Generation of optimal business processes from business rules

Master thesis

Bas Steen

Generation of optimal business processes from business rules

Author	: Bas Steen
Student number	: 0063150
Email	: b.steen@alumnus.utwente.nl
Educational institute	: University of Twente
Study	: Msc. Business Information Technology
Organization	: Logica, Arnhem
Internal Supervisors	: Ing. Martijn Vlietstra and Dr. Edwin Hendriks
First Supervisor UT	: Dr. Luís Ferreira Pires
Second Supervisor UT	: Dr. Maria-Eugenia Iacob

Arnhem, May 2009

Abstract

Business process modeling is increasingly used as a method for business process improvement, either as a means to analyze or automate the business process. Business process modeling uses business process modeling languages to define the business process. There are however several problems when using these languages, mainly that they are not understandable for business people and that they are procedural languages, which means that they specify how business processes should be executed and in what order, making the business process inflexible if implemented in a workflow management system.

Inspired by these problems declarative languages where developed. Declarative languages specify what is, must, ought and can be done, without prescribing how it should be done. In the context of business processes business rule languages are declarative languages.

Using business rule languages for specifying business processes could solve some of the problems mentioned above. However, in order to automate the specified business process with a workflow management system, some part of the how, namely the order of activities, is still needed. The business rule specification thus needs to be transformed to an business process model and this has to be done automatically and systematically so that the business rule specification is always interpreted the same way.

This thesis presents a method to automatically transform business rules, written in the PA-notation, to business processes, modeled in BPMN. The approach taken to perform the transformation is to derive the dependencies between activities from the business rules, use the dependencies to determine the optimal sequence of activities and optimal allocation of resources, and build a business process model based on that.

The method has been implemented in a prototype and verified using a case study. Given that the business rule specification is complete and consistent, our method is capable of automatically transforming business rules to optimal business processes.

Preface

This master's thesis is the result of eight months of hard work carried out at Logica in Arnhem. The results of this research would not have been possible without the help of various people.

First of all I would like to thank my supervisors at the university, Luís Ferreira Pires and Maria-Eugenia Iacob for providing me with ideas on what to research, guidance on how to structure my research and valuable comments on how I could improve my work.

I would also like to express my gratitude to my supervisors at Logica, first Martijn Vlietstra for keeping sure that the project stayed on the right course and reviewing my work and second, Edwin Hendriks for his enthusiasm, for giving me valuable insights in the field of business process improvement, for the discussions we had and for doing everything he can to keep me at Logica albeit these difficult times. Also, I would like to thank the fellow Working Tomorrow students at Logica Arnhem for making the time at Logica very enjoyable.

Finally, I would like to thank my parents for believing in me and for giving me the room and resources to go my own way.

Bas Steen
Arnhem, 6 May 2009

Contents

1 Introduction

This chapter presents the motivation and objective, the research questions that aid to reach the objective, the approach taken and the outline of the performed research.

1.1 Motivation and objective

In recent years, business process modeling is increasingly used as a method for business process improvement. There are two kinds of business process models. The first kind of model is used to document current or future business processes. This model can have several purposes, it can serve as a means for discussion between different stakeholders in order to identify possible business process improvements and it can serve as a means to capture the requirements for developing the second kind of model, an executable one. Executable business process models can be directly inserted in a workflow management system which is used to support and automate the modeled business processes (Reijers & Aalst, 2005).

Both kind of business process models are defined using a business process modeling language. A language used for documentation is, for instance, BPMN (Object Management Group, 2008). A language used for execution is, for instance, WS-BPEL (Jordan, et al., 2007)

However, there are several problems when using these kinds of modeling languages, especially if they are used as a basis for a workflow management system:

- They are not understandable for business people. This means that the business process model has to be modeled by IT-people who are not directly involved in the business process. This has two main consequences:
    1. The business process model inevitably contains false interpretations about the actual business process (Muehlen, Indulska, & Kamp, 2008) and because business people do not understand the business process model these false interpretations are not recognized;
    2. When a business process changes, due to, for example, changed regulations, the business process model has to be modified. Because these modifications cannot be made by business people themselves, but have to be made by IT-people, it takes more time and money.
- They are procedural languages, which means they specify how business processes should be executed and in what order. If the model is used for direct execution, this order is often strictly enforced in the resulting system. This is not necessarily bad as the order defined is often defined as being the most optimal order and enforcing this order thus enforces people to work in the optimal way. It does, however, leave users with little opportunity to deviate from this process making the business process less flexible.

These problems inspired the development of declarative languages that are understandable for business people. Declarative languages specify what is, must, ought and can be done, without prescribing how it should be done (Muehlen, Indulska, & Kamp, 2008). In the context of business processes, business rule

languages are declarative languages. Today there are several business rule languages, such as SBVRSE (Object Management Group, 2008) or AceRules (University of Zurich).

Some of the problems mentioned above could be solved by using business rule languages for specifying business processes. For example, a business process specified in a business rule language is understandable for both IT-people and business people and thus will contain less false interpretations.

However, in order to automate the specified business process with a workflow management system some part of the how, namely the order of activities, is still needed. The business rule specification thus needs to be transformed to an executable business process model. This transformation has to be done by IT-people which means that some of the problems related to interpretation are still present.

A better solution would be to automatically transform a business rules specification to an executable business process model. In this approach the interpretations made in the transformation process are documented and standardized which means that a business rule specification is always interpreted the same way. Another advantage of automatically transforming a business rule specification to a business process model is that it is faster than manual transformation.

When transforming a business rule specification to an executable business process model, all the information needed for the executable business process model needs to be present in the business rule specification. This is a problem because, business people, who define or at least verify the business rules, generally do not know anything about all the technical information needed, such as how to connect to a certain external system.

An alternative solution would be to transform the business rule specification to a business process model used for documentation, and let the IT-people transform that specification to an executable business process model. Using this approach the most important part of the business process model, the order of activities, is already defined and the IT-people only need to add the technical details, leaving no room for wrong interpretation.

Based on the motivation given above, the main objective of this project is defined as follows:

"To develop a method to automatically transform a specification of business rules, written in a language that is understood by business people, to an optimal business process specification, written in a business process modeling language used for documentation."

1.2 Research questions

To reach the objective stated in the previous section, several questions need to be answered. The main research question is as follows:

"*How can we automatically transform a specification of business rules, written in a business rule language, into an optimal specification of business processes, written in a business process modeling language used for documentation?*"

The research questions that can be derived from this main research question are:

1. What are business rules?
    a. What types of business rules are relevant in the context of this research?
    b. Which languages are available to specify business rules?
    c. Which language is the most suitable to specify business rules?
2. What are business processes?
    a. Which languages are available to specify business processes?
    b. Which language is the most suitable to specify business processes?
    c. Which criteria should be used to characterize an optimal process?
3. How can we transform business rules to business processes?
    a. What methods and tools are available for the transformation of different languages?
    b. How to transform business rules languages into business process languages?
    c. How can optimization criteria be incorporated in such transformations?

1.3 Research approach

To explain the approach used to reach the goal of this research the research model of (Verschuren, P. and Doorewaard, H. 2000) is used. The research model of Figure 1 shows the global steps necessary to achieve the goals of this research. To clarify the research model and our approach, each step is explained below.

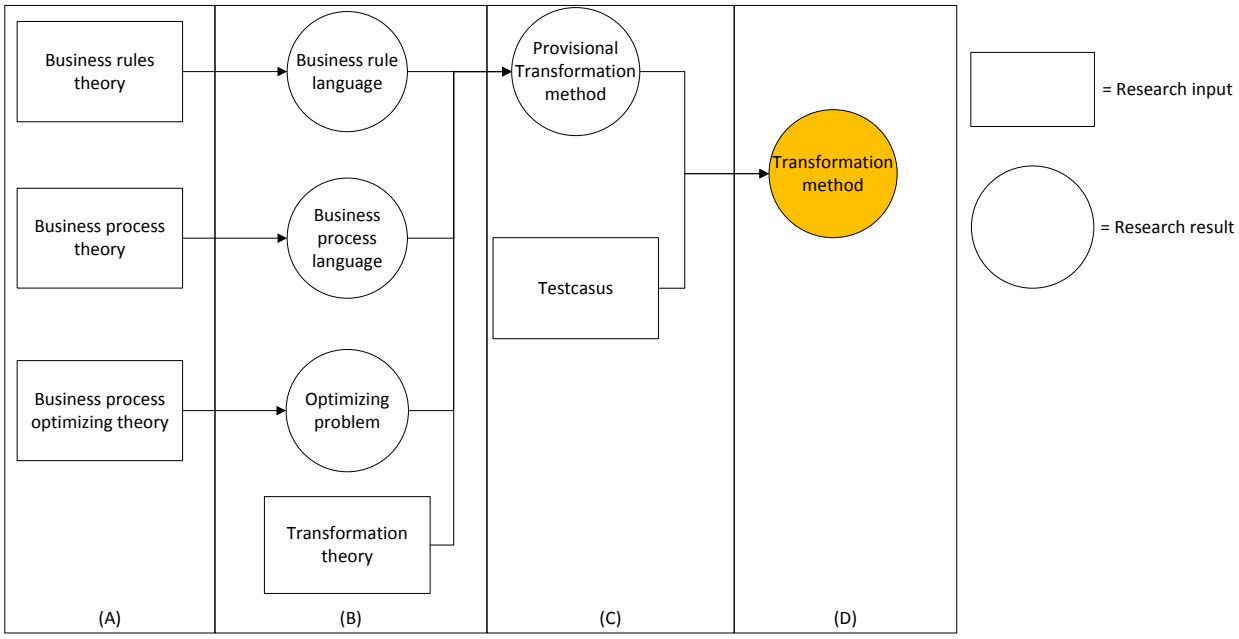

Figure 1: Research model

A. The first part of this research focuses on finding background information on business rules, business processes, and business process optimization theory. This has led to the language used for the specification of business rules and the specification of business processes and to the optimization problem.
B. In this part of the research a transformation method is developed and implemented in a prototype using the result of (A) and transformation theory.
C. The prototype created in (B) has been used together with information of a test case to verify the transformation method.

1.4 Structure of report

The structure of the report is as follows:

- Chapter 2 characterizes business rules, determines which types of business rules are relevant for this research and describes different languages for specifying business rules and justifies the choice for the business rule language that is used in the transformation method.
- Chapter 3 characterizes what business processes are, describes different languages for specifying business processes, justifies the choice for the business process language that is used in the transformation method and identifies optimization criteria for business processes.
- Chapter 4 described different methods and tools that can be used for model transformation, identifies current approaches that exist to transform a business rules specification to a business process specification and defines the approach we used to develop the transformation method.
- Chapter 5 discusses the metamodels used in the developed transformation method
- Chapter 6 discusses our transformation method in detail.
- Chapter 7 presents the developed prototype and verifies and evaluates the developed transformation method.
- Chapter 8 presents the conclusions and future work of the research.

2 Business rules

This chapter presents the basic concepts of business rules, discusses different classifications of business rules and discusses the type of business rules that are important in the context of this research. Furthermore, in this chapter different business rule languages that are relevant in the context of this research are described and a choice is made between them based on their expressiveness and support.

This chapter is further structured as follows: Section 2.1 elaborates on the basic concepts of business rules. Section 2.2 discusses different business rule classifications. Section 2.3 describes different business rule languages and section 2.4 justifies the choice for a business rule language.

2.1 Concepts

There are several definitions of business rules. The definition most commonly used is the definition of the Business Rules Group which is a group of IT-professionals and one of the developers of the SBVR standard (Business Rules Group). They state that a business rule is:

"*a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the* business".

Some other definitions that can be found are:

- Business rules are atomic, formal expressions of business policies, business regulations and common-sense constraints (Goedertier, Heason, & Vanthienen, 2007).
- A business rule is a statement that aims to influence or guide behavior and information in an organization (Muehlen, Indulska, & Kamp, 2008).

All these definitions basically state the same: a business rule is a statement that defines or guides some aspect of the business. The definition of (Goedertier, Heason, & Vanthienen, 2007) adds to that that the statement has to be atomic and formal. Atomic means that the statement cannot be broken down or decomposed further into other business rules, without losing its meaning. Consider the following rule:

```
If a customer has salary > 3000 euro per month then the customer has a
gold status else the customer has a silver status.
```

This is not an atomic rule since it can be decomposed into the following rules:

```
If customer has salary > 3000 euro per month then the customer has a gold
status.

If customer has salary <= 3000 euro per month then the customer has a
silver status.
```

It is quite normal to use if-then-else statements in business rules therefore we do not find it necessary for a business rule to be atomic.

In our research a business rules specification must be transformed automatically, this means that the business rule specification has to be parsed by machine, i.e., the machine must understand what is in the specification in order to do something with it. In order for a specification to be parsable it must be defined in a formal language, therefore, in our research, it is necessary for a business rule to be formal.

For these reasons, a business rule is defined in this research as:

"A formal statement that defines or constrains some aspect of the business"

The basic building block of a business rule is an entity: an *entity* is an object that is relevant for the business. This could be many things, such as a person, a car or a document. An entity could be categorized according to two dimensions: type or instance, and generic or specific (Business Rules Group). Table 1 gives an example of an entity for each dimension.

Table 1: Examples of entities

	Generic	Specific
Type	City	Order
Instance	"Arnhem"	"Order 123"

Using entities two types of business rules statements can be defined: facts and constraints.

A *fact* is a statement that relates entities to each other. There are two types of facts: base facts and derived facts (Business Rules Group). A base fact defines a relationship between entities, such as, "a car has an engine". A derived fact defines how the value of one entity could be inferred from other rules, such as, "The Rental charge is the base rental price + the refueling charge" or "The price of an article is the cost price + profit margin + added taxes"

Constraints are statements that concern some dynamic aspect of the business. A constraint specifies restrictions on the results that actions can produce, such as, "if order value < 500 euro and customer has gold status then approve order".

2.2 Classification of business rules

Business rules can be classified according to a number of dimensions, such as role or the business aspect that they cover. The classification used in this research is based on the business aspect that is covered by the rule and is adapted from (Weiden, Hermans, Schreiber, & Zee, 2002). Three business aspects can be identified:

1. Rules that belong to the *informational aspect* describe the static aspect of the business;
2. Rules that belong to the *behavioral aspect* describe some dynamic aspect of the business. These rules often use rules defined in the informational aspect;
3. Rules that belong to the *guidance aspect* are defined to guide the business process. Rules defined in the behavioral aspect are often based on these rules.

Below we discuss each of these aspects.

2.2.1 Informational aspect

Rules in the informational aspect describe static aspects of the business. This means that these rules describe aspects of the business that do not directly relate to activities. In the informational aspect, two types of rules can be identified:

1. *Concept structure rules* are the rules that describe the entities and facts as described in section 2.1 and constraints on facts that can be asserted. An example of a constraint on the fact, A person is married to a person, that can be asserted is “A person is married to at most one person”;
2. *Persistency type rules* describe how long information should be kept available. An example of such a rule is **“**No order must be kept longer than 5 years”.

In the classification of (Weiden, Hermans, Schreiber, & Zee, 2002) another type of rule is defined that belongs to the informational aspect called history rules. History rules state what should happen with the history of an object, e.g., is every customer a new customer or should he considered to be the same customer. In our research, history rules are considered to be a form of pre or post condition rules as discussed in the behavioral aspect. For example, the history rule “All orders made by some customer should be recorded as belonging to that same customer” can be expressed as a post-condition rule as follows: “After a customer makes an order and the person is an existing customer then the order is recorded to the existing customer”.

2.2.2 Behavioral aspect

Rules in the behavioral aspect are concerned with the execution of activities in the business. There are six types of behavioral rules:

1. *Control flow rules* control the possible execution of activities. An example of a control flow rule is “If the payment for an order is received then the order must be shipped”;
2. *Pre-conditions rules* indicate which conditions must be met or which information needs to be available before an activity can start. An example of a pre-condition rule is “A person can only apply for a drivers license if he is older than 17”;
3. *Post-condition rules* indicate which conditions must be met or which information is available after the activity is performed. An example of a post-condition rule is “After a person has passed his driver’s license test he is considered to be a legal driver”;
4. *Task knowledge rules* specify knowledge that is only used in specific activities. An example of a task knowledge rule in the context of assessing an applicant for a mortgage is “Applicants for a mortgage must be over 18 years old”;
5. *Frequency rules* define how often an activity is performed. The frequency is often related to some time period. An example of a frequency rule is “On the first day of every month each employee delivers his timesheet to his manager”;
6. *Duration rules* define the time period in which an activity has to be done. An example of a duration rule is “Any incoming protest from a participant must be handled within 3 hours by the officials”.

In the classification of (Weiden, Hermans, Schreiber, & Zee, 2002) another type of rule has been identified that belongs to the behavioral aspect, called *information flow rules*. Informational flow rules describe situations in which an activity needs information from other activities to be able to execute. In this research this is considered to be a pre- or post-condition. For example the information flow rule: “The offer must contain the same information about the customer as was on the application” can be

expressed as the following post-condition rule: “After an offer is made the order must contain the same information about the customer as was on the application”.

2.2.3 Guidance aspect

Rules in the guidance aspect are defined to guide the business process. There are five types of guidance rules.

1. *Organization rules* describe the policies and culture of the organization. An example of such a rule is “We shall not distribute private information of our clients to other parties”;
2. *Goal and value rules* describe the goals of the organization. They differ from the organizational rules in that they are aimed at one particular process or task instead of the organization as a whole. An example of such a rule is “We want to make sure that there will be no rentals turned down because there are no cars available”;
3. *Actor competences rules* define what skills an actor has to have in order to perform a certain role in the organization. An example of such a rule is “In order for a person to be a register accountant he has to have certificate x”;
4. *Resources rules* define constraints about the amount and type of resources that are used in the business. An example of such a rule is “If there are less than 10 items left in stock for product x order 10 new items”.
5. *Actor responsibilities rules* define the responsibilities of actors in terms of the activities they can or must perform. An example of an actor responsibility rule is “Assigning special cars must be done by a senior employee”.

