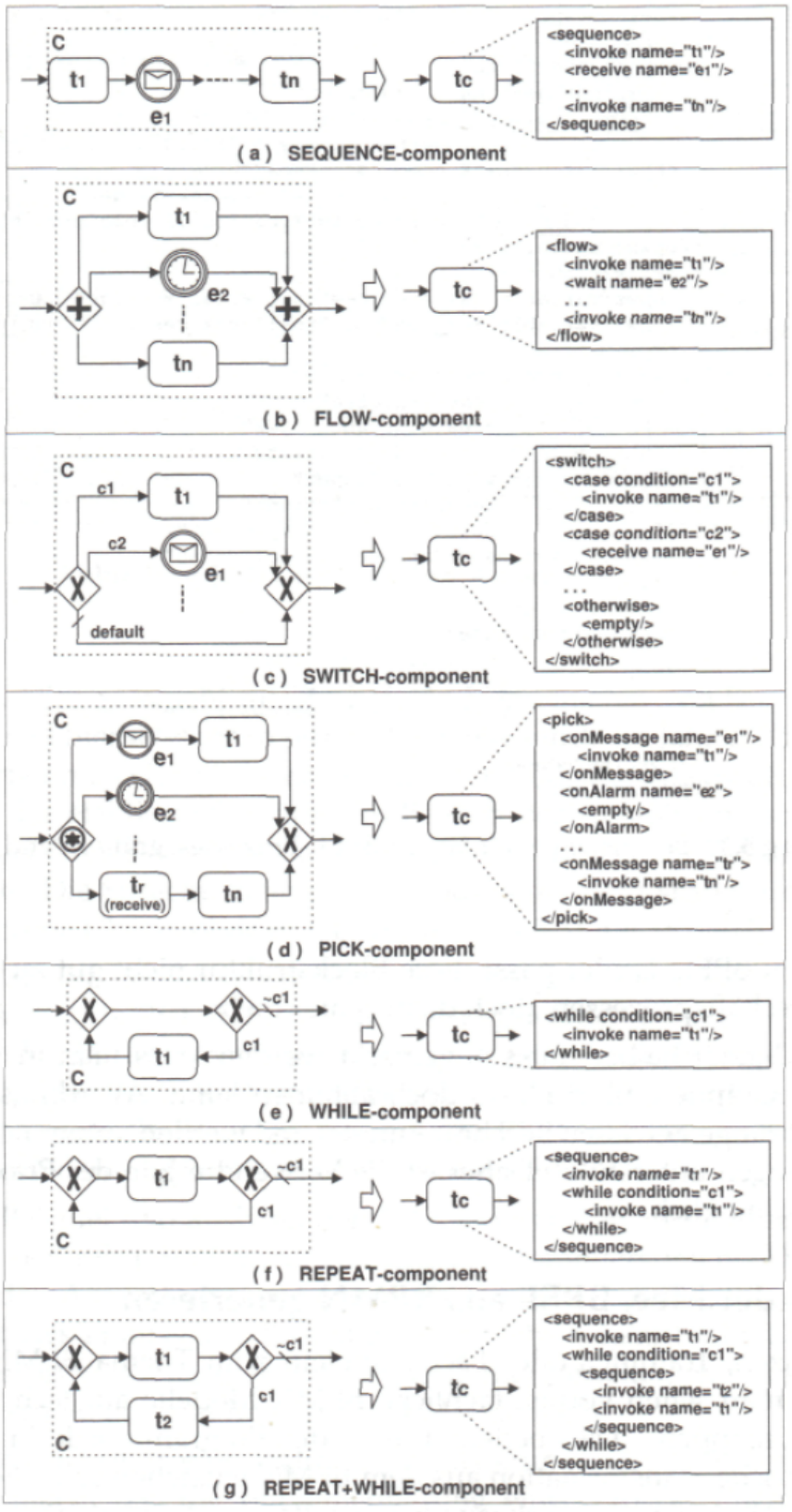




Figure 3.6: Basic transformation patterns (copied from [11, p. 5])

Further transformation problems are mentioned in the BPMN standard: *"Not all BPMN orchestration Processes can be mapped to WS-BPEL in a straight-forward way. That is because BPMN allows the modeler to draw almost arbitrary graphs to model control flow, whereas in WS-BPEL, there are certain restrictions such as control-flow being either block-structured or not containing cycles. For example, an unstructured loop cannot directly be represented in WS-BPEL."* [1, p. 445]

Also BPEL processes can be graphically represented in a BPEL designer, but this was never a main goal for the developers of BPEL and so the graphical representation was never standardized [13]. Therefore the graphically representation with a BPEL-editor-tool is usually not or very difficult to read for non-technical people (even for a very simple business processes). An example for the difference in graphical output between BPMN- and BPEL-editors is shown in Figures 3.7 and 3.8.

In both Figures 3.7 and 3.8 a normal ordering process of a company is represented by a supplier. In the ordering process is attempted (once a message "Artikel Bestellmenge" arrives from the "Lagerverwaltung") to determine the supplier who has the best price, short delivery time and where the product is available. Once the supplier is found, the item will be ordered by him. The obtained data from the supplier (available, delivery time) will be sent to the "Lagerverwaltung" and the process is finished, but if no supplier have this item available, an exception will be thrown and this message then is sent to the "Lagerverwaltung" and the process is finished.

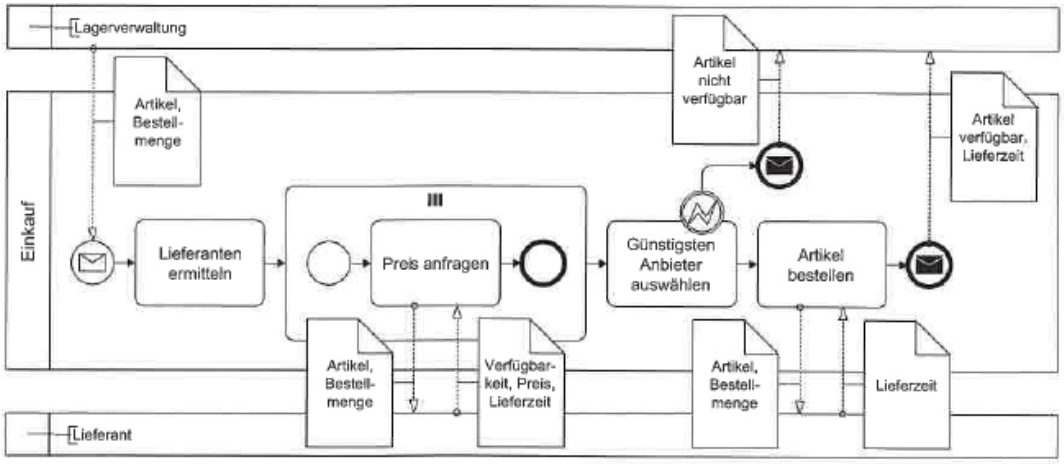


Figure 3.7: Graphical representation of a shopping process with a BPMN-editor (copied from [15, p. 36])

While in the BPMN model (Figure 3.7) a non IT-professional can understand the individual graphical steps without much explanation, the viewer needs for the graphical representation of the BPEL model (Figure 3.8) background knowledge of BPEL. Otherwise the BPEL model is very difficult or impossible to understand.

For example, the element in the BPMN model with "the letter" (circled with a line and also called "event") is mostly intuitively interpreted as a message. Depending on whether a dashed line with arrow goes in or out, this is also most intuitively interpreted as an incoming or outgoing

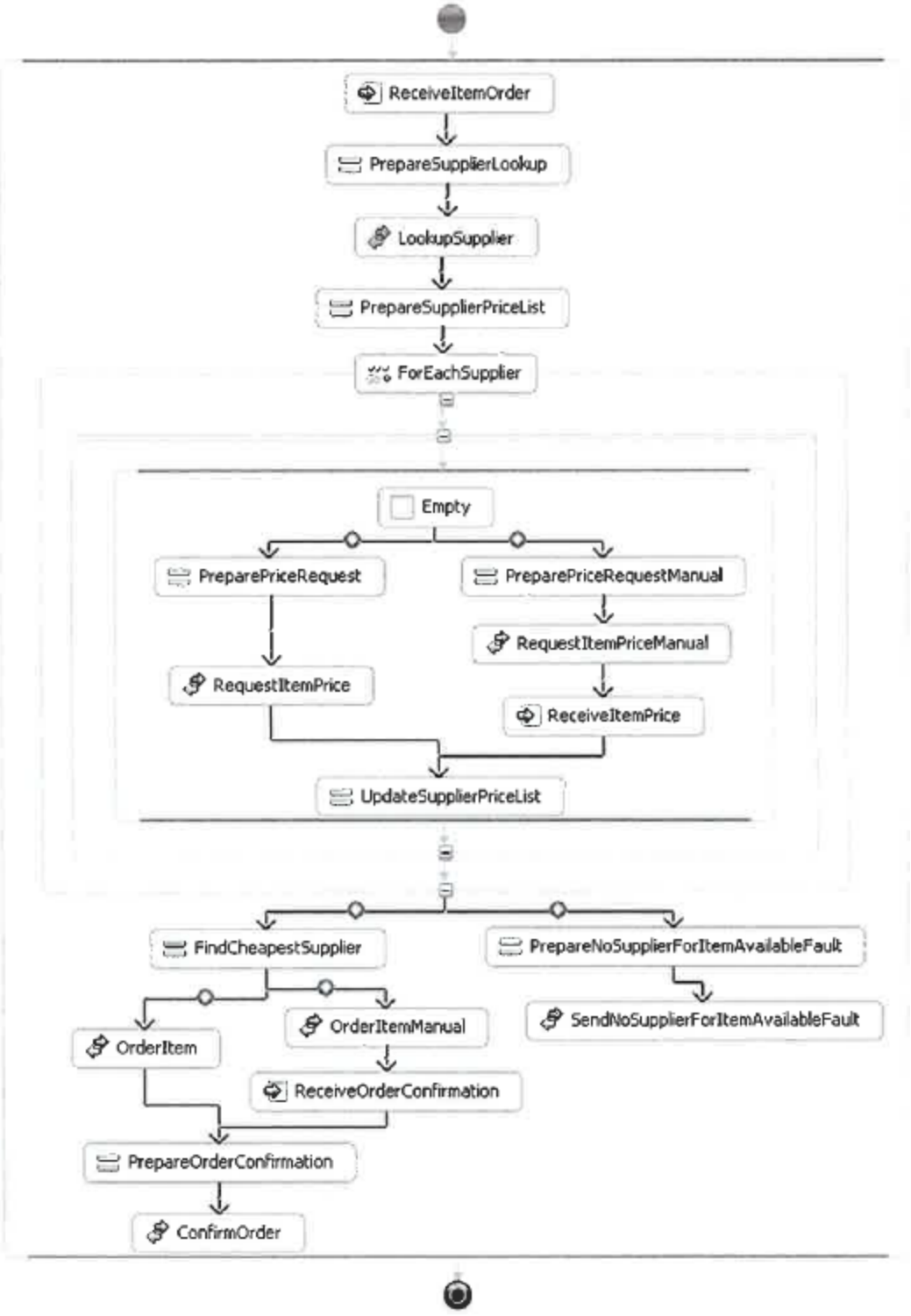


Figure 3.8: Graphical representation of the same shopping process (from Figures 3.7) with a BPEL-editor after transformation from BPMN to BPEL (copied from [15, p. 167])

message.

However, in contrast to the BPMN model the reader of the BPEL model has to know exactly which element is used for which purpose. For the case of a incoming message, the element is used where the arrow points into a rectangle. In the XML representation, both models are equally hard to read and can only be interpreted by experts.

This transformation of BPMN models to BPEL processes was up to the current version of BPMN 2.0 necessary, because additional attributes and execution semantics were missing for the execution of process models until BPMN 2.0. But since the introduction of BPMN 2.0, BPMN is now also a competitor in the field of execution, because now it has introduced an execution semantics and the missing attributes in addition to an XML-based interchange format, which is very similar to BPEL. The question is, if BPMN 2.0 can be established in these areas (exchange format and execution semantics) as a standard, is there still a need for BPEL? On the whole, the two specifications of BPEL and BPMN are similar, and many constructs in the XML representation are even quite similar in both languages. Differences between BPEL and BPMN are mainly according to ( [4] [15]):

- Control flow as a graph (graph-based BPMN, block-oriented BPEL)
- BPMN has no fixed link to Web Services and XML.
- Graphical representation

At this stage, BPEL is certainly needed, because it is integrated on the one hand in BPMS of many major manufacturers, and on the other hand BPEL currently meets certain requirements better (e.g., interoperability, interchangeability and execution) [15].

## 3.2 Execution via XML Process Definition Language (XPDL)

As mentioned above, in the previous version of BPMN (before BPMN 2.0 introduces) no attention was paid in execution, storage and exchange (with other tools) of business processes definitions (this was outside of the scope of the BPMN-standard), but only the graphical representation of business processes was important. That was the reason why XML Process Definition Language (XDPL) came to use.

XPDL was designed by Workflow Management Coalition (WfMC) in 1993 and is an XML-based language which describes business processes and related workflows. The goal of the developers was to store and exchange process models. So it should be possible to open, modify or execute models not only in the tool, in which the process was created, but also in any other tool that supports XPDL.[2,3,4]

At the beginning, XPDL was rather an exchange format, based on XML, it was seen more as an interface of Workflow Reference Models. *"XPDL sollte ein minimales Meta-Model beschreiben, das gemeinsam verwendete Konstrukte in einer Prozessdefinition identifizieren sollte [9, p. 36]."* In 2005, a new version (2.0) has been adopted, in which the focus was to provide an

[2] http://de.wikipedia.org/wiki/XML_Process_Definition_Language
[3] http://www.ebizq.net/topics/human_centric_bpm/features/7852.html
[4] http://www.wfmc.org/xpdl.html

XML-based interchange format for BPMN. It should consider both the graphical representation as well as the executable features [9].[4]

The latest version is at the moment version 2.2. *"The new XPDL 2.2 allows existing XPDL users and supporters to continue to exploit their investment in XPDL whilst extending BPMN support to encompass the new process modeling constructs of BPMN 2.0."*[4]

For ensuring a full support of BPMN, certain elements from BPMN had to be implemented in XPDL (e.g., events, gateways).

Figure 3.9 shows the Process Definition Meta-Model of XDPL [9]. In this Figure, there are some elements in the process meta-model of XPDL which also occur in the BPMN meta-model, for example Process, Activity and Sequence Flow. They also have the same semantics, e.g., that a sequence flow determines the order in which the activities should be executed. Noteworthy in the XPDL process model is, that the activities are divided into different types. A "Task/Tool" is an activity which requires no human interaction for execution. A "Block Activity" specifies sub-processes and are triggered by an "Activity Set". "Route Tasks" are responsible for join-split conditions of the control flow and "Events" were adopted, as mentioned above, from the BPMN specification.

XPDL is also used (like BPEL) for the execution of business processes. However, XPDL is graph-oriented and not block-oriented like BPEL. Due to the graph-based approach, BPMN models can be transformed better into XPDL. XPDL also offers a "one-to-one representation" for BPMN models in contrast to BPEL. Another difference is that XPDL has placed the focus on human activities, while BPEL (as mentioned in the previous section "BPEL") has the focus rather placed on the orchestration of Web Services. An advantage of BPEL over XPDL is that BPEL can handle error situations that XPDL currently does not take into account. The bottom line is that both languages have certain advantages and disadvantages.[2,3] [5]

Although both languages (BPEL and XPDL) have different approaches, it must not necessarily mean that only one of them can be used. In some BPM suites even both languages come to use (e.g., TIBCO or ActiveVOS). The new version of BPMN (2.0) tries to combine the advantages of both languages (XPDL and BPEL), to achieve the best possible execution of process models.[2]

[5] `http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20%282012-08-30%29.pdf`

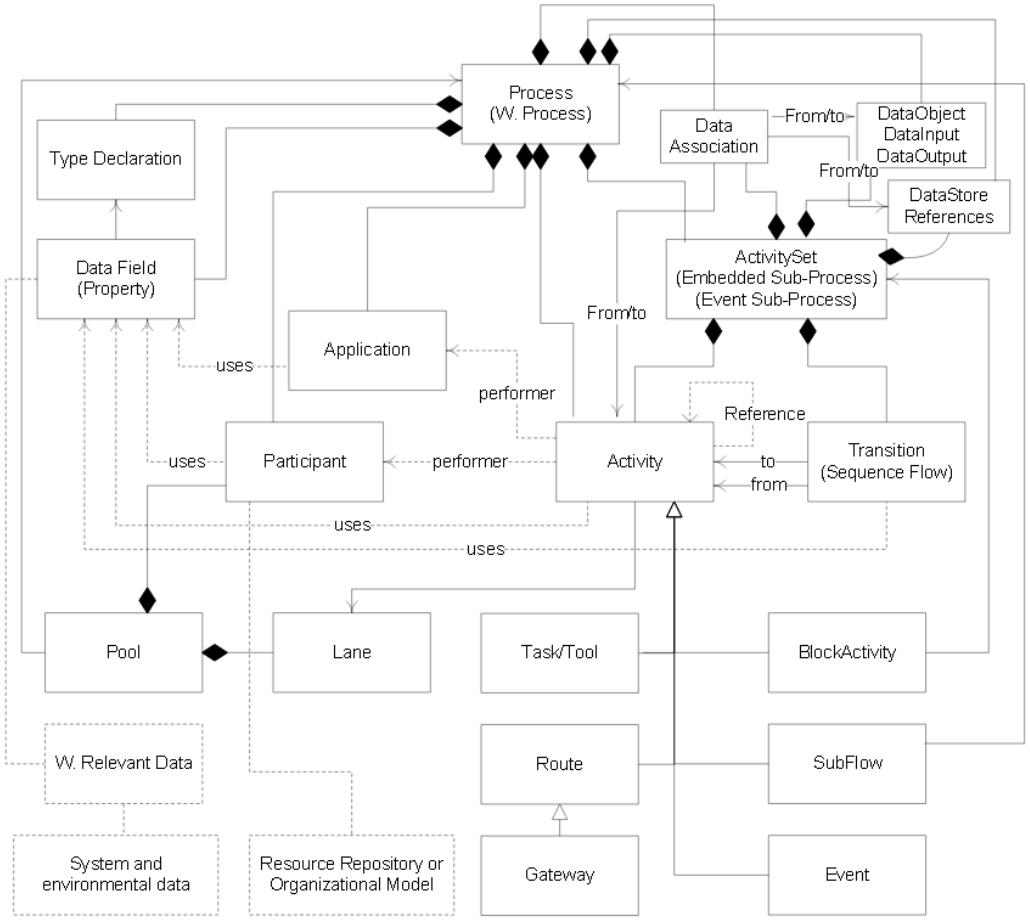


Figure 3.9: Process Definition Meta-Model (copied from [5])

CHAPTER 4

# Direct execution of BPMN 2.0 models

Since the introduction of BPMN 2.0, the created models of a business processes can be executed directly. This is made through a so-called Business Process Engine integrated in the different tools. In this chapter, the reader gets an explanation about Business Process Engines. Then a BPMN 2.0 model is defined which is used for the analysis of the various tools which provide a Business Process Engine.

A Business Process Engine or Process Engine is used for the execution of processes, therefore the processes which should be executed have to be precisely defined first (e.g., their order of tasks). This can be done in a template for processes or in a process definition. If the processes are properly defined (in a technical process model that provides no room for interpretation), then the Business Process Engine can complete all activities and their links with other activities in the course of the execution.

The people involved will be notified regarding the open tasks and the results will be processed. Moreover, the systems (internal and external) can be called via interfaces (also called service orchestration).

Figure 4.1 shown a graphical representation of a possible process execution with a Business Process Engine. Here, a technical process model is given to the Business Process Engine in the form of an XML representation. Based on this technical model, the Business Process Engine can control the business process. In this process, the Business Process Engine either informs the process participants about their tasks (in the diagram represented as “Human Workflow Management”), which they have to do in the course of the process or call internal or external services (in the diagram represented as “Service Orchestrierung”) via interfaces [4]. A more specific example with more detailed information is shown below.

The Business Process Engine decides on the basis of the technical process model which tasks or service calls to execute, and also the order in which they are made. A Business Process Engine can execute a process, but it is not always possible to allow the entire process to run fully automatically, because a business process may have tasks with human interaction or manual tasks.

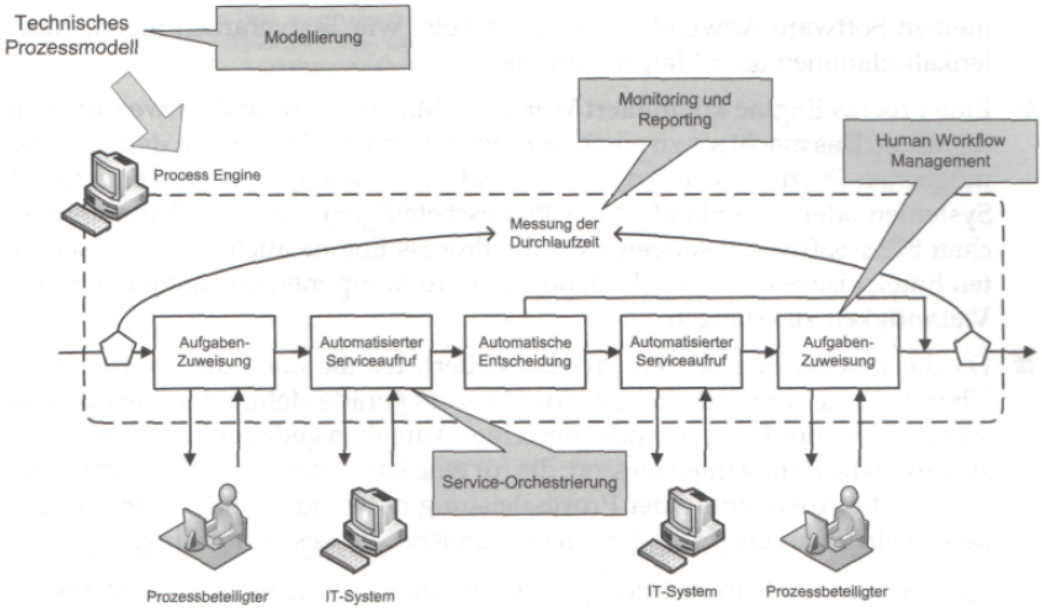


Figure 4.1: Process execution with a Business Process Engine (copied from [4, p. 7])

In a broader sense, the Business Process Engine can be seen as an interpreter (or a compiler) and the technical process model as code. The Business Process Engine can also provide important measures because the process control has a complete overview (e.g., where is the process at the moment, how long did it last, etc.). Processes that are currently in execution are also called process instances.

Basically, there are two different types of activities, human activities and machine activities. Business Process Engines are able to integrate these two types of activities during a process execution. A Business Process Engine is integrated in a workflow management system (WfMS) or a BPMS and so an important component of both. In contrast to a Business Process Engine, which is only responsible to carry out the execution of defined processes, the WFMS or BPMS has further functions to define processes and to monitor and control process instances of the Business Process Engine. This is also known as Business Activity Monitoring (BAM).

## 4.1 Defining an executable BPMN 2.0 model

Before investigating the individual tools with integrated Business Process Engines, the reader gets a fundamental understanding of how Business Process Engines work in general. For this reason, an example (called “Issuing Invoice”) has been selected/created, which is also used below for the investigation of the tools.

Before the process can run with the Business Process Engine, it must be created in a modeling tool. Most BPMS providers offer their own modeling tools for BPMN models. A modeling tool is mostly used by non-IT-experts. Therefore, their use is in general very simple. The individual elements of BPMN (e.g., tasks, gateways, events) are simply drawn and afterwards

connected. These elements are mostly offered in a palette left or right from the main windows (where the process is to be shown). By doing so, an XML-representation is created in the background.

Figure 4.2 shows the graphical output of a modeling tool. The output shows the selected example “Issuing Invoice”, which has a start-, an end-node, two Tasks, three Data Inputs and one Data Output. The first Task (“Create Invoice”) creates a invoice entity for two given inputs (Amount of money and a recipient Address). The second Task ( “Send Invoice”) sends the created invoice to a customer.

The two Tasks and nodes are connected with a Sequence Flow (line with arrow head), which shows the execution order. The Data Input and Data Output elements are connected with Tasks through a Messages Flow (dotted line with arrow head) and represent the data association.

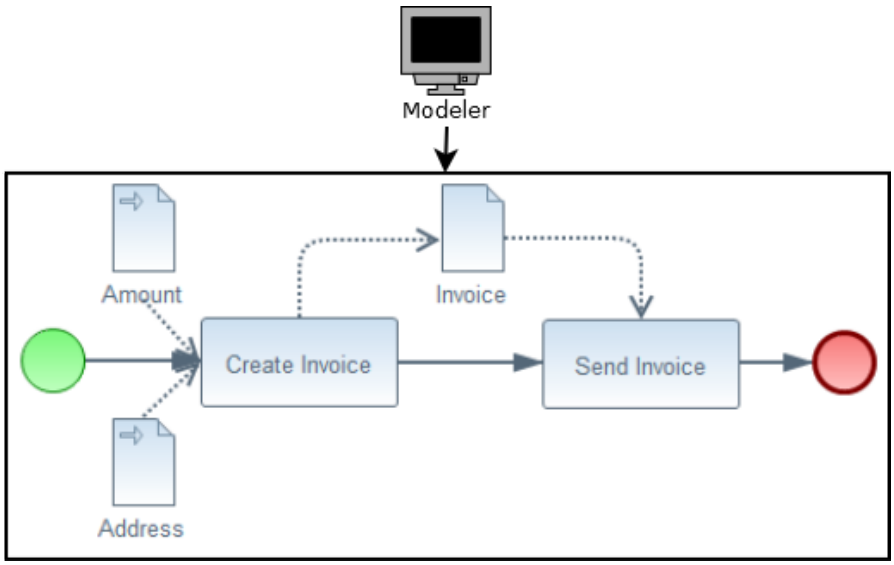


Figure 4.2: Modeler Output draw with Elcipse Modeler

As already mentioned above, an XML-file describing the “Issuing Invoice” is created in the background. A snippet of the generated XML representation is shown in Listing 4.1. To get a quick understanding about the function of the Business Process Engine. This generated XML is fed to the Business Process Engine for execution. First the Business Process Engine searches where the startEvent-tag (process entry point) is. At this startEvent tag, the Business Process Engine gets the information, which BPMN elements are connected by a Sequence Flow with the startEvent tag and which elements are to be executed next. This is possible due to the unique ID, which each BPMN element gets after generating the XML representation. With the help of this ID the Business Process Engine knows exactly which outgoing Sequences Flows connect to which Tasks, Events, Gateways, or rather which elements are to be executed next. As shown in Figure 4.2, the start node is connected to the Task “Create Invoice”. The same fact is represented in Listing 4.1 as code line 30. This Sequence Flow owns three attributes called “id”, “sourceRef”, and “targetRef”.

Due to the last two attributes (the unique ID’s “start” and “createInvoice” as shown in List-

ing 4.1 as code line 30) the Business Process Engine knows that the startEvent is followed by the Task “Create Invoice”. In BPMN 2.0 there are different types of Task, which are responsible for different functions. Such the Script Task, which can execute a piece of code. The Task “Create Invoice” is a Task without further specification (function). Therefore, it can be defined an Input Output Specification which defines the input and output of a Task (Listing 4.1, code line 14 – 16) and also Data Input and Data Output Association can be defined, which is used for the mapping of input and output. However, this make no sense for this type of Task, because the Task make nothing without further specification. A exactly specification is made below for this Task.