2.2.4 Alternative classifications

An alternative classification of business rules is based on the role of the business rule. (Charfi & Mezini, 2008) classify four types of rules: contraint, action enabler, computation and inference rules. (Taveter & Wagner, 2001) use a similar classification, they combine computation rules and inferences rules as derivation rules, which correspond to derived facts as described in section 2.1. They also add a new category of rules called deontic assignments. Therefore we come to the following classification of business rules based on the role of the business rule:

- A *constraint rule* expresses an unconditional circumstance that must be true or false. This type can be further differentiated into state or process constraints. State constraints must be valid at any time. Process constraints specify the valid state transitions. An example of a state constraint is “A person can only apply for a driver’s license if he is older than 17”. An example of a process constraint is “The lab can only go to operating mode if the temperature is below 20 degrees”.
- An *action enabler rule* defines conditions and the action that must be initiated when the conditions are true. An example of such a rule is “If the payment for an order is received then the order must be shipped”.
- A *computation rule* defines conditions and the algorithm that must be used to calculate the value of an attribute if the conditions are true. An example of such a rule is “If more than 2 persons travel together, the third person pays only half price”.

- An *inference rule* defines conditions and the fact that must be true if the conditions are true. An example of such a rule is "If a person occurs on the blacklist then his credibility is negative".
- *Deontic assignments* can be used to explicitly define authorizations. An example of such a rule is "Assigning special cars must be done by a senior employee".

2.2.5 Relevant types of business rules

Several types of business rules have been described in this section. However, not all types of rules are evenly important in the context of this research. The aim of this research is to transform a business rule specification to an optimal business process specification. The more important business rules in the context of this research are thus the business rules that actually define a part of the business process.

Based on this criterion the following rules are less important in the context of this research:

- Organization rules are less important because they do not directly relate to some activity that needs to be performed. For example, the rule "We shall not distribute private information of our clients to other parties" describes a general company guideline that is important. However, it cannot be directly transformed to a business process, one first has to define some mechanisms in other rules to ensure that no information is distributed to third parties.
- Goal and value rules also do not directly relate to some activity. For example, the rule "We want to make sure that there will be no rentals turned down because there are no cars available" describes a goal that is important. However it cannot be directly transformed to a business process, additional rules are needed for that. An example of a rule that can ensure this goal rule is the resource rule "If a branch has less than 10 cars, then it must buy 10 new cars".

2.3 Business rules specification languages

There are several business rules specification languages. However, not all languages are relevant in the context of this research. As stated in the main objective the business rule specification language used in this work has to be human understandable. Business rule languages based on XML, such as RuleML (Hirtle, et al.), are not being considered in this work because they are not intelligible for most business people. Consider for example the following simple rule:

```
If the customer has a gold status then the discount is 20%
```

Listing 1 shows how this rule is expressed In RuleML. One can see that the RuleML specification is less readable than the textual representation.

```
<Implies>
 <head>
 <Atom>
       <Rel>discount</Rel>
       <Var>customer status</Var>
       <Ind>20 percent</Ind>
 </Atom>
```

```
 </head>
 <body>
 <Atom>
       <Rel>gold</Rel>
       <Var>customer status</Var>
 </Atom>
 </body>
</Implies>
```

Listing 1: Example RuleML rule

Three business rule languages that that do fit the requirement of being human understandable are SBVRSE, EM-BrA^{2}Ce, and the PA-notation. SBVRSE is a structured English vocabulary developed by the Object Management Group (OMG) complying with the SBVR standard (Object Management Group, 2008). EM-BrA^{2}Ce is an extension of SBVRSE, and the PA-notation is a proprietary language developed by Logica that can also be considered as a business rules specification language.

The sections below describe the basic constructs of these three languages. At the end of each section the example that is described in Listing 2 is expressed in the language discussed in that section.

> Every day a bank gets several requests for credit. The standard process for handling such a request is as follows:
>
> > Based on the credibility status of the customer and the size of the requested credit the credit request is either directly approved or sent to the credit manager for evaluation. The request is directly approved if:
> >
> > > The creditability status of the customer is medium and the requested amount is lower than 500
> > > The creditability status of the customer is good and the requested amount is lower than 1000
> >
> > In all other situations the credit manager has to determine whether to approve or deny the credit request.

Listing 2: Credit request example

2.3.1 SBVRSE

SBVRSE stands for ***S***emantics of ***B***usiness ***V***ocabularies and Business ***R***ules ***S***tructured ***E***nglish. One of the techniques used in SBVRSE is font styles to interpret the meaning of words in a sentence. The font styles used in SBVRSE are.

- To represent an entity type, called terms in the SBVRSE specification, a green underlined font is used, like in entity type.
- To represent an entity instance a green double underlined font is used, like in instance.

- To represent a verb a blue italic font is used, like in *verb* .
- To represent a keyword a red bold font is used, like in **keyword**.

Entity type and entity instance

In SBVRSE the distinction between an entity instance and an entity type is made by using different fonts. However, SBVRSE does not make a distinction between general and specific entities. SBVRSE does allow one to give a definition of an entity informally, such as:

> Renter
> Definition: driver contractually responsible for a rental.

Verbs

Verbs are used in SBVRSE to relate entities to each other in order to create facts, called fact types. In SBVRSE, several types of facts can be distinguished, as it can be seen in the class diagram of Figure 2.

A characteristic fact has exactly one associative entity, such as, shipment *is late*.

A binary fact has exactly two associative entities, such as, person *is married to* person. A special type is the partitive fact, which indicates that the first entity is in the composition of the second entity, such as, car model *is included in* car group.

An *associative fact* has more than one associative entity, for instance car manufacturer *delivers* car *to* branch. The *is property of* fact indicates that the first entity is an essential quality of the second entity, such as, engine size *is property of* car model.

A *specialization fact* specifies that the first entity is a special type identified by the second entity, such as, customer *is of the category* risky customer. An instance of this fact type would be that a specific customer being a risky customer.

An assortment fact relates entity types and entity instances to each other, such as, Ford Motor Company *is a* car manufacturer.

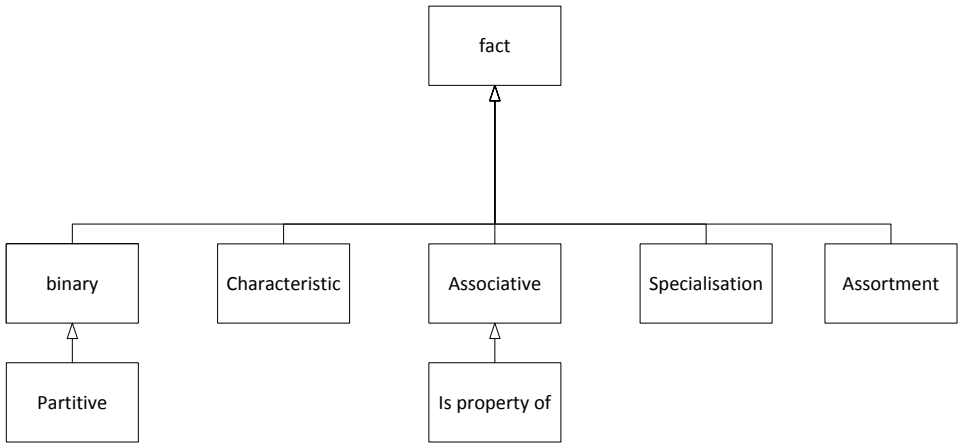

Figure 2: Facts in SBVRSE

Two other type of facts that can be distinguished in SBVRSE that are not classified as a fact type are the facts that are created using the *generalizes* and *specializes* verbs. These fact have exactly two associative entities and can be used to create facts like, airplane *specializes* means of transportation.

In some cases it is not directly clear which type of fact is used, for instance, car *has* speed, could either be an associative fact or a binary fact. To solve this, SBVRSE allows one to specify some metadata about a fact, such as:

```
car has speed
Concept Type: associative fact
```

Keywords

In SBVRSE, Keywords can be used to specify business rules. There are several types of keywords in order to specify two types of rules, namely structural rules and operative rules. Structural rules are true by definition. Operative rules can be violated, therefore, in order to fully specify an operative rule one also has to specify an enforcement level, such as strict or loose. An example of such a rule is:

```
It is prohibited that an intoxicated person is a passenger on a flight
Enforcement level: strict
```

However, this rule does not enforce some behavior of the business process. Structural rules have to be defined to make sure that no intoxicated person is a passenger on a flight. An example of such rules could be:

```
It is necessary that if a person checks in for a flight that the person
gets a blood test and that the result of the blood test is negative
It is necessary that if a person gets a blood test that the result is
negative or the result is positive
```

Operative rules can be seen as a means to express goal and value rules and are therefore not relevant in the context of this research.

Figure 3 shows the type of keywords that can be used to specify structural rules. The comments indicate the actual keywords that are used.

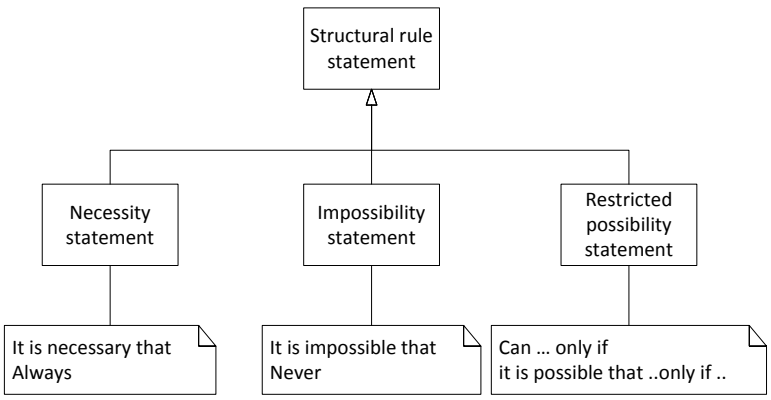

Figure 3: Structural rule types in SBVRSE

Keywords that define some quantification, logical keywords and keywords used to connect facts to each other can also be used. Table 2 gives an overview of those keywords.

Table 2: Keywords in SBVRSE

Catagory	Keyword	type
Quantification		
	each	universal quantification
	some	at least N quantification
	at least one	at least N quantification
	at least N	at least N quantification
	at most one	at most N quantification
	at most N	at most N quantification
	Exactly one	Exactly N quantification
	Exacly N	Exactly N quantification
	at least N and at most M	numeric range quantification
	more than one	at least N quantification
Logical operations		
	it is not the case that P	logical negation
	P and Q	conjuction
	P Or Q	disjuction
	P or Q but not both	exlusive disjuction
	if P then Q	implication
	Q if P	implication
	P if and only if Q	equivalance
	not both P and Q	nand formulation
	neither P nor Q	nor formulation
	P whether or not Q	whether or not formulation (P is TRUE regardless of Q)
	not is	logical negation
	does not 'other verbs'	logical negation
Other keywords		
	the	
	a, an	
	another	
	a given	
	that	
	who	
	x **is of** y	
	what	

Example

Listing 3 shows how the credit request example of Listing 2 is expressed in SBVRSE. First the entities and the facts are specified, followed by the constraints that must hold in this example.

```
requester

credit_manager

credit_request

amount

status


credit_request has amount

Synonym: amount of credit_request

requester requests credit_request

requester has status
```

```
Synonym: status of requester

credit_request is approved

credit_request is denied

credit_manager assesses credit_request of requester


It is necessary that a credit_request has an amount

It is necessary that status is low or medium or good

it is necessary that if requester requests credit_request and the status
of a requester is medium and the amount of credit_request is at most 500
then credit_request is approved

it is necessary that if requester requests credit_request and the status
of a requester is good and the amount of credit_request is at most 1000
then credit_request is approved

it is necessary that if requester requests credit_request and the status
of a requester is not good or medium then credit_manager assesses
credit_request of requester

it is necessary that if requester requests credit_request and the status
of a requester is medium and the amount of credit_request is least 501
then credit_manager assesses credit_request of requester

it is necessary that if requester requests credit_request and the status
of a requester is good and the amount of credit_request is least 1001
then credit_manager assesses credit_request of the requester

it is necessary that if the credit_manager assesses the credit_request of
the requester then credit_request is approved or credit_request is denied
```

Listing 3: Credit request example expressed in SBVRSE

2.3.2 EM-BrA²Ce

(Goedertier, Heason, & Vanthienen, 2007) have specified an extension to the vocabulary defined in SBVRSE to make it more suitable to describe processes in a declarative manner. This extension is called EM-BrA^{2}Ce which stands for "Enterprise Modeling using business Rules, Agents, Activities, Concepts and Events'. The authors found SBVRSE unsuitable because it does not have standard constructs to directly express process-related concepts such as agent, activity, events and deontic assignments.

In EM-BrA^{2}Ce some entities and facts related to business processes are predefined. The most important added entities are:

- An `activity type` represents an unit of work to be performed by an agent, such as, `place order activity`.
- An `event type` corresponds to an instantaneous discrete state change of a concept in the world, such as, `violation event`. There are two specific events:
    1. Activity events, which represent a change of a state in the activity. These states have to be specified in the business rule specification. In the specification of the EM-BrA^{2}Ce vocabulary (Goedertier, Heason, & Vanthienen, 2007) give the following examples of activity states: `created`, `scheduled`, `assigned`, `revoked`, `started`, `factAdded`, `factRemoved`, `factUpdated`, `aborted`, `skipped`, `completed` and `redone`;

2. Business fact events, which represent the state change of a fact, such as created or changed.

- An agent represents a specific actor or a group of actors who can perform activities. The agent type only reflects entities of the instance type, such as, WorkerX.
- A role is a concept that represents authorizations which regard to the execution of activities, for instance the role seller that can perform the activity sell product: seller *can perform* sell product activity.

The type of an entity can be indicated by stating the type of the entity directly after the entity, for instance place order activity or by adding metadata when defining an entity, such as,

```
place order
Concept Type: activity
```

The EM-BrA^{2}Ce vocabulary also predefines a set of facts related to these predefined entities. For instance, role *can perform* activity type.

Using those extra entities and facts one can create rules like:

```
If accept order activity is started, it is necessary that a place order
activity has been completed and that no accept order activity or reject
order activity has been started.
```

Example in EM-BrA^{2}Ce

Listing 4 shows how the credit request example of Listing 2 is expressed in EM-BrA^{2}Ce. First the entities and the facts are specified, followed by the constraints that must hold in the example

```
Requester
Concept type: role
Request credit activity
credit_manager
Concept type: role
credit_request
judge credit request activity
amount
status
credit_request has requester
Synonym: requester of credit_request
credit_request has amount
Synonym: amount of credit_request
requester requests credit_request
requester has status
Synonym: status of requester
```

```
credit_request is approved

credit_request is denied


It is necessary that status is low or medium or good

Requester can perform Request credit activity

credit_manager can perform judge credit request activity

it is necessary that a credit_request exist if a request credit activity
is completed

It is necessary that a credit_request has an amount

It is necessary that a credit_request has a requester

If a request credit activity is completed then it is necessary that
credit_request is approved if the status of a requester is medium and the
amount of credit_request is at most 500

If a request credit activity is completed then it is necessary
credit_request is approved if the status of a requester is good and the
amount of credit_request is at most 1000

If a request credit activity is completed then it is necessary that a
judge credit request activity is started if the status of a requester is
not good or medium

If a request credit activity is completed then it is necessary that a
judge credit request activity is started if the status of a requester is
medium and the amount of credit_request is least 501

If a request credit activity is completed then it is necessary that a
judge credit request activity is started if the status of a requester is
good and the amount of credit_request is least 1001

To complete a judge credit request activity it is necessary that
credit_request is approved or credit_request is denied
```

Listing 4: Credit request example in EM-BrA^{2}Ce

2.3.3 PA-notation

The PA-notation is a language developed by Logica for specifying business processes. Unlike SBVRSE and EM-BrA^{2}Ce, the PA-notation is not completely based on natural language.

The PA-notation consists of the following syntax elements:

- Specification start
- Sentences and their definitions
- Entities and attributes
- Input
- Operators

Specification start

The statements “The following applies” and “For each A in B applies” often determine/define the beginning of a PA-specification. In the second statement a context is added to the business process. For example, if a specification begins with “For each E in EMPLOYEES applies” it means that the specified

process only applies to each employee in the collection EMPLOYEES. The expression that follows after these statements has to be evaluated to true in order to reach some end result. An example of a PA-specification with a start is given below:

```
The following applies:
       If "Credit card details have been entered"
       Then "Credit card date is validated"
```

This means that if credit card details have been entered then the credit card details must also be validated in order to reach the end result.

Sentences

Every statement between " " is a sentence. Sentences are used to represent expressions that need to be evaluated or performed in a textual form. The expression that must be evaluated is defined in the sentence definition. An example of a sentence and its definition is given below.

```
……
Then "Credit is approved"                    <-Sentence usages
"Credit is approved" =                       <-Sentence definition
       CR.requeststatus = 'approved'
```

The return type of a sentence is quite important. In the example above the return type is a Boolean indicating if the action has been performed. However, the return type can also be a date, number, a collection or an element in a collection depending on the definition of the sentence.

Definition of collections and attributes

Every word written in capitals can be (a reference to) a (element in a) collection. This collection can be classified as an entity. One can also specify some properties of the collection, which are called attributes. An example of a collection is PERSONS, which can have the attribute firstname. A collection having attributes can be classified as a fact.

In order to use collections and their attributes they have to be explicitly defined, such as,

<COLLECTIONNAME> description '<description>' =

<attributename>:<attribute type> <attribute property>

The type of an attribute can be text, number, date, boolean, a (element of) another collection or calculated attribute. The property of the attribute can be identifies, required or can specify a constraint, if the attribute type is another collection. A calculated attribute is specified as follows:

<attributename> = <calculation>

An example of a collection specification is given below:

```
PERSONS description 'Natural persons' =
       Bsnr: number identifies required
```

```
Firstname: text required
Lastname: text required
Birthdate: date required
MARRIEDTO = PERSONS (1)
Age = NOW - Birthdate
```

The convention is that if the attribute type is a collection then the attribute name is written in capitals.

Creation of an element in a collection and attributes

In the PA-notation, it is also possible to specify that at some point in time an element in a collection needs to be created, changed or deleted. This is done using the keywords one/multiple exist and there existed. For example:

```
One P exists in PERSONS with:
Bsnr                = 123456789
Firstname           = 'Bas'
Lastname            = 'Steen'
Birthdate           = 24-12-1984
```

Usage of collections and attributes

Once a collection and its attributes are defined, the collection can be used in expressions and sentence definitions. One can refer to a collection by using the collection name. One can refer to an attribute of a collection by using the collection name followed by a dot and the attribute name, such as, P.age .

When referring to collection one can also use filters. For example PERSONS (age = 18) will result in a collection of all the persons with the age 18.

Predefined collections

The PA-notation also consists of several predefined collections. The most important is the collection TIMES, which can be used to refer to specific dates or time moments. A special element in that collection is the element NOW which is used to represent the current date and time.

Input

To indicate that at some point in time input is needed from the environment the following keyword is used, "input from <ROLENAME>". The rolename indicates which entity type needs to deliver the input. Optionally one can also use the following keywords when defining an input.

- "chosen from <COLLECTION>" can be used to limit the input that the entity can give to a certain collection. This can be a previously defined collection such as PERSONS or a collection indicated by using the keywords "[<value1 , value2, etc]". For example,

```
input from CREDITMANAGER chosen from ['approved', 'denied']
```

- "Based on <EXPRESSION>" can be used to indicate that the entity needs some information in order to give the correct input. For example,

```
input from CREDITMANAGER based on CR (CR is a specific credit request)
```

- “labeled <label>“ can be used to indicate the label that must be given to the input. The label keyword is added with the thought in mind that a PA-specification at some point is transferred to an application and can be seen as a requirement for the user interface. For example the following input definition:

```
Firstname = input from USER labeled ‘First name’
```

Must be transformed to the following input field:

Operators

PA operators are used in sentence definitions and expressions. Table 3 gives the operators that can be used in the PA-notation.

Table 3: Operators in the PA-notation

Logical operators	Math
If	A + B
then	A - C
else	A * B
and	A / B
or	A % B
not	A = B
	A < B
	A <= B
	A >= B
	A <> B
	biggest A
	smallest A
	A in B
	A exists

Example in the PA-notation

Listing 5 shows how the credit request example of Listing 2 is expressed in the PA-notation. This specification first defines the global rules, followed by the definitions of the sentences used and specification of the collections and their attributes.

```
The following applies:
“Credit CR is requested”
and
 If “Credit CR is requested” then
       If “Status requester” == ’medium’ and “Request amount” < 500 or
       “Status requester” == ’good’ and “Request amount” < 1000
       Then “Credit is approved”
       Else
       “Credit request is judged”
```

```
"Status requester" =
       CR.REQUESTER.status

"Request amount" =
       CR.amount

"Credit CR is requested" =
       One CR exists in CREDITREQUESTS with:
              amount = input from REQUESTER
              REQUESTER = input from REQUESTER chosen from REQUESTERS

"Credit is approved" =
       CR.requeststatus = 'approved'

"Credit request is judged" =
       CR.requeststatus = input from CREDITMANAGER chosen from
                     ['approved', 'denied']

CREDITREQUESTS =
       REQUESTER: REQUESTERS(1) required
       amount: number required
       requeststatus:text

REQUESTERS =
       Status: text
```

Listing 5: Credit request example in the PA-Notation

2.4 Business rule language choice

This section makes a choice for a business rule language. In order to make a justified decision the three languages described in section 2.3 are evaluated according to the following criteria:

- Expressiveness: the ability of the language to express the concepts needed to describe a business process declaratively.
- Support: the language support by vendors or communities, and the tools available to support the specification of business rules for that language.

In this research the different business rule languages are compared on their expressiveness by using an ontology analysis, as originally described by (Weber, 1997) and extended by (Sinderen, Ferreira Pires, & Guizzardi, 2005).

According to this technique, languages are compared to a reference ontology. Any deviation from an one-to-one mapping between a concept of the reference ontology and a construct of the language is called a deficit. There are four types of deficits, as can be seen in Figure 4. This technique is extended by (Sinderen, Ferreira Pires, & Guizzardi, 2005) by also determining the properties that the resulting specifications composed using the language should have.

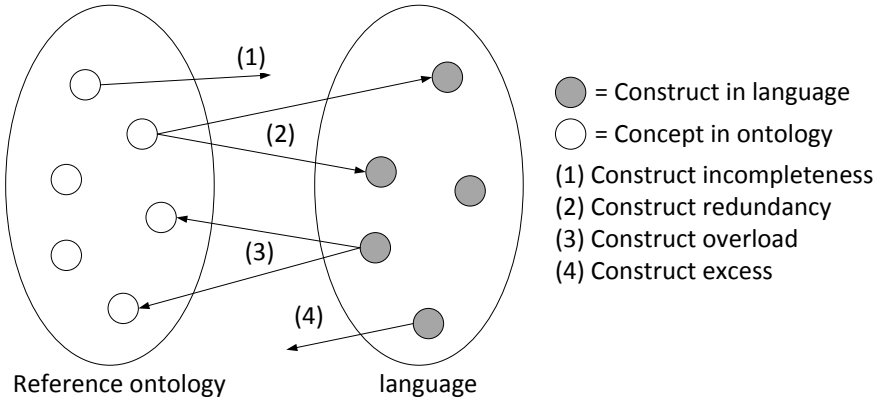

Figure 4: Ontology analysis deficits

The deficits are:

1. *Construct incompleteness,* in which an ontology concept does not have a corresponding construct in the language;
2. *Construct redundancy*, in which there is more than one alternative language construct for one ontological concept;
3. *Construct overload* in which, there is more than one ontological concept for one language construct;
4. *Construct excess,* in which there is a construct in the language that is have no counterpart in the ontology.

In this research the ontology analysis mainly focuses on the construct incompleteness deficit. This deficit is important because it determines the expressiveness of a language, here less deficits means a better expressiveness (Sinderen, Ferreira Pires, & Guizzardi, 2005). The construct overload is also important because, if the language has construct overload, then the language is ambiguous (Sinderen, Ferreira Pires, & Guizzardi, 2005). Ambiguousness in a language decreases its suitability to automated transformation because the automated process then may not know how to interpret the specification. It then has to choose one of the possible interpretations, which might be the wrong one. Occurrences of the other two deficit types mean that the language is overcomplete, which decreases the clarity of the specification, but has no effect on its ability to transform it to a business process modeling language. Therefore those types of deficits are out of the scope of our ontological analysis.

The first step in performing an ontology analysis is to determine the reference ontology. The ontology most widely used in the comparison of modeling languages is the Bunge Wand Weber (BWW) ontology (Weber, 1997). This ontology consists of a set of basic real world concepts that information system models should be able to express. These concepts are however very generic. Consequently it is possible

that a business rule language supports most concepts of the BWW model, but does not support important concepts of a business process. This is, for instance, the case for the concept of an event. In the BWW ontology there is only one concept for event, while in business processes there are multiple event types that are important and should be represented in business rules, such as time events or message events (Söderström, Andersson, Johannesson, Perjons, & Wangler, 2002). Therefore, we decided to define the most important concepts that must be supported in a business rule language and base our analysis based on these concepts. In the next section the identified concepts are discussed followed by the analysis of SBVRSE, EM-BrA^{2}Ce and the PA-notation based on these concepts.

2.4.1 Ontology of business rules

The concepts that we identified consists of two categories, similar to the classification of business rules as described in 2.2. The first category describes the concepts in the informational aspect of a business process. The second category describes the concepts of the behavioral and guidance aspects of a business process.

Informational aspect

In the informational aspect two rule types can be identified: Persistency type rules and concept structure rules.

Persistency type rules describe how long information should be kept available and thus can be seen as a time duration event as discussed in the next section. An example is a rule that states that "if the difference between current date and the creation date of the information is 5 years trigger the delete information activity"

Concept structure rules are the rules that describe the entities and facts as described in section 2.1 and constraints on facts that can be asserted. This is similar to specifying a domain model in an IT project. Therefore the concepts of the informational aspect are similar to the concepts used in, for instance, Entity Relationship Diagrams or class diagrams.

The most important concept is the concept of Entities. Entities can be either types or instance of types, where each instance is at least one type. Entities can have attributes and relationships with other Entities. A special type of relationship is the relationship where one entity is a special type of another Entity. Furthermore, concept structure rules define the constraints on facts that can be asserted, i.e., it must be possible to define constraints on attributes, entities and relationships.

Table 4 summarizes the concepts identified in the informational aspect.

Table 4: Concepts related to informational aspect

Informational elements
Entity
type
instance
attribute
value
relationship
specializes
Constraints

Behavioral and guidance aspect

In the behavioral and guidance aspect several rule types can be identified, below the concepts that are important for each rule type are identified.

Control flow rules can be described as relationships between events and activities. (Grossmann, Schrefl, & Stumptner, 2008) identified 6 different type of events:

1. Activity is started;
2. Activity is running;
3. Activity is completed;
4. Attribute enters value;
5. Attribute has value;
6. Attribute leaves value.

The first three events are related to the state that an activity can have: it can either be started, it can be running or it can be completed. If an activity enters a certain state then this is an event. The other events are related to changes in attributes that entities can have. These attributes are often related to a certain state of an entity, such as, the state student for a person who is studying at some university. Another activity state event that can be identified is, activity is cancelled.

Next to events related to the state of an activity and the state of some attribute (Söderström, Andersson, Johannesson, Perjons, & Wangler, 2002) have identified two types of time events:

1. *Time point events* occur when it becomes a certain point in time;
2. *Time duration events* occur when a predefined difference between two time points becomes true.

Another event that can be identified is the case that some message from the environment arrives at the organization.

(Grossmann, Schrefl, & Stumptner, 2008) also identified four different actions on activities initiated by events. An event can either trigger or cancel an activity which means that some work must be done, or undone. An event can also enable or disable some activity which means that some work can be done or cannot be done anymore, for example students can or cannot enroll for courses.

pre and post condition rules can be described as some condition that must hold, respectively, before and after an activity. This condition is defined as something in the informational aspect. For example, the condition “entity instance y must exist”.

Task knowledge rules can be described as something in the informational aspect that is only valid the context of a specific activity.

Frequency rules can be expressed as control flow rules where the event is a time point event, time duration event or an attribute value event. For example in the rule “on the first day of every month each

employee delivers his timesheet to his manager" the time point event is "when it is the first day of the month".

Duration rules can be expressed as duration time events, e.g., "3 hours after activity x is started, if activity is not completed then do something"

Actor responsibilities are rules that determine which actor must perform an activity. As with entities there is also a distinction between type and instance with actors. The type, or role in the context of actor assignment, determines the organizational role that must perform the activity. The instance, or resource in the context of actor assignment, is the specific resource that performs the activity.

Resource rules can be expressed as attribute value events. For example the rule "if there are less than 10 items left in stock for product x order 10 new items" can be expressed as the attribute value event "number of stock items enters value 9" and based on this event an order new products activity can be triggered.

Table 5 summarizes the concepts and relationships between concepts identified in the behavioral and guidance aspect.

Table 5: Concepts and relations in Behavioral and guidance aspect

Single aspects	Relationships
Activity	On event perform action on activity
Associated informational aspect	Activity Y is performed by role Z
Event	resource performs role in context of activity
change event	Activity has precondition
activity starts	Activity has postcondition
during activity	
activity finished	
activity cancelled	
attribute enters value	
attribute has value	
attribute leaves value	
time event	
point in time	
duration	
message event	
Action	
trigger	
cancel	
enable	
disable	
Role	
Resource	

2.4.2 Analysis of SBVRSE

Table 6 shows the result of the ontological analysis of SBVRSE. The results are explained in the following segments.

Information aspect

Most of the informational aspect concepts are available in SBVRSE. The entity type is defined in SBVRSE by a `term type`. An entity instance is defined in SBVRSE by the `individual type`. Relationships between entities are described in SBVRSE using fact types. The concept of attributes is not available in SBVRSE because it does not make a difference between an entity having a relationship with another entity or an entity being an attribute of another attribute. For example, `Person` *`has`* `dog`, could mean that dog is an attribute of a person or that a dog and a person have a relationship with each other. SBVRSE does support specifying constraints. The specializes concept is supported in SBVRSE using the specialization fact, e.g., `airplane` *`specializes`* `means of transportation`.

Behavioral and guidance aspect

In this aspect all concepts are related to the activity concept. There is however no construct in SBVRSE that can be used to express the activity concept. (Raj, Prabhakar, & Hendryx, 2008) have solved this by assuming that every fact type that is written using verbs in a transitive form, e.g. *`inserts`*, *`requests`*, *`enters`*, represents an activity. Given the context of our research, this is not considered to be an adequate solution because one would need to add a mechanism to recognize that a verb is in a transitive form. Therefore we consider that none of the concept in the behavioral and guidance aspect are supported by SBVRSE.

Support

There are some tools that support SBVRSE. One of the main reasons for this is that the language and the standard it complies with (SBVR) are developed in cooperation with several industry players and is adopted by the OMG. Two tools use SBVRSE as the language for specifying some part of the business rules:

- *Knowngravity*, which is a commercial tool. In this tool, SBVRSE is used for specifying rules that belong to the informational aspect, thus the entities, facts and some constraints.
- *SBeaVeR*, which is an open source tool that aids in specifying business rules with SBVRSE. However, work on this tool seems to have stopped since no updates have been made since 2006.

In the future several other tools and further standardization of SBVRSE can be expected. The Eclipse community is working on a SBVR component that may include tools to support business rules specification in SBVRSE (The Model Development Tools project). A standard vocabulary for specifying dates and times is also in the process of being standardized (Linehan, M.H., 2008).

2.4.3 Analysis of EM-BrA²Ce

Table 6 shows the result of the ontological analysis of EM-BrA^{2}Ce. EM-BrA^{2}Ce does not add any language construct to SBVRSE that relates to the informational aspect, the support of EM-BrA^{2}Ce for the informational aspect concepts is thus identical to the support of SBVRSE.

The behavioral and guidance aspect consists of a number of relationships, and concepts needed to express these relationships. The analysis of the behavioral and guidance aspect is performed according to these relationships.

Event, Action, Activity

The relationship between the concepts of Activity, Action and Event can be expressed in EM-BrA^{2}Ce by using the following rule:

```
It is necessary that if event then activity is triggered
```

As opposed to the overall relationship concept not all individual concepts of activity, event and action can be expressed using EM-BrA^{2}Ce.

The activity concept is supported. This concept can be expressed by the activity type construct, e.g. purchase order activity. The concept of relating certain information aspect to certain activities is not supported in EM-BrA^{2}Ce.

EM-BrA^{2}Ce has an event construct. It is also possible to define activity events and business fact events which, in theory, could cover the change event concepts of our ontology. In the specification of the EM-BrA^{2}Ce vocabulary (Goedertier, Heason, & Vanthienen, 2007) give several examples of activity state events but it is unclear whether they are all actually present in the language. It is assumed that the start and completed activity event types are available in the EM-BrA^{2}Ce vocabulary because these two are also used to specify other rule types, such as the pre and post condition rules. This means that the change event concepts activity starts and activity completed are covered by EM-BrA^{2}Ce. There are no constructs available in EM-BrA^{2}Ce to express the time and message events concepts.

Of the action concepts only the trigger concept is covered in EM-BrA^{2}Ce. This concept is covered by the start construct.

Activity, Role, Resource, Attribute

The relationship between the concepts of Activity and Role can be expressed in EM-BrA^{2}Ce using "Role *can perform* activity", where Role is the EM-BrA^{2}Ce construct that is used to express the role concept.

The relationship between the concepts of Activity, Role and Resource can be expressed in EM-BrA^{2}Ce using "Agent *has* role *in the context of* activity", where the agent construct is used to express the resource concept.

Activity, Conditions

The activity and condition relation concepts are expressed in EM-BrA^{2}Ce using pre and post condition rules. These rules specify which condition must hold before an activity can start, its precondition, and which conditions must hold after an activity is considered to be completed, its post condition. The pre and post condition rules are expressed in EM-BrA^{2}Ce as follows:

```
To start an activity it is necessary that….. condition that must hold
before the activity is started

To complete an activity is it necessary that……. condition that must hold
if the activity is considered to be completed.
```

Support

EM-BrA²CE is only supported by a single party, the Katholieke Universiteit Leuven which defined the language. There are no tools available that support EM-BrA²Ce.

2.4.4 Analysis of the PA-notation

Table 6 shows the result of the ontological analysis of PA-notation. The results are explained in the remainder of this section.

Informational aspect

Most of the informational aspect concepts can be expressed in the PA-notation. The entity type, his attributes and the relationship it has with other entities can be expressed in the PA-notation in the collection specification. For example, in the example given below the entity type person is specified that has several attributes and a relationship with another entity in the same collection

```
PERSONS description 'Natural persons' =
       Bsnr: number identifies required
       Firstname: text required
       FATHER: PERSONS(1)
       MOTHER: PERSONS(1)
       Lastname: text required
```

The entity instance and the actual value of attributes can be specified by creating a collection instance. The constraint concept is only partly supported. In the PA-notation it is only possible to specify the maximum amount of elements a collection attribute can have. In the example given above the collection attribute FATHER can consist of at most one other person, i.e., a person can have at most one person as its father. The concept of the informational aspect that cannot be expressed in the PA-notation is the specialized relationship construct.

Behavioral and guidance aspect

In the PA-notation it is not explicitly mentioned what is considered to be an activity or not. However, three constructs can be considered as an activity, namely the 'input from', 'there exist/existed' and assign construct. The 'input from' concept means something needs to deliver some input. This can be seen as an activity. With the 'there exist/existed' one specifies that something in the environment needs to be created/updated or deleted, this can also be seen as an activity. The assign construct means that some attribute in a collection needs to be updated, which also can be seen as an activity.

The event action activity rule concept cannot be fully expressed in the PA notation. In the PA-notation it is only possible to relate events and activities to each other using if statement, such as, If "event" then "activity". The implicit action used here is the trigger action, i.e., if event happens then trigger activity.

Of the event concepts the PA-notation contains constructs to express the activity finished, attribute enters or leaves value and the time events concepts.

The event finished event is an implicit event the PA-notation, if an input is delivered or the collection element in the environment is created/deleted/updated then the activity is finished.

The attribute enters value event can be checked in an if statement in the PA-notation. For example, if person.status == “something” then “do something”. The attribute leaves value can be expressed in the PA-notation by referring to the old value of an attribute. If person.status != old person.status and old person.status == “something” then “do something”.