But back to our example, when the Task “Create Invoice” is finished, the Business Process Engine (based on the outgoing Sequence Flow) goes to the next Task “Send Invoice”. This is also a Task without further specification and it has the same behavior like the Task “Create Invoice”. After this Task is finished and the Business Process Engine tries to find the next BPMN elements based on the outgoing Sequence Flows. Since this was the last Task, the Business Process Engine achieves the endEvent and the process is completed/finished, and the Business Process Engine stops the execution.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <definitions xmlns="..."
3         xmlns:tns="..."
4         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6         id="..."
7         name="Issuing Invoice">
8
9    <process id="PROCESS_1" isClosed="false" isExecutable="true">
10
11     <startEvent id="start"name="Start Event"/>
12
13     <task  id="createInvoice" name="Create Invoice">
14       <ioSpecification id="IO1">
15           ...
16       </ioSpecification>
17       <dataInputAssociation>
18           ...
19       </dataInputAssociation>
20       <dataOutputAssociation>
21           ...
22       </dataOutputAssociation>
23     </task>
24     <task  id="sendInvoice" name="Send Invoice"/>
25       ...
26     </task>
27
28     <endEvent id="end" name="End Event">
```

```
29
30      <sequenceFlow id="sf_6" sourceRef="start" targetRef="
            createInvoice"/>
31      <sequenceFlow id="sf_7" sourceRef="createInvoice" targetRef
            ="sendInvoice"/>
32      <sequenceFlow id="sf_8" sourceRef="sendInvoice" targetRef="
            end"/>
33    </process>
34    ...
35 </definitions>
```

**Listing 4.1:** Generated XML representation of the Issuing Invoice.

But this BPMN model "Issuing Invoice" (shown in Figure 4.2) is still not executable, because there are still some attributes missing in the implementation, also the model has to be made more specific/accurate. In this example, the BPMN element "Task" was defined in such a way, that some of the required attributes for execution are missing.

Generally in BPMN 2.0, there are different types of Task elements which are in use for automatic execution. For the example "Issuing Invoice" the "Service Task" and the "Script Task" can be used. If the Service Task is used, the Task can be linked with a Web Service or with an automated application (like Java), which does the necessary work without human interaction. The Script Task is used to be directly executed by a Business Process Engine. The Script Task is only used to execute small and simple pieces of code. It is important that the modeler uses a scripting language which can be interpreted by the Business Process Engine [1].

The two Tasks can be distinguished graphically by an icon, which is located on the top left side. The various symbols are shown in Figure 4.3. In the generated XML code, they are represented differently as shown in Listing 4.2. Since both elements are specializations of the Task element, there are only a few attributes not contained in both.

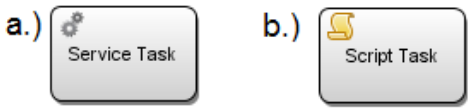


Figure 4.3: a.) Service Task, b.) Script Task

A Service Task exclusively uses the attributes "implementation" (shows what kind of service the Task is connected) and "operationRef" (where an operation reference can be specified). In the given example, the Service Task is associated with a Web Service (implementation = "##WebService") by the Web Service Description Language (WSDL) operation (operationRef="createInvoice"), which is defined by the interface (also defined in the XML representation) and related directly to the Web Service.

*"In contrast to a Service Task, a Script Task is designed for execution of simple tasks that are usually implemented in a scripting language. Therefore, it has an attribute "scriptFormat" instead of the attribute "operationRef". This attribute specifies the language in which the code is written and which the process engine has to interpret and execute."* [7, p. 4] In Listing 4.2 the language "groovy" is used for the Script Task.

*"Furthermore, the Script Task has another important attribute called "script", in which the script itself is implemented. This is the main part of the Script Task and is used during execution."* [7, p. 4]

```
1 <serviceTask id="createInvoice" name="Create Invoice"
    implementation="##WebService" operationRef="createInvoiceWS"
    >
2  ...
3 </serviceTask>
4
5 <scriptTask id="createInvoice" name="Create Invoice"
    scriptFormat="groovy">
6 ...
7 <script>...</script>
8 </scriptTask>
```

**Listing 4.2:** Difference in XML representation between Service and Script Task

For executing the example "Issuing Invoice" automatically, Tasks are replaced by Service Tasks (illustrated in Figure 4.4). The graphical difference between the Task symbols is minor, but in the XML representation now the missing attribute needed to link the Task with a Web Service or automated application are added.


Figure 4.4: Replace (General) Task with Service Task (copied form [7, p. 3]).

Listing 4.3 shows an example of how an XML representation of a Service Task connected with a Web Service looks like. The link to the Web Service has already been explained above, but there are other things that are relevant for execution. There is an Input Output Specification, which defines the input and output for the Service Task. In Listing 4.3 the two parameters and the result for the Web Service are defined as Data Inputs (code lines 4 and 5) and Data Output (code line 6). The Data Input and Data Output Association can map the Data Input and Data Output to local variables, which can be used then in the further course of the process. In Listing 4.3 two parameters (amount and address) are provided for the Web Service and the result is a invoice, which is stored in a local variable named "invoice".

In BPMN 2.0 Data Associations are used for moving data. For doing this, the BPMN 2.0 standard defines two ways. Listing 4.3 shows the simple variant in the Data Input and Data Output Association. Here the Data Input "amount" (code line 16) is directly mapped to a local variable "input_WS_amount"(code line 17). The second variant is the use of so-called Assignments. Here the Data Input is mapped to a local variable with the help of Expressions. For both variants it is important that the Data Input structure is the same as the local variable. The structure is defined in an Item Definition.

1

```
2 <serviceTask id="createInvoice_ServiceTask" implementation="##
    WebService" operationRef="createInvoice"  name="Create
    Invoice">
3  <ioSpecification>
4   <dataInput id="dataInputCreateInvoiceServiceTaskAmount"
       itemSubjectRef="createInvoiceInvoiceInputSoap"/>
5   <dataInput id="dataInputCreateInvoiceServiceTaskAddress"
       itemSubjectRef="createInvoiceInvoiceInputSoap"/>
6   <dataOutput id="dataOutputCreateInvoice"  itemSubjectRef="
       createInvoiceBooleanOutputSoap"/>
7   <inputSet>
8     <dataInputRefs>dataInputCreateInvoiceServiceTaskAmount</
         dataInputRefs>
9     <dataInputRefs>dataInputCreateInvoiceServiceTaskAddress</
         dataInputRefs>
10  </inputSet>
11  <outputSet>
12    <dataOutputRefs>dataOutputCreateInvoice</dataOutputRefs>
13  </outputSet>
14 </ioSpecification>
15 <dataInputAssociation id="dataInpAssoc_22">
16  <sourceRef>amount</sourceRef>
17  <targetRef>input_WS_amount</targetRef>
18 </dataInputAssociation>
19 <dataInputAssociation id="dataInpAssoc_23">
20  <sourceRef>address</sourceRef>
21  <targetRef>input_WS_address</targetRef>
22 </dataInputAssociation>
23 <dataOutputAssociation id="dataInpAssoc_25">
24  <sourceRef>output_WS_invoice</sourceRef>
25  <targetRef>invoice</targetRef>
26 </dataOutputAssociation>
27 </serviceTask>
```

**Listing 4.3:** XML representation of a Service Task.

Before a Web Service (like the WSDL specification in the "Issuing Invoice" example) can be used, it has to be imported by an import statement into the BPMN model. An example is shown in Listing 4.4, in code line 1.

The attribute "importType" defines the type of document, which should be imported. According to the BPMN 2.0 standard, at least the following three types of document have be supported [1]:

- Web Service Description Language 2.0 (WSDL 2.0)
- BPMN 2.0

- XML Schema 1.0

But also all other document types can be supported. The next attribute “location” defines the location of the imported service. The last attribute “namespace” defines the namespace of the imported service.

Also the Data structures can be imported from a WSDL specification and reused in BPMN. For example, *structureRef=“createInvoiceOperationResponse”* references to the structure defined in WSDL.

For access to the WSDL operation an interface is used, which defines the required WSDL operations. Listing 4.4 (code line 15) shows an example for an interface implementation. This interface has only one operation with the ID “createInvoice” defined. This operation is also referenced in the preceding Service Task (Listing 4.3, line code 2, operationRef=“createInvoice”).

The attribute “implementationRef” of the interface allows referencing a particular artifact in the underlying implementation technology represented by that interface (e.g., a WSDL porttype) [1]. The attribute “implementationRef” allows the same with an operation. Figure 4.5 illustrates the link between the operation of the interface and the operation of the WSDL.

For communication between the operation of the Web Service and the operation of the BPMN model interface, so-called messages (Listing 4.4, code lines 10 and 11) are used. Every operation needs two messages, for the request and the response. Respectively the message must be compatible with the structure in the Web Service (in the example with the WSDL) or with the structure in the automated application. This is done with the attribute *itemRef=“...”* of the message. The message structure is defined in the item definition (Listing 4.4, code lines 6 and 7). It is defined by the attribute “structureRef” shown above. The incoming and outgoing message of the operation is defined with the attributes “inMessageRef“ (input of the operation) and “outMessageRef“ (output of the operation) in the operation. This is shown in Listing 4.4, code lines 17 and 18.

According to the BPMN 2.0 Standard an operation has to have exactly one input message, while an output message is not necessarily required [1].

Figure 4.6 shows an example illustrating the relationship between the structure of the item definition, the structure of the WSDL specification, messages and the associated operation defined in the interface. In Figure 4.5 is shown that in the interface of the BPMN model XML-code an ID with “sendInvoice” exists, and this operation references the WSDL operation “sendInvoiceOperation”. Also it is shown that the WSDL operation has an output message with the name “sendInvoiceOperationResponse”. The structure of response is defined in the WSDL specification. Figure 4.6 shows that the structure is boolean (upper framing in red) and referenced to an item definition (with the ID “sendInvoiceOutputSoap”). This is made with the attribute “structureRef“ in the XML-code. The blue framing shows which message references which item definition (by the attribute “itemRef“). And the green framing shows which operation uses which message. Based on this information, the output of the operation (with the id “sendInvoice”) has to be boolean. If is tried to use another structure at the Item Definition with the ID “sendInvoiceInvoiceOutputSoap” (e.g. String), it receives an error message.

An good description with respect to the import, messages, item definitions and the use of Web Services can be found in the BPMN 2.0 standard. A specific overview can be found in the user guides of the particular tools.

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="..."
        namespace="..."/>

<!-- Item Definitions -->
<itemDefinition id="createInvoiceInputSoap" structureRef="
    createInvoiceOperation"/>
<itemDefinition id="createInvoiceInvoiceOutputSoap"
    structureRef="createInvoiceOperationResponse"/>

<!-- Messages -->
<message id="createInvoiceRequestMessage" itemRef="
    createInvoiceInputSoap" name="createInvoiceRequestMessage"/>
<message id="createInvoiceResponseMessage" itemRef="
    createInvoiceInvoiceOutputSoap" name="
    createInvoiceResponseMessage"/>


<!-- WSDL-Interface -->
<interface id="Interface_1" implementationRef="IssuingInvoice"
    name="IssuingInvoice">
 <operation id="createInvoice" implementationRef="
     createInvoiceOperation" name="createInvoiceOperation">
  <inMessageRef>createInvoiceRequestMessage</inMessageRef>
  <outMessageRef>createInvoiceResponseMessage</outMessageRef>
 </operation>
</interface>
```

**Listing 4.4:** XML representation of a Service Task

If the Service Task is connected with a Web Service, the client applications require a Web Service stub. These stubs handle the connection to the service implementations and they have to be generated or created according to the WSDL specification of the Web service. *"There are several frameworks available that allow automatic generation of a client stub from a WSDL specification during runtime. These stubs are then used to create service calls and hide the marshalling of objects and messages that are exchanged between client and server. A call to a Web service then acts like a local call of a procedure."* [7, p. 2]

As already mentioned, Service Tasks can be linked not only with Web Services, but also with automated applications. Figure 4.7 shows an example, where the Service Task is linked with a Java class. When calling an automated application, the attribute "implementation" of the Service Task has the value "##unspecified". This means that the implementation technology is left open and the tool provider can decide how to link the Service Task and the automated application. In Figure 4.7 the example was created with the tool "Activiti". Here the tool extends the Meta-Model of BPMN 2.0 with the attribute "class" and use this attribute to link a Java class, where the value given is equal to the Java class name (marked in red in Figure 4.7).

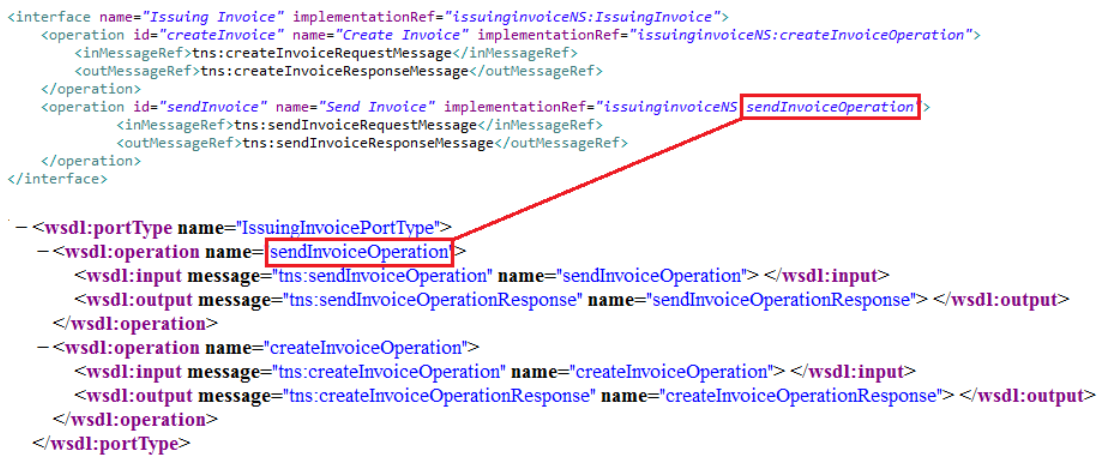


Figure 4.5: Illustration of link between interface operation and WSDL operation


Figure 4.6: Illustration of link between WSDL, message and item definition

According to BPMN 2.0 standard the tools (which are BPMN 2.0 compliant) have to support the link to Web Services or automated applications (like Java). While the link to Web Services is explicitly mentioned in the standard, the link to alternative implementation technologies for automated applications is open and, therefore, it is tool dependent. However, the BPMN 2.0 standard also leaves open how exactly Web Services are linked with Service Tasks. There is indeed the attribute "operationRef" which is linked to the invoked operation, but it is not described in detail. Several attributes that are necessary for an automatic invocation (like WSDL encod-

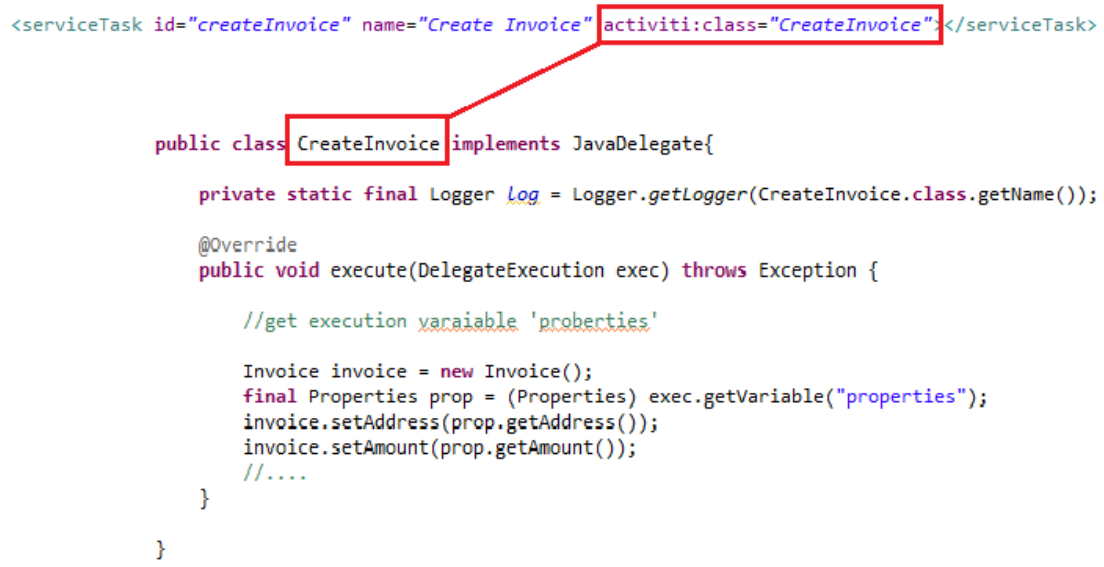


Figure 4.7: Service Task linked with Java created with the tool Activiti

ing or WSDL namespaces) are not mentioned at all. Thus the tools can interpret the BPMN 2.0 specification in various ways, which results in different XML-representations of the same model.

Since the link from a Service Task to an automated application is not specified at all, the implementation is tool dependent. Therefore, the investigation of the tools lead to a problem with the portability and so the Service Tasks of the running example "Issuing Invoice" are linked with Web Services (WSDL) which are supported by the BPMN 2.0 standard. If a tool also supports the link with an automated application, it was also tried to execute the example with this approach. For this purpose, the programming language "Java" was used, since it was supported by the majority of the tools.

A running example of an "Issuing Invoice" has been created and was used for the investigation of the different tools. The complete created example can be found in Listing A.1.

## 4.2 A Pitfall in BPMN 2.0

This section is based on the paper [7].

When the Tasks are replaced in the "Issuing Invoice" example above with Service Tasks an adapted model is created (shown in Figure 4.8). The graphical representation is similar to the model of Figure 4.8 and the model of Figure 4.2, but Figure 4.8 represents a business process with embedded service call specifications, which are only seen in the XML representation.

So far, everything looks plausible and the process model more or less straight-forward. However, simply replacing the Tasks from Figure 1 with Service Tasks actually leads to a model that is not compliant with the BPMN 2.0 standard. This very standard imposes an additional constraint on Service Tasks, i.e., it does not allow more than one input set and a single Data Input

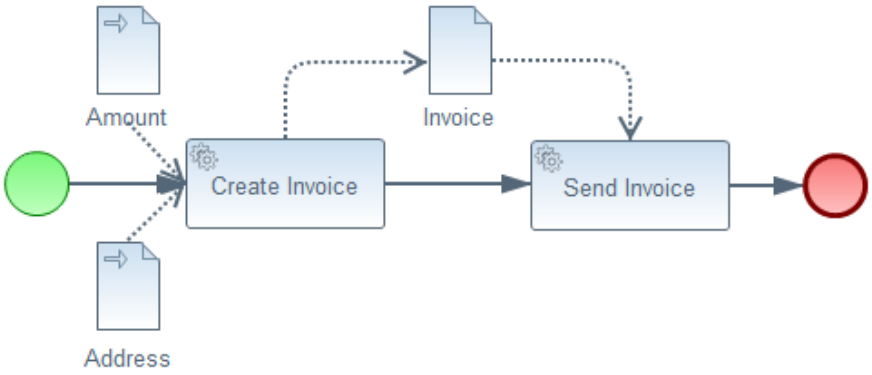


Figure 4.8: Adapted BPMN 2.0 model of the Issuing Invoice example process with Service Tasks

per Service Task. Even this simple process model, however, has two input parameters for the Service Task implementing the creation of an Invoice.

This constraint can be found in the BPMN 2.0 standard on page number 158 at the Service Task. [1]

This constraint might be a reaction to Web Service specifications where the SOAP message only consists of one body part. However, the Service Task itself should not be involved with the actual encoding and message passing of a Web Service and should only deal with the parameters themselves. If the Service Task is supposed to receive an already fully constructed message object with all parameters embedded, it would make it difficult for somebody without technical background to understand how this service is invoked. Furthermore, the specification above does not only hinder invoking Web Services but also simple procedure calls in a programming language, which BPMN 2.0 is capable of. Since the encoding of parameters to message objects (in case of WSDL/SOAP) is not at the same level of abstraction as input parameters of methods/operations, this should not be mixed up. So, the definition of parameters of a Service Task should not involve any information about the actual encoding or how they are transported. This should be part of the Service Task and its specification (which BPMN already handles through an import statement and additional attributes).

The question now is how to solve the problem and still remain standard compliance.

There are several possibilities to accomplish this, all of which involve changes to the BPMN process as well. One possible solution is to change the WSDL specification of the Web service so that only one parameter is passed as input. This would make the service itself compatible with the BPMN Service Task specification, but would still require additional mapping techniques in the BPMN process to combine all inputs into one single input. Furthermore, it puts an uncommon constraint on the development of Web services, and only a small subset of available services can be integrated into business processes modeled in BPMN 2.0.

Figure 4.9 shows a modified process model for our running example, which uses a wrapper so that the Service Task does not directly have more than one input. In our example, a Script

[1] `http://www.bpmi.org/`

Task is used as the wrapper, because a Script Task can have, in contrast to a Service Task, more than one input set and more than one Data Input. The wrapper Script Task is added before the Service Task, and it receives all the inputs which the Service Task needs for its operation. The function of the Script Task is to prepare the inputs for the Service Task in such a way that they are combined into a single input. In this case, all inputs are combined into one map. Viewed from a higher level, however, this modified model is not really appropriate any more for a business expert, since preparing input for another Task is not really a business-relevant Task per se.

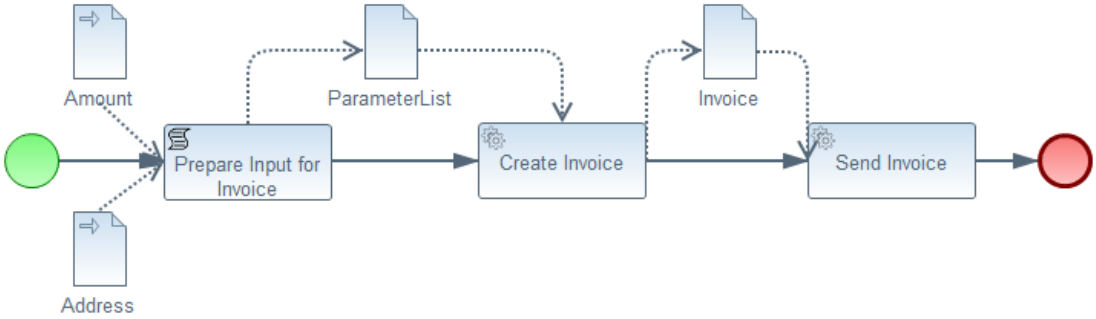


Figure 4.9: Modified BPMN 2.0 model of the Issuing Invoice example process with a wrapper

Listing 4.5 shows the XML representation of the Script Task used as a wrapper for the Service Task Create Invoice. Since both Script Task and Service Task are derived from the more general Task specification in the BPMN 2.0 Meta-Model, their syntax is similar. Both have an Input Output Specification and associated Data Input and Data Output Association. This Input Output Specification defines the parameters of the Script Task which are combined in a map variable.

But the main task is shown in the attribute “script”. Listing 4.5 shows that in this attribute first an object of type “InvoiceParameterList” is created and stored in a variable “ParameterList”. The object “InvoiceParameterList” has two operations — one to add parameters and one to delete parameters. The behavior is similar to a Java Hashmap, and is required to operate with WSDL specifications, since there simple or well-defined parameter types are allowed only. The two inputs of the Script Task are then added to the created object via the “addParameter” operation. In a last step, the object is set as the process variable that corresponds to the Data Output of the Script Task.

```
1  <!-- Script Task -->
2    <scriptTask id="PrepareParameters" scriptFormat="groovy">
3     <ioSpecification id="InputOutputSpecification_5">
4          <dataInput id="dataInputScriptTaskAmount"
              itemSubjectRef="inputScriptTaskAmount" name="
              Amount"/>
5          <dataInput id="dataInputScriptTaskAddress"
              itemSubjectRef="inputScriptTaskAddress" name="
              Address"/>
```

```
6           <dataOutput id="dataOutputScriptTask" itemSubjectRef=
               "outputScriptTask" name="Result"/>
7             <inputSet id="InputSet_5">
8               <dataInputRefs>dataInputScriptTaskAmount<
                   dataInputRefs>
9           <inputSet>
10          <inputSet id="InputSet_9">
11              <dataInputRefs>dataInputScriptTaskAddress<
                   dataInputRefs>
12          <inputSet>
13          <outputSet id="OutputSet_5">
14              <dataOutputRefs>dataOutputScriptTask<
                   dataOutputRefs>
15          <outputSet>
16        <ioSpecification>
17        <dataInputAssociation id="DataInputAssociation_8">
18           <sourceRef>amount</sourceRef>
19        <dataInputAssociation>
20        <dataInputAssociation id="DataInputAssociation_10">
21           <sourceRef>address<sourceRef>
22        <dataInputAssociation>
23        <dataOutputAssociation id="DataOutputAssociation_9">
24           <targetRef>ParameterList<targetRef>
25        <dataOutputAssociation>
26        <script>
27
28           def parameters = new at.ac.tuwien.ict.proreuse.
                webservices.icas.icasservices.InvoiceParameterList
                ();
29           parameters.addParameter("amount", amount);
30           parameters.addParameter("address", address);
31
32           execution.setVariable("ParameterList", parameters);
33      <script>
34    <scriptTask>
```

**Listing 4.5:** XML Representation of a Script Task used as a Wrapper.

The approach described in Listing 4.5 shows how a complex object of type InvoiceParameterList is created and other parameters, which have been specified as the input of the Script Task PrepareParameters, are attached to it. After all parameters have been added in the "script" section, the resulting variable is set in the execution engine. Note, that the name of this resulting variable is set according to the output specification in the dataOutputAssociation. Thus the input and output of the Script Task can be specified according to the script in the "script" section, and the input/output behavior of the Script Task is fully specified. Since the script directly oper-

ates in the execution engine, it would also be possible to set other variables. Some frameworks automatically define an additional output specification for them.

However, this approach has a drawback. It requires the Web service to only take a single parameter, where all other parameters are embedded in one single object. A better solution would be to call the Web service directly from a Script Task or maybe an attached Java class. The idea is to hide the call to the Web service in a Script Task and thus avoid the unusual constraint with just one parameter. Figure 4.10 gives an example for the graphical notation for such a wrapper. As it shows, there is no indication for a Web service call given, which makes it harder to understand. Another major advantage is that this process model fits better the modeled process of our running example. A business expert can still see the essential tasks in this model and not an extra one irrelevant for the process. The specializations of Task to Script Task and to Service Task, respectively, are only visualized as small icons and should not be irritating to the business expert.

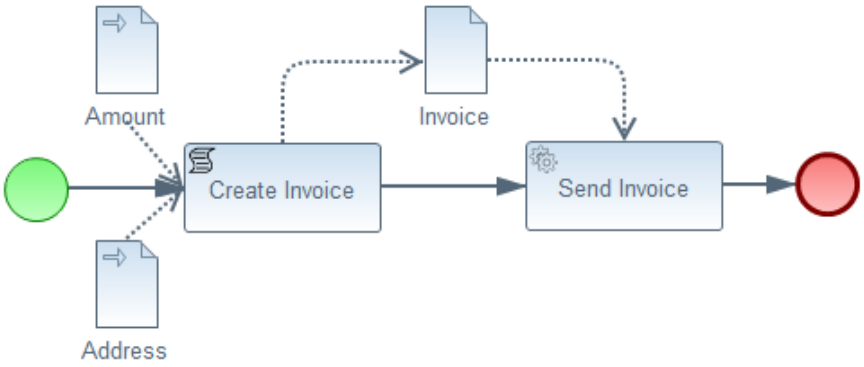


Figure 4.10: Modified BPMN 2.0 Model of the Issuing Invoice example process with Script Task instead of Service Task

Another approach would be to use Java implementations as a back-end for Script Tasks and to employ them to call the Web services. Since Java classes are executed in the same runtime environment, they can access all available properties and values during the execution. In this case, the definition of the Data Input and Data Output could technically be omitted (which makes the overview of the business process difficult, of course). The BPMN 2.0 standard does not provide a clear specification on how other implementations can be attached to a Script Task or Service Task, but simply states that other implementations can be included. This means that the technical implementation of calling back-end Java classes is permitted, though not entirely specified. Using this method has the advantage that during execution there is full control on all variables and definitions but that the graphical notation does not fully represent the business process.

Both techniques, calling the Web service directly from a Script Task or a reference Java class, require to make manual Web service calls. This means that there needs to be an implementation available for the client part of the service call. This is in contrast to the normal Service Task where most frameworks support automatic generation of the required client stub from the referenced WSDL file. Since most frameworks already provide such a generation, their imple-

mentation can be reused. However, the call to create the client stub has to be provided manually. In addition, it is possible to use external frameworks to generate all necessary classes on the client side.

In addition, the BPMN 2.0 standard provides another possibility for mapping Data Objects, the Assignment construct in the DataInputAssociation. This construct allows mapping one Item Definition to another one and thus enables BPMN users to specify how locally defined Data Objects can be transferred to (parts of) input parameters. This is similar to what we accomplish with the ScriptTask but has the disadvantage that only short expressions can be used and that still the input parameter of a Service Task has to be defined as a more complex structure that embeds the local objects into one single object. Additionally, this approach requires the process designer to know of the existence and state of local Data Object definitions and thus the transparency of the process is obscured.