The point in time and duration events can be expressed in the PA-notation using the predefined collection TIMES. If there is referred to a certain point in time in an if statement then it is assumed that this is a point in time event, for example, if TIMES (datum = “somedate”) then “do something”. The time duration construct can be expressed in the PA-notation by doing math with dates, for example if (date1 – date2 >3 days) then “do something”.

Most of the concept relating activities to roles and resources cannot be expressed in the PA-notation. In the PA-notation it is only possible to couple input activities to roles by using the expression: input from ROLENAME.

The precondition concept can be implicitly expressed in the PA-notation in two ways. For the input activity the precondition can be indicated using the ‘input from ROLE based on information’ expression. The information part specifies which information is needed to deliver the correct input and is the precondition. For the create/update/delete activities the precondition is specified by using referrals to other create/update activities. For example in the example below the precondition for the activity “Employee is better and this is registered” is that there is a sick notice (SN).

```
“There is a sick notice registered” =
There exist a SN in SICKNOTICES with:
             date                              = NOW
             startdate, reason, workability%   = Input from van SECRETARY


“Employee is better and this is registered” =
      SN.enddate = input from SECRETARY
```

The post condition of an activity is expressed in the PA-notation by the activity itself, for an input activity the post condition is the input itself, for a create/update/delete activity the post condition is a created/updated or deleted entity.

Support

The PA-Notation is only supported by Logica and some customers of Logica. There are no tools available to aid in the specification of rules in the PA-notation.

Table 6: Ontology analysis SBVRSE, EM-BrA^{2}Ce and the PA-Notation

Informational aspect	SBVRSE	EM-BrA^{2}Ce	PA-Notation
Entity			
type	+	+	+
instance	+	+	+
attribute	-	-	+
value	-	-	+
relationship	+	+	+
specializes	+	+	-
Constraints	+	+	+/-
construct supported	**5**	**5**	**5,5**

Behavioral aspect	SBVRSE	EM-BrA^{2}Ce	PA-Notation
Activity	-	+	+
Associated informational aspect	-	-	-
Event			
change event			
activity starts	-	+	-
during activity	-	-	-
activity finished	-	+	+
activity cancelled	-	-	-
attribute enters value	-	+	+
attribute has value	-	-	-
attribute leaves value	-	-	+
time event			
point in time	-	-	+
duration	-	-	+
message event	-	-	-
Action			
trigger	-	+	+
cancel	-	-	-
enable	-	+	-
disable	-	-	-
Role	-	+	+
Resource	-	+	-
Relationships			
On event perform action on activity	-	+	+/-
Activity Y is performed by role Z	-	+	+/-
resource performs role in context of activity	-	+	-
Activity has precondition	-	+	+
Activity has postcondition	-	+	+
Constructs supported	**0**	**13**	**11**

2.4.5 Conclusion

In this chapter SBVRSE, EM-BrA²CE and the PA-notation have been evaluated based on their expressiveness and their support. The expressiveness of the languages has been evaluated based on an ontology analysis and the support has been analyzed based on the number of parties that support the language and the number of tools that are available.

From our ontology analysis we conclude that SBVRSE is not suitable for describing business processes in a declarative way and thus, although it has the most support, cannot be used in the context of this research.

The other two business rule languages, EM-BrA²CE and the PA-notation are suitable for describing business rules, with EM-BrA²CE supporting eighteen and the PA-notation supporting sixteen and a half of the thirty concepts identified. The biggest flaw of the PA-notation in terms of expressiveness is that it does not support other actions on events then triggers and that it does not have constructs to indicate the relationship between activities and resources. The biggest flaw of EM-BrA²CE in terms of expressiveness is that it does not allow one to specify time events.

EM-BrA²CE and the PA-notation have the same amount of support, for both languages holds that they are supported by a single party and there are not any tools available.

Based on expressiveness and support EM-BrA²CE and the PA-notation are comparable. Since the PA-notation is developed by the company where our research is performed we decided to use the PA-notation as the business rule language.

3 Business processes

This chapter presents the concepts of business process, describes different business process languages that are relevant in the context of this research, and makes a choice between them based on their expressiveness and support. Furthermore, this chapter determines the optimization problem used in this research by discussing optimization criteria, the metrics related to these criteria and methods to determine an optimal business process.

This chapter is further structured as follows: Section 3.1 elaborates on the concept of business processes. Section 3.2 describes two business process modeling languages and section 3.3 justifies the choice for a business process modeling language. Section 3.4 discusses business process optimization.

3.1 Concepts

In literature the two definitions of business process that are most quoted are from (Hammer & Champy, 1993) and (Davenport, 1993).

(Hammer & Champy, 1993) state that: *"a business process is a collection of activities that takes one or more kinds of inputs and creates an output that is of value to the customer".*

(Davenport, 1993) uses a similar definition but adds that the activities in a business process are linked to each other, forming a chain of activities. The definition used by Davenport is: *"a business process is defined as the chain of activities whose final aim is the production of a specific output for a particular customer or market".*

We have added to these definitions that someone or something, denoted as an entity, actually has to perform the activity. This leads to the following definition:

"A business process is a chain of activities, performed by entities that takes one or more kinds of inputs and creates an output that is of value for one or more entities"

The basic concepts used in business processes can be derived from the definition:

- An *activity* is some work that needs to be done. An activity can be triggered by an event, which is something that happens, such as, it is the first day of the month 1pm.
- An *entity* is something or someone, such as a machine, person, customer or organization.
- *Inputs and outputs* are artifacts that are needed or created by activities. Some examples of artifacts are a document, the outcome of a decision or information from an IT-system.

3.2 Business process modeling languages

Many business process modeling languages are used for documentation. Some of the most noticeable are BPMN (Object Management Group, 2008), YAWL (van der Aalst & Hofstede, 2005), EPC's (Keller & Nüttgens, 1992) and Amber (the modeling notation used in BIZZdesigner) (Eertink, Janssen, Luttighuis, Teeuw, & Vissers, 1999). In this research, BPMN and YAWL are considered. AMBER and EPC's are not considered because they are mainly used in commercial tools.

The sections below describe BPMN and YAWL. At the end of each section a business process is defined that complies with the rules defined in Listing 2, expressed in the language of that section.

3.2.1 BPMN

The Business Process Modeling notation (BPMN) (Object Management Group, 2008) is developed by the Business Process Management Initiative. In 2006 the first version of the BPMN standard was adopted by the OMG. This first version has been further developed by the BPMI resulting in the adoption of BPMN 1.1 by the OMG (Object Management Group, 2008).

BPMN defines a notation that can be used to create a Business Process Diagram (BPD), which is a graphical representation of a business process. BPMN defines four basic types of elements to create a BPD diagram:

1. Flow objects
2. Connection objects
3. Data objects
4. Swimlanes
5. Other objects

Flow objects

Flow Objects are the elements that define the behavior of a Business Process. There are three types of Flow Objects: Events, Activities and Gateways.

An *event* represents something that happens. In BPMN there are three types of events: a start event, indicating the start of a (sub) process, the end event, indicating the end of a (sub) process and an intermediate event, indicating that something has happened that does not directly start or end a process. Figure 5 shows the symbols for the start, end and intermediate event types.

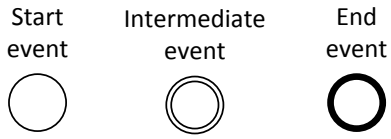

Figure 5: BPMN 1.1 event types

An *activity* represents some work that is performed within a business process. Figure 6 shows the symbol used in BPMN for an activity. This symbol can represent a group of other activities, called sub-process or a single activity.

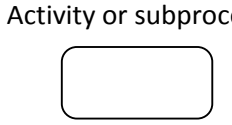

Figure 6: BPMN 1.1 symbol for activity or subprocess

A gateway is used to represent a decision point in the business process from which the process flow can continue down to one or more paths. There are four types of gateways:

1. An *exclusive gateway* represents a point of decision where, at most one of the possible outgoing paths is enabled. This decision can either be *data-based* or *event-based*. If the gateway is used for joining paths it means that all the possible incoming paths enable the outgoing path.
2. An *inclusive gateway* represents a point of decision where, several possible outgoing paths can be enabled or several possible incoming paths can enables the outgoing path. The decision to enable an alternative path is data-based. If the gateway is used for joining incoming paths it can also represent a situation where all incoming paths that have been enabled previously in the process must arrive at the gateway before the process can continue;
3. A c*omplex gateway* is used to handle situations that are not easily handled through the other types of gateways, such as, choosing a path using a combination of data and events;
4. An *and gateway* represents a point of decision where all alternative outgoing paths are enabled by default. If the gateway is used for joining paths it represents a situation where all incoming paths must arrive at the gateway before the process can continue.

Figure 7 shows the symbols used for each type of gateway.

Figure 7: BPMN 1.1 gateway symbols

Connecting objects

Flow Objects can be connected to each other or other information in three ways: Sequence Flow, Message Flow and Association. A sequence flow is used to show that two flow objects are related to each other. A Message Flow is used to show the flow of messages between two flow objects. It is not allowed to use a message flow to connect a gateway to other objects. An Association is used to associate information and/or Artifacts with Flow Objects. Figure 8 shows the symbols used for each type of flow.

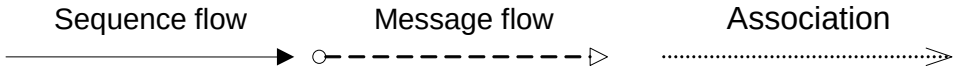

Figure 8: BPMN 1.1 Symbols for connecting objects

The usage of sequence flows arrows in comparison with using gateways is discussed below. For example consider the BPDs of Figure 9 and Figure 10:

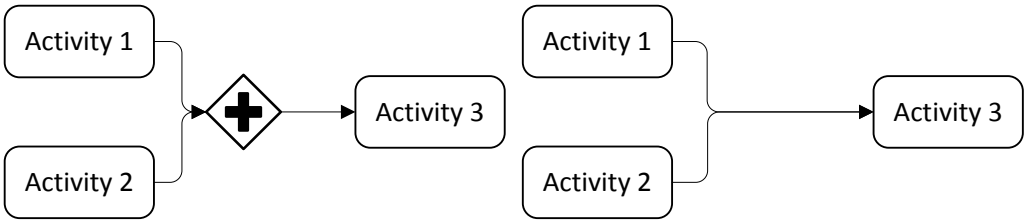

Figure 9: BPMN process model using and-gateway

Figure 10: BPMN process model using sequence flow arrows

At first sight they seem to model the same behavior: after both activities 1 and 2 are completed activity 3 is enabled. This is, however only the case in model of Figure 9. In the model of Figure 10 activity 3 is enabled after activity 1 is completed and after activity 2 is completed, and thus it is performed twice.

Data Object

Data objects are used to represent some data. Data object can be related to activities to indicate that some activity read/writes or updated some data, but is can also be associated to message flows to indicate that some data is transferred between activities. Figure 11 shows the symbol used to represent data objects.

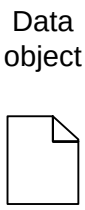

Figure 11: BPMN 1.1 Symbol used for data objects

Swimlanes

Swimlanes are used to represent entities. Activities are assigned to entities in BPMN by drawing an activity inside a swimlane. In order to represent that a certain entity consists of several other entities BPMN has two types of swimlanes: pools and lanes. For example, a pool called company x that consist of the lanes buyer and seller, is used to represent that the entity “company x” consists of an entity “buyer” and an entity “seller”, as shown in Figure 12.

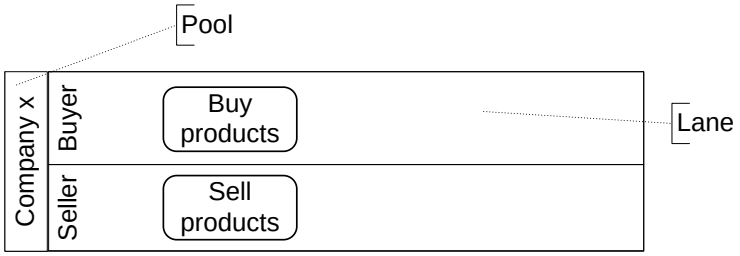

Figure 12: BPMN 1.1 example of the usage of pools and lanes

Other

Other symbols that can be used in BPMN are:

1. *Annotations* are used to give some additional information, for example to explain a certain decision, and can be seen as comments;
2. The *group* artifact is used to informally group elements together.

Figure 13 shows the symbols used for each type of artifact.

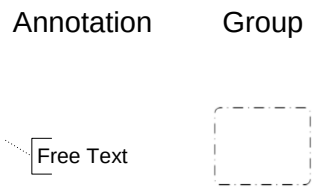

Figure 13: BPMN 1.1 symbols used for artifacts

Example in BPMN

Figure 14 shows a business process defined in BPMN that complies with the rules defined in the credit request example of Listing 2.

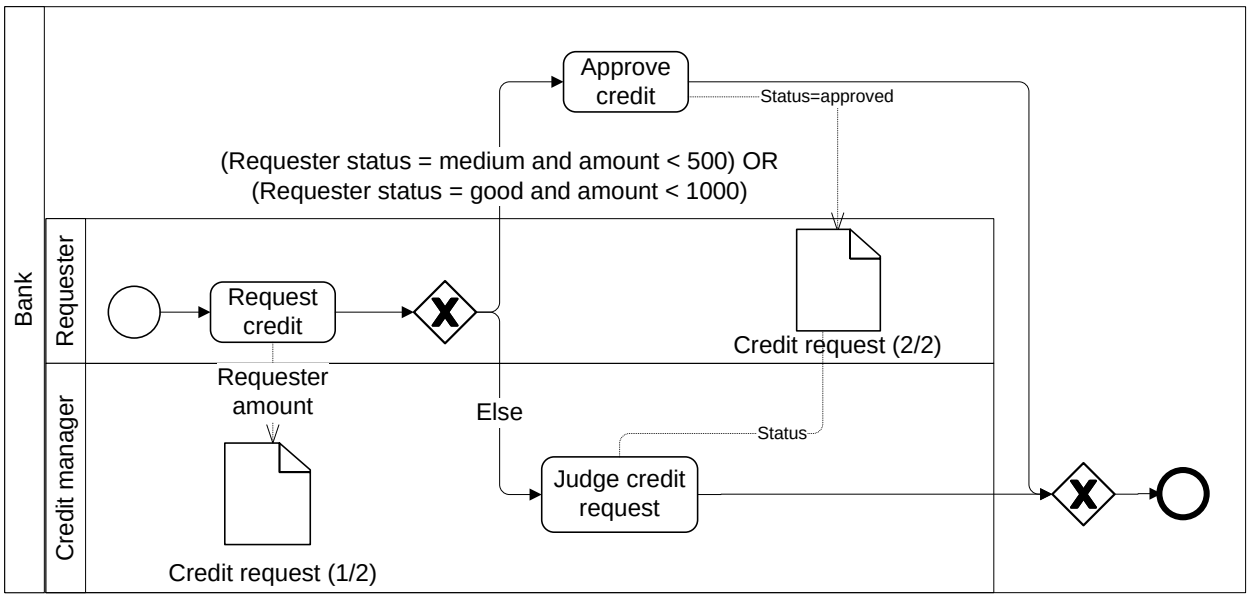

Figure 14: Credit request example in BPMN

3.2.2 YAWL

The Yet Another Workflow Language (YAWL) is a language developed at the Queensland University of Technology and Eindhoven University. There are currently three version of the YAWL language: (1) The language as described in (van der Aalst & Hofstede, 2005); (2) the language implemented in the YAWL system and (3) newYAWL as proposed in (Russell, Aalst, & Edmond, 2007). The language described in this section is the language as implemented in the YAWL system, which supports data and resource allocation and can be used to specify business processes.

The YAWL language defines of three different types of graphical elements, which can be connected to each other using arrows: conditions, tasks and sequence elements. Figure 15 shows the symbols that are used in YAWL.

There are three types of *conditions*: input conditions, output conditions and normal conditions. The input and output conditions serve to specify the start and endpoint for a process. The normal condition indicates some state in the process.

A *Task* represents some work that needs to be done. A task can be atomic or composite and can have single or multiple instances. A composite task consists of several other atomic tasks, while an atomic task cannot be decomposed any further. Single instance tasks are performed only once in a process instance while multiple instance tasks are performed several times in one process instance.

A task can also be connected to a cancellation region. A cancellation region is represented by a dotted area. If the task to where the region is connected to is activated, all the tasks within the cancellation region are cancelled.

To specify *sequences* of tasks YAWL supports three constructs that can be used for both the convergence and divergence of sequences: AND, OR and XOR.

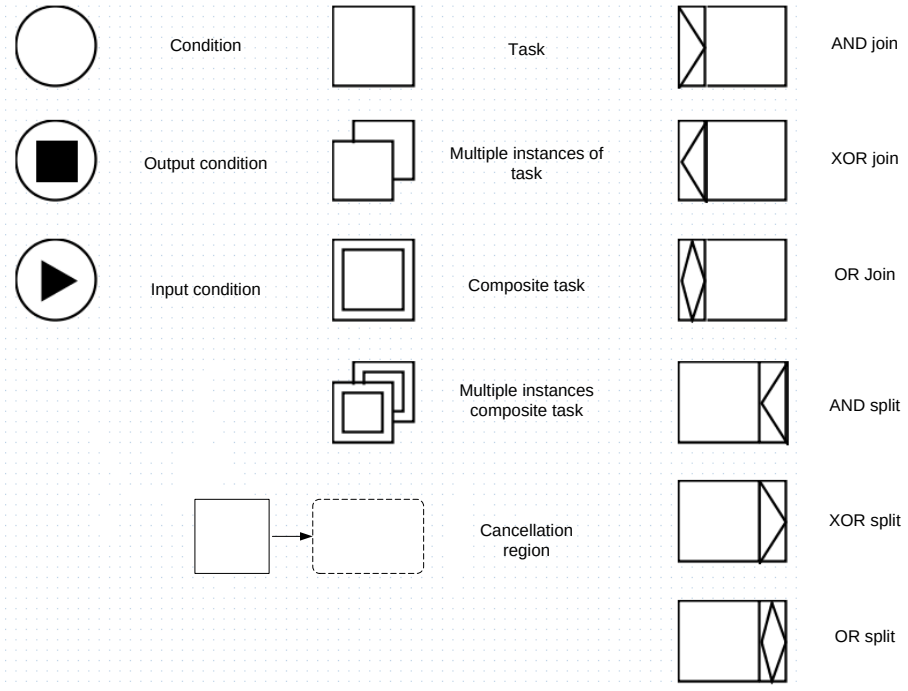

Figure 15: YAWL symbols

There is no graphical representation of data in the YAWL system. In the YAWL system, two types of variables can be specified that contain data local variables and task variables. These data elements can be specified by editing the properties of the Task or the complete diagram. XQuery (The World Wide Web Consortium) is used to manipulate the value of variables.

For the resource perspective there is no graphical representation. Assigning a resource to a task is done by specifying a resource allocation behavior for that specific activity.

Example in YAWL

Figure 16 shows a business process defined in YAWL that complies with the rules defined in the credit request example of Listing 2. Figure 17 shows an example of a data assignment in the YAWL system. In this case the result of the judge credit request, which is either an approved or a denied credit request is represented with to the parameter Creditapproved, which indicates if the credit request has been approved or denied.

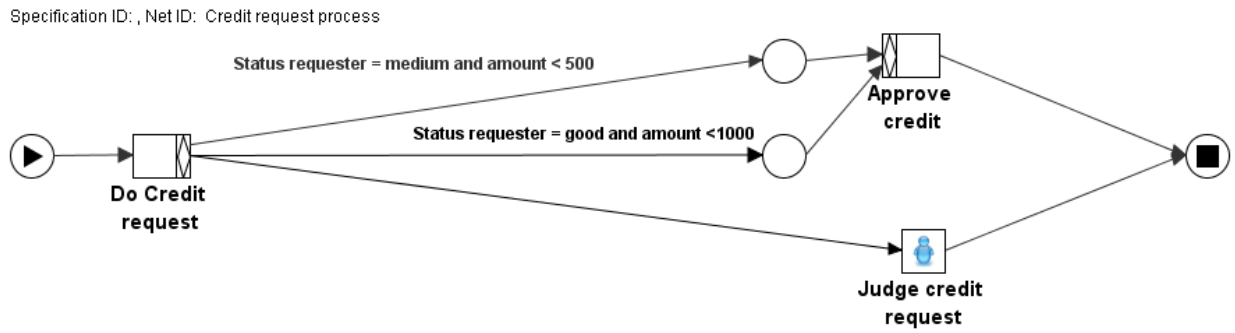

Figure 16: Credit request example in YAWL

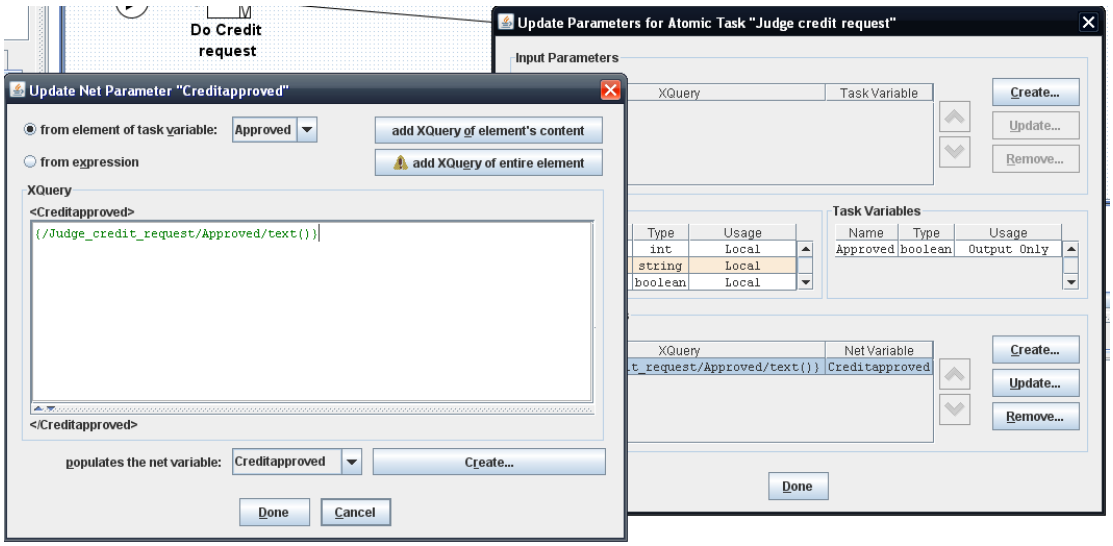

Figure 17: Data assignment in the YAWL system

3.3 Business process modeling language choice

This section makes a choice for a business modeling language. In order to make a justified decision, the two languages described in section 3.2 are evaluated according to the same criteria as the business rule languages:

- Expressiveness, the ability of the language to express the constructs needed to describe a business process in a procedural manner.

- Support, which parties support the language, and whether there are any tools available that support the specification of business processes with the language.

3.3.1 Expressiveness

In current research a lot of work has been devoted to comparing different business process modeling languages. The common approach to compare business process modeling languages, in terms of their expressiveness, is to analyze which of the workflow patterns, indentified by members of the workflow initiative (Initiative, 2008), the language can express. Patterns have been introduced that cover the control flow (Aalst, Hofstede, Kiepuszewski, & Barros, 2003), data (Edmond, Russell, Hofstede, & Aalst, 2004) and resource perspective (Hofstede, Russel, Edmond, & Aalst, 2004). Table 7 shows the result of analyzing BPMN and YAWL against the different patterns, a more detailed analysis can be found in 0.

Table 7: Result patterns analysis BPMN, YAWL

Perspective	BPMN	YAWL
	Number of patterns supported	
Control flow	29	31
Data	20	12
Resource	8	9
Total	57	52

3.3.2 Support

BPMN is a language that is in cooperation with several industry players and is adopted by the OMG. Because BPMN is an OMG standard there are many tools that support it. In fact, the official BPMN web site[1] lists 53 vendors of process tools support BPMN. Next to the support of modeling with BPMN some tools also support other features like the transformation of BPMN diagrams to an initial BPEL diagram (Lonneke Dikmans, Oracle) or a YAWL diagram (Process mining group, 2008).

YAWL is a language developed in cooperation with several researchers of the Queensland University of Technology and the Technical University of Eindhoven. Together they have developed several tools that support YAWL, such as the YAWL editor for editing YAWL specifications, the YAWL engine which can be used to execute YAWL specifications (when modeled with enough detail), and the ProM system (Process mining group, 2008) that can, among other things, convert YAWL diagrams to BPMN diagrams and analyze the modeled business process based on logs produced by the YAWL engine.

3.3.3 Conclusion

We evaluated BPMN and YAWL based on their expressiveness and support. Table 7 shows the result of the expressiveness evaluation. From the result can be concluded that, although YAWL supports more control flow and resource patterns, in total BPMN supports more patterns and thus is more expressive than YAWL.

BPMN is supported by large industry partners and is supported by over 53 tools. YAWL is only supported by two universities that have developed a small group of tools. YAWL does have the advantage that it is

[1] www.bpmn.org

directly executable when modeled with enough detail while BPMN first has to be transformed to an executable language. However, we conclude that BPMN has a better support than YAWL.

Overall it can be concluded that based on the expressiveness and support criteria BPMN is the best choice. Therefore BPMN is used in this research as the business process modeling language.

3.4 Business process optimization

This section determines the optimization problem that is used in this research. In order to determine this problem we first discuss the criteria that can be used to determine an optimal process. Next, we describe some business process optimization techniques. Based on this information section 3.4.3 defines the optimization problem.

3.4.1 Business processes criteria

The criteria used to determine the optimal business process depend on what an organization wants to achieve with (new) business processes. (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007) concluded that the criteria generally used in business process (re) design are the dimensions as depicted in the model of Brand and Van der Kolk namely time, quality, costs and flexibility (Brand & Kolk, 1995). These dimensions, however, can influence each other; improving in one dimension may have a negative effect on the others.

In order to compare several business processes in a dimension, metrics have to be identified for this dimension. These metrics depend on the information that is available in the organization at the moment of comparison. Below we discuss the metrics that can be identified for each dimension and the metrics used in this research.

Time

For the time dimension of a business process, a number of perspectives can be distinguished and each perspective has different (but related) time metrics. (Jonkers & Franken, 1996) (Iacob & Jonkers, 2009)

- The time metric from a *customer perspective* is response time, which is the time between the point in time when a customer issues a request and the point in time when he receives the result.
- The time metric from a *process perspective* is the completion time, which is the time required to complete one instance of a process.
- The time metric from a *product perspective* is the processing time which is the amount of time that actual work is performed on the realization of a certain product or result.
- The time metric from a *system perspective* is throughput, which is the number of requests that are completed per time unit.
- The time metric from a *resource perspective* is utilization, which is the percentage of the operational time that a resource is busy

In our research, we view a business process from a process perspective. The most important metric in this research is thus completion time. The completion time can be further decomposed into several other time metrics (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007),

- *Activity* time is the time a resource spends on performing the activity including setup time.
- *Service time* is the time that a resource actually spends performing the activity, excluding setup time.
- *Setup time* is the time it takes to setup an activity, such as the time to get acquainted with the order that you have to process.
- *Move time* is the time it takes to move products and/or information between activities.
- *Queue time* is the time between two activities the process is stopped waiting to be serviced because the resource that needs to perform the next activity is busy with another activity.
- *Wait time* is all other delays in a business process, e.g., the time an activity has to wait for other activities to complete in order to synchronize.

In our research, move time, queue time and wait time are not considered for the following reasons:

- Move time is the time it takes for information and/or products to move from one activity to the next. If the transformation of information is supported by an information system, this time is insignificant, therefore this metric can be neglected. If transferring information or products do take a significant amount of time then we impose that transportation is described as a separate activity.
- Queue time can be calculated based on the time an activity takes, the availability of resources and the number of business process instances per period (Verbeek, Hattem, Reijers, & Munk, 2005). The last metrics however cannot be determined without actually executing the business process over a period of time.
- Wait time is a metric that is specific to a workflow instance and whose value will probably vary from one workflow instance to another depending on availability of resources. Therefore this metric is not considered.

In a business process there are usually different execution paths with different resulting completion times, at forehand one cannot know which execution path is taken. In this research we consider the worst case scenario, i.e., the execution path resulting in the highest completion time. Therefore the following simplified version of completion time is used as the metric for the time dimension:

The completion time is calculated as the sum of activity times of all activities in the business processes, considering the worst case scenario.

Costs

Total costs are often mentioned as a means to measure a business process (Goedertier, Heason, & Vanthienen, 2007) (Vergidis, Tiwari, & Majeed, 2006) (Doshi, Goodwin, Akkiraju, & Verma, 2005). The total costs are often calculated as the sum of the costs of all participating activities, and these costs are not decomposed any further, into, for example, labor, machine, and transportation costs or fixed and variable costs (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007). Although decomposing the cost dimension into other metrics would allow a more detailed definition of an optimal process (for example, a business process with the least amount of labor cost), in this research the more integral metric for cost is used in which costs are not decomposed.

Quality

The quality dimension of a business process design can be assessed from three perspectives:

1. *External quality* where the quality of the process is assessed from the customer side. A measure for this perspective could be customer satisfaction, with number of complaints as a metric (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007);
2. *Internal quality* where the quality of the process is assessed from the side of the employees. A metric in this perspective could be task variety, which is the degree in which an employee has to carry out the same activity (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007);
3. *Design* where the quality of the business process design is assessed. In (Vanderfeesten, Cardoso, Mendling, Reijers, & Aalst, 2007) it is suggested that design quality is related to understandability, maintenance costs and errors. Design quality is measured using five metrics: coupling, complexity, cohesion, modularity and size. In recent research (Cardoso, Vanderfeesten, Reijers, Mendling, & Aalst, 2008) the cross-connectivity metric has been added as enhancement of the coupling metric to indicate the understandability of a business process design.

In this research the internal and external quality measures are not used. The values for the metrics mentioned above for internal and external quality metrics cannot be determined without actually executing the business process and are thus not applicable in our research.

The value of the metrics for business process design quality can be determined but we decided not to use this metric in our research.

Flexibility

Flexibility is often mentioned as a goal for process improvement. The question how to measure the flexibility of a business process is, however, not widely addressed. Measures of the flexibility of a business process are discussed in (Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007).

(Jansen-Vullers, Kleingeld, Loosschilder, & Reijers, 2007) mention flexibility metrics for three levels: for individual resources, for individual activities, and for the business process as a whole. For each level the flexibility is measured using the following metrics:

- The amount of time required to adapt to change;
- The amount of money required to adapt to change;
- The range of variations, e.g., volume, different type of cases, which can be handled in a business process.

In this research none of these flexibility metrics are used. The metrics mentioned above cannot be determined without actually executing the business process and going through changes, which is outside the scope of this research.

3.4.2 Methods for process optimization

In current literature several methods for process optimizing are described. (Reijers & Mansar, 2005) look at process optimization from the perspective of business process redesign. They have described 29 best practices, used in business process redesign, and qualitatively evaluated their impact on the cost, quality, flexibility and time dimension.

Vergadis, Tawari, Roy and Majeed use a mathematical approach to determine the optimal business process design in terms of optimal allocation of resources. The optimal process is determined by solving a multi-objective optimization problem. They have described two different approached based on two different notations of a business process (Vergidis, Tiwari, & Majeed, 2006), (Tiwari, Vergidis, Majeed, & Roy, 2007) (Vergadis, Tiwari, & Majeed, 2007). Below we discuss the best practices that could be used in this research and the two mathematical approaches.

Sequence versus parallel

In a business process there are usually some activities that check some condition and upon finding it true the process is ended. If there are several of these activities, and these activities do not depend on each other, from a cost perspective, it is better to arrange them in order of expected knockout effort. The expected knockout effort is the chance that the process is ended divided by the required effort, for example costs, to check the condition. From a time perspective, it is better to perform independent activities in parallel.

An example is registration for the army. In order for a person to be fit for the army he has to pass several tests, if the person fails one test then the person is unfit for the army. Table 8 shows three test that have to be performed and for each test, the costs (effort) required to perform the test and the chance, based on statistics, that a person fails the test.

Table 8: Effort and expected knockout when registering for the army

Test	Costs for performing test(effort)	Chance that person fails test and process is ended	Knock out ratio
Personality test	6	50	8,3
Physical test	8	30	3,8
Family history test	7	10	1,4

Consider the situation where the order is the same as the order given in Table 8, i.e., in the order of knock out ratio. In this situation the expected costs of testing a person are:

$$6+(0{,}5*8)+(0{,}5*0{,}7*7)=12{,}45$$

To explain the calculation:

- 6 is the costs of performing the personality test, since it is the first test this test always has to be performed.
- After the first test there is a 50% chance that the second test needs to be performed, hence the 0,5 * 8.
- After the second test there is a 50% * 70% chance that the third test needs to be performed hence the 0.5*0,5*7.

This calculation shows that it is beneficial, in terms of total costs, to perform activities that have a low cost and high chance of ending the process at the beginning of the process, i.e., from a costs perspective it is better to perform activities in the order of the knockout ratio.

Another best practice related to the knockout ratio is to execute activities as late as possible. See, for example, the business process of Figure 18a. In this business process activity B can be performed after activity A is completed. However, the information produced in activity B is only needed in activity E, which is much later in the business process. If activities C and/or D are activities where the business process might be ended then it is better, from a cost perspective, to perform activity B after activity D as in Figure 18b.

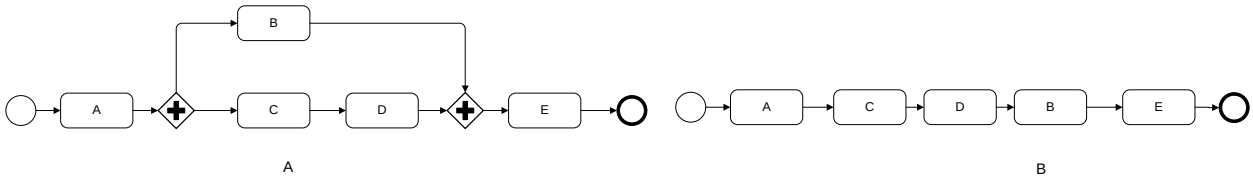

Figure 18: Example business process

Attribute optimizing in a fixed process design

In this approach, described in (Vergadis, Tiwari, & Majeed, 2007), a business process is a sequence of activities and each activity is performed by an entity against certain attributes, such as costs. The optimal business process design is the combination of entities in a fixed process design that meets specific predefined criteria.

The general optimization criterion used in this approach is:

$$\min pt_i = \min \sum_{j=1}^{n} at_{ij}$$

Where Pt_i is a to be minimized process attribute and at_{ij} is the value of the process attribute for a specific activity performed by a specific entity. This criterion is repeated for each process metric. Thus if one has two process attributes the optimization criteria are:

$$\min pt_1 = \min \sum_{j=1}^{n} at_{1j} \quad \text{and} \quad \min pt_2 = \min \sum_{j=1}^{n} at_{2j}$$

The two input variables of the approach are a set of activities described in a visual representation of the order of activities and a library with alternatives for each activity. In our example there are three fictional activities: A, B and C, two process attributes 1 and 2, and per activity there are two different entities that can perform the activity. Figure 19 shows the order of activities in our example and Table 9 gives the library of alternatives of our example.

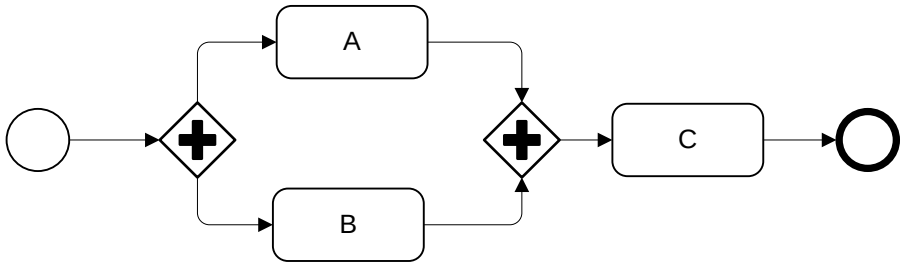

Figure 19: Given process design

Table 9: Alternatives for each activity

Activity	Alternative	Attribute 1	Attribute 2
A			
	1	4	4
	2	2	3
B			
	4	6	8
	5	3	9
C			
	6	6	9
	7	3	8

Based on these inputs a multi-objective optimization algorithm, such as NSGA2 or SPEA2 (Deb, 2001) is used to determine the optimal business process designs. The reason for using an algorithm instead of just generating all possible solutions and picking the most optimal is performance. If one, for instance, have ten activities and ten alternatives per activity there are 10.000.000.000 different process alternatives. Comparing all these alternatives would take a large amount of time.