Since all these possible solutions rely on active tool support, it is necessary to state that the BPMN 2.0 standard does not allow for a compliant solution to this only-one-parameter problem it poses, so that solutions are tool-specific. However, the general idea behind the wrappers should work for all available frameworks. In addition, the frameworks provide the means to call the Web services and thus it might be necessary, depending on the framework, to provide client stub implementations or the shared objects in the same class path.

## 4.3 Execution of BPMN 2.0 models with Business Process Management Systems

In this section, some tools with integrated Business Process Engines are investigated. Most tools extended their Business Process Engine so that they can offer a complete BPMS. Table 4.1 shows the various found tools listed and some important properties.

Certain criteria are defined:

- The tool must be standard conform with BPMN 2.0.
- The tool must either be open source or offer a trial version.
- The tool must support modeling and direct execution of BPMN 2.0 models without transformation.

According to this selection, the following tools are investigated below.

- Activiti [4.3.1]
- jBPM [4.3.2]
- Bonita BPM [4.3.3]
- Camunda BPM [4.3.4]

[2] "+" == commercial | "+/-" == partially commercial | "-" == open source

[3] no longer available

Table 4.1: Overview of the founded tools with integrated business process engine

| BPMS | BPMN 2.0 Model direct execution | Older version of BPMN | BPEL execution | commercial [2] | Trial version | Web-service link | Java link |
|---|---|---|---|---|---|---|---|
| AristaFlow | | | | + | | X | X |
| Appian BPM Suite | | | | + | | X | |
| IBM WebSphere Process Manager | | | X | + | | | |
| Fuege (AquaLogic BPM) | | | | + | | X | X |
| inubit BPM | X | | | + | X | X | X |
| Fujitsu Interstage | | X | | + | | X | X |
| Oracle Business Process Manager Suite | X | | X | + | X | | |
| SAP NetWeaver BPM | | | X | + | | X | X |
| SoftProject X4 BPM Suite | X | | | + | | X | |
| Pegasystems | | | | + | | | |
| Software AG webMethods BPMS | X | | | + | | | |
| Stolz IT Consulting Process Engine | | | | + | | | |
| Tibco iProcess Suite | | X | | + | | X | X |
| Vitria iBPMS | X | | | + | | X | |
| IYOPRO BPMS | X | | | + | | | |
| Sydlee Seed | X | | | +/- | X | X | X |
| Intalio BPMS | X | | X | + | X | X | |
| ActiveVOS | X | | X | + | | X | |
| jBPM | X | | | - | | X | X |
| Activiti | X | | | - | | X | X |
| Bonita BPM | X | | | +/- | | X | X |
| Camunda BPM | X | | | +/- | X | | X |
| Roubroo [3] | X | | | - | | X | X |

- Sydlee Seed [4.3.5]
- Inubit BMP [4.3.6]

For the investigation of the selected tools the created "Issuing Invoice" business process is used. Figure 4.9 shows the business process. This business process is imported and tried to execute in the selected tools. If the process is not executed, the BPMN model is adapted so that the BPMN model can be executed.

### 4.3.1 Activiti

*"Activiti is a light-weight workflow and Business Process Management (BPM) Platform targeted at business people, developers and system admins. Its core is a super-fast and rock-solid BPMN 2 process engine for Java. It's open-source and distributed under the Apache license. Activiti runs in any Java application, on a server, on a cluster or in the cloud. It integrates perfectly with Spring, it is extremely lightweight and based on simple concepts."* [4]

Activiti is written in Java and consists of a number of specific components and applications. These are used in the whole process from definition to the execution of a Business Process. Figure 4.11 shows the various components and applications. Table 4.2 gives their description [12].

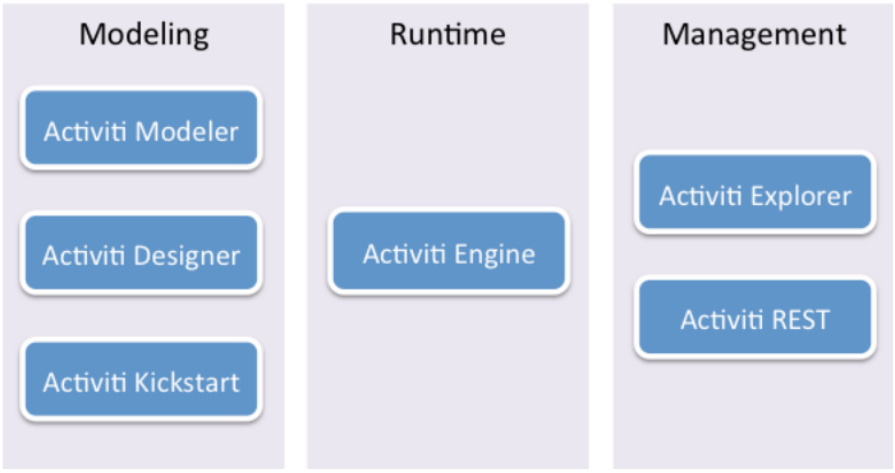


Figure 4.11: Different components and applications of Activiti (copied from [5])

[4] http://activiti.org/
[5] http://www.activiti.org/components.html

| Name | Description |
|---|---|
| Activiti Modeler | *"a web-based modeling environments for creating BPMN 2.0-compliant business process diagrams. This component is donated by Signavio."* [12, p. 5] |
| Activiti Designer | an Eclipse plugin for the creation of BPMN 2.0 business process diagrams [12]. |
| Activiti REST | *"a web application that provides REST interface on top of the Activiti engine."* [12, p. 5] |
| Activit Explorer | a web application used to access the Activiti Engine at runtime. This allows task management, gathering reports, monitoring the system or the state of process instances, starting process instance, aso [12]. |
| Activiti Engine | *"the core component of the Activiti to stack that performs the process engine functions."* [12, p. 5] |

Table 4.2: Description of Activiti components and applications.

The current version of Activiti is 5.14 and was introduced at the end of 2013. It can be downloaded from [6].

A more detailed overview and description of Activiti and its components can be found in the book [12] or on the Activiti homepage [4]. On this homepage also the user guide can be found, as well as various examples and explanations of the different BPMN elements. Also the installation process of Activiti and the implementation in Eclipse is shown. Activiti needs the following components:

- JDK 6+
- Eclipse (Indigo or Juno)
- Apache Tomcat

After the installation and configuration, which are described in the user guide Chapter 1 (Introduction) and 2 (Getting Started), the business process "Issuing Invoice" can be imported as an example in Eclipse (shown in Figure 4.9). After the import, the business process can be opened with the Activiti Designer.

Figure 4.12 shows the opened business process with the Activiti Designer. It is immediately apparent that in the process the Tasks, Sequence Flows and Events appear, but all Data Inputs and Data Outputs are missing in this representation. The reason is, as shown in Figure 4.13 (which lists all possible graphical representations for Activiti) that the Activiti Designer does not provide a graphical representation for Data Input and Data Output.

As already mentioned above, when a diagram is created (graphically), also an XML representation is generated in the background with all BPMN elements, which are needed. And so the Data Inputs and Data Outputs are still in this XML representation. Figure 4.14 shows a part of the Script Task "Prepare Input for Invoice" in XML-representation. However, such an XML representation is very difficult to read for non IT-experts.

[6] `http://www.activiti.org/download.html`

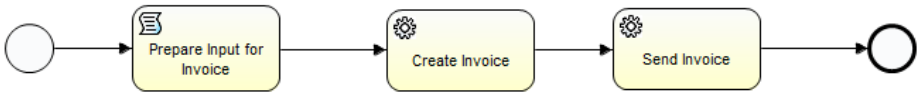


Figure 4.12: Business process Issuing Invoice opened with Activiti Designer in Eclipse

Another problem with Activiti Designer is that a Service Task cannot link with a Web Service (e.g., WSDL-file), but only with Java classes, expressions or delegate expressions (shown in Figure 4.15). To import Web Service (e.g., WSDL) and integrate Data-Input and Data-Output in a BPMN model, one can use the two following possibilities. One possibility is to implement these functionalities directly in the generated XML representation. The second possibility is to use another modeler in Eclipse (e.g., Eclipse Modeler for BPMN2 Project/Diagram).

In Activiti there are two ways to define DataInputAssociation and DataOutputAssociation (using expression or using the simplistic approach), which are responsible for the mapping of the required Data Inputs or Data Output. For both approaches, examples can found in the user guide of Activiti [7]. In the example "Issuing Invoice" the simplistic approach is used. Here it is important when the Service Task is linked to a WSDL specification (as in the example "Issuing Invoice") that the attributes of a DataInputAssociation or DataOutputAssociation from a Service Task ("sourceRef" and "targetRef") have to refer to an Item Definition or Property. In the example they refer to Item Definitions.

However, it needs to be considered that the structure and the naming of the WSDL input/output message types and especially their subtype definitions have to be equivalent to the Item Definition of the corresponding BPMN Data Input/Output association. In case of the input association the target reference and in case of the output association, the source reference has to have the same name.

The reason is shown graphically in Figure 4.16. The Service Task "Send Invoice" is associated with the WSDL operation "sendInvoiceOperation", which has an input and an output. These are defined by Messages. The messages are in turn linked to an Item Definition (link is marked in blue in Figure 4.16), in which the structure of the Message based on the WSDL is defined (green marked in Figure 4.16). Therefore an object "invoice" is passed. The red mark in Figure 4.16 shows, why the Item Definition ID (which is used in Service Task "Send Invoice" for the Data Input) in Activiti have to have the same name as in WSDL defined. If another name is chosen the process throws an error message (org.apache.cxf.binding.soap.SoapFault) during the execution and the process stops.

Otherwise, the imported business process "Issuing Invoice" should be executable without further adjustments. Listing A.2 shows the complete executable BPMN 2.0 model of the "Issuing Invoice".

An advantage of Activiti is that the mentioned constraint (which allows only one parameter for the Service Task, explained in the section 4.2) is not implemented in Activiti and so also the BPMN 2.0 model shown in Figure 4.8 is executable in Activiti, but the BPMN model is not

[7] http://www.activiti.org/userguide/

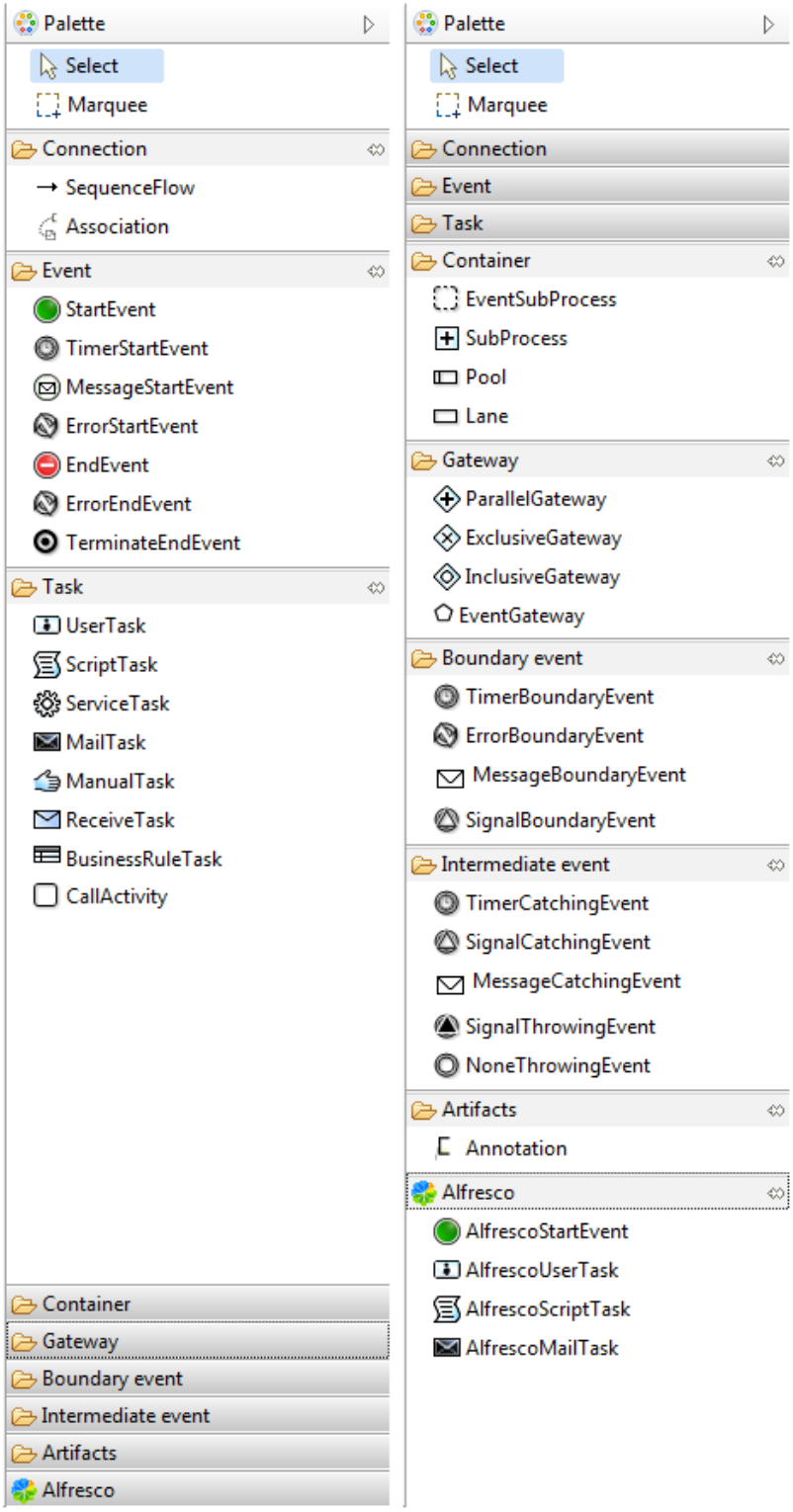


Figure 4.13: All elements provided by the Activiti Designer in Eclipse. The Figure contains two screenshots with different opened rubric of the Activiti Designer

```
<!-- Script Task -->
<scriptTask id="prepareInput" name="Prepare Input for Invoice"  scriptFormat="groovy">
    <ioSpecification id="InputOutputSpecification_5">
        <dataInput id="dataInputScriptTaskAmount" itemSubjectRef="inputScriptTaskAmount" name="Amount"/>
        <dataInput id="dataInputScriptTaskAddress" itemSubjectRef="inputScriptTaskAddress" name="Address"/>
        <dataOutput id="dataOutputScriptTask" itemSubjectRef="outputScriptTask" name="Result"/>
        <inputSet id="InputSet_5">
            <dataInputRefs>dataInputScriptTaskAmount</dataInputRefs>
        </inputSet>
        <inputSet id="InputSet_9">
            <dataInputRefs>dataInputScriptTaskAddress</dataInputRefs>
        </inputSet>
        <outputSet id="OutputSet_5">
            <dataOutputRefs>dataOutputScriptTask</dataOutputRefs>
        </outputSet>
    </ioSpecification>
    <dataInputAssociation id="DataInputAssociation_8">
        ...
</scriptTask>
```

Figure 4.14: The XML-representation of Script Task “Prepare Input for Invoice” from the business process Issuing Invoice

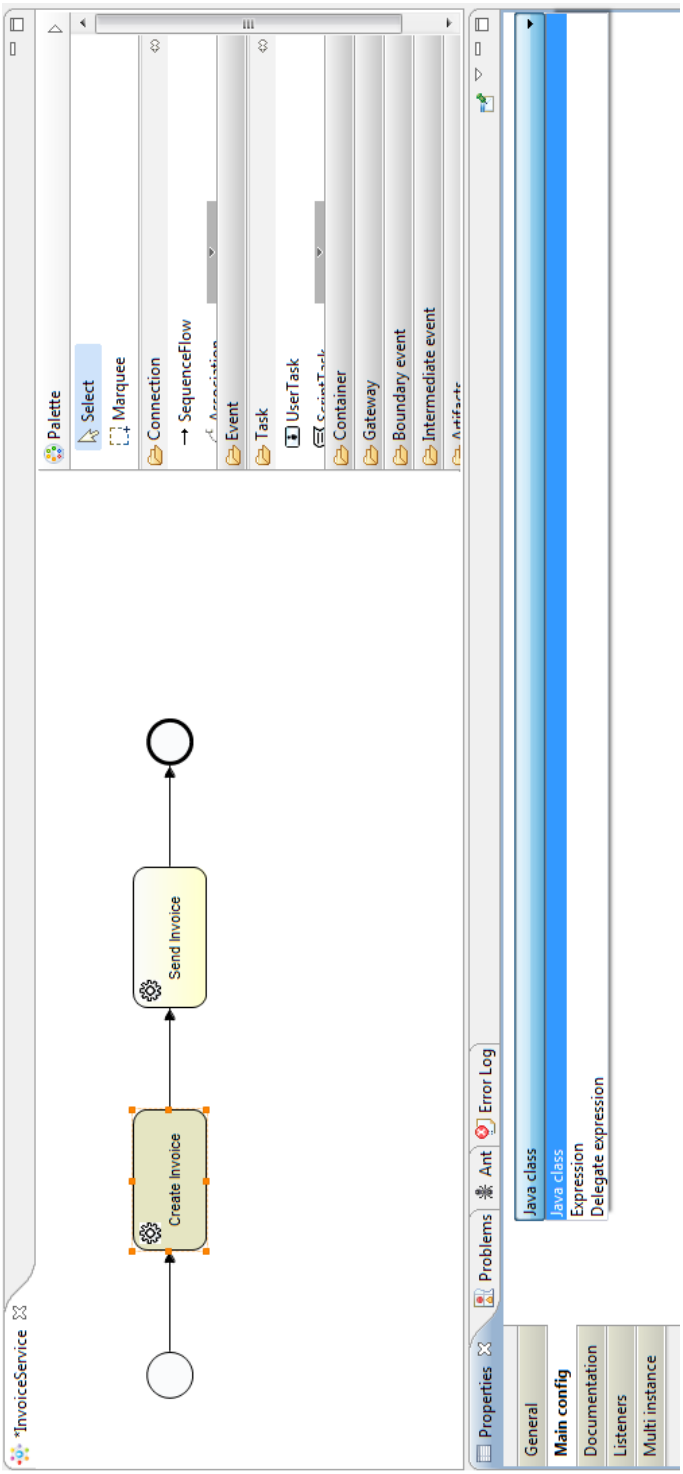


Figure 4.15: Service Task properties in Activiti Designer

Figure 4.16: XML representation of the Service Task “Send Invoice” and the link to WSDL

standard compliant. This has at least an advantage for the graphical representation of large and complex BPMN models, because the Script Task is no longer needed for preparing the inputs of the Service Task. The fully generated and executable XML representation of the example "Issuing Invoice" in Activiti with integrated Web Service and more than one Parameter passed to the Service Task can be found in Listing A.1.

Another advantage in Activiti is that during the execution of the business process, the stub classes for the Web Services are automatically generated.

As mentioned above, there is in addition to the link of the Service Task with a Web Service also the possibility to link the Service Task with an automated application. But how this link looks like is not defined in the BPMN standard. In Activiti this is a link to a Java class and, therefore, two types of Service Task (which is used for automatically execution) are distinguished in Activiti.

- Web Service Task
- Java Service Task

The graphical representation of these Tasks is the same. The difference can only be seen in the generated XML representation. Listing 4.6 shows the different XML representations. The attribute of both Service Tasks were already explained in section 4.1 above. The generated and executable XML representation of the example "Issuing Invoice" with integrated Java can be found at Listing A.3.

```
1 <!-- WSDL -->
2 <serviceTask id="createInvoice" name="Create Invoice"
     implementation="##WebService" operationRef="tns:
     createInvoice">
3 ...
4 </serviceTask>
5
6 <!-- JAVA -->
7 <serviceTask id="createInvoice" name="Create Invoice" activiti
     :class="CreateInvoice">
8 ...
9 </serviceTask>
```

**Listing 4.6:** Generated XML representation of a Web- and Java-Service Task

However, this link with Java is tool-specific, because is done via the attribute "class" in Activiti and this is not BPMN 2.0 Standard compliant. In Activiti, there are four different ways to connect Java Service Task with Java Logic: [7]

- *"Specifying a class that implements JavaDelegate or ActivityBehavior*
- *Evaluating an expression that resolves to a delegation object*
- *Invoking a method expression*

- *Evaluating a value expression"*[7]

In Listing A.3 the most common way to integrate a Java classes is shown. Here it is important that the Java class has the same name as the value of the attribute "class" in the Java Service Task. Listing 4.7 shows an example of a Java class that is integrated into a BPMN model. In addition to the Java class name, an interface called "org.activiti.engine.delegate.JavaDelegate" is implemented at the used Java class. This interface provides the required operation "execute" which is called during the execution from the Process Engine. In this operation, the needed logic for this Service Task is implemented. The passed Parameter (exec) of the type "DelegateExecution" is used to store or receive the important process instance information. Listing 4.7 shows that first a object "Invoice" is created in the Java Class with the name "CreateInvoice" (which is the same used in the example "Issuing Invoice"). After that the required input variables (amount and address) are assigned to the invoice object. Subsequently, the invoice object is stored back to the process instance variable (exec) and the Service Task is finished. The Process Engine searches for the next BPMN element following the outgoing Sequence Flows.

```
1
2 public class CreateInvoice implements JavaDelegate{
3
4   private static final Logger log = Logger.getLogger(
       CreateInvoice.class.getName());
5
6   @Override
7   public void execute(DelegateExecution exec) throws Exception
       {
8
9     //get execution varaiable
10    Invoice invoice = new Invoice();
11    invoice.setAddress((String) exec.getVariable("address"));
12    invoice.setAmount((Float) exec.getVariable("amount"));
13
14    exec.setVariable("invoiceProperties", invoice);
15  }
16 }
```

**Listing 4.7:** Java class for the Service Task "Create Invoice"

A more detailed description of all four types for integrated Java can be found in the current user guide of Activiti (under the section Java Service Task [7]).

Another type of executable Task in Activiti is the Script Task. As mentioned in the previous Section 4.2, the Script Task is a specialization of the Task and, therefore, it is very similar to the Service Task. The difference between and the important attribute of the generated XML representation for the Script Task is explained in the section 4.1. In Activiti, "groovy is used as scripting language, which is very similar to Java.

For generating and starting a process instance one needs another Java class. Listing 4.8 shows a possibility how such a Main Java class can look like. First, in code line 12 the Process

Engine is constructed by the command “ProcessEngines.getDefaultProcessEngine()“. With this command a file named “activiti.cfg.xml“ is searched in the project. In this file the configuration of the Process Engine is defined. Several examples of how this file may look like, can be found in the user guide of Activiti. [7]

If the Process Engine is defined, the next step is shown in code line 17. Here the Activiti Services Repository is created, which provides operations for manipulating and managing deployments and process definitions. It can be used to deploy one, or multiple BPMN 2.0 XML files, or other resources. This process is also called deployment. The created files are uploaded to the Process Engine and there all resources are audited and parsed before they are stored into database. If an error occurs, an error message is thrown. If no error message is thrown then the deployment is known for the Process Engine and every process (which is in this deployment) can be started. In code line 21 the statement is used to deploy the BPMN model. [7]

After that, the Runtime Service is created. It is responsible for the storage and using of the process variables. In addition, the Runtime Service can be used for queries on the process instances and the execution and it is also used if a waiting process instance is to be continued (e.g. due to a wait state). But it is also used to start a process instance, this is shown in Listing 4.8 (code line 31). In this statement a process definition with the name “IssuingInvoice” is searched in the loaded BPMN model. Furthermore, the required variables are passed, which are needed for the business process. After that, a process instance of the process definition is started.[7]

```
1
2
3  public class Main {
4
5    /**
6     * @param args
7     */
8
9    public static void main(String[] args) {
10
11     //create Activiti process engine
12     ProcessEngine processEngine = ProcessEngines.
          getDefaultProcessEngine();
13
14
15     // get Activiti services Repository, Runtime
16     // operations for managing and manipulating deployments and
          process definitions
17     RepositoryService reposService = processEngine.
          getRepositoryService();
18
19
20     //Deploy the process definition
21     reposService.createDeployment().addClasspathResource("
```

```
        diagrams/IssuingInvoice.bpmn").deploy();
22
23      //starting new Process Instance of process definitions
24      RuntimeService runtiService = processEngine.
            getRuntimeService();
25
26      Map<String, Object> variables = new HashMap<String, Object
            >();
27      variables.put("amount", 13f);
28      variables.put("address", "Am Hof 544, 1020 Wien");
29
30      //Start process instance
31      ProcessInstance procInstance = runtiService.
            startProcessInstanceByKey("IssuingInvoice", variables);
32
33    }
34  }
```

**Listing 4.8:** Main java class for starting a business process in Activiti.

### 4.3.2 jBPM

jBPM is a business process management suite (BPMS) developed from JBoss. It is an open source project for the execution of worklfows. It is written in Java and it supports multiple process languages natively (e.g., BPMN 2.0, jPDL, BPEL, etc.). jBPM is used to model, execute and monitor business processes throughout their life cycle.[8]

*"The core of jBPM is a light-weight, extensible workflow engine written in pure Java that allows you to execute business processes using the latest BPMN 2.0 specification. It can be run in any Java environment, embedded in your application or as a service."*[8]

*"Since jBPMN 5 the tool offers open-source business process execution and management, including*

- *an embeddable, lightweight Java process engine, supporting native BPMN 2.0 execution*
- *BPMN 2.0 process modeling in Eclipse (developers) and the web (business users)*
- *process collaboration, monitoring and management through the Guvnor repository and the web console*
- *human interaction using an independent WS-HT*[9] *task service*
- *tight, powerful integration with business rules and event processing"*[8]

[8] `https://www.jboss.org/jbpm`
[9] WebService-HumanTask

The latest version at the time of this writing is jBPM 6, which was introduced in late 2013. The current version of jBPM can be download from [10]. The example "Issuing Invoice" was tested in the previous version 5.4. jBPM can also be run on other application servers, servlet containers or SE environments since version 5. This is possible, because jBPM is no longer dependent on the JBoss application server.[8]

jBPM is composed of various components. The most important are:[11]

- *"Core engine: used to execute your business processes"*
- *"Eclipse plugin: graphical modeling, development and debugging of your processes"*
- *"Designer: web-based editing of your processes"*
- *"Console: web-based management of your processes, user tasks, reports, etc."*
- *"Other components that the jBPM project integrates with"*[11]

Figure 4.17 shows the various components and their interaction.

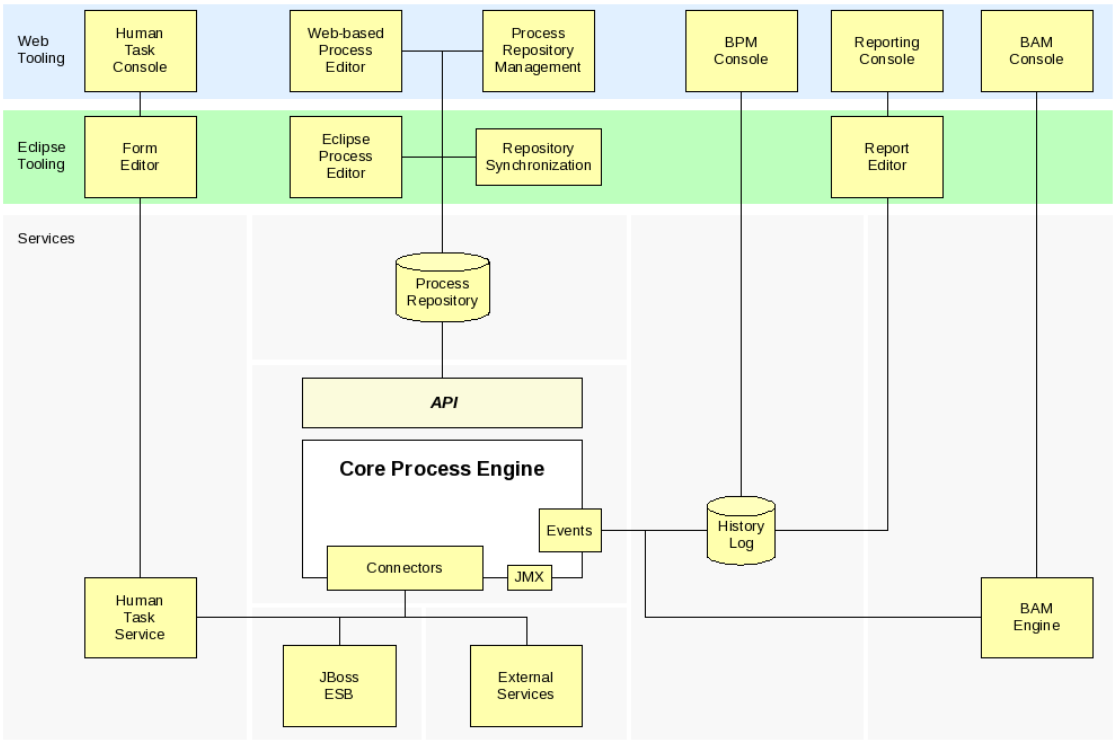


Figure 4.17: jBPM Components (copied from[12])

[10] http://sourceforge.net/projects/jbpm/files/
[11] https://www.jboss.org/jbpm/components
[12] http://www.activiti.org/components.html

*"jBPM can be combined with the Drools project to support one unified environment that integrates these paradigms where you model your business logic as a combination of processes, rules and events."*[8] Drools is a business rule management system (BRMS). It is a so-called production rule system which uses an enhanced implementation of the Rete algorithm.[13] It offers a unified and integrated platform for workflow, rules and event processing.[14]

A more detailed overview and description of jBPM and its components can be found on the homepage [8]. Furthermore, the user guide of various jBPM versions can be found here, as well as various examples and explanations of the different BPMN elements. Also the installation process of jBPM and the implementation in Eclipse are shown. jBPM needs the following components for its installation:

- Java 1.5+
- Eclipse
- Ant 1.7+

After the installation of jBPM, Eclipse can be started and the example business process "Issuing Invoice" can be imported as a jBPM project. Then the imported example can be opened with the jBPM designer for Eclipse.