The optimal business process designs are all designs that are on the Pareto front, which are all solutions where one cannot make a value of an attribute better without making another attribute worse.

Table 10 gives all business process alternatives and the sum for each attribute of our example. The Solutions that are on the Pareto front are solutions 4, 6 and 8.

Table 10: Process design alternatives

Solution	Chosen alternatives	Attribute 1	Attribute 2
1	1,4,6	16	21
2	1,4,7	13	20
3	1,5,6	13	22
4	1,5,7	10	21
5	2,4,6	14	20
6	2,4,7	11	19
7	2,5,6	11	21
8	2,5,7	8	22

A downside of this approach is that time is not treated as a separate attribute and the approach does not consider the actual sequence of activities. This combination is important, since the total value of most attributes are calculated by taking the sum of the attribute for each activity. However, if two

activities are performed in parallel and a subsequent activity needs both activities in order to start, then, for the time attribute, only the activity with the longest time should be considered.

For example, for the business process of Figure 20 in combination with the alternatives for activity C as given in Table 11, by taking the approach as explained in this section all alternatives are on the Pareto front. However, considering that, in parallel executing, only the activity that takes the longest time is important for the total process time, the optimal choice is alternative 3. Because activity B takes 10 minutes to complete and activity D needs both activity B and C, the time it takes for alternative C does not matter that much as long shorter than 10 minutes, and the only important attribute is attribute 2. Therefore the alternative with the lowest value for attribute 2 is the optimal which in this case is alternative 3.

Table 11: Example alternatives activity C

Activity	Alternative	Time in minutes	Attribute 2
C			
	1	6	10
	2	8	7
	3	10	6

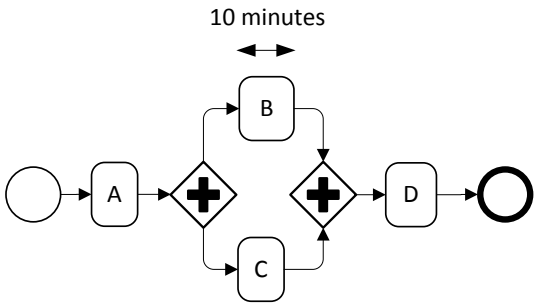

Figure 20: Example business process design

Costs and time optimization

In this approach, described in (Vergidis, Tiwari, & Majeed, 2006) and (Tiwari, Vergidis, Majeed, & Roy, 2007), a business process is a collection of activities, where each activity needs some inputs and produce some outputs in a certain amount of time against certain costs. In this approach the optimal business process design is a collection of activities that can transform a given set of inputs to a given set of outputs in an optimal way, in terms of costs and lead time.

The two optimizing criteria used in this approach are:

$$\min costs = \min \sum u_{i1} x_i$$

$$\min time = \min(\max(q_j)) \text{ where.} \forall j : b_j \in go_j$$

In the cost function, u_{i1} is the cost of activity i and x_i is a binary value indicating if activity i is participating in a design. In the time function, q_j is the time that a resource is produced. The latter of the time function is a constraint indicating that each given output needs to be produced. Next to the optimizing criteria some constrains are specified to ensure that each design created is feasible.

The starting point of this approach is a given set of inputs and outputs. Figure 21 shows the starting point in our example. There are four inputs: A, B, C and D, available at the beginning of the business process that needs to be transformed to two outputs: F and E.

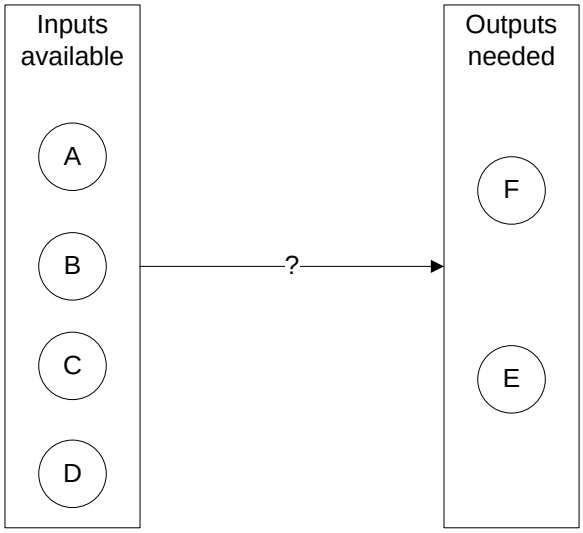

Figure 21: Starting point

Next to the given inputs and outputs there is a library of activities, where for each activity the required input, the resulting output and the time and costs of the activity are defined. Table 12 gives the library of activities of our example.

Table 12: Library of activities

Activity	Inputs	Outputs	Costs	Time
1	A, B	F	5	7
2	C,D	E	10	5
3	A,B,C	Z	7	7
4	Z,D	F, E	6	6
5	C,D	E	6	9
6	A, C	E	8	9

A multi-objective optimization algorithm is used to determine the optimal business process designs. The optimal business process designs are the ones that are on the Pareto front.

Table 13 shows all the feasible business process designs of our example. The optimal business process designs that are on the Pareto front are alternatives 1 and 2.

Table 13: Process alternatives

Alternative	Path	Total Costs	Total Time
1	1,2	15	7
2	1,5	11	9
3	3,4	13	13
4	1,6	13	9

A downside of this approach is that, in its current form, one cannot incorporate OR splits and joins and loops common in many business processes, e.g., in this approach one cannot indicate that an activity needs input A OR B.

3.4.3 Conclusion

In order to compare several business processes, metrics have to be identified. Current research suggest several metrics for each dimension, however some of the metrics cannot be used in this research because they can only be measured if the business process is executed. In our research we only design business processes to-be and do not execute it.

Given the metrics that can be used, we decided to compare the different business processes, in this research, on the time and cost dimension. For these dimensions the following metrics are used:

- The metric used for the time dimension is completion time. In this research completion time is calculated as the sum of activity times of all activities in the business process, considering the worst case scenario;
- The metric used for the cost dimension is total amount of costs, which is the sum of the costs of all possible participating activities.

The optimization criterion used in this research is defined based on the metrics above. The optimization criterion used in this research is to minimize the completion time and the total cost.

The optimization techniques described in this sections show that there are two ways in which a business process can be optimized. At the one hand one can change the order of activities and on the other hand one can change the allocation of resources to activities. The order of activities can be determined knowing the activities and the dependencies between activities and, given a set of entities that each performs one activity against a certain costs in a certain time, the optimal allocation of resources can be determined knowing the above. Therefore we define the optimization problem as follows:

Given:

- A set of activities.
- The dependencies between the activities.
- A set of entities that each performs one activity against a certain costs in a certain time

The optimal solution is the solution where:

$$(a * Ct) + (b * Tc) = \min$$
$$a + b = 1$$
$$0 \leq a \leq 1$$
$$0 \leq b \leq 1$$

Here Ct is the completion time, Tc is the total costs, and a, b are fixed weights to indicate the relative importance between completion time and costs. The relative importance is added because improvement in completion time may have a negative effect on costs and vice versa.

This section also discussed some approaches that could be used to determine the optimal business process. However, none of these approaches can be directly used in our research:

- The optimal sequence can depend on the knockout ratio of different activities. However this attribute is not considered in this research because the information needed to calculate this value is often not available without the business process having executed for a period of time.
- The first mathematical approach cannot be used because all process attributes are calculated as the sum of the attribute of all individual activities. As we have taken time as an optimization criterion this method does not give proper results.
- The second mathematical approach cannot be used because one cannot incorporate OR splits and joins and loops common in many business processes, furthermore one needs to know the starting inputs and outputs in the beginning to create the process design, which might not be the case in our research.

4 Approach

This chapter presents the basic concept of model transformation, some tools that can be used in model transformation, lists some current approaches to automatically transform business rules to business processes and presents our transformation approach.

This chapter is further structured as follows: Section 4.1 describes the basic concepts in model transformation. Section 4.2 describes some tools that can be used in model transformation, section 4.3 discusses current approaches used to automatically transform business rules to business processes and section 4.4 discusses our approach.

4.1 Model transformation

In the context of Model Driven Engineering and the Model Driven Architecture, many transformation techniques are available that are based on automatic model transformation (Czarnecki & Helsen, 2006). The general approach for model transformation as suggested by the OMG is to transform a model expressed in the source language (source model) to a model expressed in the target language (target model).

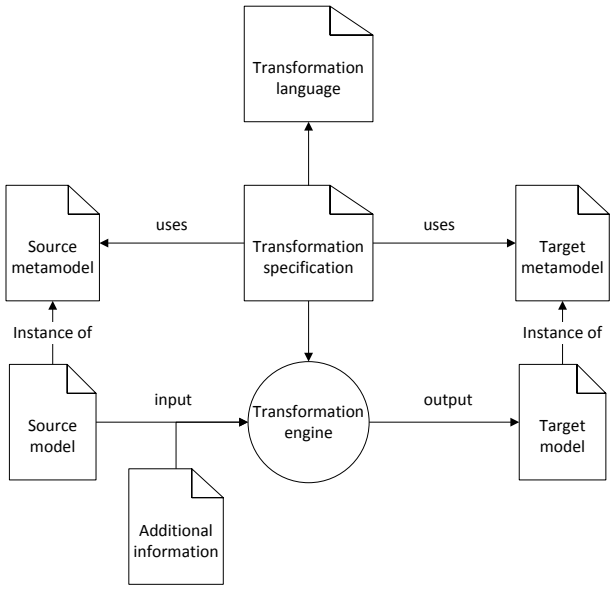

Figure 22: Automatic transformation method adopted from (Czarnecki & Helsen, 2006)

The automatic transformation method can be described using the transformation pattern, as shown in Figure 22. The input of the transformation is the source model that is an instance of the source metamodel, and possibly some additional information. The transformation engine performs the transformation as defined in the transformation specification, which consist of some transformation rules written in a transformation language, and generates the target model, which is an instance of the target metamodel

4.2 Model transformation tools

In literature, three types of model transformations can be identified: text to model (t2m), model to model (m2m) and model to text (m2t). The difference between the approaches is the way they use

metamodels. In a m2m transformation the metamodels of both the source and target model are explicitly defined. In the case of transformation to/or from text the metamodel of the textual part is often implicitly defined in the transformation definition itself (Czarnecki & Helsen, 2006). In our research we need tools that support t2m and m2m transformations. We need t2m transformation since the input for the transformation method, the PA-specification, is in a textual form that needs to be transformed to a BPMN model, which is a model. We need m2m tools because the t2m transformation cannot be specified in such a way that it directly results in an optimal BPMN model. Below we discuss some tools that can be identified for each type of transformation.

4.2.1 Text to model tools

There are not many approaches/tools available that support t2m transformations. One tool that can be found is *oAW Xtext*, implemented in openArchitectureWare (oAW) (openArchitectureWare.org). oAW Xtext is a framework/tool for the development of textual languages. The language that is used in the input of the transformation must be described in an Extended Backus-Naur Form (EBNF) like grammar language. Based on this description, the tool generates a metamodel (implemented in EMF), an Eclipse Text Editor and a parser. The parser is responsible for reading the text and transforming it to an instance of the generated metamodel.

Another tool that could be used for text to model transformation is *ANother Tool for Language Recognition* (ANTLR) (Parr). As with Xtext the language to be transformed must be described using EBNF. Based on this specification ANTLR creates a parser that transforms the specification to an abstract syntax tree, implemented in an ANTLR-specific language.

4.2.2 Model to model tools

QVT

Query/View/Transformation (QVT) (Object management Group, 2008) is the model transformation standard defined by the OMG. The QVT standard consists of several languages that are organized as shown in Figure 23.

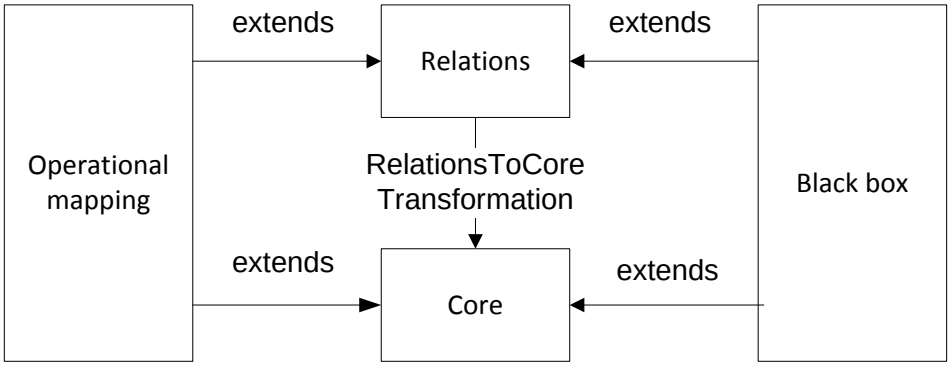

Figure 23: Structure QVT

The Relations language provides capabilities for specifying transformations as a set of relations among models. Relations contain a set of object patterns. These patterns can be matched against existing model elements, instantiated to model elements in new models, and may be used to apply changes to existing models.

The Core language is declarative and is the foundation language for two other two languages. The Core language is as powerful as the Relations language, and is relative simple compared to Relations. However, transformation definitions written in Core tend to be longer than the equivalent definitions written in the Relations language. (Kurtev, 2008)

The Operational Mappings language extends both the Relations as the Core language with procedural constructs and OCL (Object Management Group, 2006) constructs. The syntax of Operational Mappings language provides constructs commonly found in procedural languages (loops, conditions, etc.).

The Black Box mechanism allows the plugging-in and execution of external code, such as Java, during the transformation execution.

The reason that several languages have been adopted instead of just one language is that model transformation is a relatively new area and the OMG did not know what to standardize as the model transformation language.

Some tools that implement a QVT language are SmartQVT (SmartQVT), which implements the QVT-Operational language and Medini QVT (IKV++ technologies), which implements the Relations language.

ATL

Atlas transformation language (ATL) is a language and toolkit developed by the ATLAS Group (INRIA & LINA) as a response to the QVT Request for Proposal. It consists of both imperative and declarative constructs.

The declarative part of ATL is based on matching rules. Such a rule consists of a pattern that is compared to the source model and of a target pattern that is created in the target model for every match.

ATL offers two procedural constructs, namely rule and action block. A called rule is explicitly called, like a procedure, but its body may be composed of a declarative rule. Action blocks are sequences of procedural instructions that can be used in either matched or called rules.

The ATL language is used in the m2m project of the Eclipse Foundation (ATLAS Group).

Xtend

Xtend is the language used in oAW for m2m transformation. It is a textual procedural language, similar to OCL. Next to Xtend, oAW (openArchitectureWare.org) also allows one to specify additional (more complex) transformations using Java.

4.2.3 Conclusion

In this section we have described two tools that can be used for t2m transformation, oAW xText and ANTLR. From these tools we have decided to choose oAW xText. The main reason for this choice is that it creates the model in a format that can be used in the other model transformation tools.

Also for m2m transformation several tools can be used. From these tools we decided to use oAW XTend. The reason for choosing this tool is that Xtend is relatively simple in comparison with other tools (Mitoussis & Macos, 2008) and that xText and Xtend are part of the same platform, oAW.

4.3 Methods to transform business rules to business processes

Two methods can be found in current literature to transform business rules to business processes. Both approaches use SBVRSE as business rule language. Although we de not use SBVRSE in this research, the general approach might still be useful as an inspiration for our transformation method.

4.3.1 From SBVRSE to UML activity diagram

(Raj, Prabhakar, & Hendryx, 2008) suggest an approach to transform SBVRSE specifications to an UML activity diagram (Object Management Group, 2005). Their approach is based on the following assumptions:

- Only operative rules with the enforcement level automatable are relevant for the activity diagram. The enforcement level automatable is not a defined in the SBVRSE standard, but has been added by the authors.
- Only rules with an if then statement are transformed, the authors assume that those rules are written in the format “it is obligatory that x if y” as opposed to “it is obligatory that if y then x”.
- Facts which include transitive verbs are considered to be activities, e.g. `user` *`inserts`* `card`.

Furthermore, the approach only describes business rules that lead to a diagram where one activity enabled at most one other activity. The authors do mention what should happen if one activity enables multiple other activities or vice versa (add Fork or Join) but do not mention how this is incorporated in their method.

Figure 24 shows the approach used by (Raj, Prabhakar, & Hendryx, 2008) to derive the activity diagram from the rules specification. Each step is further explained below:

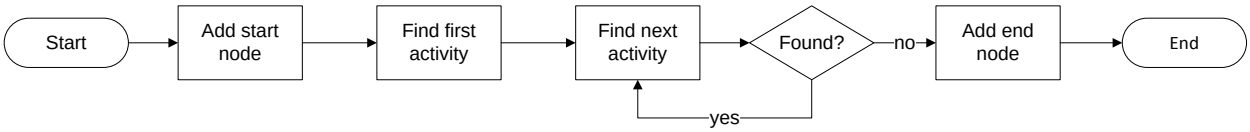

Figure 24: Approach used by (Raj, Prabhakar, & Hendryx, 2008) to transform a SBVRSE specification to UML activity diagram

- First a start node is created. Then all rules are searched for an automatable rule with no if statement. If such an activity is found that this is considered to be the first activity.
- Then based on the found activity all rules are searched for an automatable rule that has that activity in the if condition. If such an activity is found it is considered to be the next activity.
- The above is redone until there are not any automatable rules left that have the last found activity in their if statement.
- Then an end node is added to complete the diagram.

Thus, for example, the automatable rules that can be seen in Listing 6 are transformed to the activity diagram as shown in Figure 25.

```
user
card
```

```
atm
bank
account
password
status-message
user inserts card
atm requests password
user enters password
atm verifies account
bank returns status-message
It is obligatory that each user inserts exactly one card
It Is obligatory that atm requests exactly one password if user inserts exactly one card
It is obligatory that user enters exactly one password if atm requests exactly one password
It is obligatory that each atm verifies at least one account if user enters exactly one password
It is obligatory that each bank returns at least one status-message if atm verifies at least one account
```

Listing 6: SBVRSE atm example

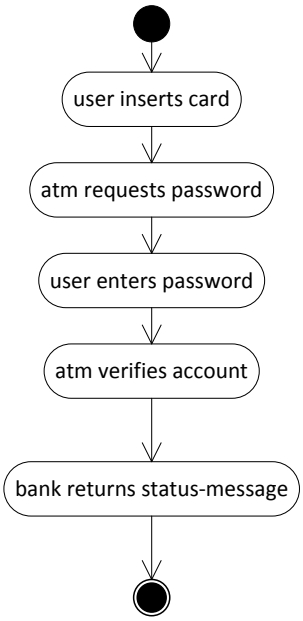

Figure 25: UML activity diagram atm example

4.3.2 From SBVRSE to BPMN

(Eder, Filieri, Kurz, Heistracher, & Pezzuto, 2008) suggest an approach to transform a SBVRSE specification to a BPMN model. Their approach is based on the following assumptions:

- Each binary fact is transformed to an activity. The first entity mentioned by the fact is considered to be the performer of the activity, the verb and the second entity is the activity. For example, in the fact `user inserts card` the BPMN activity is “inserts card” and the performer of the activity is user. Each performer of an activity is transformed to a BPMN lane.
- Each characteristic fact is considered to be a condition that can be true or false. Therefore this fact is mapped to a BPMN gateway that has two outputs, one for the truth value of the condition one for the false value.
- The rules are written in the format “**if** x **then** y” without using keywords like **it is necessary that**.

The sequence of activities is derived using the following approach:

- Each rule that has the pattern “**if** x **then** y” means that activity y has to be performed after x. if there are more than two if statement with the same x then a fork is used and the activities are performed in parallel.
- Each rule that has the pattern “**if** x **and** y **and** z **then** a” is mapped to a join.

Thus, for example, the rules from Listing 7 are transformed to the BPMN diagram as shown in Figure 26.

```
user
card
atm
password
account
user inserts card
atm requests password
user enters password
atm verifies account
account is valid
atm does rollback
user selects amount
atm gives amount

if the user inserts card then atm requests password
if the atm requests a password then user enters password
if the user enters the password then atm verifies account
if the atm verifies the account and account is valid then user selects
amount else atm does rollback
if user selects amount then the atm gives amount
```

Listing 7: SBVRSE atm example 2

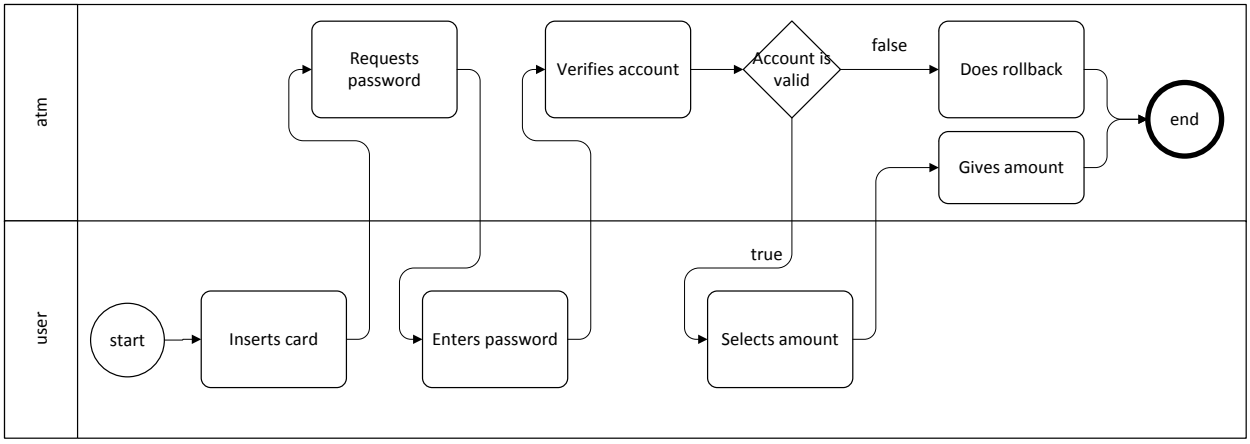

Figure 26: BPMN atm example 2

4.3.3 Conclusion

In this section two methods to transform business rules to business processes are described. Both approaches have in common in that they use the "if then else" construct present in business rule to derive the sequence of activities. However, both approaches also have in common that they do not discuss difficult situations in where the condition part of the if statement are used in multiple rules. We do find the general idea (finding first activity and then recursively find the next activity until there are no more activities left) behind the approach used by (Raj, Prabhakar, & Hendryx, 2008) useful in the context of this research.

4.4 Our approach

This section discusses the approach taken to develop our transformation method. Figure 27 shows the goal of our transformation method; we want to automatically transform a business rule specification, written in the PA-notation, to a business process specification, modeled in BPMN.

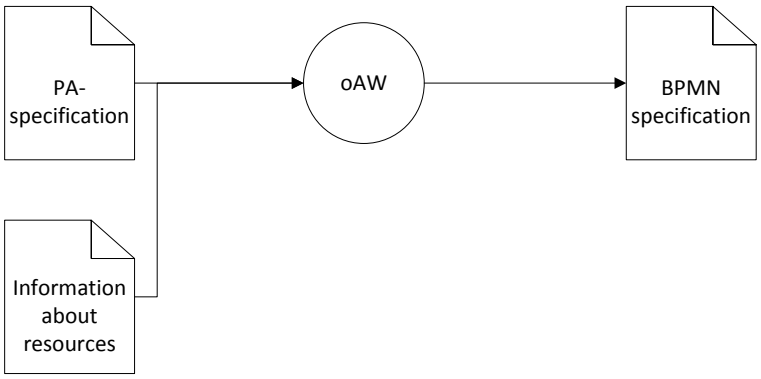

Figure 27: Transformation method used in this research

This transformation could be performed in one step. However, we found it easier to separate the transformation in several other transformations. Figure 28 shows the steps taken, in the following sections we discuss each step and why it is needed.

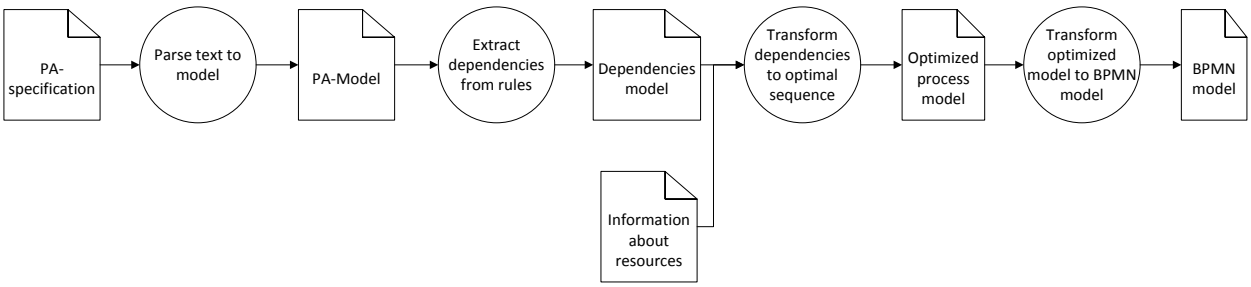

Figure 28: Transformation process

4.4.1 Parse text to model

The first step in our approach is to parse the PA-specification to its model representation. Figure 29 shows the transformation process we used in this step.

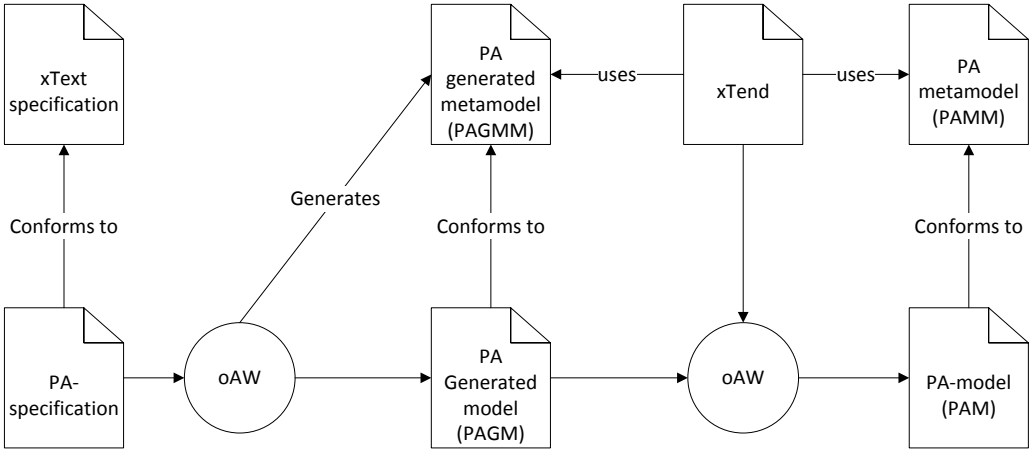

Figure 29: Transformation from PA-specification to PA-model

In order to parse the PA-specification we have defined the grammar of the PA-notation in xText. oAW then generates a metamodel based on this specification and a parser. The metamodel generated by oAW is not easy to use as each expression is of the same type (assign Expr), that contains another expression type through an attribute relation. For example, the Exists expression of Listing 8 results in the nested list as can be seen in Figure 30. In order to find what type an expression really is we have to go down the attribute relations, until an expression actually contains information related to that specific type.

Because we do not want to perform this process in the ongoing transformation, we have decided to define a more heterogeneous PA-metamodel and a transformation between the generated metamodel (PAGMM) and the PA-metamodel (PAMM) so that finding the real element is done once for every expression. Figure 31 shows the result of such a transformation for the PAGM of Figure 30. Section 5.1.6 discusses this transformation step in more detail.

```
The following applies:
       One CR exists in CREDITREQUESTs with:
              amount= 500
CREDITREQUESTs =
```

```
        amount:int
```

Listing 8: Example PA-notation, ExistsExpr

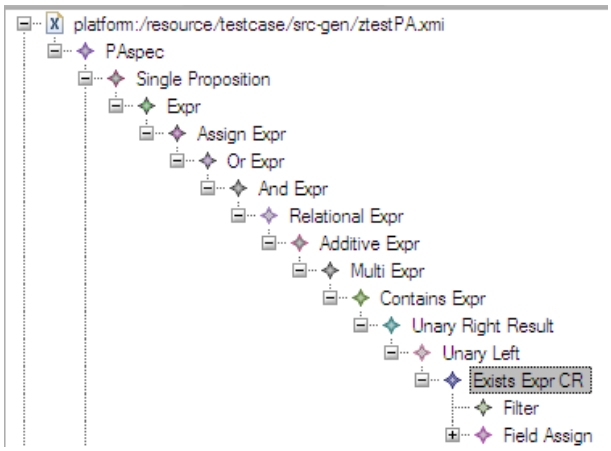

Figure 30: PAGM of the ExistsExpr of Listing 8

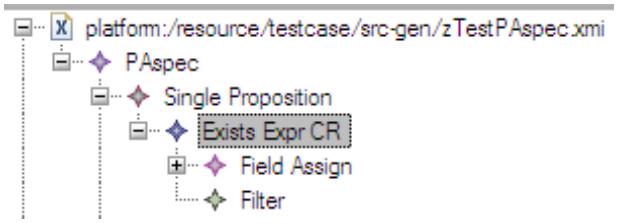

Figure 31: PAM of the ExistsExpr of Listing 8

4.4.2 Extract dependencies from rules

The second step that we have defined, is the transformation from the PAM to a process model that only contains the dependencies between activities. Figure 32 shows the transformation process we used in this step. Section 6.2 discusses this transformation step in detail.

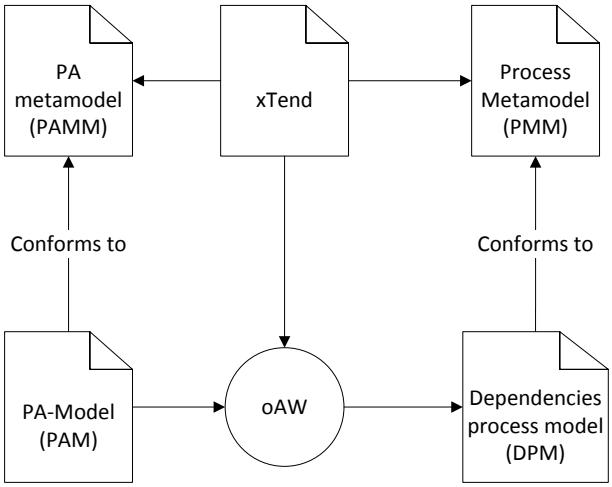

Figure 32: Transformation from PA-Model to dependencies process model

The reasons for introducing the self defined process metamodel (PMM) as discussed in 5.2, instead of transforming a PAM directly to a BPMN model is that we wanted to separate deriving the dependencies

between the activities that can be found in a PA-specification from deriving the optimal sequence of activities as presented in a BPMN model, so that, when the optimization criteria change, the transformation method can be changed more easily.

To explain the difference between dependencies and sequence consider the PA-specification as can be seen in Listing 9. This specification can only result in one dependency set, namely that in order to reach result 3, result 1 and result 3 are needed. However, in terms of sequence of activities, the same specification can lead to three different process models, as shown in Figure 33. What is considered to be optimal determines which process model is optimal.

```
The following applies:
"Result 1"
And
"Result 2"
and
(if "Result 1" and "Result 2" then "Result 3")
```

Listing 9: Example PA-specification

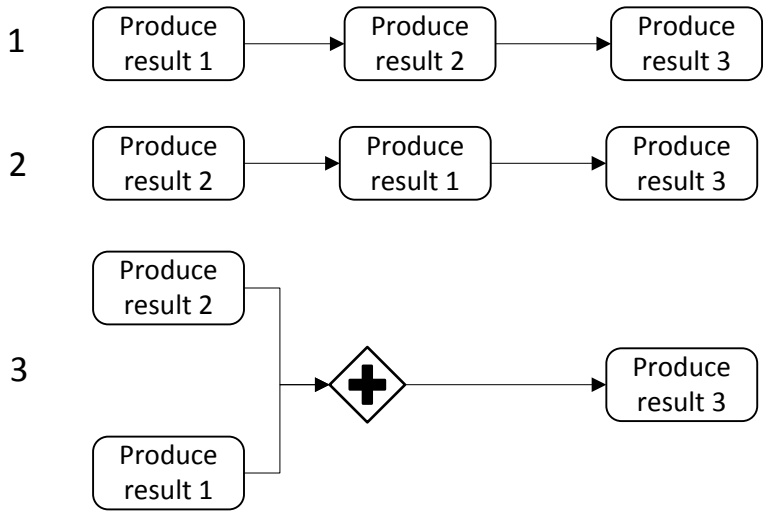

Figure 33: Example process models that comply with the rules of Listing 9

4.4.3 Transform dependencies to optimal sequence

The third step that we have defined, is deriving the optimal process model from the dependencies process model (DPM) leading to an optimized process model (OPM) that conforms to the PMM. Figure 34 shows the transformation process we used in this step. Section 6.3 discussed this transformation step in detail.

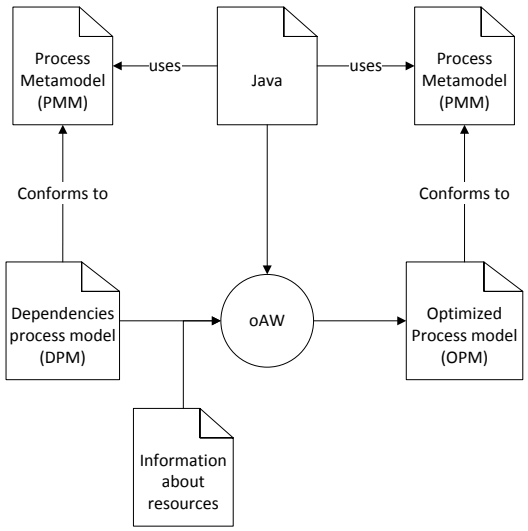

Figure 34: Transformation from dependencies process model to optimized process model

The reason for transforming the DPM to an OPM instead of transforming it to a BPMN model mainly is convenience. An example that shows an aspect in which the PMM is more convenient than the BPMN metamodel is:

- In the PMM the link between process elements is defined through a list attribute, while in the BPMN metamodel process elements do not have direct links with other process elements, but links with edges (represented by arrows) which have two links with process elements (source and target). If we thus want to have all incoming process elements of one process element, in a OPM, we simply get the list of incoming activities. However in a BPMN model, we have to iterate through the list of incoming edges and get the process element that is contained in the source attribute of the edge.

4.4.4 Transform optimized model to BPMN model

The last step we have defined is to transform the OPM to a BPMN model. Figure 35 shows the transformation process we used in this step. Section 6.4 discusses this transformation step in detail.

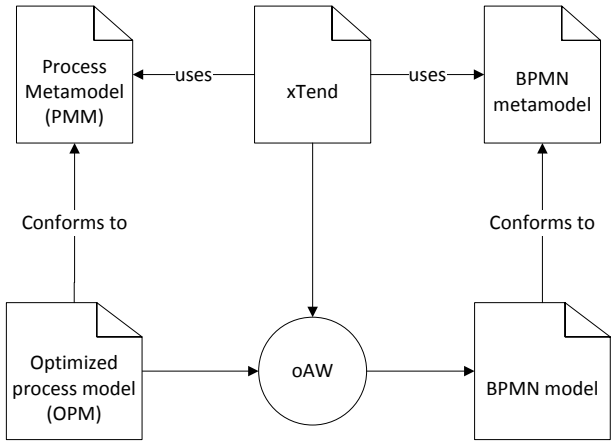

Figure 35: Transformation from OPM to BPMN model

5 Metamodels

This chapter describes the different metamodels that are used in our transformation method. Section 5.1 describes the metamodels used to capture the PA-notation, section 5.2 describes the metamodel used to capture the dependencies between activities and the sequence between activities and section 5.3 describes the BPMN metamodel that we used.

5.1 PA-notation metamodels

In our transformation method two metamodels are used to capture the elements in the PA-notation. (1) the metamodel generated by oAW, the PA-generated metamodel (PAGMM), and (2) the metamodel we defined, the PA-metamodel (PAMM). The complete PAGMM and PAMM can be found in Appendix B and Appendix C respectively. Because the constructs in the two metamodels are similar we only discuss the PAMM here.