In contrast to Activiti, all elements are shown (also Data Inputs and Data Outputs). However, if one wants to create a new model, no Data Input and Data Output can be added, because the provided Designer for Eclipse has no graphical representation for them (shown in Figure 4.18). The Designer has only a graphical representation for Data Objects.

In contrast to Activiti Designer, the Service Task can link with a Web Service (e.g., WSDL file). In jBPM the client stub class for the Web Service has to be generated before the execution of the business process (e.g., using Maven). So it is possible to integrate the Web Service using the previously created client stub class.

To make the business process model "Issuing Invoice" executable with jBPM, a few tool-specific modifications in the XML representation had to be made. While it does not matter in Activiti if the Item Definitions for the Messages (needed for the communication between Web Service and BPMN model) come before or after the Message, in jBPM these Item Definitions have to be given before the Messages. Furthermore, in jBPM Properties are used for Data Input and Data Output instead of Item Definitions at the Task. The difference is shown in Figure 4.19. Here, in the DataInputAssociation a Property "amount_pro" (blue framing) is used instead of an Item Definition as used in Activiti (red framing). However, the BPMN 2.0 standard does support both possibilities.

In Activiti, the Item Definition has to have the same name as in the WSDL file for mapping the DataInputAssociation and DataOuputAssocation. In jBPM the ID of the Data Input and Data Output at the Service Task is used. The ID's are defined in the Service Task in the IOSpecification. Figure 4.20 shows the difference between jBPM (blue framing) and Activiti (red framing).

[13] `http://en.wikipedia.org/wiki/Drools`
[14] `http://www.jboss.org/drools/`

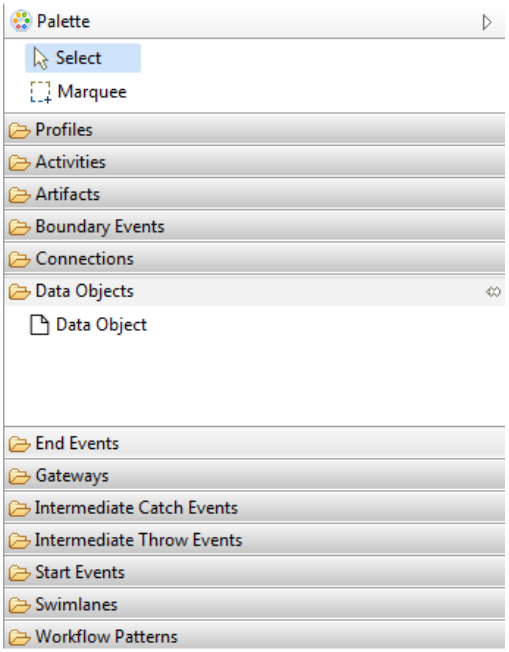


Figure 4.18: All data elements which the jBPM Designer provides in Eclipse

a.) jBPM

```
<property id="amount_pro"/>

<dataInputAssociation>
    <sourceRef>amount_pro</sourceRef>
    <targetRef>dataInputOfCreateInvoiceServiceTask</targetRef>
</dataInputAssociation>
```

b.) Activiti

```
<itemDefinition id="amount"/>

<dataInputAssociation>
    <sourceRef>amount</sourceRef>
    <targetRef>createInvoiceOperationAmountInput</targetRef>
</dataInputAssociation>
```

Figure 4.19: Differences between using Property and Item Definition at the DataAssociation in Activiti and jBPM

A further modification regarding the Script Task is that jBPM uses Java as script language. The example “Issuing Invoice” uses groovy, which is similar to Java. Listing 4.9 shows an example of a Script Task in jBPM, in which Java is used as script language.

a.) Activiti

```
<itemDefinition id="createInvoiceOperationInvoiceInput" .../>

 <!-- Service Tasks -->
<serviceTask id="ServiceTask_1" name="Create Invoice" implementation="##WebService" operationRef="tns:createInvoice">
   <ioSpecification id="InputOutputSpecification_3">
    <dataInput id="dICreateInvoice" itemSubjectRef="tns:createInvoiceInputSoap"/>
    <dataOutput id="dOCreateInvoice" itemSubjectRef="tns:createInvoiceInvoiceOutputSoap"/>
    <inputSet id="InputSet_3">
      <dataInputRefs>dICreateInvoice</dataInputRefs>
    </inputSet>
    <outputSet id="OutputSet_3">
      <dataOutputRefs>dOCreateInvoice</dataOutputRefs>
    </outputSet>
  </ioSpecification>
  <dataInputAssociation id="DataInputAssociation_1">
    <sourceRef>hashmap</sourceRef>
    <targetRef>createInvoiceOperationInvoiceInput</targetRef>
  </dataInputAssociation>
  ...
</serviceTask>
```

b.) jBPM

```
<serviceTask id="ServiceTask_1" name="Create Invoice" implementation="##WebService" operationRef="createInvoice">
  <ioSpecification id="InputOutputSpecification_3">
    <dataInput id="dICreateInvoice" itemSubjectRef="createInvoiceInvoiceInputSoap" name="Parameter"/>
    <dataOutput id="dOCreateInvoice" itemSubjectRef="createInvoiceOutputSoap" name="Result"/>
    <inputSet id="InputSet_3">
      <dataInputRefs>dICreateInvoice</dataInputRefs>
    </inputSet>
    <outputSet id="OutputSet_3">
      <dataOutputRefs>dOCreateInvoice</dataOutputRefs>
    </outputSet>
  </ioSpecification>
  <dataInputAssociation id="DataInputAssociation_1">
    <sourceRef>hashmap</sourceRef>
    <targetRef>dICreateInvoice</targetRef>
  </dataInputAssociation>
  ...
</serviceTask>
```

Figure 4.20: Difference between the mapping at the DataAssociation in Activiti and jBPM

```
1     <scriptTask id="PrepareParameters" name="Prepare Parameters
      for Input" scriptFormat="java">
2       <ioSpecification id="InputOutputSpecification_3">
3         <dataInput id="dataInputScriptTaskAmount"
            itemSubjectRef="inputScriptTaskAmount" name="Amount
            "/>
4         <dataInput id="dataInputScriptTaskAddress"
            itemSubjectRef="inputScriptTaskAddress" name="
            Address"/>
5         <dataOutput id="dataOutputScriptTask" itemSubjectRef="
            outputScriptTask" name="Result"/>
6         ...
7        <script>
8          at.ac.tuwien.ict.proreuse.
             invoiceissuingcomplexwithoneparameter.
```

```
            InvoiceParameterList parameters = new at.ac.tuwien
            .ict.proreuse.
            invoiceissuingcomplexwithoneparameter.
            InvoiceParameterList();
9         parameters.addParameter(&amp;quot;amount&amp;quot;,
            amount_pro);
10        parameters.addParameter(&amp;quot;address&amp;quot;,
            address_pro);
11        kcontext.setVariable(&amp;quot;invoice_pro&amp;quot;,
            invoice_pro);
12      </script>
13    </scriptTask>
```

**Listing 4.9:** Script Task for "Create Invoice".

The complete executable example is shown in Listing A.4.

In jBPM a Java class is used to start a process instance of a business process. Listing 4.10 shows an example of such a Java main class. First, a so-called knowledge builder is generated. It can load the needed resources from a specified location. After this, a so-called knowledge base is created that contains all the required process definitions. The commands are shown in Listing 4.10, code lines 10 – 12. Then a session is created using the knowledge base, which communicates with the Process Engine (Listing 4.10, code line 14). A session can also start a process instance. This is made in Listing 4.10, code line 25. Furthermore, the required parameters are passed to the process.

```
1
2 public class Example_Main_jBPM {
3
4   /**
5    * @param args
6    */
7   public static void main(String[] args) {
8     // TODO Auto-generated method stub
9
10    KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.
        newKnowledgeBuilder();
11    kbuilder.add(ResourceFactory.newClassPathResource("
        Example_WS_jBPM.bpmn"), ResourceType.BPMN2);
12    KnowledgeBase kbase = kbuilder.newKnowledgeBase();
13
14    StatefulKnowledgeSession ksession = kbase.
        newStatefulKnowledgeSession();
15
16    Map<String, Object> params = new HashMap<String, Object>();
17    params.put("employee", "krisv");
```

```
18      params.put("address_pro", "Weg 1, 1010 Wien");
19      params.put("amount_pro", 33f);
20
21      ServiceTaskHandler stHandler = new ServiceTaskHandler(
            ksession);
22
23      ksession.getWorkItemManager().registerWorkItemHandler("
            Service Task", stHandler);
24
25      ProcessInstance processInstance = ksession.startProcess("
            CallInvoiceIssuingWithOneParameter", params);
26
27    }
28
29  }
```

**Listing 4.10:** Main java class for starting a business process in jBPM.

An important point in jBPM is that for the execution of a Service Task a so-called Service Task Handler is needed. This handler must be created and registered for the session. This is made in Listing 4.10 in code line 21 and code line 23. Here the general Service Task Handler implemented in jBPM (org.jbpm.process.workitem.bpmn2.ServiceTaskHandler) is used.

In contrast to Activiti, the Service Task in jBPM has the constraint (according the BPMN 2.0 standard) implemented that only one Parameter can be passed. When the general Service Task Handler is used in jBPM, a few points have to be changed in the XML representation due to its implementation. The example "Issuing Invoice" has two Service Tasks. Each Service Task in the model must have the attribute "name" with the value "Parameter" at the Data Input. Also each Service Task in the model must have the attribute "name" with the value "Result" at the Data Output. An example is shown in Listing A.4, code lines 69 and 70 at the Service Task "Send Invoice". If the attribute "name" is missing or the attribute has another value, then an error message is thrown. The reason for this is given in the implementation of the Service Task Handler in the operation "executeWorkItem". The complete implementation of the operation is shown in Listing A.6.

The operation "executeWorkItem" has two input parameters "WorkItem" and "WorkItemManager". The WorkItemManager ensures, that the program is continued if the WorkItem is complete. It is also responsible for the registration of WorkItemHandler (e.g., Service Task Handler). The WorkItem is a Service Task in this case. Furthermore, it includes all the important parameters from the BPMN model for the Service Task (e.g., Implementation, interfaceImplementationRef, Interface, Operation, Inputs, etc.).

In the course of the operation, first a check is done if the passed WorkItem is connected with a Web Service (shown in Listing A.6, code line 3) or another service like Java (then another operation "executeJavaWorkItem" is called, but both operations only one parameter can be passed to the service).

After this check, the needed parameters are stored in variables. In jBPM, a Service Task can be executed in three different ways (synchronous, asynchronous or one way). In the business

process “Issuing Invoice”,none of them is specified at the Service Tasks. In this case, the synchronous way is used by default. But all three ways have similar structure. First, a client object is created and then the operation “invoke” is called. This operation is responsible for the link with the required operation of the Web Service and gets a result back (not in “one way”). After this, the Service Task is finished.

The question may above why the attribute “name” is needed at the Data Input and why the attribute has to have the value “Parameter“. This is shown in code line 6 in Listing A.6. Here an object from the WorkItem is retrieved by the name “Parameter” and is stored in a variable. So, when another value has been chosen or the attribute “name” is missing at Data Input, “null” is stored in the variable. The variable and the associated operation are then passed to the client operation “invoke” (shown in Listing A.6, code line 13). This operation links to the Web Service operation. If the type/structure of the variable is not compatible with the type/structure defined in the WSDL file an exception is thrown. If it is compatible a return value is expected. This return value is stored in a Hashmap with the key “Result”, and the Data Output has to have the attribute “name” and the value “Result”. Otherwise, the BPMN model has no result at the Service Task. Further modifications are not required and the BPMN model is executable.

In jBPM there is also the possibility to link a Service Task with an automated application (e.g., Java), but as mentioned above this is not specified in the BPMN 2.0 standard and so it is tool specific.

For linking a Service Task with Java. the XML-representation of the BPMN model must be adapted first. An interface is needed for every Java class used. The attribute “implementationRef” of the interface has to refer to the relevant Java class. The attribute “implementationRef“ of the operation has to refer to the operation of the Java class. In the Service Task, the value of the attribute “implementation“ has to be changed to the value “##unspecified“. Furthermore, the structure of the Item Definition must also be changed to the required types/structure (e.g., String). The changes and the link between Java class, Service Task and interface are shown in Figure 4.21. The Service Task handler does not need any change, because (as mentioned above) the Service Task Handler uses another operation if the Service Task is not linked with a Web Service. But the operation can also have only one parameter.

The BPMN 2.0 standard allows extensions to the base element (abstract type of most BPMN elements), but the standard does not define precisely how do implement this extension. jBPM has implemented the two extensions “onEntry-script“ and “onExit-script”. The extension “onEntry-script” is executed before the Service Task begins its work, while the extension “onExit-script” is executed when the Service Task has finished its work. Due to this extension, the Script Task as Wrapper is no longer needed, because the required parameters can be merged into one object and this object is passed to the Service Task before it is executed. With this approach, the constraint of the BPMN 2.0 standard can be circumvented. One drawback is that this approach is tool-specific and so causes problems for the portability.

In the example “Issuing Invoice” only the Service Task “Create Invoice” needs this extension. Here two Data Inputs (amount and address) are required. An example of the extension of the Service Task “Create Invoice” is shown in Listing 4.11. Here only the “onEntry-script” extension is needed and implemented. In this extension the attribute “script” is executed. This attribute defines the code which should be executed before the Service Task is executed. Much


Figure 4.21: Link between Java class and BPMN model

as for the Script Task, Java is used as script language. In Listing 4.11, an object “InvoiceParameterList”(similar to Hashmap) is created and is filled with the required Data Inputs (amount and address). The object is stored in a local variable (“parameters_pro”) in code line 8. The same is done with the Script Task “Prepare Parameters for Input” in the BPMN model “Issuing Invoice”. After that the DataInputAssociation passes the local variable to the Web Service.

```
1  <serviceTask id="createInvoice" name="Create Invoice"
       implementation="##WebService"   operationRef="createInvoice"
       >
2    <extensionElements>
3        <onEntry-script scriptFormat="http://www.java.com/java">
4          <script>
5              InvoiceParameterList parameters_pro = new
                   InvoiceParameterList();
6              parameters_pro.addParameter(&amp;quot;amount&amp;quot;,
                   amount_pro);
7              parameters_pro.addParameter(&amp;quot;address&amp;quot;,
                   address_pro);
8              kcontext.setVariable(&amp;quot;parameters_pro&amp;quot;,
                   parameters_pro);
9          </script>
10       </onEntry-script>
11   </extensionElements>
12   <ioSpecification>
13       <dataInput  id="dataInputOfCreateInvoiceServiceTask"
           itemSubjectRef="createInvoiceInputSoap" name="Parameter"
```

```
        />
14    <dataOutput id="dataOutputOfCreateInvoiceServiceTask"
        itemSubjectRef="createInvoiceInvoiceOutputSoap" name="
        Result"/>
15      <inputSet>
16        <dataInputRefs>dataInputOfCreateInvoiceServiceTask</
            dataInputRefs>
17      </inputSet>
18      ...
19      </ioSpecification>
20        <dataInputAssociation>
21          <sourceRef>parameters_pro</sourceRef>
22          <targetRef>dataInputOfCreateInvoiceServiceTask</
              targetRef>
23      </dataInputAssociation>
24      ...
25 </serviceTask>
```

**Listing 4.11:** Example of an extension at the Service Task in jBPM.

Much as in Activiti, also in jBPM more than one parameter can be passed to the Service Task. One possibility to achieve this is to rebuild the Service Task Handler, but this solution is not BPMN 2.0 standard compliant. If one only wants to use a Web Service, only the operation "executeWorkItem" has to be changed. Listing 4.12 shows the modified operation of the Service Task Handler. The other parts of the Service Task Handler do not need to be changed. The first change is shown in Listing 4.12 in code line 7, where the retrieval of the parameter from the WorkItem is commented out, because more than one parameter should be passed to the Web Service. For this, one needs a Hashmap in which all parameters of the WorkItem are stored (code line 10). In this case also parameters from the WorkItem (Interface, interfaceImplementationRef, etc.) are stored into the Hashmap, which are not needed for the Web Service. Therefore, these parameters have to be deleted from the Hashmap (shown in Listing 4.12, code lines 12 – 17). Because the client operation ("invoke") receives an object array, the Hashmap with the required parameters has to be rebuilt to an object array "params" (shown in Listing 4.12, code lines 19 – 26) and this new object array is passed to the client operation (shown in Listing 4.12, code line 32). After this modification, more than one parameter can be passed to a Web Service.

```
1  public void executeWorkItem(WorkItem workItem, final
       WorkItemManager manager) {
2        workI = workItem;
3          String implementation = (String) workItem.getParameter(
             "implementation");
4          if ("##WebService".equalsIgnoreCase(implementation)) {
5              String interfaceRef = (String) workItem.
                 getParameter("interfaceImplementationRef");
```

```
6               String operationRef = (String) workItem.
                   getParameter("operationImplementationRef");
7   //            Object parameter = workItem.getParameter("
        Parameter");
8               WSMode mode = WSMode.valueOf(workItem.getParameter(
                   "mode") == null ? "SYNC" : ((String) workItem.
                   getParameter("mode")).toUpperCase());
9
10              Map<String, Object> parameters = new HashMap<String
                   , Object>(workItem.getParameters());
11
12              parameters.remove("Interface");
13              parameters.remove("Operation");
14              parameters.remove("ParameterType");
15              parameters.remove("implementation");
16              parameters.remove("operationImplementationRef");
17              parameters.remove("interfaceImplementationRef");
18
19              Object[] params = null;
20
21              params = new Object[parameters.size()];
22              int counter =0;
23              for (Object param : parameters.values()) {
24                      params[counter] = param;
25                      counter++;
26               }
27
28              try {
29                   Client client = getWSClient(workItem,
                        interfaceRef);
30                   switch (mode) {
31                  case SYNC:
32                      Object[] result = client.invoke(
                            operationRef, params);
33                    ...
34                    }
35              } catch (Exception e) {
36                   logger.error("Error when executing work item",
                         e);
37              }
38      } else {
39          executeJavaWorkItem(workItem, manager);
40      }
```

```
41 }
```

**Listing 4.12:** Service Task Handler adaption.

When a Service Task should be associated with a Java class and multiple parameter should be allowed, then only the Service Task Handler operation "executeJavaWorkItem" needs to be changed. An example of the use of a changed operation is shown in Listing A.7. The modifications are similar to those for the Web Service described above. First, all parameters of the WorkItem are stored in a Hashmap. Then the parameters not needed are deleted from the Hashmap. After that, the required parameters are rebuilt from the Hashmap, an object array is created and then passed to the operation of the Java class. Listing 4.14 shows an example of a Java class including an operation with multiple parameters.

One difference to the Service Task Handler operation for the Web Service is that no client object is needed. The Java classes and their operations can simply be integrated. A problem here is that the operation of the interface in BPMN models may have only one input message according to the BPMN 2.0 standard. This message is referenced to an Item Definition, which defines the structure of the message (using the attribute "structureRef") and so also the structure of the parameter(s) passed. The structure of the parameter(s) is taken out of the WorkItem in the Service Task Handler (shown in Listing A.7, code line 4). The problem here is that the Service Task Handler expects only one particular structure for the parameter(s) without modifications.This means that indeed one can pass multiple parameters, but the structure of these parameters has to be the same (e.g., String). This problem is not caused by the integration of a Web Service. The structure of the message is defined in the WSDL file and thus can be a complex type, which can have more elements (parameters) with different structure.

To solve this problem, first the value of the attribute "strutureRef" of the Item Definition in the XML-representation responsible for the Input Message has to be changed. In business process "Issuing Invoice", for example, this is only the operation "createInvoice", which requires more than one parameter (amount and address). These parameters have different structures (amount is a float and address is a string). Therefore, in the value of the attribute "structureRef" one has to define both structures separated by a dash ("–"). Listing 4.13 shows the modification.

```
1  <itemDefinition id="createBillInput" structureRef="java.lang.
    String-java.lang.Float"/>
```

**Listing 4.13:** Adaption Item Definition.

It is important, that the attributes (parameters) defined are given in the same order as they are implemented in the Java class. Listing 4.14 shows the corresponding Java class. It can be seen that in the operation "createInvoiceOperation" the first parameter defined as String and the second parameter as Float. The same sequence has to be defined in the Item Definition in Listing 4.13. If the parameters are given in the wrong order, an error message is thrown (operation not found).

The String in Listing A.7, code line 4 "java.lang.String–java.lang.Float" contains two different structures for the parameters. It is split on the dash ("–") and the result is stored in a String array (shown in Listing A.7, code line 15). After this, the different structures/types as well as the parameters are stored into a new array. This is made in the "for" loop between code lines

26 and 31 in Listing A.7. Then the new arrays are passed to the associated operation (shown in code lines 32 and 33) and so more than one parameter with different structures can be passed to the Service Task.

```
1 public class CallCreateInvoiceClass {
2
3   public Invoice createInvoiceOperation(String address, Float
       amount) {
4
5     Invoice invoice = new Invoice();
6     invoice.setAddress(address);
7     invoice.setAmount(amount);
8
9     return invoice;
10  }
11
12 }
```

**Listing 4.14:** Example for Service Task connect with Java Class.

A advantage of jBPM is that one can create custom Service Tasks. Here the number and structure of the passed parameter and also the result of the Service Task can be defined. As a drawback, the BPMN model is then no longer BPMN 2.0 standard conform and also tool dependent.

To create a custom Service Task, first a work definition has to be created. Listing 4.15 shows an example of a work definition for the Service Task “Create Invoice”. First, the name of the custom Service Task is defined. This name must be unique. Then the name, number and types/structure of the parameters are defined. After that, the result and the type/structure of the result is defined. Finally, the name of the graphical representation of the custom Service Task is defined. The example of the work definition in Listing 4.15 defines the custom Service Task “createInvoice” with two parameters (a Float “amount” and a String “address”), the result (a Object “Invoice”) and the displayed name “Create Invoice”.

```
1 import org.drools.process.core.datatype.impl.type.FloatDataType
    ;
2 import org.drools.process.core.datatype.impl.type.
    ObjectDataType;
3 import org.drools.process.core.datatype.impl.type.
    StringDataType;
4 [
5   // the Create Invoice work item
6   [
7     "name" : "createInvoice",
8     "parameters" : [
9       "amount" : new FloatDataType(),
10      "address" : new StringDataType(),
```

```
11      ],
12      "results" : [
13        "invoice" : new ObjectDataType(),
14      ],
15      "displayName" : "Create Invoice",
16    ]
17
18 ]
```

**Listing 4.15:** Example for custom Service Task Work Definition.

After the work definition has been created, it has to be registered before it can be used. To link the Service Task with the work definition, only the first and the last line in the XML-representation of the Service Task have to be changed (shown in Listing 4.16).

```
1 <task id="Task_1" taskName="createInvoice" tns:displayName="
    Create Invoice" icon="task.png" name="Create Invoice">
2 ...
3 </task>
```

**Listing 4.16:** Adaption Service Task for Linking with Work Definition.

For using a custom Service Task, a custom Service Task Handler has to be written. Listing 4.17 shows a custom Service Task Handler example for the custom Service Task "Create Invoice". This handler is also derived from WorkItemHandler as the general Service Task Handler. Therefore, the handler has also the operation "executeWorkItem" with the two parameters "workItem" and "manager", which has the same properties as explained above for the general Service Task Handler. The two parameters (amount and address) first are retrieved from the passed WorkItem and are stored in variables. An invoice is created and stored in a Hashmap. This Hashmap is passed to the WorkItemHandler with the ID of the WorkItem. After that, the Service Task is finished.

```
1 public class CallWorkItemCreateInvoice implements
    WorkItemHandler{
2
3   public void executeWorkItem(WorkItem workItem,
       WorkItemManager manager) {
4
5         // extract parameters
6         Float amount = (Float) workItem.getParameter("amount");
7
8         String address = (String) workItem.getParameter("
             address");
9
10        Invoice invoice = new Invoice();
11        invoice.setAddress(address);
12        invoice.setAmount(amount);
```

```
13
14          Map<String,Object> result = new HashMap<String,Object
                >();
15          result.put("invoice", invoice);
16
17          // notify manager that work item has been completed
18          manager.completeWorkItem(workItem.getId(), result);
19
20        }
21        public void abortWorkItem(WorkItem workItem,
              WorkItemManager manager) {
22
23          // Do nothing this task cannot be aborted
24
25        }
26  }
```

**Listing 4.17:** Example for Service Task Handler for the custom Service Task.

For using this defined Work Definition and the associated custom Service Task Handler, they have to be registered in the main java class with the following command shown in Listing 4.18.

```
1   ksession.getWorkItemManager().registerWorkItemHandler("
        createInvoice", new CallWorkItemCreateInvoice());
```

**Listing 4.18:** Example for Registration of custom Service Task handler .

Detailed instructions for creating and registering a custom Service Task can be found in the user guide on the homepage of jBPM [15].

### 4.3.3 Bonita BPM

Bonita BPM is developed by Bonitasoft and combines three solutions in one:

- Bonita Studio
- Bonita User Experience Portal
- Bonita Execution Engine

Bonita Studio is responsible for process modeling. There the process can be created graphically on a whiteboard. For connection to other technologies (e.g., database, Web Service, Java, etc.) Bonita BPM use so-called connectors. Over 100 connectors are available in Bonita Studio at the moment. Also custom-created connectors can be easily created whit the connector creator, which is part of Bonita Studio. Furthermore, it is also possible to create a so-called form in Bonita Studio, which supports user interaction, but also templates of forms can be imported.

[15] http://docs.jboss.org/jbpm/v5.4/userguide/ch.domain-specific-processes.html

After creating a BPMN model in Bonita Studio, the model can be directly executed here without further initial aid.[16]

The Bonita User Experience Portal shows the daily tasks for the user. Through its user-friendly interface, the tasks are easy to manage and track.[16]

The Bonita Execution Engine is the heart of Bonita BPM. It can be continuously extended with new services and BPMN standard. The performance is very good so that the utilization is very high.[16]

Bonita BPM supports a business process in every phase of its life cycle: modeling, development, execution and monitoring. Bonita BPM is available in four different editions:

- Community
- Teamwork
- Efficiency
- Performance

A comparison between the editions can be found on the Website under product comparison. [17] For our comparison with the business process "Issuing Invoice" the Community edition was used because only this version is free to download from [18]. Here it is possible to choose between different version of operation systems. The downloaded file is a so-called installer, which installs the Bonita BPM Community edition completely. No further configuration is necessary. Each edition contains at least the Bonita BPMN Studio to create and test processes and the Bonita BPM Platform containing the server.

The Community edition of Bonita BPM contains Bonita BPM Studio, Bonita BPM Platform and includes an Apache Tomcat application server, a database (h2), Bonita BPM Portal [19] and the Bonita BPM Engine. It is also possible to integrate another application server in Bonita BPM. The documentation of Bonita BPM can be found on the homepage [18].

After installation and starting Bonita BPM Studio, the business process "Issuing Invoice" can be imported. Figure 4.22 shows the business process with Bonita BPM Studio. Here one can see the same problem as in Activiti above. The Data Inputs and Data Outputs are not shown. The reason can be seen in the palette on the left side of the Whiteboard. There are no graphical representations available for Data Inputs and Data Outputs. However, Data Inputs and Data Outputs can be shown using the "properties view", which is located below the Whiteboard (Figure 4.23 shows an example), but cannot be displayed graphically. Furthermore, a new symbol appears which is not BPM 2.0 Standard conform on the Service Task representation (marked in red in Figure 4.24). The symbol means that the Service Task contains a connector.

In Figure 4.22 an icon (framed in green) in the main menu is shown for starting the execution of the business process. When the icon is pushed and the execution starts, the model is checked first and an error appears. In order to obtain more precise information about the error(s), the "more details" button has to be clicked. According to this information, the error only occurs at

[16] http://de.bonitasoft.com/produkte/bonita-open-solution-open-source-bpm

[17] http://de.bonitasoft.com/produkte/produktvergleich

[18] http://de.bonitasoft.com/produkte/bpm-software-und-dokumentation-herunterladen

[19] Bonita BPM Portal shows task for users and helps installing, deploying and managing process

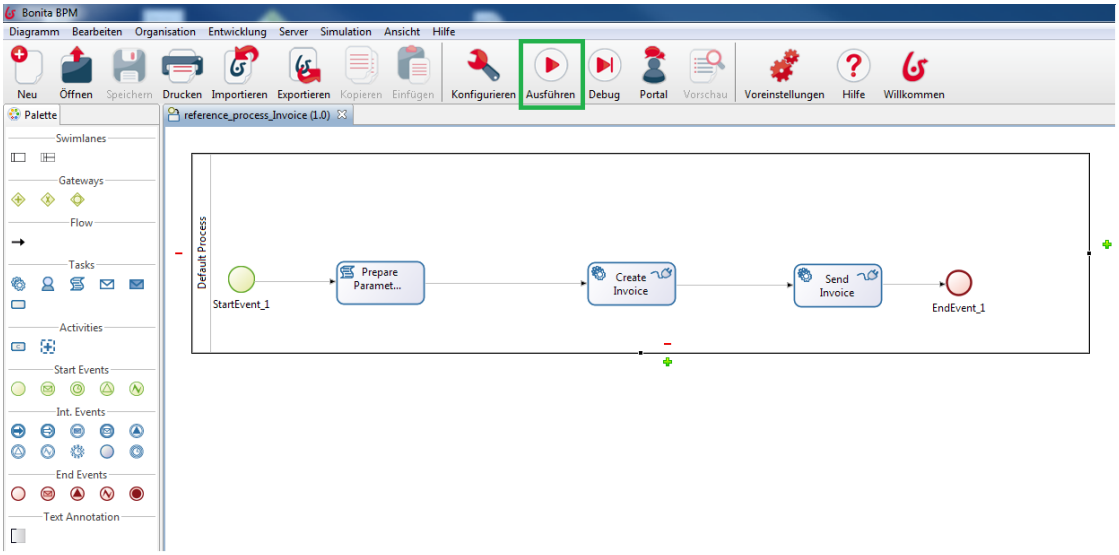


Figure 4.22: The example “Issuing Invoice” is shown with Bonita BPM Studio

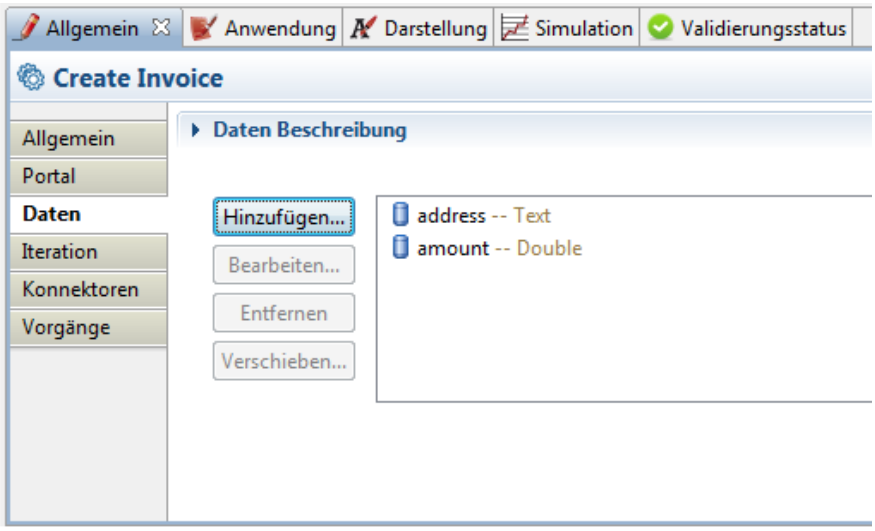


Figure 4.23: Property view for Service Task “Create Invoice”

the Service Tasks. The reason is that the Bonita BPM connectors were used for linking Service Tasks with external services.

If the model is created by Bonita BPM, these connectors are linked in the import statement of the generated XML-representation as XML schema. When importing the business process “Issuing Invoice” in to Bonita BPM Studio, the Studio recognizes that the Service Tasks are connected with Web Services, but the import statement of the XML-representation of the business process “Issuing Invoice” contains a link to a WSDL file and not an XML schema. That is why an error is thrown.


Figure 4.24: Graphical representation of Service Task “Create Invoice”

For being able to execute the business process “Issuing Invoice”, the Service Tasks need to add a Web Service connector. Then a click on the Service Task is performed in the Whiteboard, the Property-view appears below of the individual Service Task. In this view, a category “connectors” is available. Figure 4.25 shows this for the Service Task “Create Invoice”. It shows that no connector could be found.

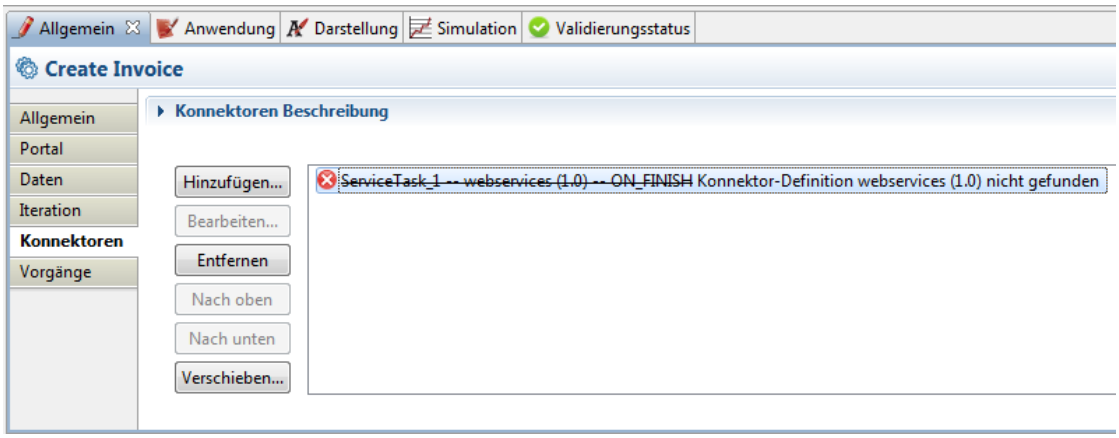


Figure 4.25: Property view “connectors” for Service Task “Create Invoice”

For the execution, first a new connector must be created and added to the Service Tasks. This is made with a Connector Wizard, which has already various predefined connectors defined (shown in Figure 4.26). Here also a connecter for a Web Service is predefined. This connector needs some information (e.g., namespace, porttype, parameter, result, operation, etc.). An advantage of Bonita BPM is that the created Web Service connector can be tested in the wizard immediately and so one can check the settings and the result for the Web Service. A more detailed description of the different steps and tutorials for creating a connector can be found on the BonitaSoft homepage. Also many videos are available showing the creation of Service Tasks with a connector.

An advantage of Bonita BPM is that a custom connector can be easily created in the Bonita BPM Studio. Also the import of externally created connectors is possible. To create a connector, first a connector definition must be done. After that, an implementation for the definition can be created. Separating definition and implementation has the advantage that one can change the implementation without changing the definition. The implementation is done in a Java class, which automatically opens when a new implementation for a definition should be created. In this way, Java can be easily integrated.

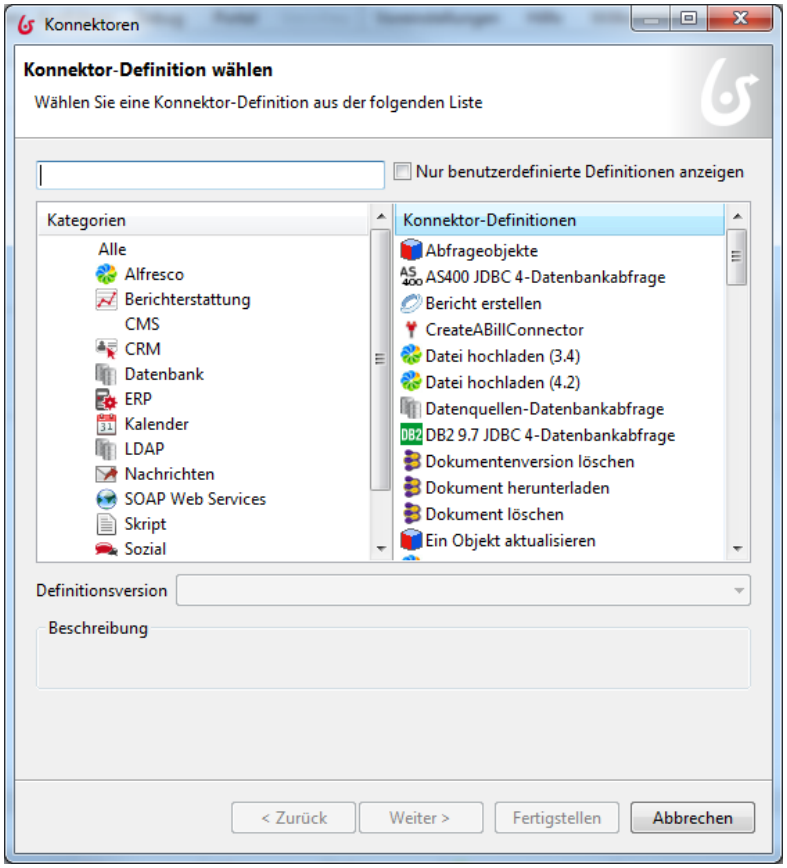


Figure 4.26: The Connector Wizard

In Bonita BPM, Service Tasks are allowed to get multiple inputs. All inputs are mapped to a complex object using Assignments in the DataInputAssociation. This complex object is managed internally and is passed to the linked services of the Service Tasks. This approach only defines one Data Input for the Service Task and this fact makes the model BPMN 2.0 standard compliant. So the example “Issuing Invoice” does not need the Script Task as wrapper here.

Listing 4.19 shows a part of the BPMN model of the business process “Issuing Invoice” created in Bonita BPM Studio. This model has only two Service Tasks (no Script Task) and these Service Tasks are connected with a created custom connector. As in Listing 4.19, code line 6 is shown, in the import statement the connector is defined. In Bonita BPM, for each connector an import statement is required.

Another difference is that the Service Task “Create Invoice” has one Data Input and one Data Output, but in the graphical model the Service Task has defined two inputs. As mentioned above, in Bonita BPM Assignments are used to map the inputs to an Item Definition. The structure of the Item Definition is created from the connector definition and it is a complex object.

The mapping of the inputs is done in the XML representation in the DataInputAssociation (Listing 4.19, code lines 35 – 48). First the target object for the inputs is defined (code line 37). Then the two inputs are mapped to a target object. This process (shown in Listing 4.19 between

code lines 39 and 46) is made by Assignments in the “from” and “to” attributes. For each input an assignment is required. After that, the target object is passed to the linked service of the Service Task by the Data Input. In Bonita BPM, Data Objects are used for Data Inputs and Data Outputs.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2 <definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3 xmlns:bonitaConnector="http://www.bonitasoft.org/studio/
    connector/definition/6.0"
4 ...>
5
6  <import importType="http://www.w3.org/2001/XMLSchema"
      location="connectorDefs/CreateInvoiceConnector-1.0.0.
      defconnectors.xsd" namespace="http://www.bonitasoft.org/
      studio/connector/definition/6.0"/>
7  ...
8
9  <itemDefinition id="CreateInvoiceConnectorConnectorInput"
      structureRef="bonitaConnector:
      CreateInvoiceConnectorInputType"/>
10 <itemDefinition id="CreateInvoiceConnectorConnectorOutput"
      structureRef="bonitaConnector:
      CreateInvoiceConnectorOutputType"/>
11
12 <message id="CreateInvoiceConnectorConnectorMessageInput"
      itemRef="CreateInvoiceConnectorConnectorInput"/>
13 <message id="CreateInvoiceConnectorConnectorMessageOutput"
      itemRef="CreateInvoiceConnectorConnectorOutput"/>
14
15 <interface id="
      CreateInvoiceConnector_Bonita_Connector_Interface" name="
      CreateInvoiceConnector_Bonita_Connector_Interface">
16   <operation id="ExecCreateInvoiceConnector" name="
        ExecCreateInvoiceConnector">
17     <inMessageRef>CreateInvoiceConnectorConnectorMessageInput
          </inMessageRef>
18     <outMessageRef>
          CreateInvoiceConnectorConnectorMessageOutput</
          outMessageRef>
19   </operation>
20 </interface>
21
22 <process id="_p3tasJoqEeO7WOWBtN4Sjg" name="Issuing Invoice">
```

```
23    ....
24     <serviceTask id="_1WOFYJoqEeO7WOWBtN4Sjg" name="Create
          Invoice" implementation="BonitaConnector" operationRef="
          ExecCreateInvoiceConnector">
25       <ioSpecification id="_GwtiIJoyEeO7WOWBtN4Sjg">
26         <dataInput id="_GwtiIZoyEeO7WOWBtN4Sjg" itemSubjectRef=
             "CreateInvoiceConnectorConnectorInput"/>
27         <dataOutput id="_GwtiI5oyEeO7WOWBtN4Sjg" itemSubjectRef
             ="CreateInvoiceConnectorConnectorOutput"/>
28         <inputSet id="_GwtiIpoyEeO7WOWBtN4Sjg">
29           <dataInputRefs>_GwtiIZoyEeO7WOWBtN4Sjg</dataInputRefs
               >
30         </inputSet>
31         <outputSet id="_GwtiJJoyEeO7WOWBtN4Sjg">
32           <dataOutputRefs>_GwtiI5oyEeO7WOWBtN4Sjg</
                dataOutputRefs>
33         </outputSet>
34       </ioSpecification>
35       <dataInputAssociation>
36
37         <targetRef>_GwtiIZoyEeO7WOWBtN4Sjg</targetRef>
38
39         <assignment>
40           <from xsi:type="tFormalExpression" id="
                _GwtiJZoyEeO7WOWBtN4Sjg" evaluatesToTypeRef="java:
                java.lang.String" language="http://www.w3.org
                /1999/XPath">address</from>
41           <to id="_GwtiJpoyEeO7WOWBtN4Sjg">getDataInput('
                _GwtiIZoyEeO7WOWBtN4Sjg')/bonitaConnector:address<
                /to>
42         </assignment>
43         <assignment>
44           <from xsi:type="tFormalExpression" id="
                _GwtiJ5oyEeO7WOWBtN4Sjg" evaluatesToTypeRef="java:
                java.lang.Float" language="http://www.w3.org/1999/
                XPath">amount</from>
45           <to id="_GwtiKJoyEeO7WOWBtN4Sjg">getDataInput('
                _GwtiIZoyEeO7WOWBtN4Sjg')/bonitaConnector:amount</
                to>
46         </assignment>
47
48       </dataInputAssociation>
49       ....
```

```
50      </serviceTask>
51
52      ...
53    </process>
54    ....
55  </definitions>
```

**Listing 4.19:** Part of issuing invoice created with Bonita BPM.

This approach is similar to the one proposed above in Section 4.2, where the Script Task is used as wrapper. However, the creation of a BPMN 2.0 model in Bonita BPM with a connector is tool-specific and tool-dependent. The created models in Bonita Studio are indeed BPMN 2.0 compliant, because the XML-described connectors are linked in the import statement as XML-schema. This is allowed according to the BPMN 2.0 standard, because it allows to import all XML-based structure, but these created models cannot interpret through the process engines of other tools (which do not support this concept).

### 4.3.4 Camunda BPM

Camunda BPM is a Java-based Business Process Management platform, which is aimed specifically at Java software developers. It is not a closed suite, but an open source framework. The platform supports BPMN 2.0 in the modeling, execution and monitoring.[20] Also Camunda BPM provides a commercial edition. The most important components of Camunda BPM are:

- BPMN 2.0 Process Engine
- Flexible framework for developers
- Camunda Cockpit
- Modeler as Eclipse Plugin

The BPMN 2.0 Process Engine is the core of Camunda BPM. It is implemented in Java and is a lightweight, native process engine. The code of the framework is available for download and can be edited. Camunda Cockpit is a tool to manage the business process (e.g., to monitor, cancel, kick back or manipulate business process instances). The modeler can be used in Eclipse and is responsible for support of the uses for process modeling.[21]

A comparison between the commercial enterprise edition and the open source edition (free of charge) can be found on the homepage under [22], and the user guide for Camunda BPM under [23], where in addition to various examples also the installation of Camunda BPM and the implementation in Eclipse can be found.

The current version of Camunda BPM is 7.1.0 and can be downloaded under [24]. For the installation of Camunda BPM, the following set of tools must be installed:

[20] http://camunda.com/
[21] http://camunda.com/bpm/features/
[22] http://camunda.com/de/bpm/enterprise/
[23] http://docs.camunda.org/latest/guides/user-guide/
[24] http://camunda.org/download/

- Java JDK 1.6+
- Apache Maven
- Modern Web browser (Google Chrome or latest Mozilla Firefox or Internet Explore 9+)
- Eclipse (Indigo, Juno, or Kepler)

After the installation of Camunda BPM according of the user guide, Eclipse can be started and the example business process "Issuing Invoice" can be imported as a BPMN 2.0 project. After this, the imported example can be opened with the included Camunda Modeler for Eclipse. In contrast to Activiti, all elements are shown (also Data Inputs and Data Outputs). In contrast to jBPM, also the Data Input and Data Output can be modeled when a new model is created.

A problem of Camunda Modeler is that a Service Task cannot be linked directly with a Web Service, but only with Java classes, expressions or delegate expressions. Furthermore, Camunda Modeler cannot import a Web Service and, therefore, the import and the linking of a Service Task with a Web Service must be made directly in the XML representation or using another modeler in Eclipse (e.g., Eclipse Modeler for BPMN2 Project/Diagram).

For generating and starting a process instance in Camunda BPM, a Java class is need. Listing 4.20 shows one possibility how such a Main Java class can look like. First, in code line 12 a process engine is created based on the file "camunda.cfg.xml". In this file the configuration of the Process Engine is defined. Several examples of how this file may look like, can be found in the user guide of Camunda.[23]

When the Process Engine is defined, the next step is (Listing 4.20, code line 17) that the Camunda Services Repository is created, which provides operations for manipulating and managing deployments and process definitions. It can be used to deploy one or multiple BPMN 2.0 XML files, or other resources. This process is also called deployment. The created files are uploaded to the Process Engine and there all resources are audited and parsed before they are stored into a database. If an error occurs, an error message is thrown. If no error message is thrown then the deployed files or resources are known by the Process Engine and every process (which was deployed) can be started. The statement in code line 20 shows the deploying of the BPMN model.[23]

After that, the Runtime Service is created. It is responsible for the storage and the use of the process variables. In addition, the Runtime Service can be used for queries on the process instances and the execution, and it is also used for controlling process instances (e.g., resuming a waiting process to a wait state). But it is also used to start a process instance. In the statement in Listing 4.20, code line 30, a process definition with the name "IssuingInvoice" is searched in the loaded BPMN model. Furthermore, the variables needed for the business process are passed. After that, a process instance of the process definition is started. [23]

```
1
2
3 public class Main {
4
5   /**
6    * @param args
```

```
7     */
8
9   public static void main(String[] args) {
10
11      //create Activiti process engine
12      ProcessEngine processEngine =  ProcessEngineConfiguration.
           createProcessEngineConfigurationFromInputStream(new
           FileInputStream("resources/camunda.cfg.xml")).
           buildProcessEngine();
13
14
15      // get Camunda services Repository, Runtime
16      // operations for managing and manipulating deployments and
            process definitions
17      RepositoryService repositoryService = processEngine.
           getRepositoryService();
18
19      //Deploy the process definition
20      repositoryService.createDeployment().addClasspathResource("
           IssuingInvoice.bpmn").addClasspathResource("
           IssuingInvoice.png").deploy();
21
22      RuntimeService runtimeService = processEngine.
           getRuntimeService();
23
24      Map<String, Object> variableMap = new HashMap<String,
           Object>();
25      variableMap.put("name", "Activiti");
26      variableMap.put("amount", 50f);
27      variableMap.put("address", "Linke Wienzeile 22, 1040 Wien")
           ;
28
29        //Start process instance
30      ProcessInstance processInstance = runtimeService.
           startProcessInstanceByKey("IssuingInvoice", variableMap)
           ;
31    }
32  }
```

**Listing 4.20:** Main Java class for starting a business process in Camunda.

Then a process instance of the business process “Issuing Invoice” can be started, but an error message (org.camunda.bpm.engine.ProcessEngineException) is thrown, because the process engine cannot import the WSDL file. This a major drawback of Camunda BPM, because the Service Task in Camunda BPM can only be connected with Java classes, expressions or delegated

expressions. So, if one wants to execute the business process in Camunda BPM, the Service Tasks have to be connected with Java classes. In these Java classes (each Service Task needs one Java class), the invoice can be created and then the invoice can be sent. But the integration at the Service Tasks of Java classes is done via the attribute "class" in Camunda BPM and this is a tool-specific solution and so it is not BPMN 2.0 standard compliant. Listing A.8 shows the complete XML-representation of the business process "Issuing Invoice", where the two Service Tasks are connected with two Java classes.

In Camunda BPM, four different ways are available to connect a Service Task with Java Logic:[23]

- *"Specifying a class that implements JavaDelegate or ActivityBehavior*
- *Evaluating an expression that resolves to a delegation object*
- *Invoking a method expression*
- *Evaluating a value expression"*[23]

In Listing A.8, the most commonly used way to integrate a Java class is shown. It is important that the Java class has the same name as the value of the attribute "class" in the Service Task. Listing 4.21 shows an example of a Java class that is integrated into a BPMN model. In addition to the Java class name, an interface called "org.activiti.engine.delegate.JavaDelegate" is implemented in the used Java class. This interface provides the required operation "execute", which is called during the execution from the Process Engine. In this operation, the needed logic for this Service Task is implemented. The passed Parameter (execution) of type "DelegateExecution" is used to store or receive the important process instance information.

Listing 4.21 shows that first an invoice object is created in the Java Class with the name "CreateInvoice", which is the same used in the example "Issuing Invoice". After that, the required input variables (amount and address) are assigned to the invoice object. Subsequently, the invoice object is stored back to the process instance variable (execution) and the Service Task is finished. The Process Engine searches for the next BPMN element following the outgoing Sequence Flows.

```
1
2 public class CreateInvoice implements JavaDelegate {
3
4   public void execute(DelegateExecution execution) throws
       Exception {
5   Invoice invoice = new Invoice();
6
7   InvoiceParameterList parameters = (InvoiceParameterList)
       execution.getVariable("hashmap");
8
9   invoice.setAddress((String)parameters.getValue("address"));
10  invoice.setAmount((Float)parameters.getValue("amount"));
11
```

```
12      execution.setVariable("invoice", invoice);
13    }
14
15  }
```

**Listing 4.21:** Java class for the Service Task “Create Invoice”

A more detailed description of all four ways to link a Service Task with Java can be found in the user guide of Camunda under “Service Task”.[25]

Another type of executable Task in Camunda BPM is the Script Task. It is used in the example “Issuing Invoice”. As mentioned in the previous Section 4.2, the Script Task is a specialization of Task and, therefore, it is similar to the Service Task. The difference between and the important attribute of the generated XML representation for the Script Task is explained in Section 4.2. In Camunda BPM, groovy is used as scripting language, which is similar to Java.

The Script Task in the business process “Issuing Invoice” is not needed, since in Camunda BPM, when linking the Service Task with Java, only one parameter as Object is passed (shown in Listing 4.21). Thus, the mapping of the Data Inputs to an object (which is made in the Script Task in example “Issuing Invoice”) is no longer necessary, because all Data Inputs and Data Outputs are stored in the passed object “DelegateExecution”. Therefore, the Data Inputs and Data Outputs can be used at each Service Task in the process linked with Java in Camunda BPM. However, these models (which need more Data Inputs at the Service Task) are not BPMN 2.0 standard compliant.

### 4.3.5 Sydlee Seed

Sydlee Seed is a BPMS for modeling, documentation, automation, execution and monitoring of business processes. In contrast to the tools discussed above, Sydlee Seed is used online. This means that, after the registration, one can directly log in on the provided Website. Also no further installation of tools is necessary. Sydlee Seed is available in four different editions:

- Community
- On demand
- Private cloud
- On site

Only the Community edition is free of charge. The user guide for the tool Sydlee Seed can be found under [26].

After registration and logging in, the welcome page of Sydlee Seed is displayed (shown in Figure 4.27).

There are five different categories:

- Seeds (Create and manage Instance of business process, monitor its activities)

[25] http://docs.camunda.org/latest/api-references/bpmn20/#tasks-service-task
[26] http://cloud.sydle.com/seed/cm/help/en/gettingToKnowSeed.html

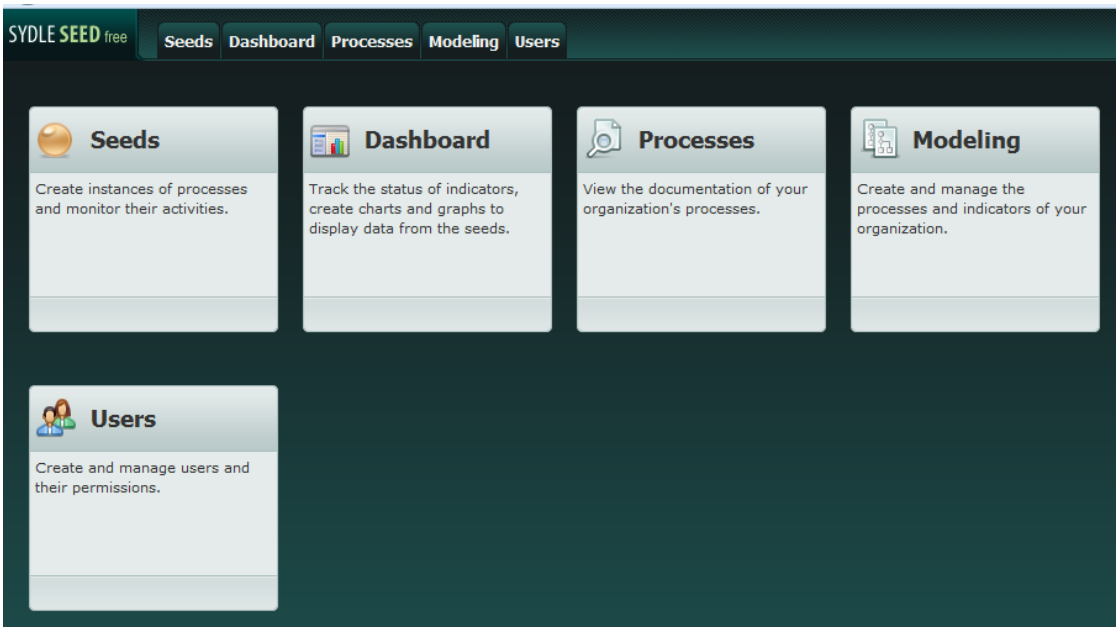


Figure 4.27: Welcome page of Sydlee Seed

- Dashboard (monitoring business processes)
- Processes (documentation of the created business processes)
- Modeling (create, execute and manage business processes)
- Users (Create and manage users)

To execute the business process “Issuing Invoice” with Sydlee Seed, its model has to be imported. The import should be done in the category “Modeling”, but as can be seen in this category there is no import available for BPMN models. Figure 4.28 shows the “Modeling” view. The reason is that importing BPMN models is only possible for the edition “on site”. As a workaround, the business process “Issuing Invoice” can be created in Sydlee Seed again. Sydlee Seed is a closed suite and so it is not possible to investigate how the tool stores and executes the created BPMN model. Also the export of created BPMN models is not possible in the Community edition and so no statement can be made about the portability.

To link the Service Task with an external service, a connector is used in Sydlee Seed (as in Bonita BPM). First the connector has to be created and then the connector can be linked to a Service Task. Each Service Tasks needs a connector for the link to an external service. In the Community edition the connector can only link Service Tasks with Web Services. In the other editions of Sydlee Seed, it is possible to link Service Tasks with Java through connectors.

Due to the fact that the export function (BPMN 2.0 model or connector) is not available in the Community edition, the XML-representation of the example “Issuing Invoice” cannot be investigated (e.g., is the XML-representation BPMN 2.0 standard compliant or how connectors are integrated in the model). Nevertheless, the concept of connectors is not described in the

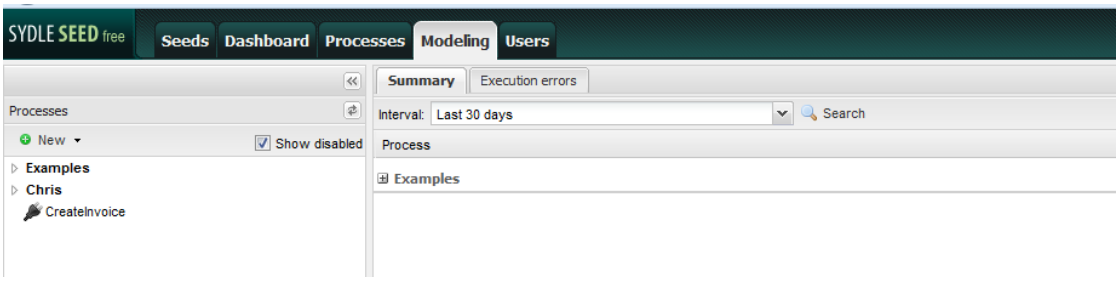


Figure 4.28: Modeling view of Sydlee Seed

BPMN 2.0 standard, but connectors can be linked in a standard-compliant way by the import statement that allows any XML-structure. The XML-representation of the business process models are indeed BPMN 2.0 standard compliant, but the implementation of the connector is tool-dependent and so this models cannot be interpreted by another process engines.

In Sydlee Seed, the Script Task uses Javascript as script language.

### 4.3.6 Inubit BPM

Inubit BPM is a BPMS developed by Bosch. It is used for modeling, simulation, execution, monitoring and reporting. Inubit BPM supports a business process throughout its life cycle. This allows a good collaboration between business users, developers and system administrators over the whole process cycle. The Inubit BPM contains the following components:

- Modeling Center
- Process Center
- Solution Center
- Integration Center

The Modeling Center is responsible for modeling, simulation and documentation of the business processes. The Solution Center is a Web application and used for the creation of Web-based solutions. The Integration Center offers a variety of connectors, which can integrate different services. The Process Center contains the necessary components for modeling, validation and simulation of business processes. Furthermore, the created business processes can be executed and monitored with the integrated Process Engine in the Process Center.[27]

Inubit BPM is a commercial BPMS, but it offers a trail license for 30 days. After obtaining the trail license and full installation of the Inubit BPM (it is also possible to install the individual components mentioned above) one has four icons available on the Desktop (shown in Figure 4.29). The icon “Process Engine start” has to be executed before one can use the Inubit BPM. Here the process engine and the application server are started and the required resources

[27] http://www.bosch-si.com/media/de/bosch_software_innovations/documents/brochure/inubit_suite/inubit_allgemeine_dokumente_uebergang/inubit_Suite_61_transition_Dec12_web.pdf

are deployed. The icon “Process Engine stop” is used to stop the process engine and to shut down the application server. It should be executed as soon as Inubit BPM is no longer used. A reason is that in the full installation the Modeling Center requires at least 8GB Ram. For only installing the Process Center, Solution Center and Integration Center, just 2GB Ram is indicated as minimal system requirement. The icon “Enterprise Portal” starts the Web portal (Solution Center) in a Web browser. There a user can log in and then execute, manipulate and monitor business processes.


Figure 4.29: Components for Inubit BPM

The icon “Workbench” can be used to start the Inubit BPM. Figure 4.30 shows the welcome page. Here different types of models (e.g., BPEL-processes, XPDL-model) can be imported, but BPMN models cannot be imported. Figure 4.31 shows the available types which can be imported in Inubit BPM.

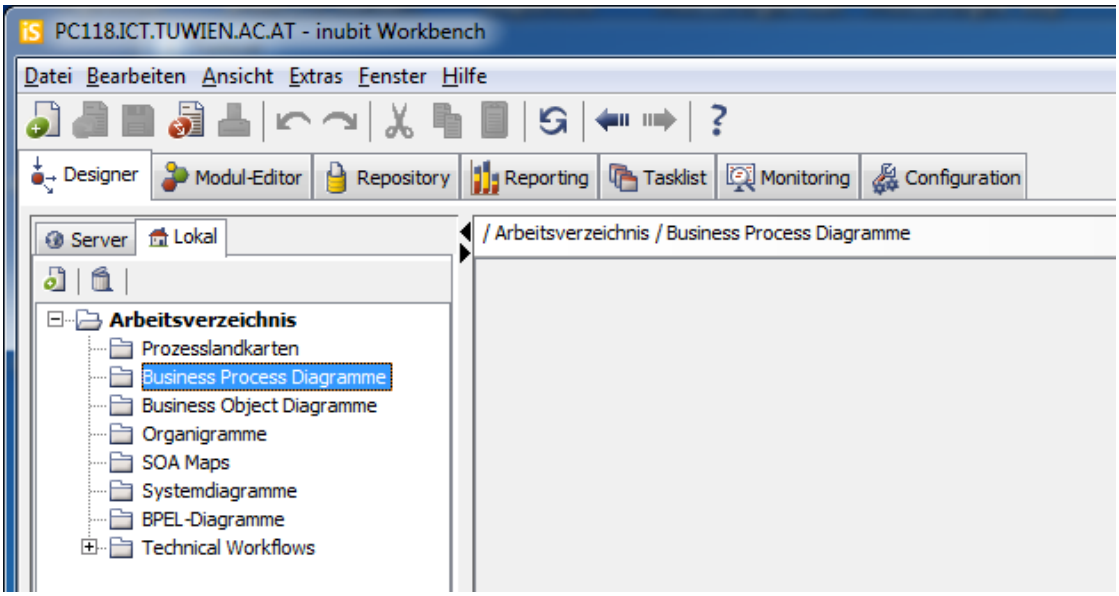


Figure 4.30: Welcome page for Inubit BPM

The business process “Issuing Invoice” has been modeled again. This was done in the category “Business Process Diagrams” (BPD). The available BPMN notation for modeling is shown in Figure 4.33 on the right side marked in red. The BPMN model should be executable and so only a part of the BPMN notation is available. The newly created business process “Issuing Invoice” is shown in Figure 4.32. In this BPMN model, it can be seen that Inubit BPM uses Data Objects instead of Data Inputs and Data Outputs. For example, in Figure 4.32 Data Objects are

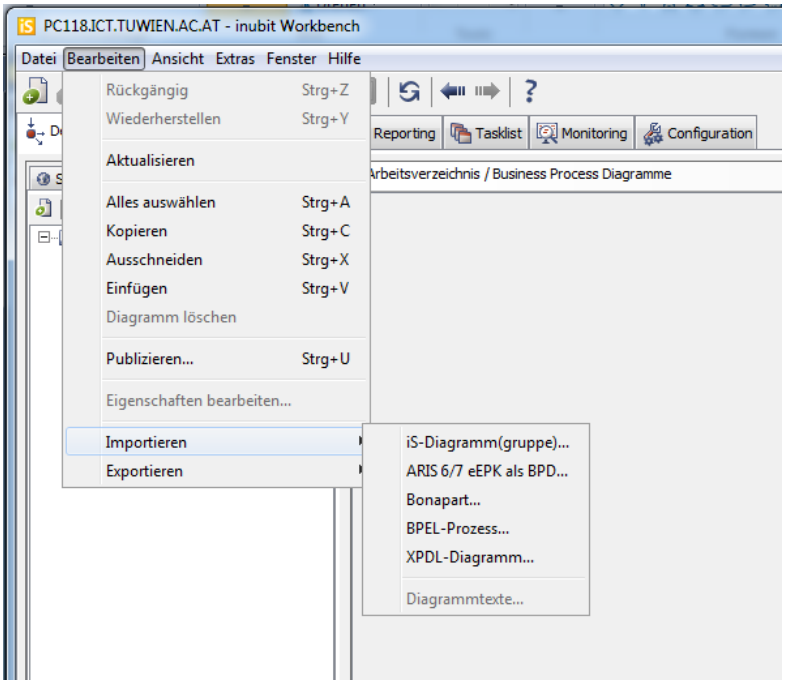


Figure 4.31: The available types of model which can be imported into Inubit BPM

used instead of the two Data Inputs (amount and address). Also in Inubit BPMN connectors are used to link the Service Tasks to an external service. Then the created BPMN 2.0 model has to be linked with technical workflows, which are responsible for the execution of the created BPD model.

In Inubit BPM, there are already several predefined connectors. For Inubit BPM connectors, a custom documentation is given under [28]. In this documentation also the creation of custom and the use of the predefined connectors is explained. The created BPMN 2.0 model is stored in a proprietary XML-file and so the XML-representation is not BPMN 2.0 standard compliant. Therefore, the created BPMN 2.0 models through Inubit BPM are tool-dependent and cannot be imported into other tools and cannot be executed through process engines of the other tools.

All documentations for using and installing Inubit BPM can be found under [29].

The focus of Inubit BPM is in the graphical representation (BPMN 2.0 compliant) and the generation (or the linking) of created model with technical workflows. The generated XML-representation of the created models represents the XML-format only for persistence an not prepared for portability.

[28] http://www.inubit.com/inubit-Suite/6.1/onlinehelp/pdf/de/inubitSystemConnectorGuide_DE.pdf

[29] http://www.inubit.com/inubit-Suite/6.1/onlinehelp/start_de.html

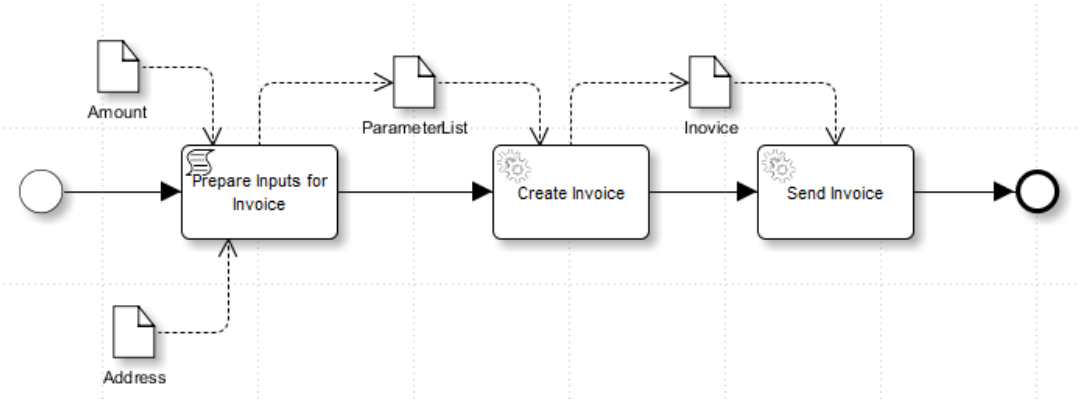


Figure 4.32: The created business process “Issuing Invoice” in Inubit BPM

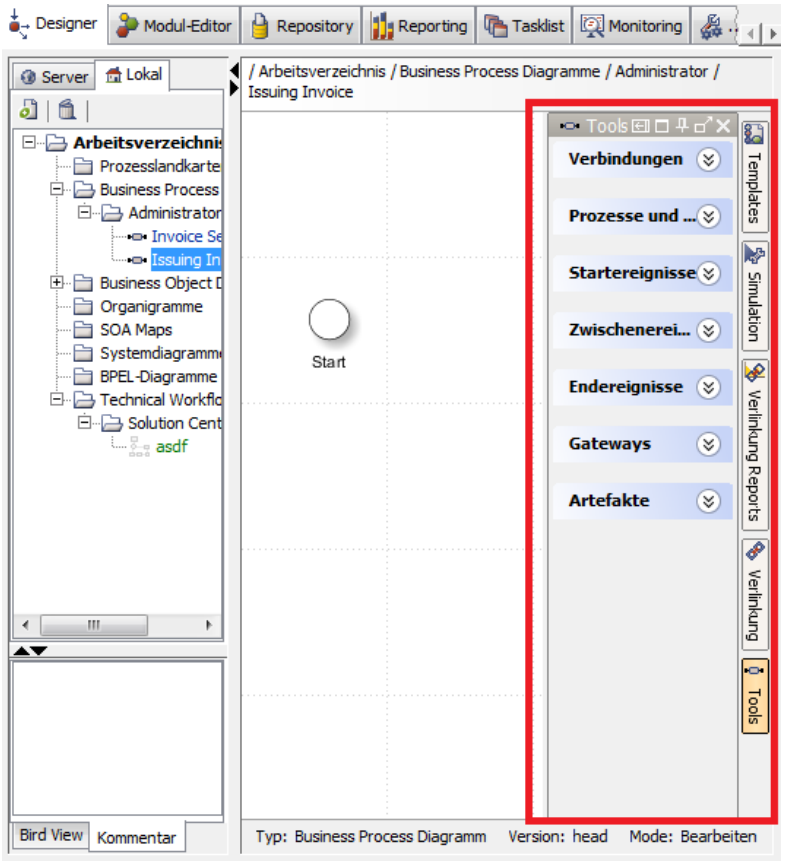


Figure 4.33: The available BPMN notations in Inubit BPM

### 4.3.7 Tool Comparison

In this subsection, the tools are compared based on the created business process example “Issuing Invoice”. Table 4.3 shows the selected tools, the main criteria and the result of the investiga-

tion.

| BPMS | Import running example | Connector to link software | Support Web Service | Script language | Open source | Adjustments for execution | Tool-specific designer (notation) |
|---|---|---|---|---|---|---|---|
| Inubit BPM | | X | X | Javascript | | major | part |
| Sydlee Seed | | X | X | Javascript | | major | part |
| jBPM | X | | X | Java | X | minor | all |
| Activiti | X | | X | Groovy | X | minor | part |
| Bonita BPM | X | X | X | Javascript | X | major | part |
| Camunda BPM | X | | | Groovy | X | minor | all |

Table 4.3: Tool comparsion based on the business process “Issuing Invoice”

In the first column in Table 4.3 (BPMS), the investigated tools are shown.

The second column (Import “Issuing Invoice”) shows which tool was able to import the created business process “Issuing Invoice”. No import was possible with the tools Inubit BPM and Sydlee Seed. The reason for Inubit BPM was that it is not able to import BPMN 2.0 models. The graphical creation of BPMN 2.0 models is possible. In Sydlee Seed, the import and export of BPMN 2.0 models is only available for the commercial edition “On site”. In Activiti, Camunda BPM, Bonita BPM and jBPM importing the business process example was possible without problems.

The third column (Connector to link services) shows which tools use a connector to link a Service Task with a software (e.g., WSDL, Java, etc.).

Connectors are simple to create and are used to connect external services with BPMN. These connectors are often built via XML-structures that can be imported into BPMN models using the import statement of BPMN 2.0. This means that the imported XML-structures are tool-dependent but the overall BPMN models with the import statement are BPMN 2.0 compliant. Therefore, the created models cannot be interpreted by another process engine.

The fourth column (Support Web Service) shows that only Camunda BPM currently does not support links to Web Service, but links only to Java. The other tools support links to Web Services.

The fifth column (Script language) contains the name of the scripting language used by the different tools. The tools use different script languages which is a problem for the portability between the tools.

The sixth column (open source) shows which tools are open source and which are not. Inubit BPM and Sydlee Seed are closed suites, while the other four tools are open source tools. In the latter, the execution of BPMN 2.0 models can be traced, while in closed suites this is impossible.

The seventh column (Adjustments for execution) shows which tools required adjustments to the imported business process “Issuing Invoice” before it could be executed. Here two different types (minor, major) are possible.

“Minor” means that only a few syntax adjustments had to be made. For example, jBPM uses Properties for Data Input and Data Output, while Activiti provides the possibility to use Properties or Item Definitions for Data Input and Data Output. Furthermore, mapping Data Inputs and Data Outputs at the Tasks is solved in different ways in Activiti and jBPM. However, these differences are a result of the BPMN 2.0 specification, because both approaches mentioned

are allowed. Also in jBPM, Data Input and Data Output need to have the attribute “name” for the Service Task in the XML-representation. This attribute is defined in the BPMN 2.0 standard as optional. In jBPM, the value of the attribute “name” has to be “Parameter” for Data Input and “Result” for Data Output. Otherwise, an exception is thrown. Activiti does not need these additional attributes. The reason for this is the different implementation of the process engines of the two tools. A further tool-specific difference between jBPM and Activiti is that the Service Task in jBPM implements the BPMN 2.0 standard exactly, so only one parameter can be passed to the Service Task. In Activiti it is possible to pass multiple parameters.

“Major” means that in addition to the adaptation of the XML-representation of the created example, other resources (like connectors) are needed or that the created business process could not be imported (in case of the tools Inubit BPM and Sydlee Seed). In Bonita BPM, the created business process “Issuing Invoice” can be imported, but the Service Task has to be linked with a connector before the BPMN model can be executed. The connector for the Service Task in Bonita BPM is linked in the import statement of the XML-representation and the implementation of this connector is stored in the tool. The XML-representation of the example is actually BPMN 2.0 standard compliant even though using connectors, but the implementation of the connector is tool-dependent and so the created model cannot be interpreted or executed by another process engines.

The last column (tool-specific designer) describes the ability of the tool-specific designer program to show the notation of the business process “Issuing Invoice” graphically. In some tools, the created business process was directly used or if importing was not possible, the business process was created anew with the tool itself. Here two different arguments (all, part) are possible. The type “all” means that all notations are shown. Only the tools jBPM and Camunda BPM were able to do this. The argument “part” means that only a part of the notation describing the business process was shown. For example Activiti, Bonita BPM and Sydlee Seed Designer could not display Data Inputs, Data Outputs and Data Objects. Inubit BPM could not display Data Inputs and Data Outputs.

Based on Table 4.3, a statement about portability between the tools can be made. Portability is high for Activiti, jBPM and Camunda as the comparison in Table 4.3 suggests. Importing the example model in each of these tools was possible, but minor adaptions had to be done (e.g., script language, Camunda does not support Web Service, etc.) before it could be executed.

Also Inubit BPM, Sydlee Seed and Bonita BPM show similarities as indicated in Table 4.3. They use connectors for links to software, support Web Services, and use the same script language. But in contrast to Bonita BPM, in Inubit BPM the created BPMN 2.0 model is stored in a non-compliant BPMN 2.0 XML-format. This leads to tool-dependent model specifications and the process engine of Bonita BPM is not able to interpret these. Sydlee Seed does not support export in the community version, and so the generated XML-representation could not be analyzed. Therefore, no statements about the portability can be made for Sydlee Seed in this master’s thesis.

Due to the created business process example “Issuing Invoice”, the following notations of BPMN 2.0 were tested explicitly in the selected tools with respect to direct execution. Table 4.4 shows that Inubit BPM, jBPM and Camunda BPM support all of the used notations in the business process example, while Sydlee Seed, Activiti and Bonita BPM do not support Data Object.

Instead of Data Object, Data Input and Data Output can be used here.

| BPMS | Sequence Flow | Association | Service Task | Script Task | Start Event | End Event | Data Object | Data In-/Output |
|---|---|---|---|---|---|---|---|---|
| Inubit BPM | X | X | X | X | X | X | X | X |
| Sydlee Seed | X | X | X | X | X | X | | X |
| jBPM | X | X | X | X | X | X | X | X |
| Activiti | X | X | X | X | X | X | | X |
| Bonita BPM | X | X | X | X | X | X | | X |
| Camunda BPM | X | X | X | X | X | X | X | X |

Table 4.4: Overview of the tested notation in the business process “Issuing Invoice” which can be executed directly

CHAPTER 5

# Conclusion

Since the introduction of BPMN 2.0, this standard provides better support of portability than previous BPMN versions. The graphical representation of the BPMN notation is specified very strictly there. However, the description of some aspects in the BPMN 2.0 standard lacks a comprehensive specification and should be described in more detail. This leads to slightly different XML-representations of the same business process depending on the interpretation of the standard. Additionally, this standard allows specifying business processes in more than one way, which makes it more difficult for tools to be fully BPMN 2.0 compliant, and they often implement only a subset of the specification of BPMN 2.0.

In this master's thesis, the execution of parts of BPMN was investigated. Here different possibilities for the execution of BPMN models were found. In previous versions of BPMN (1.x), BPMN models could only be executed, if they had been transformed into another, executable format (e.g., BPEL). Since the introduction of BPMN 2.0, BPMN models can be executed directly. For this purpose, a BPMN 2.0 standard conform running example was created. The example was kept simple but included all aspects necessary for automatic execution. This example showed that, despite its simplicity, it causes problems with different tools.

The comparison between several BPMN 2.0 compliant tools has been performed with an emphasis on their ability to directly execute BPMN 2.0 models. The above mentioned example was used as a reference business model and it was tried to import and execute it with different tools. Since BPMN 2.0 supposedly supports portability of business process models, it should be possible to pass models to different tools and execute them with every process engine that supports BPMN 2.0. Today, there are several tools on the market for creating and executing BPMN 2.0 models. A subset of the tools was selected for investigation according to specific criteria (Open Source or trial version available, modeling of BPMN 2.0 and direct execution of BPMN 2.0 models).

So, the exchange of BPMN 2.0 models is, in principle, possible between tools. However, the execution of this simple example is not possible without further adjustments, since most tools only support a subset of the BPMN 2.0 standard. While the graphical representations of BPMN 2.0 models are similar with all tools (with exceptions, for example some tools do not provide the

graphical representation of Data Input and Data Output), the XML-representations of BPMN 2.0 models are different. This is an obstacle to the portability of BPMN 2.0 models between tools, and especially it is not possible to execute them with another process engine without specific changes to the XML-representation.

During the investigation of the BPMN 2.0 standard, a pitfall was found. This pitfall relates to the parameters passed to a Service Task and restricts their number to one. Several work-arounds are possible, but using a wrapper to bypass this constraint proved to be the most flexible way to achieve BPMN 2.0 compliant models that can deal with multiple parameters.

In summary, the direct execution of BPMN 2.0 models is possible, but for the execution of the same model in different tools adjustments are needed. To achieve better portability between tools, the tools have to implement support for the entire standard specification and not only a subset of it. In addition, the BPMN 2.0 standard has to be more specific in certain aspects (e.g., how a Web Service is linked to a Service Task) that are important for automated execution.

# List of Figures

APPENDIX A

# Appendix - Tool-specific XML-representations of the running example

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL
    "
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xmlns:activiti="http://activiti.org/bpmn"
6     xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
7     xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
8     xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
9     typeLanguage="http://www.w3.org/2001/XMLSchema"
10    expressionLanguage="http://www.w3.org/1999/XPath"
11    xmlns:tns="http://ict.tuwien.ac.at/proreuse/"
12    targetNamespace="http://ict.tuwien.ac.at/proreuse/"
13    xmlns:invoiceIssuingNS="http://ict.tuwien.ac.at/proreuse/
        InvoiceIssuingMultipleParameters"
14    xmlns:typesInvoiceIssuing="http://ict.tuwien.ac.at/proreuse
        /InvoiceIssuingMultipleParameters/types">
15
16    <import importType="http://schemas.xmlsoap.org/wsdl/"
17            location="http://localhost:9898/ProReUseWebServices
                /services/InvoiceIssuingMultipleParameters?wsdl"
18            namespace="http://ict.tuwien.ac.at/proreuse/
                InvoiceIssuingMultipleParameters"/>
```

```
19
20
21      <message id="createInvoiceRequestMessage" itemRef="tns:
           createInvoiceAmountInputSoap"/>
22      <message id="createInvoiceResponseMessage"  itemRef="tns:
           createInvoiceInvoiceOutputSoap"/>
23      <message id="sendInvoiceRequestMessage" itemRef="tns:
           sendInvoiceAmountInputSoap"/>
24      <message id="sendInvoiceResponseMessage" itemRef="tns:
           sendInvoiceInvoiceOutputSoap"/>
25
26      <itemDefinition id="createInvoiceAmountInputSoap"
           structureRef="invoiceIssuingNS:createInvoiceOperation"/>
27      <itemDefinition id="createInvoiceInvoiceOutputSoap"
           structureRef="invoiceIssuingNS:
           createInvoiceOperationResponse"/>
28      <itemDefinition id="sendInvoiceAmountInputSoap"
           structureRef="invoiceIssuingNS:sendInvoiceOperation"/>
29      <itemDefinition id="sendInvoiceInvoiceOutputSoap"
           structureRef="invoiceIssuingNS:
           sendInvoiceOperationResponse"/>
30
31      <interface name="Issuing Invoice" implementationRef="
           invoiceIssuingNS:InvoiceIssuingMultipleParameters">
32        <operation id="createInvoice"   name="Create Invoice"
             implementationRef="invoiceIssuingNS:
             createInvoiceOperation">
33          <inMessageRef>tns:createInvoiceRequestMessage </
               inMessageRef>
34          <outMessageRef>tns:createInvoiceResponseMessage </
               outMessageRef>
35        </operation>
36        <operation id="sendInvoice" implementationRef="
             invoiceIssuingNS:sendInvoiceOperation" name="Send
             Invoice">
37          <inMessageRef>tns:sendInvoiceRequestMessage</
               inMessageRef>
38          <outMessageRef>tns:sendInvoiceResponseMessage</
               outMessageRef>
39        </operation>
40      </interface>
41
42      <itemDefinition id="amount" structureRef="float"/>
```

```
43      <itemDefinition id="createInvoiceOperationAmountInput"
           structureRef="float"/>
44      <itemDefinition id="address" structureRef="string" />
45      <itemDefinition id="createInvoiceOperationAddressInput"
           structureRef="string"/>
46      <itemDefinition id="createInvoiceOperationInvoiceOutput"
           structureRef="at.ac.tuwien.ict.proreuse.
           InvoiceIssuingMultipleParameters.Invoice"/>
47      <itemDefinition id="invoice" structureRef="at.ac.tuwien.ict
           .proreuse.InvoiceIssuingMultipleParameters.Invoice"/>
48      <itemDefinition id="sendInvoiceOperationInvoiceInput"
           structureRef="at.ac.tuwien.ict.proreuse.
           InvoiceIssuingMultipleParameters.Invoice"/>
49      <itemDefinition id="sendInvoiceOperationOutput"
           structureRef="boolean" />
50      <itemDefinition id="sendInvoiceAmount" structureRef="
           boolean" />
51
52    <process id="CallInvoiceMutlipleParametersWebService" name="
        Call Invoice Complex WebService" isExecutable="true">
53
54      <startEvent id="start" name="Start the Process Issuing
           Invoice"></startEvent>
55      <serviceTask id="createInvoice" name="Create Invoice"
           implementation="##WebService"   operationRef="tns:
           createInvoice" >
56          <ioSpecification>
57            <dataInput itemSubjectRef="tns:
                 createInvoiceAmountInputSoap" id="
                 dataInputOfCreateInvoiceServiceTask"/>
58          <dataInput itemSubjectRef="tns:
               createInvoiceAmountInputSoap" id="
               dataInputOfCreateInvoiceServiceTaskAddress"/>
59          <dataOutput itemSubjectRef="tns:
               createInvoiceInvoiceOutputSoap" id="
               dataOutputOfCreateInvoiceServiceTask"/>
60          <inputSet>
61              <dataInputRefs>dataInputOfCreateInvoiceServiceTask<
                   /dataInputRefs>
62              <dataInputRefs>
                   dataInputOfCreateInvoiceServiceTaskAddress</
                   dataInputRefs>
63            </inputSet>
```

```
64          <outputSet>
65              <dataOutputRefs>
                    dataOutputOfCreateInvoiceServiceTask</
                    dataOutputRefs>
66            </outputSet>
67          </ioSpecification>
68
69          <dataInputAssociation>
70            <sourceRef>amount</sourceRef>
71            <targetRef>createInvoiceOperationAmountInput</
                  targetRef>
72          </dataInputAssociation>
73          <dataInputAssociation>
74            <sourceRef>address</sourceRef>
75            <targetRef>createInvoiceOperationAddressInput</
                  targetRef>
76          </dataInputAssociation>
77          <dataOutputAssociation>
78            <sourceRef>createInvoiceOperationInvoiceOutput</
                  sourceRef>
79            <targetRef>invoice</targetRef>
80          </dataOutputAssociation>
81      </serviceTask>
82
83      <serviceTask id="sendInvoice" name="Send Invoice"
           implementation="##WebService" operationRef="tns:
           sendInvoice" >
84          <ioSpecification>
85            <dataInput itemSubjectRef="tns:
                  sendInvoiceAmountInputSoap" id="
                  dataInputOfSendInvoiceServiceTask"/>
86          <dataOutput itemSubjectRef="tns:
                sendInvoiceInvoiceOutputSoap" id="
                dataOutputOfSendInvoiceServiceTask"/>
87          <inputSet>
88              <dataInputRefs>dataInputOfSendInvoiceServiceTask</
                    dataInputRefs>
89            </inputSet>
90          <outputSet>
91              <dataOutputRefs>dataOutputOfSendInvoiceServiceTask<
                    /dataOutputRefs>
92            </outputSet>
93          </ioSpecification>
```

```
94
95          <dataInputAssociation>
96            <sourceRef>invoice</sourceRef>
97            <targetRef>sendInvoiceOperationInvoiceInput</
                 targetRef>
98          </dataInputAssociation>
99          <dataOutputAssociation>
100           <sourceRef>sendInvoiceOperationOutput</sourceRef>
101           <targetRef>sendInvoiceAmount</targetRef>
102         </dataOutputAssociation>
103     </serviceTask>
104
105
106     <endEvent id="end" name="End of Process Issuing Invoice"></
          endEvent>
107     <sequenceFlow id="flow1" name="" sourceRef="start"
          targetRef="createInvoice"></sequenceFlow>
108     <sequenceFlow id="flow2" name="" sourceRef="createInvoice"
          targetRef="sendInvoice"></sequenceFlow>
109     <sequenceFlow id="flow4" name="" sourceRef="sendInvoice"
          targetRef="end"></sequenceFlow>
110   </process>
111   <bpmndi:BPMNDiagram id="
        BPMNDiagram_CallInvoiceMutlipleParametersWebService">
112     <bpmndi:BPMNPlane bpmnElement="
          CallInvoiceMutlipleParametersWebService" id="
          BPMNPlane_CallInvoiceMutlipleParametersWebService">
113       <bpmndi:BPMNShape bpmnElement="start" id="BPMNShape_start
            ">
114         <omgdc:Bounds height="35" width="35" x="100" y="110"></
              omgdc:Bounds>
115       </bpmndi:BPMNShape>
116       <bpmndi:BPMNShape bpmnElement="createInvoice" id="
            BPMNShape_createInvoice">
117         <omgdc:Bounds height="55" width="105" x="230" y="100"><
              /omgdc:Bounds>
118       </bpmndi:BPMNShape>
119       <bpmndi:BPMNShape bpmnElement="sendInvoice" id="
            BPMNShape_sendInvoice">
120         <omgdc:Bounds height="55" width="105" x="400" y="100"><
              /omgdc:Bounds>
121       </bpmndi:BPMNShape>
122       <bpmndi:BPMNShape bpmnElement="end" id="BPMNShape_end">
```

```
123         <omgdc:Bounds height="35" width="35" x="720" y="110"></
               omgdc:Bounds>
124       </bpmndi:BPMNShape>
125       <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
126         <omgdi:waypoint x="135" y="127"></omgdi:waypoint>
127         <omgdi:waypoint x="230" y="127"></omgdi:waypoint>
128       </bpmndi:BPMNEdge>
129       <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
130         <omgdi:waypoint x="335" y="127"></omgdi:waypoint>
131         <omgdi:waypoint x="470" y="127"></omgdi:waypoint>
132       </bpmndi:BPMNEdge>
133       <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
134         <omgdi:waypoint x="575" y="127"></omgdi:waypoint>
135         <omgdi:waypoint x="720" y="127"></omgdi:waypoint>
136       </bpmndi:BPMNEdge>
137     </bpmndi:BPMNPlane>
138   </bpmndi:BPMNDiagram>
139 </definitions>
```

**Listing A.1:** Generated XML of the issuing invoice with Service Task and multiple parameter

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-
     instance"
3         xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
4         xmlns:activiti="http://activiti.org/bpmn"
5         xmlns:invoiceIssuingns="http://ict.tuwien.ac.at/proreuse
            /InvoiceIssuingComplexWithOneParameter"
6         xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
7         xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
8         xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
9         xmlns:tns="http://ict.tuwien.ac.at/proreuse/"
10        targetNamespace="http://ict.tuwien.ac.at/proreuse/"
11        typeLanguage="http://www.w3.org/2001/XMLSchema"
12        expressionLanguage="http://www.w3.org/1999/XPath">
13
14  <import importType="http://schemas.xmlsoap.org/wsdl/"
15          location="http://localhost:9898/ProReUseWebServices/
      services/InvocieServiceComplexWithOneParameter?wsdl"
16          namespace="http://ict.tuwien.ac.at/proreuse/
      InvoiceIssuingComplexWithOneParameter"/>
17
18
19  <message id="createInvoiceRequestMessage" itemRef="tns:
```

```
    createInvoiceInputSoap"/>
20  <message id="createInvoiceResponseMessage" itemRef="tns:
    createInvoiceInvoiceOutputSoap"/>
21  <message id="sendInvoiceRequestMessage" itemRef="tns:
    sendInvoiceInputSoap"/>
22  <message id="sendInvoiceResponseMessage" itemRef="tns:
    sendInvoiceInvoiceOutputSoap"/>
23
24  <itemDefinition id="createInvoiceInputSoap" structureRef="
    invoiceIssuingns:createInvoiceOperation"/>
25  <itemDefinition id="createInvoiceInvoiceOutputSoap"
    structureRef="invoiceIssuingns:
    createInvoiceOperationResponse"/>
26  <itemDefinition id="sendInvoiceInputSoap" structureRef="
    invocieIssuingns:sendInvoiceOperation"/>
27  <itemDefinition id="sendInvoiceInvoiceOutputSoap"
    structureRef="invoiceIssuingns:sendInvoiceOperationResponse
    "/>
28
29  <interface  id="Interface_1" name="Issuing Invoice"
    implementationRef="invoiceIssuingns:
    InvoiceIssuingComplexWithOneParameter">
30   <operation id="createInvoice"   name="Create Invoice"
     implementationRef="invoiceIssuingns:createInvoiceOperation
     ">
31    <inMessageRef>tns:createInvoiceRequestMessage </
       inMessageRef>
32    <outMessageRef>tns:createInvoiceResponseMessage </
       outMessageRef>
33   </operation>
34   <operation id="sendInvoice" implementationRef="
     invoiceIssuingns:sendInvoiceOperation" name="Send Invoice"
     >
35     <inMessageRef>tns:sendInvoiceRequestMessage</inMessageRef
        >
36     <outMessageRef>tns:sendInvoiceResponseMessage</
        outMessageRef>
37    </operation>
38  </interface>
39
40   <itemDefinition id="createInvoiceOperationOutput"
     structureRef="at.ac.tuwien.ict.proreuse.
     invoiceissuingcomplexwithoneparameter.Invoice"/>
```

```
41  <itemDefinition id="createInvoiceOperationInvoiceInput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingcomplexwithoneparameter.InvoiceParameterList
       "/>
42  <itemDefinition id="sendInvoiceOperationInvoiceInput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingcomplexwithoneparameter.Invoice" />
43  <itemDefinition id="sendInvoiceOperationOutput" structureRef=
       "Boolean" />
44  <itemDefinition id="inputScriptTaskAmount" structureRef="
       Float"/>
45  <itemDefinition id="inputScriptTaskAddress" structureRef="
       String"/>
46  <itemDefinition id="outputScriptTask" structureRef="at.ac.
       tuwien.ict.proreuse.invoiceissuingcomplexwithoneparameter.
       InvoiceParameterList"/>
47  <itemDefinition id="address" structureRef="String"/>
48  <itemDefinition id="amount" structureRef="Float"/>
49  <itemDefinition id="hashmap" structureRef="at.ac.tuwien.ict.
       proreuse.invoiceissuingcomplexwithoneparameter.
       InvoiceParameterList"/>
50  <itemDefinition id="invoice" structureRef="at.ac.tuwien.ict.
       proreuse.invoiceissuingcomplexwithoneparameter.Invoice"/>
51  <itemDefinition id="isInvoicesent" structureRef="Boolean"/>
52
53  <process id="IssuingInvoice" name="Call Issuing Invoice with
       WebService" isExecutable="true">
54
55    <!-- Events -->
56    <startEvent id="StartEvent_1">
57      <outgoing>tns:SequenceFlow_1</outgoing>
58    </startEvent>
59    <endEvent id="EndEvent_1">
60      <incoming>tns:SequenceFlow_4</incoming>
61    </endEvent>
62
63    <!-- Script Task -->
64    <scriptTask id="ScriptTask_1" name="Prepare Input for
         Create Invoice"  scriptFormat="groovy">
65     <ioSpecification id="InputOutputSpecification_5">
66          <dataInput id="dataInputScriptTaskAmount"
               itemSubjectRef="inputScriptTaskAmount" name="
               Amount"/>
```

```
67          <dataInput id="dataInputScriptTaskAddress"
               itemSubjectRef="inputScriptTaskAddress" name="
               Address"/>
68          <dataOutput id="dataOutputScriptTask" itemSubjectRef=
               "outputScriptTask" name="Result"/>
69            <inputSet id="InputSet_5">
70              <dataInputRefs>dataInputScriptTaskAmount</
                   dataInputRefs>
71          </inputSet>
72           <inputSet id="InputSet_9">
73              <dataInputRefs>dataInputScriptTaskAddress</
                   dataInputRefs>
74          </inputSet>
75          <outputSet id="OutputSet_5">
76              <dataOutputRefs>dataOutputScriptTask</
                   dataOutputRefs>
77          </outputSet>
78        </ioSpecification>
79        <dataInputAssociation id="DataInputAssociation_8">
80           <sourceRef>amount</sourceRef>
81          <targetRef>amount</targetRef>
82      </dataInputAssociation>
83      <dataInputAssociation id="DataInputAssociation_10">
84       <sourceRef>address</sourceRef>
85          <targetRef>address</targetRef>
86      </dataInputAssociation>
87      <dataOutputAssociation id="DataOutputAssociation_9">
88      <sourceRef>hashmap</sourceRef>
89          <targetRef>hashmap</targetRef>
90      </dataOutputAssociation>
91      <script>
92
93          def hashmap = new at.ac.tuwien.ict.proreuse.
               invoiceissuingcomplexwithoneparameter.
               InvoiceParameterList();
94          hashmap.addParameter("amount", amount);
95          hashmap.addParameter("address", address);
96      execution.setVariable("hashmap", hashmap);
97
98      </script>
99    </scriptTask>
100
101     <!-- Service Tasks -->
```

```
102     <serviceTask id="ServiceTask_1" name="Create Invoice"
           implementation="##WebService" operationRef="tns:
           createInvoice">
103
104        <ioSpecification id="InputOutputSpecification_3">
105         <dataInput id="dataInputOfCreateInvoiceServiceTask"
               itemSubjectRef="tns:createInvoiceInputSoap"/>
106         <dataOutput id="dataOutputOfCreateInvoiceServiceTask"
               itemSubjectRef="tns:createInvoiceInvoiceOutputSoap"/
               >
107         <inputSet id="InputSet_3">
108           <dataInputRefs>dataInputOfCreateInvoiceServiceTask</
                 dataInputRefs>
109         </inputSet>
110         <outputSet id="OutputSet_3">
111           <dataOutputRefs>dataOutputOfCreateInvoiceServiceTask<
                 /dataOutputRefs>
112         </outputSet>
113       </ioSpecification>
114       <dataInputAssociation id="DataInputAssociation_1">
115         <sourceRef>hashmap</sourceRef>
116         <targetRef>createInvoiceOperationInvoiceInput</
              targetRef>
117       </dataInputAssociation>
118       <dataOutputAssociation id="DataOutputAssociation_3">
119         <sourceRef>createInvoiceOperationOutput</sourceRef>
120         <targetRef>invoice</targetRef>
121       </dataOutputAssociation>
122     </serviceTask>
123
124     <serviceTask id="ServiceTask_2" name="Send Invoice"
           implementation="##WebService" operationRef="tns:
           sendInvoice">
125       <ioSpecification>
126       <dataInput  id="dataInputOfSendInvoiceServiceTask"
            itemSubjectRef="tns:sendInvoiceInputSoap"/>
127     <dataOutput id="dataOutputOfSendInvoiceServiceTask"
           itemSubjectRef="tns:sendInvoiceInvoiceOutputSoap"/>
128     <inputSet>
129         <dataInputRefs>dataInputOfSendInvoiceServiceTask</
              dataInputRefs>
130       </inputSet>
131     <outputSet>
```

```
132         <dataOutputRefs>dataOutputOfSendInvoiceServiceTask</
               dataOutputRefs>
133       </outputSet>
134      </ioSpecification>
135    <dataInputAssociation>
136       <sourceRef>invoice</sourceRef>
137       <targetRef>sendInvoiceOperationInvoiceInput</targetRef>
138      </dataInputAssociation>
139      <dataOutputAssociation>
140       <sourceRef>sendInvoiceOperationOutput</sourceRef>
141       <targetRef>isInvoicesent</targetRef>
142      </dataOutputAssociation>
143     </serviceTask>
144
145      <!-- Sequence Flows -->
146     <sequenceFlow id="SequenceFlow_1" sourceRef="StartEvent_1"
           targetRef="ScriptTask_1"/>
147     <sequenceFlow id="SequenceFlow_2" name="" sourceRef="
           ScriptTask_1" targetRef="ServiceTask_1"/>
148     <sequenceFlow id="SequenceFlow_3" name="" sourceRef="
           ServiceTask_1" targetRef="ServiceTask_2"/>
149     <sequenceFlow id="SequenceFlow_4" name="" sourceRef="
           ServiceTask_2" targetRef="EndEvent_1"/>
150   </process>
151
152 </definitions>
```

**Listing A.2:** Generated XML of the issuing invoice with Script Task as Wrapper in Activiti

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL
     "
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:activiti="http://activiti.org/bpmn"
5         xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
6         xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
7         xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
8         typeLanguage="http://www.w3.org/2001/XMLSchema"
9         expressionLanguage="http://www.w3.org/1999/XPath"
10        targetNamespace="http://www.activiti.org/test">
11
12   <process id="CallInvoiceMutlipleParametersJava" name="Create
        and Send an Invoice" isExecutable="true">
13     <startEvent id="start" name="Start"></startEvent>
```

```
14
15      <serviceTask id="createInvoice" name="Create Invoice"
           activiti:class="CreateInvoice"></serviceTask>
16      <serviceTask id="sendInvoice" name="Send Invoice" activiti:
           class="SendInvoice"></serviceTask>
17
18      <receiveTask id="waitState" />
19
20      <sequenceFlow id="flow1" sourceRef="start" targetRef="
           createInvoice"></sequenceFlow>
21      <sequenceFlow id="flow2" sourceRef="createInvoice"
           targetRef="sendInvoice"></sequenceFlow>
22      <sequenceFlow id="flow4" sourceRef="sendInvoice" targetRef=
           "end"></sequenceFlow>
23
24      <endEvent id="end" name="End"></endEvent>
25
26    </process>
27    <bpmndi:BPMNDiagram id="
         BPMNDiagram_CallInvoiceMutlipleParametersJava">
28      <bpmndi:BPMNPlane bpmnElement="
           CallInvoiceMutlipleParametersJava" id="
           BPMNPlane_CallInvoiceMutlipleParametersJava">
29        <bpmndi:BPMNShape bpmnElement="start" id="BPMNShape_start
             ">
30          <omgdc:Bounds height="35.0" width="35.0" x="130.0" y="
               180.0"></omgdc:Bounds>
31        </bpmndi:BPMNShape>
32        <bpmndi:BPMNShape bpmnElement="createInvoice" id="
             BPMNShape_createInvoice">
33          <omgdc:Bounds height="55.0" width="105.0" x="210.0" y="
               170.0"></omgdc:Bounds>
34        </bpmndi:BPMNShape>
35        <bpmndi:BPMNShape bpmnElement="sendInvoice" id="
             BPMNShape_sendInvoice">
36          <omgdc:Bounds height="55.0" width="105.0" x="370.0" y="
               170.0"></omgdc:Bounds>
37        </bpmndi:BPMNShape>
38        <bpmndi:BPMNShape bpmnElement="end" id="BPMNShape_end">
39          <omgdc:Bounds height="35.0" width="35.0" x="690.0" y="
               180.0"></omgdc:Bounds>
40        </bpmndi:BPMNShape>
41        <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
```

```
42          <omgdi:waypoint x="165.0" y="197.0"></omgdi:waypoint>
43          <omgdi:waypoint x="210.0" y="197.0"></omgdi:waypoint>
44        </bpmndi:BPMNEdge>
45        <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
46          <omgdi:waypoint x="315.0" y="197.0"></omgdi:waypoint>
47          <omgdi:waypoint x="370.0" y="197.0"></omgdi:waypoint>
48        </bpmndi:BPMNEdge>
49        <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
50          <omgdi:waypoint x="475.0" y="197.0"></omgdi:waypoint>
51          <omgdi:waypoint x="690.0" y="197.0"></omgdi:waypoint>
52        </bpmndi:BPMNEdge>
53      </bpmndi:BPMNPlane>
54    </bpmndi:BPMNDiagram>
55 </definitions>
```

**Listing A.3:** Generated XML of the issuing invoice in Activiti with Java integrated

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3             xmlns:invoiceIssuingns="http://ict.tuwien.ac.at/
              proreuse/InvoiceIssuingComplexWithOneParameter"
4             xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
5             xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/
    DI"
6             xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
7             xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
8             xmlns:drools="http://www.jboss.org/drools"
9             xmlns:tns="http://ict.tuwien.ac.at/proreuse/"
10            xmlns:typesInvoiceIssuing="http://ict.tuwien.ac.at/
    proreuse/InvoiceIssuingComplexWithOneParameter/types"
11            xsi:schemaLocation="http://www.omg.org/spec/BPMN
    /20100524/MODEL BPMN20.xsd http://www.jboss.org/drools
    drools.xsd http://www.bpsim.org/schemas/1.0 bpsim.xsd"
12            id="Definitions_1"
13            targetNamespace="http://ict.tuwien.ac.at/proreuse/">
14
15
16  <import importType="http://schemas.xmlsoap.org/wsdl/"
17      location="http://localhost:9898/ProReUseWebServices/
    services/InvoiceIssuingComplexWithOneParameter?wsdl"
18      namespace="http://ict.tuwien.ac.at/proreuse/
    InvoiceIssuingComplexWithOneParameter"/>
19
```

```
20  <itemDefinition id="createInvoiceInputSoap" structureRef="
       invoiceIssuingns:createInvoiceOperation"/>
21  <itemDefinition id="createInvoiceInvoiceOutputSoap"
       structureRef="invoiceIssuingns:
       createInvoiceOperationResponse"/>
22  <itemDefinition id="sendInvoiceInputSoap" structureRef="
       invoiceIssuingns:sendInvoiceOperation"/>
23  <itemDefinition id="sendInvoiceInvoiceOutputSoap"
       structureRef="invoiceIssuingns:
       sendInvoiceOperationResponse"/>
24
25  <itemDefinition id="createInvoiceOperationAmountInput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingcomplexwithoneparameter.InvoiceParameterList
       "/>
26  <itemDefinition id="createInvoiceOperationInvoiceOutput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingcomplexwithoneparameter.Invoice"/>
27  <itemDefinition id="sendInvoiceOperationInvoiceInput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingcomplexwithoneparameter.Invoice"/>
28  <itemDefinition id="sendInvoiceOperationOutput" structureRef=
       "Boolean"/>
29  <itemDefinition id="inputScriptTaskAmount" structureRef="
       Float"/>
30  <itemDefinition id="inputScriptTaskAddress" structureRef="
       String"/>
31  <itemDefinition id="outputScriptTask" structureRef="at.ac.
       tuwien.ict.proreuse.invoiceissuingcomplexwithoneparameter.
       InvoiceParameterList"/>
32  <itemDefinition id="address" structureRef="String"/>
33  <itemDefinition id="amount" structureRef="Float"/>
34  <itemDefinition id="invoice" structureRef="at.ac.tuwien.ict.
       proreuse.invoiceissuingcomplexwithoneparameter.Invoice"/>
35  <itemDefinition id="sendInvoiceAmount" structureRef="Boolean"
       />
36  <itemDefinition id="parameterList" structureRef="at.ac.tuwien
       .ict.proreuse.invoiceissuingcomplexwithoneparameter.
       InvoiceParameterList"/>
37
38  <message id="createInvoiceRequestMessage" itemRef="
       createInvoiceInputSoap" name="createInvoiceRequestMessage"
       />
```

```
39  <message id="createInvoiceResponseMessage" itemRef="
       createInvoiceInvoiceOutputSoap" name="
       createInvoiceResponseMessage"/>
40  <message id="sendInvoiceRequestMessage" itemRef="
       sendInvoiceInputSoap" name="sendInvoiceRequestMessage"/>
41  <message id="sendInvoiceResponseMessage" itemRef="
       sendInvoiceInvoiceOutputSoap" name="
       sendInvoiceResponseMessage"/>
42
43  <interface id="Interface_1" implementationRef="
       InvoiceIssuingComplexWithOneParameter" name="Issuing
       Invoice">
44    <operation id="createInvoice" implementationRef="
         createInvoiceOperation" name="createInvoiceOperation">
45      <inMessageRef>createInvoiceRequestMessage</inMessageRef>
46      <outMessageRef>createInvoiceResponseMessage</
           outMessageRef>
47    </operation>
48    <operation id="sendInvoice" implementationRef="
         sendInvoiceOperation" name="sendInvoiceOperation">
49      <inMessageRef>sendInvoiceRequestMessage</inMessageRef>
50      <outMessageRef>sendInvoiceResponseMessage</outMessageRef>
51    </operation>
52  </interface>
53
54  <process id="CallInvoiceIssuingWithOneParameter" name="Call
       Issuing Invoice" isExecutable="true">
55    <property id="address_pro" itemSubjectRef="address"/>
56    <property id="amount_pro" itemSubjectRef="amount"/>
57    <property id="invoice_pro" itemSubjectRef="invoice"/>
58    <property id="sendInvoiceAmount_pro" itemSubjectRef="
         sendInvoiceAmount"/>
59    <property id="parameterList_pro" itemSubjectRef="
         parameterList"/>
60
61    <startEvent id="start" name="Start the Process Issuing
         Invoice">
62    </startEvent>
63
64    <endEvent id="end" name="End of Process Issuing Invoice">
65    </endEvent>
66
67    <serviceTask id="sendInvoice" name="Send Invoice"
```

```
        implementation="##WebService" operationRef="sendInvoice"
        >
68        <ioSpecification id="InputOutputSpecification_2">
69          <dataInput id="dataInputOfSendInvoiceServiceTask"
              itemSubjectRef="sendInvoiceInputSoap" name="
              Parameter"/>
70          <dataOutput id="dataOutputOfSendInvoiceServiceTask"
              itemSubjectRef="sendInvoiceInvoiceOutputSoap" name="
              Result"/>
71          <inputSet id="InputSet_2">
72            <dataInputRefs>dataInputOfSendInvoiceServiceTask</
                dataInputRefs>
73          </inputSet>
74          <outputSet id="OutputSet_2">
75            <dataOutputRefs>dataOutputOfSendInvoiceServiceTask</
                dataOutputRefs>
76          </outputSet>
77        </ioSpecification>
78        <dataInputAssociation id="DataInputAssociation_3">
79          <sourceRef>invoice_pro</sourceRef>
80          <targetRef>dataInputOfSendInvoiceServiceTask</targetRef
              >
81        </dataInputAssociation>
82        <dataOutputAssociation id="DataOutputAssociation_2">
83          <sourceRef>dataOutputOfSendInvoiceServiceTask</
              sourceRef>
84          <targetRef>sendInvoiceAmount_pro</targetRef>
85        </dataOutputAssociation>
86      </serviceTask>
87
88      <serviceTask id="createInvoice" name="Create Invoice"
          implementation="##WebService" operationRef="
          createInvoice">
89        <ioSpecification id="InputOutputSpecification_1">
90          <dataInput id="dataInputOfCreateInvoiceServiceTask"
              itemSubjectRef="createInvoiceInputSoap" name="
              Parameter"/>
91          <dataOutput id="dataOutputOfCreateInvoiceServiceTask"
              itemSubjectRef="createInvoiceInvoiceOutputSoap" name
              ="Result"/>
92          <inputSet id="InputSet_1">
93            <dataInputRefs>dataInputOfCreateInvoiceServiceTask</
                dataInputRefs>
```

```
94          </inputSet>
95          <outputSet id="OutputSet_1">
96            <dataOutputRefs>dataOutputOfCreateInvoiceServiceTask<
                /dataOutputRefs>
97          </outputSet>
98        </ioSpecification>
99        <dataInputAssociation id="DataInputAssociation_1">
100         <sourceRef>parameterList_pro</sourceRef>
101         <targetRef>dataInputOfCreateInvoiceServiceTask</
              targetRef>
102       </dataInputAssociation>
103       <dataOutputAssociation id="DataOutputAssociation_1">
104         <sourceRef>dataOutputOfCreateInvoiceServiceTask</
              sourceRef>
105         <targetRef>invoice_pro</targetRef>
106       </dataOutputAssociation>
107     </serviceTask>
108
109     <scriptTask id="prepareParameters" name="Prepare Parameter
          for Service Task" scriptFormat="java">
110           <ioSpecification id="InputOutputSpecification_3">
111            <dataInput id="dataInputScriptTaskAmount"
                 itemSubjectRef="inputScriptTaskAmount" name="
                 Amount"/>
112            <dataInput id="dataInputScriptTaskAddress"
                 itemSubjectRef="inputScriptTaskAddress" name="
                 Address"/>
113            <dataOutput id="dataOutputScriptTask"
                 itemSubjectRef="outputScriptTask" name="Result"/
                 >
114            <inputSet id="InputSet_5">
115            <dataInputRefs>dataInputScriptTaskAmount</
                 dataInputRefs>
116            </inputSet>
117             <inputSet id="InputSet_6">
118            <dataInputRefs>dataInputScriptTaskAddress</
                 dataInputRefs>
119            </inputSet>
120            <outputSet id="OutputSet_5">
121              <dataOutputRefs>dataOutputScriptTask</
                   dataOutputRefs>
122            </outputSet>
123         </ioSpecification>
```

```
124          <dataInputAssociation id="DataInputAssociation_8">
125             <targetRef>amount_pro</targetRef>
126          </dataInputAssociation>
127          <dataInputAssociation id="DataInputAssociation_10">
128              <targetRef>address_pro</targetRef>
129          </dataInputAssociation>
130          <dataOutputAssociation id="DataOutputAssociation_9">
131             <targetRef>parameterList_pro</targetRef>
132          </dataOutputAssociation>
133          <script>
134             at.ac.tuwien.ict.proreuse.
                   invoiceissuingcomplexwithoneparameter.
                   InvoiceParameterList parameters = new at.ac.tuwien
                   .ict.proreuse.
                   invoiceissuingcomplexwithoneparameter.
                   InvoiceParameterList();
135             parameters.addParameter(&amp;quot;amount&amp;quot;,
                   amount_pro);
136             parameters.addParameter(&amp;quot;address&amp;quot;,
                   address_pro);
137             kcontext.setVariable(&amp;quot;parameterList_pro&amp;quot;,
                   parameters);
138
139       </script>
140       </scriptTask>
141
142       <sequenceFlow id="flow1" drools:priority="1" name=""
             sourceRef="start" targetRef="prepareParameters"/>
143       <sequenceFlow id="flow4" drools:priority="1" name=""
             sourceRef="sendInvoice" targetRef="end"/>
144       <sequenceFlow id="flow2" drools:priority="1" name=""
             sourceRef="createInvoice" targetRef="sendInvoice"/>
145       <sequenceFlow id="SequenceFlow_1" drools:priority="1" name=
             "" sourceRef="prepareParameters" targetRef="
             createInvoice"/>
146
147    </process>
148    <bpmndi:BPMNDiagram id="
          BPMNDiagram_CallInvoiceMutlipleParametersWebService">
149       <bpmndi:BPMNPlane id="
             BPMNPlane_CallInvoiceMutlipleParametersWebService"
             bpmnElement="CallInvoiceMutlipleParametersWebService">
150          <bpmndi:BPMNShape id="BPMNShape_start" bpmnElement="start
```

```
      ">
151     <dc:Bounds height="35.0" width="35.0" x="100.0" y="
          110.0"/>
152   </bpmndi:BPMNShape>
153   <bpmndi:BPMNShape id="BPMNShape_createInvoice"
        bpmnElement="createInvoice">
154     <dc:Bounds height="55.0" width="105.0" x="340.0" y="
          100.0"/>
155   </bpmndi:BPMNShape>
156   <bpmndi:BPMNShape id="BPMNShape_sendInvoice" bpmnElement=
        "sendInvoice">
157     <dc:Bounds height="55.0" width="105.0" x="530.0" y="
          100.0"/>
158   </bpmndi:BPMNShape>
159   <bpmndi:BPMNShape id="BPMNShape_end" bpmnElement="end">
160     <dc:Bounds height="35.0" width="35.0" x="720.0" y="
          110.0"/>
161   </bpmndi:BPMNShape>
162   <bpmndi:BPMNEdge id="BPMNEdge_flow1" bpmnElement="flow1"
        targetElement="BPMNShape_prepareParameters">
163     <di:waypoint xsi:type="dc:Point" x="135.0" y="127.0"/>
164     <di:waypoint xsi:type="dc:Point" x="190.0" y="128.0"/>
165   </bpmndi:BPMNEdge>
166   <bpmndi:BPMNEdge id="BPMNEdge_flow2" bpmnElement="flow2">
167     <di:waypoint xsi:type="dc:Point" x="445.0" y="127.0"/>
168     <di:waypoint xsi:type="dc:Point" x="530.0" y="127.0"/>
169   </bpmndi:BPMNEdge>
170   <bpmndi:BPMNEdge id="BPMNEdge_flow4" bpmnElement="flow4">
171     <di:waypoint xsi:type="dc:Point" x="635.0" y="127.0"/>
172     <di:waypoint xsi:type="dc:Point" x="720.0" y="127.0"/>
173   </bpmndi:BPMNEdge>
174   <bpmndi:BPMNShape id="BPMNShape_prepareParameters"
        bpmnElement="prepareParameters">
175     <dc:Bounds height="50.0" width="110.0" x="190.0" y="
          103.0"/>
176   </bpmndi:BPMNShape>
177   <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_1" bpmnElement
        ="SequenceFlow_1" sourceElement="
        BPMNShape_prepareParameters" targetElement="
        BPMNShape_createInvoice">
178     <di:waypoint xsi:type="dc:Point" x="300.0" y="128.0"/>
179     <di:waypoint xsi:type="dc:Point" x="340.0" y="127.0"/>
180   </bpmndi:BPMNEdge>
```

```
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

**Listing A.4:** Generated XML of the issuing invoice in jBPM with Script Task which prepare the parameters for the Service Task

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
             xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
             xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/
             DI"
             xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
             xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
             xmlns:drools="http://www.jboss.org/drools"
             xsi:schemaLocation="http://www.omg.org/spec/BPMN
             /20100524/MODEL BPMN20.xsd http://www.jboss.org/
             drools drools.xsd http://www.bpsim.org/schemas/1.0
             bpsim.xsd"
             id="Definitions_1"
             xmlns:tns="http://ict.tuwien.ac.at/proreuse/"
             targetNamespace="http://ict.tuwien.ac.at/proreuse/"
             xmlns:invoiceIssuingNS="http://ict.tuwien.ac.at/
             proreuse/InvoiceIssuingMultipleParameters"
             xmlns:typesInvoiceIssuing="http://ict.tuwien.ac.at/
             proreuse/InvoiceIssuingMultipleParameters/types">


    <import importType="http://schemas.xmlsoap.org/wsdl/"
             location="http://localhost:9898/ProReUseWebServices
                /services/InvoiceIssuingMultipleParameters?wsdl"
             namespace="http://ict.tuwien.ac.at/proreuse/
                InvoiceIssuingMultipleParameters"/>

  <itemDefinition id="createInvoiceInputSoap" structureRef="
     invoiceIssuingNS:createInvoiceOperation"/>
  <itemDefinition id="createInvoiceInvoiceOutputSoap"
     structureRef="invoiceIssuingNS:
     createInvoiceOperationResponse"/>
  <itemDefinition id="sendInvoiceInputSoap" structureRef="
     invoiceIssuingNS:sendInvoiceOperation"/>
  <itemDefinition id="sendInvoiceInvoiceOutputSoap"
     structureRef="invoiceIssuingNS:
```

```
    sendInvoiceOperationResponse"/>
24
25
26  <message id="createInvoiceRequestMessage" itemRef="
       createInvoiceInputSoap"/>
27  <message id="createInvoiceResponseMessage"  itemRef="
       createInvoiceInvoiceOutputSoap"/>
28  <message id="sendInvoiceRequestMessage" itemRef="
       sendInvoiceInputSoap"/>
29  <message id="sendInvoiceResponseMessage" itemRef="
       sendInvoiceInvoiceOutputSoap"/>
30
31
32
33  <interface name="Issuing Invoice" implementationRef="
       InvoiceIssuingMultipleParameters">
34    <operation id="createInvoice"   name="
         createInvoiceOperation" implementationRef="
         createInvoiceOperation">
35      <inMessageRef>createInvoiceRequestMessage </inMessageRef>
36      <outMessageRef>createInvoiceResponseMessage </
           outMessageRef>
37    </operation>
38    <operation id="sendInvoice" implementationRef="
         sendInvoiceOperation" name="sendInvoiceOperation">
39      <inMessageRef>sendInvoiceRequestMessage</inMessageRef>
40       <outMessageRef>sendInvoiceResponseMessage</
             outMessageRef>
41      </operation>
42  </interface>
43
44  <!--<itemDefinition id="amount" structureRef="float"/>
45  <itemDefinition id="address" structureRef="string" />  -->
46  <itemDefinition id="createInvoiceOperationAmountInput"
       structureRef="float"/>
47  <itemDefinition id="createInvoiceOperationAddressInput"
       structureRef="string"/>
48  <itemDefinition id="createInvoiceOperationInvoiceOutput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingmultipleparameters.Invoice"/>
49  <itemDefinition id="sendInvoiceOperationInvoiceInput"
       structureRef="at.ac.tuwien.ict.proreuse.
       invoiceissuingmultipleparameters.Invoice"/>
```

```
50  <itemDefinition id="sendInvoiceOperationOutput" structureRef=
       "boolean" />
51
52
53  <process id="CallInvoiceMutlipleParametersWebService" name="
       Call Invoice Complex WebService" isExecutable="true">
54
55    <property id="address_pro"/>
56    <property id="amount_pro"/>
57    <property id="invoice_pro"/>
58    <property id="sendInvoiceAmount_pro"/>
59
60
61    <startEvent id="start" name="Start the Process Issuing
         Invoice"></startEvent>
62    <serviceTask id="createInvoice" name="Create Invoice"
         implementation="##WebService"   operationRef="
         createInvoice" >
63       <ioSpecification>
64         <dataInput itemSubjectRef="createInvoiceInputSoap" id
              ="dataInputOfCreateInvoiceServiceTask" name="
              Amount"/>
65       <dataInput itemSubjectRef="createInvoiceInputSoap" id="
            dataInputOfCreateInvoiceServiceTaskAddress" name="
            Address"/>
66       <dataOutput itemSubjectRef="
            createInvoiceInvoiceOutputSoap" id="
            dataOutputOfCreateInvoiceServiceTask" name="Result"/
            >
67       <inputSet>
68           <dataInputRefs>dataInputOfCreateInvoiceServiceTask<
                /dataInputRefs>
69           <dataInputRefs>
                dataInputOfCreateInvoiceServiceTaskAddress</
                dataInputRefs>
70         </inputSet>
71       <outputSet>
72           <dataOutputRefs>
                dataOutputOfCreateInvoiceServiceTask</
                dataOutputRefs>
73         </outputSet>
74       </ioSpecification>
75
```

```
76          <dataInputAssociation>
77            <sourceRef>amount_pro</sourceRef>
78            <targetRef>dataInputOfCreateInvoiceServiceTask</
                 targetRef>
79          </dataInputAssociation>
80          <dataInputAssociation>
81            <sourceRef>address_pro</sourceRef>
82            <targetRef>dataInputOfCreateInvoiceServiceTaskAddress
                 </targetRef>
83          </dataInputAssociation>
84          <dataOutputAssociation>
85            <sourceRef>dataOutputOfCreateInvoiceServiceTask</
                 sourceRef>
86            <targetRef>invoice_pro</targetRef>
87          </dataOutputAssociation>
88      </serviceTask>
89
90      <serviceTask id="sendInvoice" name="Send Invoice"
           implementation="##WebService" operationRef="sendInvoice"
            >
91          <ioSpecification>
92            <dataInput itemSubjectRef="sendInvoiceInputSoap" id="
                 dataInputOfSendInvoiceServiceTask" name="Invoice"/
                 >
93          <dataOutput itemSubjectRef="
               sendInvoiceInvoiceOutputSoap" id="
               dataOutputOfSendInvoiceServiceTask" name="Result"/>
94          <inputSet>
95              <dataInputRefs>dataInputOfSendInvoiceServiceTask</
                   dataInputRefs>
96            </inputSet>
97          <outputSet>
98              <dataOutputRefs>dataOutputOfSendInvoiceServiceTask<
                   /dataOutputRefs>
99            </outputSet>
100         </ioSpecification>
101
102         <dataInputAssociation>
103           <sourceRef>invoice_pro</sourceRef>
104           <targetRef>dataInputOfSendInvoiceServiceTask</
                 targetRef>
105         </dataInputAssociation>
106         <dataOutputAssociation>
```

```
107           <sourceRef>dataOutputOfSendInvoiceServiceTask</
                 sourceRef>
108           <targetRef>sendInvoiceAmount_pro</targetRef>
109         </dataOutputAssociation>
110     </serviceTask>
111
112
113     <endEvent id="end" name="End of Process Issuing Invoice"></
           endEvent>
114     <sequenceFlow id="flow1" name="" sourceRef="start"
           targetRef="createInvoice"></sequenceFlow>
115     <sequenceFlow id="flow2" name="" sourceRef="createInvoice"
           targetRef="sendInvoice"></sequenceFlow>
116     <sequenceFlow id="flow4" name="" sourceRef="sendInvoice"
           targetRef="end"></sequenceFlow>
117   </process>
118   <bpmndi:BPMNDiagram id="
         BPMNDiagram_CallInvoiceMutlipleParametersWebService">
119     <bpmndi:BPMNPlane bpmnElement="
           CallInvoiceMutlipleParametersWebService" id="
           BPMNPlane_CallInvoiceMutlipleParametersWebService">
120       <bpmndi:BPMNShape bpmnElement="start" id="BPMNShape_start
             ">
121         <dc:Bounds height="35" width="35" x="100" y="110"></dc:
               Bounds>
122       </bpmndi:BPMNShape>
123       <bpmndi:BPMNShape bpmnElement="createInvoice" id="
             BPMNShape_createInvoice">
124         <dc:Bounds height="55" width="105" x="230" y="100"></dc
               :Bounds>
125       </bpmndi:BPMNShape>
126       <bpmndi:BPMNShape bpmnElement="sendInvoice" id="
             BPMNShape_sendInvoice">
127         <dc:Bounds height="55" width="105" x="400" y="100"></dc
               :Bounds>
128       </bpmndi:BPMNShape>
129       <bpmndi:BPMNShape bpmnElement="end" id="BPMNShape_end">
130         <dc:Bounds height="35" width="35" x="720" y="110"></dc:
               Bounds>
131       </bpmndi:BPMNShape>
132       <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
133         <di:waypoint x="135" y="127"></di:waypoint>
134         <di:waypoint x="230" y="127"></di:waypoint>
```

```
135       </bpmndi:BPMNEdge>
136       <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
137         <di:waypoint x="335" y="127"></di:waypoint>
138         <di:waypoint x="470" y="127"></di:waypoint>
139       </bpmndi:BPMNEdge>
140       <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
141         <di:waypoint x="575" y="127"></di:waypoint>
142         <di:waypoint x="720" y="127"></di:waypoint>
143       </bpmndi:BPMNEdge>
144     </bpmndi:BPMNPlane>
145   </bpmndi:BPMNDiagram>
146 </definitions>
```

**Listing A.5:** Generated XML of the issuing invoice in jBPM with multiple Parameter at Service Task

```
1   public void executeWorkItem(WorkItem workItem, final
       WorkItemManager manager) {
2          String implementation = (String) workItem.getParameter(
              "implementation");
3          if ("##WebService".equalsIgnoreCase(implementation)) {
4              String interfaceRef = (String) workItem.
                  getParameter("interfaceImplementationRef");
5              String operationRef = (String) workItem.
                  getParameter("operationImplementationRef");
6              Object parameter = workItem.getParameter("Parameter
                  ");
7              WSMode mode = WSMode.valueOf(workItem.getParameter(
                  "mode") == null ? "SYNC" : ((String) workItem.
                  getParameter("mode")).toUpperCase());
8
9              try {
10                  Client client = getWSClient(workItem,
                       interfaceRef);
11                  switch (mode) {
12                 case SYNC:
13                     Object[] result = client.invoke(
                           operationRef, parameter);
14
15                     Map<String, Object> output = new HashMap<
                           String, Object>();
16
17                     if (result == null || result.length == 0) {
18                       output.put("Result", null);
```

```
19                      } else {
20                        output.put("Result", result[0]);
21                      }
22
23                      manager.completeWorkItem(workItem.getId(),
                           output);
24                      break;
25                  case ASYNC:
26                      final ClientCallback callback = new
                           ClientCallback();
27                      final long workItemId = workItem.getId();
28                      client.invoke(callback, operationRef,
                           parameter);
29                      new Thread(new Runnable() {
30
31                         public void run() {
32
33                             try {
34
35                                 Object[] result = callback.get(
                                       asyncTimeout, TimeUnit.
                                       SECONDS);
36                                 Map<String, Object> output = new
                                        HashMap<String, Object>();
37                                 if (callback.isDone()) {
38                                     if (result == null) {
39                                       output.put("Result", null)
                                            ;
40                                     } else {
41                                       output.put("Result",
                                            result[0]);
42                                     }
43                                 }
44                                 ksession.getWorkItemManager().
                                       completeWorkItem(workItemId,
                                       output);
45                             } catch (Exception e) {
46                                logger.error("Error encountered
                                     while invoking ws operation
                                     asynchronously ", e);
47                             }
48
49
```

```
50                          }
51                      }).start();
52                      break;
53                  case ONEWAY:
54                      ClientCallback callbackFF = new
                            ClientCallback();
55
56                      client.invoke(callbackFF, operationRef,
                            parameter);
57                      manager.completeWorkItem(workItem.getId(),
                             new HashMap<String, Object>());
58                      break;
59                  default:
60                      break;
61                  }
62
63              } catch (Exception e) {
64                  logger.error("Error when executing work item",
                        e);
65              }
66          } else {
67              executeJavaWorkItem(workItem, manager);
68          }
69
70
71      }
```

**Listing A.6:** Methode executeWorkItem of the general Service Task Handler for Web Service

```
1     public void executeJavaWorkItem(WorkItem workItem,
          WorkItemManager manager) {
2       String i = (String) workItem.getParameter("Interface");
3         String operation = (String) workItem.getParameter("
              Operation");
4         String parameterType = (String) workItem.getParameter("
              ParameterType");
5 //          Object parameter = workItem.getParameter("Parameter")
      ;
6
7         Map<String, Object> parameters = new HashMap<String,
              Object>(workItem.getParameters());
8         parameters.remove("Interface");
9         parameters.remove("Operation");
10        parameters.remove("ParameterType");
```

```
11          parameters.remove("implementation");
12          parameters.remove("operationImplementationRef");
13          parameters.remove("interfaceImplementationRef");
14
15          String [] paramTyp = parameterType.split("-");
16
17          try {
18              Class<?> c = Class.forName(i);
19              Object instance = c.newInstance();
20              Class<?>[] classes = null;
21              Object[] params = null;
22              if (parameterType != null) {
23                  classes = new Class<?>[parameters.size()];
24                  params = new Object[parameters.size()];
25                  int counter =0;
26                  for (Object param : parameters.values()) {
27                    params[counter] = param;
28                    classes[counter] = Class.forName(paramTyp[
                          counter]);
29                    counter++;
30                  }
31              }
32              Method method = c.getMethod(operation, classes);
33              Object result = method.invoke(instance, params);
34              Map<String, Object> results = new HashMap<String,
                    Object>();
35              results.put("Result", result);
36              manager.completeWorkItem(workItem.getId(), results)
                    ;
37          } catch (ClassNotFoundException e) {
38              System.err.println(e);
39          } catch (InstantiationException e) {
40              System.err.println(e);
41          } catch (IllegalAccessException e) {
42              System.err.println(e);
43          } catch (NoSuchMethodException e) {
44              System.err.println(e);
45          } catch (InvocationTargetException e) {
46              System.err.println(e);
47          }
48      }
```

**Listing A.7:** Methode executeWorkItem of the general Service Task Handler for automated application

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpmn2:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3 xmlns:activiti="http://activiti.org/bpmn"
4 xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
5 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
6 xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
7 xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
8 xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
     BPMN20.xsd" id="_TFtssJryEeOf74lvtDWE5g" targetNamespace="
    http://activiti.org/bpmn">
9
10  < bpmn2:itemDefinition id="inputScriptTaskAmount" structureRef
      ="Float"/>
11   <bpmn2:itemDefinition id="inputScriptTaskAddress"
       structureRef="String"/>
12   <bpmn2:itemDefinition id="outputScriptTask" structureRef="
       InvoiceParameterList"/>
13   <bpmn2:itemDefinition id="hashmap" structureRef="
       InvoiceParameterList"/>
14   <bpmn2:itemDefinition id="address" structureRef="String"/>
15   <bpmn2:itemDefinition id="amount" structureRef="Float"/>
16   <bpmn2:itemDefinition id="createInvoiceOperationInvoiceInput"
        itemKind="Physical" structureRef="InvoiceParameterList"/>
17   <bpmn2:itemDefinition id="createInvoiceOperationOutput"
       itemKind="Physical" structureRef="Invoice"/>
18   <bpmn2:itemDefinition id="invoice" structureRef="Invoice" />
19   <bpmn2:itemDefinition id="sendInvocieOperationOutput"
       structureRef="Boolean" />
20   <bpmn2:itemDefinition id="isInvoiceSent" itemKind="Physical"
       structureRef="Boolean"/>
21   <bpmn2:itemDefinition id="sendInvoiceOperationInvoiceInput"
       structureRef="Invoice" />
22
23   <bpmn2:process id="Issuing Invoice" isExecutable="true">
24
25     <bpmn2:startEvent id="StartEvent_1" name="Start Event">
26       <bpmn2:outgoing>SequenceFlow_1</bpmn2:outgoing>
27     </bpmn2:startEvent>
28
29     <bpmn2:scriptTask id="ScriptTask_1" name="Script Task"
          scriptFormat="groovy">
30       <bpmn2:ioSpecification id="InputOutputSpecification_5">
```

```
31          <bpmn2:dataInput id="dataInputScriptTaskAmount"
               itemSubjectRef="inputScriptTaskAmount" name="Amount"
               />
32          <bpmn2:dataInput id="dataInputScriptTaskAddress"
               itemSubjectRef="inputScriptTaskAddress" name="
               Address"/>
33          <bpmn2:dataOutput id="dataOutputScriptTask"
               itemSubjectRef="outputScriptTask" name="Result"/>
34           <bpmn2:inputSet id="InputSet_5">
35               <bpmn2:dataInputRefs>dataInputScriptTaskAmount</
                    bpmn2:dataInputRefs>
36            </bpmn2:inputSet>
37            <bpmn2:inputSet id="InputSet_9">
38               <bpmn2:dataInputRefs>dataInputScriptTaskAddress</
                    bpmn2:dataInputRefs>
39            </bpmn2:inputSet>
40            <bpmn2:outputSet id="OutputSet_5">
41               <bpmn2:dataOutputRefs>dataOutputScriptTask</bpmn2
                    :dataOutputRefs>
42            </bpmn2:outputSet>
43         </bpmn2:ioSpecification>
44        <bpmn2:dataInputAssociation id="DataInputAssociation_1">
45          <bpmn2:sourceRef>amount</bpmn2:sourceRef>
46          <bpmn2:targetRef>amount</bpmn2:targetRef>
47        </bpmn2:dataInputAssociation>
48        <bpmn2:dataInputAssociation id="DataInputAssociation_2">
49          <bpmn2:sourceRef>address</bpmn2:sourceRef>
50             <bpmn2:targetRef>address</bpmn2:targetRef>
51        </bpmn2:dataInputAssociation>
52        <bpmn2:dataOutputAssociation id="DataOutputAssociation_1"
             >
53          <bpmn2:targetRef>hashmap</bpmn2:targetRef>
54        </bpmn2:dataOutputAssociation>
55        <bpmn2:script>
56            def hashmap = new InvoiceParameterList();
57        hashmap.addParameter(&amp;quot;amount&amp;quot;, amount);
58        hashmap.addParameter(&amp;quot;address&amp;quot;, address);;
59        execution.setVariable(&amp;quot;hashmap&amp;quot;, hashmap);
60        </bpmn2:script>
61      </bpmn2:scriptTask>
62
63      <bpmn2:sequenceFlow id="SequenceFlow_1" name="" sourceRef="
           StartEvent_1" targetRef="ScriptTask_1"/>
```

```
64    <bpmn2:sequenceFlow id="SequenceFlow_2" name="" sourceRef="
         ScriptTask_1" targetRef="ServiceTask_1"/>
65
66    <bpmn2:serviceTask id="ServiceTask_1" activiti:class="
         CreateInvoice" name="Create Invoice">
67      <bpmn2:ioSpecification id="InputOutputSpecification_3">
68        <bpmn2:dataInput id="dICreateInvoice" itemSubjectRef="
            createInvoiceOperationInvoiceInput" name="Parameter"
            />
69        <bpmn2:dataOutput id="dOCreateInvoice" itemSubjectRef="
            createInvoiceOperationOutput" name="Result"/>
70        <bpmn2:inputSet id="InputSet_3">
71          <bpmn2:dataInputRefs>dICreateInvoice</bpmn2:
              dataInputRefs>
72        </bpmn2:inputSet>
73        <bpmn2:outputSet id="OutputSet_3">
74          <bpmn2:dataOutputRefs>dOCreateInvoice</bpmn2:
              dataOutputRefs>
75        </bpmn2:outputSet>
76      </bpmn2:ioSpecification>
77      <bpmn2:dataInputAssociation id="DataInputAssociation_3">
78        <bpmn2:sourceRef>hashmap</bpmn2:sourceRef>
79        <bpmn2:targetRef>createInvoiceOperationInvoiceInput</
            bpmn2:targetRef>
80      </bpmn2:dataInputAssociation>
81      <bpmn2:dataOutputAssociation id="DataOutputAssociation_4"
           >
82        <bpmn2:sourceRef>createInvoiceOperationOutput</bpmn2:
            sourceRef>
83        <bpmn2:targetRef>invoice</bpmn2:targetRef>
84      </bpmn2:dataOutputAssociation>
85    </bpmn2:serviceTask>
86
87    <bpmn2:serviceTask id="ServiceTask_2" activiti:class="
         SendInvoice" name="Send Invoice">
88        <bpmn2:ioSpecification>
89      <bpmn2:dataInput  id="dISendInvocice" itemSubjectRef="
          sendInvoiceOperationInvoiceInput" name="Parameter"/>
90    <bpmn2:dataOutput id="dOSendInvoice" itemSubjectRef="
         sendInvocieOperationOutput" name="Result"/>
91    <bpmn2:inputSet>
92        <bpmn2:dataInputRefs>dISendInvocice</bpmn2:
            dataInputRefs>
```

```
93        </bpmn2:inputSet>
94      <bpmn2:outputSet>
95          <bpmn2:dataOutputRefs>dOSendInvoice</bpmn2:
              dataOutputRefs>
96        </bpmn2:outputSet>
97       </bpmn2:ioSpecification>
98     <bpmn2:dataInputAssociation>
99        <bpmn2:sourceRef>invoice</bpmn2:sourceRef>
100       <bpmn2:targetRef>sendInvoiceOperationInvoiceInput</bpmn2:
            targetRef>
101      </bpmn2:dataInputAssociation>
102      <bpmn2:dataOutputAssociation>
103       <bpmn2:sourceRef>sendInvocieOperationOutput</bpmn2:
            sourceRef>
104       <bpmn2:targetRef>isInvoiceSent</bpmn2:targetRef>
105      </bpmn2:dataOutputAssociation>
106    </bpmn2:serviceTask>
107
108    <bpmn2:sequenceFlow id="SequenceFlow_3" name="" sourceRef="
         ServiceTask_1" targetRef="ServiceTask_2"/>
109    <bpmn2:endEvent id="EndEvent_1" name="End Event">
110      <bpmn2:incoming>SequenceFlow_4</bpmn2:incoming>
111    </bpmn2:endEvent>
112    <bpmn2:sequenceFlow id="SequenceFlow_4" name="" sourceRef="
         ServiceTask_2" targetRef="EndEvent_1"/>
113
114  </bpmn2:process>
115
116 </bpmn2:definitions>
```

**Listing A.8:** Generated XML of the issuing invoice in Camunda with Java integrated

# Bibliography


[1] Business Process Model and Notation (BPMN) Version 2.0. OMG http://www.omg.org/spec/BPMN/2.0/, 2011. [Online; accessed 15-March-2014].

[2] Thomas Allweyer. *BPMN 2.0 - Business Process Model and Notation-Einführung in den Standard für die Geschäftsmodellierung*. Books on Demand GmbH, Norderstedt, 2nd edition edition, 2009.

[3] Michele Chinosi and Alberto Trombetta. BPMN: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124 – 134, 2012.

[4] Jakob Freund, Bernd Rücker, and Thoma Henninger. *Praxishandbuch BPMN*. Hanser Verlag München Wien, 2010.

[5] Orlin Genchev and John Galletly. XPDL: bringing business and software together – a case study. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '09, pages 32:1–32:6, New York, NY, USA, 2009. ACM.

[6] Manuel Götz. BPMN 2.0 Tutorial – Kompakte Einführung in die BPMN 2.0. ITransparent GmbH. http://www.itransparent.de/sites/default/files/BPMN_2_0_Tutorial_Business_Process_Modeling_Notation_Deutsch.pdf, 2011. [Online; accessed 17-March-2014].

[7] Christian Gutschier, Ralph Hoch, Hermann Kaindl, and Roman Popp. A Pitfall with BPMN Execution. In *The Second International Conference on Building and Exploring Web Based Environments*, WEB '14 [1], 2014.

[8] Matthias Knoll. Modellgetriebene Spezifikation von BPMN und Transformation von BPMN zu BPEL mit openArchitectureWare. Master's thesis, Vienna University of Technology, 2008.

[9] Agnes Koschmider. *Ähnlichkeitsbasierte Modellierungsunterstützung für Geschäftsprozesse*. Universitätsverlag Karlsruhe, 2007.

[10] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. From business process models to process-oriented software systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1):2:1–2:37, August 2009.


[1] to appear

[11] Chun Ouyang, Marlon Dumas, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. From BPMN Process Models to BPEL Web Services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 285–292, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Tijs Rademakers. *Activiti in Action: Executable Business Process in BPMN 2.0*. Manning Publications Company, 2012.

[13] D. Schumm, D. Karastoyanova, F. Leymann, and J. Nitzsche. On Visualizing and Modelling BPEL with BPMN. In *Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the*, pages 80–87, 2009.

[14] Thomas Tschach. Übersetzung von graf- und blockorientierten Prozessmodellierungssprachen. Vienna University of Technology. http://www.cvast.tuwien.ac.at/sites/default/files/StARF_SE_0525381_Tschach_Thomas.pdf, 2012. [Online; accessed 13-March-2014].

[15] Tammo van Lessen, Daniel Lübke, and Jörg Nitzsche. *Geschäftsprozesse automatisieren mit BPEL*. dpunkt.verlag, 1st edition, 2011.