Because the complete PAMM is rather large, we introduce the constructs in steps. First a very simple metamodel showing the main elements is described, followed by a description of the constructs related to Collection, Unary, Binary and Primary expressions. Furthermore, each section describes the constraints of the PA-notation that could not be captured in the metamodel. The last section discusses the transformation between the PAGMM and the PAMM.

5.1.1 Main elements

A PA-specification consists of a set of statements, as shown in Figure 38. A statement can either be a Collection (Collec[2]) or a Proposition. A Collection consists of a set of Features, and a Proposition consist of one Expression (Expr). A Proposition can either be a SingleProposition, e.g., “the following applies” or a MultiProposition, e.g., “For each x in Collection applies”.

A Proposition contains one Expression. However, not every expression type is allowed. The expressions that are allowed to be used in Propositions are: AndExpr, IfExpr and ExistsExpr.

Two examples that show the relation between PA-notation and PA-metamodel constructs are:

1. Figure 36 shows how the single proposition of Listing 10 is represented as an instance of the PA-metamodel;

```
The following applies:
       someExpression
;
```

Listing 10: PA-notation single proposition example

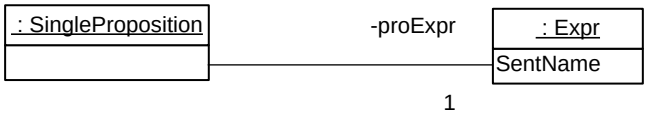

Figure 36: Object diagram showing how the example Listing 10 is represented in a PAM

[2] The name Collec is used instead of Collection to avoid name conflicts with the internal type Collection used in oAW.

2. Figure 37 shows how the MultiProposition of Listing 11 is represented as an instance of the PA-metamodel.

```
For each CR in CREDITREQUESTs applies:
       someExpression
;
CREDITREQUESTs =
       amount:int
```

Listing 11: PA-notation MultiProposition example

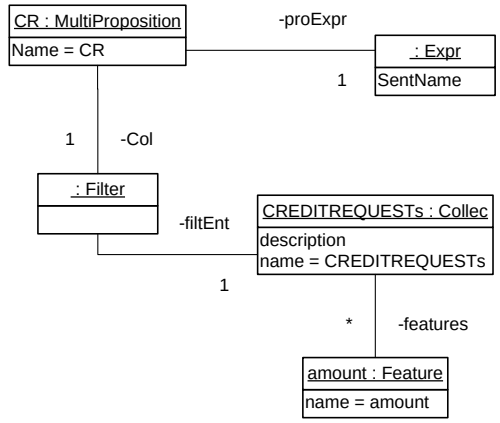

Figure 37: Object diagram showing how the example of Listing 11 is represented in a PAM

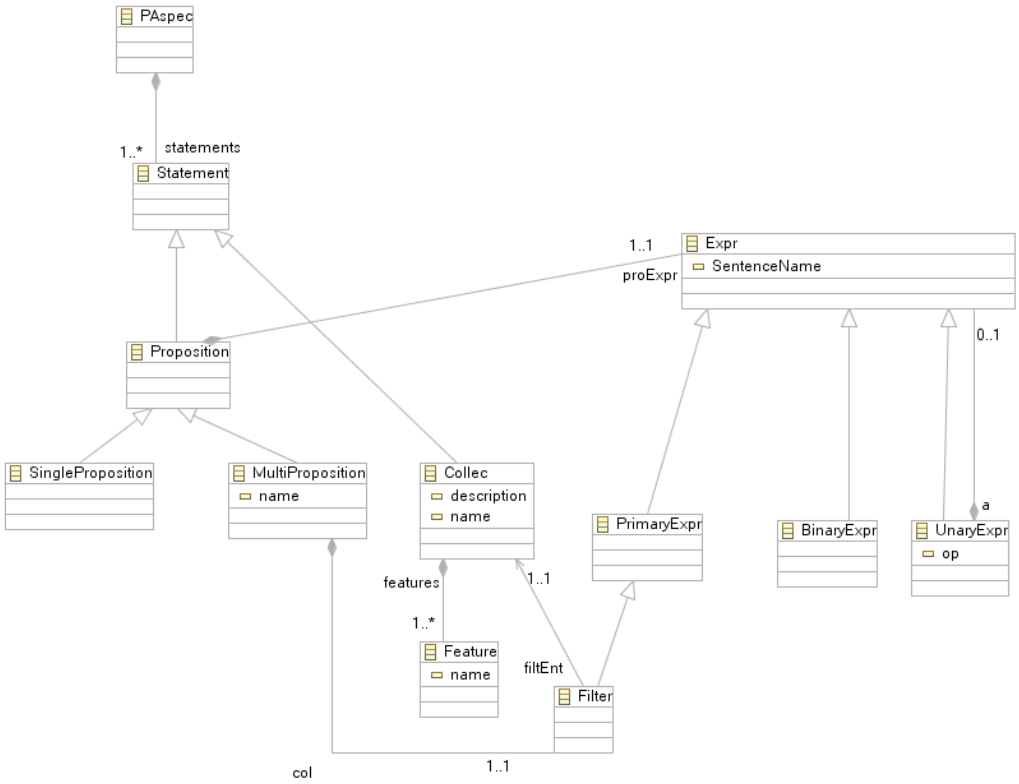

Figure 38: Main elements of the PA-notation

5.1.2 Collection elements

Figure 39 shows the PA-metamodel elements related to specifying collections.

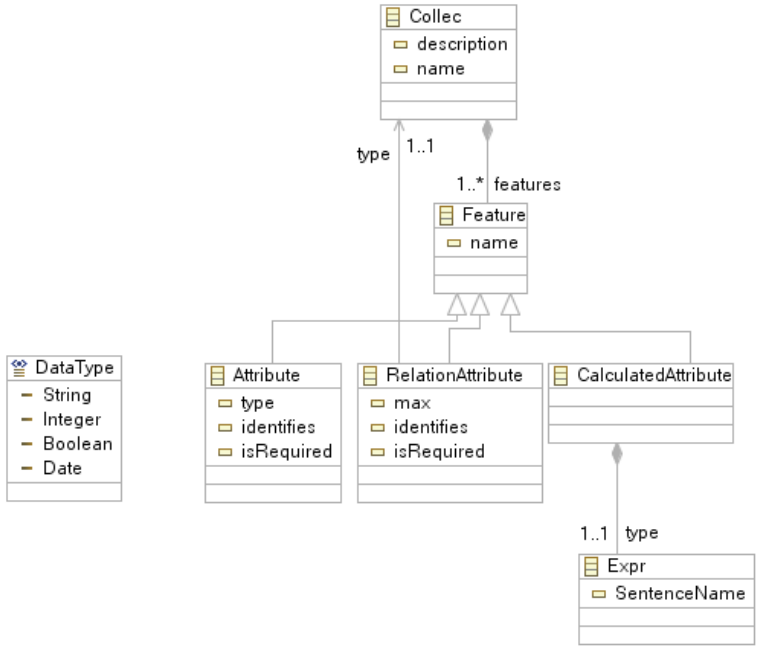

Figure 39: PA-notation Collection elements

A Collection has a name, a description and a set of features. There are three types of features:

1. *Attribute*, which consists of a name, a Boolean indicating if the attribute identifies an instance of the collection, a Boolean indicating if the attribute is required and a type. This type can only have the values that are specified in the enumeration DateType, which contains the values String, Integer, Boolean and Date;
2. *RelationAttribute,* which consists of a name, a Boolean indicating if the attribute identifies an instance of the collection, a Boolean indicating if the attribute is required, a reference to another collection and an Integer max that constrains the maximum amount of instances the attribute can hold;
3. *CalculatedAttribute,* which consists of a name and an expression that indicates how to determine the value of the Calculated attribute. The expression types that are allowed to be used for a calculated attribute are: UnaryNumberExpr, UnaryBooleanExpr, BinaryBooleanExpr, BinaryResultExpr, IfExpr, Literal, Filter and ListExpr.

Figure 40 shows how the collection specification of Listing 12 is represented as an instance of the PA-metamodel.

```
PERSONs description 'Natural persons' =
       bsnr: int identifies required
       firstname: String required
       birthdate: Date required
```

```
        marriedto: PERSONs(1)
        age: NOW - birthdate
```

Listing 12: PA-notation collection example

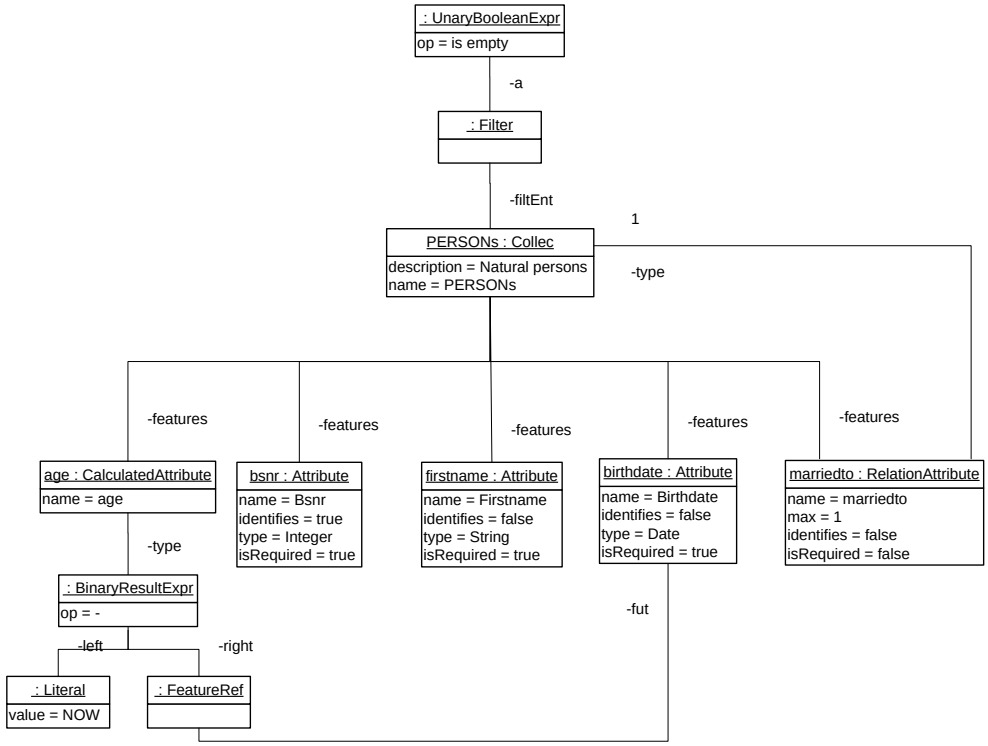

Figure 40: Object diagram showing how the example of Listing 12 is represented in a PAM

5.1.3 Unary expressions

Figure 41 shows the PA-metamodel elements related to unary expressions.

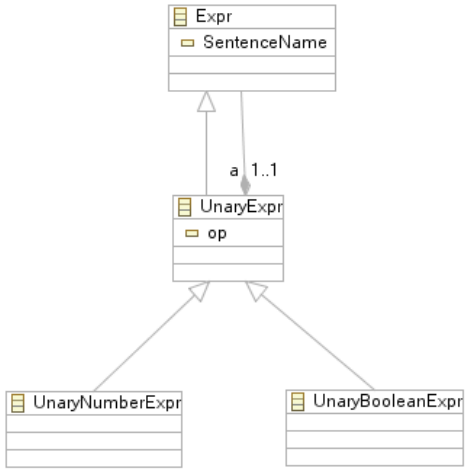

Figure 41: PA-notation Unary expressions

Unary expressions are either UnaryNumber or UnaryBoolean expressions. The difference between the two is that an UnaryNumber expression returns a result that has the type number and an UnaryBoolean returns a Boolean value.

Figure 42 shows examples of how an UnaryNumber and an UnaryBoolean expression in the PA-notation are represented as an instance of the PA-metamodel.

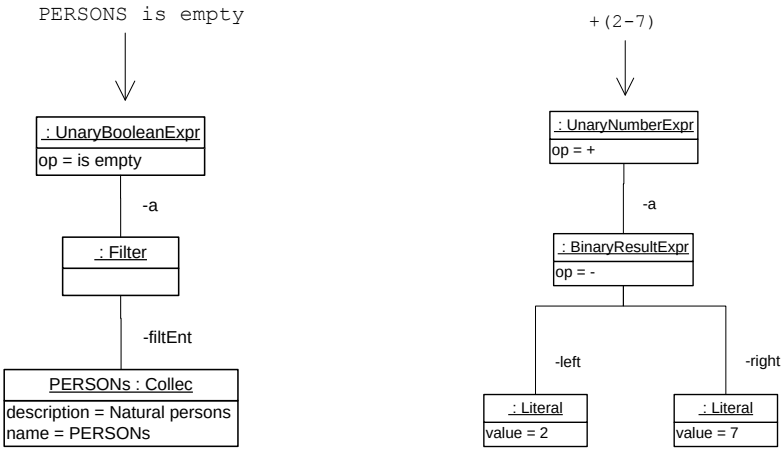

Figure 42: Object diagrams showing how an UnaryBooleanExpr and an UnaryNumberExpr are represented in a PAM

Not all expressions are allowed to be used in an Unary expression. For example, the expression: "'some string' is empty", is an allowed expression according to the metamodel, however in an actual specification this expression is meaningless.

- The expression types that are allowed to be used in UnaryNumberExpr are: UnaryNumberExpr, BinaryResultExpr, and VariableFeatureRefExpr and Literal, if they return a number.
- The expression types that are allowed to be used in UnaryBooleanExpr are: UnaryBoolaanExpr, OrExpr, AndExpr, BinaryBoolanExpr, Filter and VariableFeatureRefExpr and literal, if they return a Boolean.

5.1.4 Binary expressions

Figure 43 shows the PA-metamodel elements related to binary expressions.

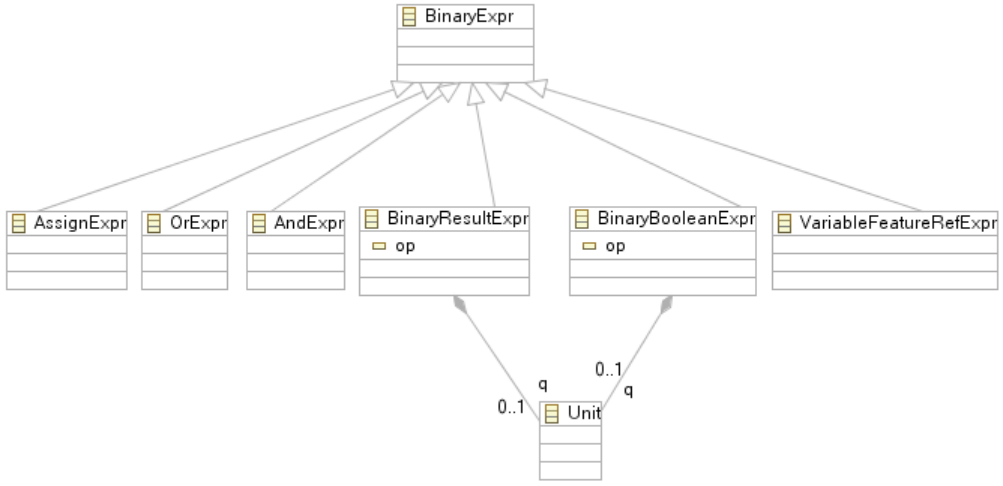

Figure 43: PA-notation Binary expressions

There are six types of Binary expressions:

1. *AssignExpr*, which is used to assign a value to a variable, e.g., CR.status='approved';
2. *OrExpr*, which is used to represent OR expressions, e.g., "CR.status== medium *or* CR.status == good" ;
3. *AndExpr*, which is used to represent AND expressions, e.g., "status == medium *and* amount > 500" ;
4. BinaryResultExpr, which is used to represent a mathematical operation between two expressions, e.g., 8+3. Q represents a time unit that can be used in the calculation, e.g., "date1 + 5 *days*";
5. *BinaryBooleanExpr*, which is used to represent a comparison between two expressions. The op attribute is the operator used in the comparison, e.g., "==". The attribute Q holds the time unit that can be used in the comparison, e.g., "date-date> 5 *days*";
6. *VariableFeatureRefExpr*, which is used to refer to a certain value of an attribute of certain collection instance created by an ExistsExpr or a multi proposition statement, e.g., CR.amount;

Not all expressions are allowed to be used in each Binary expression. Table 14 gives an overview of the allowed expressions to be used in Binary expressions.

Table 14: Allowed expression to use in Binary expressions

Expression	AssignExpr		BinaryBooleanExpr		BinaryResultExpr		VariableFeatureRefExpr	
	left	right	left	right	left	right	left	right
UnaryExpr								
UnaryNumberExpr		*	v	v	v	v		
UnaryBooleanExpr		*						
BinaryExpr								
AssignExpr								
OrExpr								
AndExpr								
BinaryBooleanExpr		*						
BinaryResultExpr		*	v	v	v	v		
VariableFeatureRefExpr	v	*	*	*	v	v		
PrimaryExpr								
IfExpr								
InputExpr		v	v	v	v	v		
ExistsExpr		*						
Literal		*	v	v	v	v		
Filter		*						
ListExpr		*						
VariableRefExpr		*					v	
FeatureRef							v	v

To explain the *'s in the table above.

- The allowed expressions in the right part of an AssignExpr depend on the variable that is assigned. For example, if the assigned variable has the type Boolean then only expressions are allowed that result in a Boolean value such as Binary and Unary Boolean expressions.
- In a BinaryBooleanExpr only numbers, strings, dates and Booleans can be compared. A VariableFeatureRefExpr is thus not allowed if it refers to a RelationAttribute.

Figure 42 shows how the example of Listing 13 containing an AndExpr, OrExpr, a BinaryBooleanExpr and a VariableFeatureRefExpr is represented as an instance of the PA-metamodel.

```
"Credit is requested"
And
If CR.status== medium or CR.status == good then ……

"Credit is requested" =
One CR exists in CREDITREQUESTs with:
       status= 'Good'
```

Listing 13: PA-notation binary expressions example

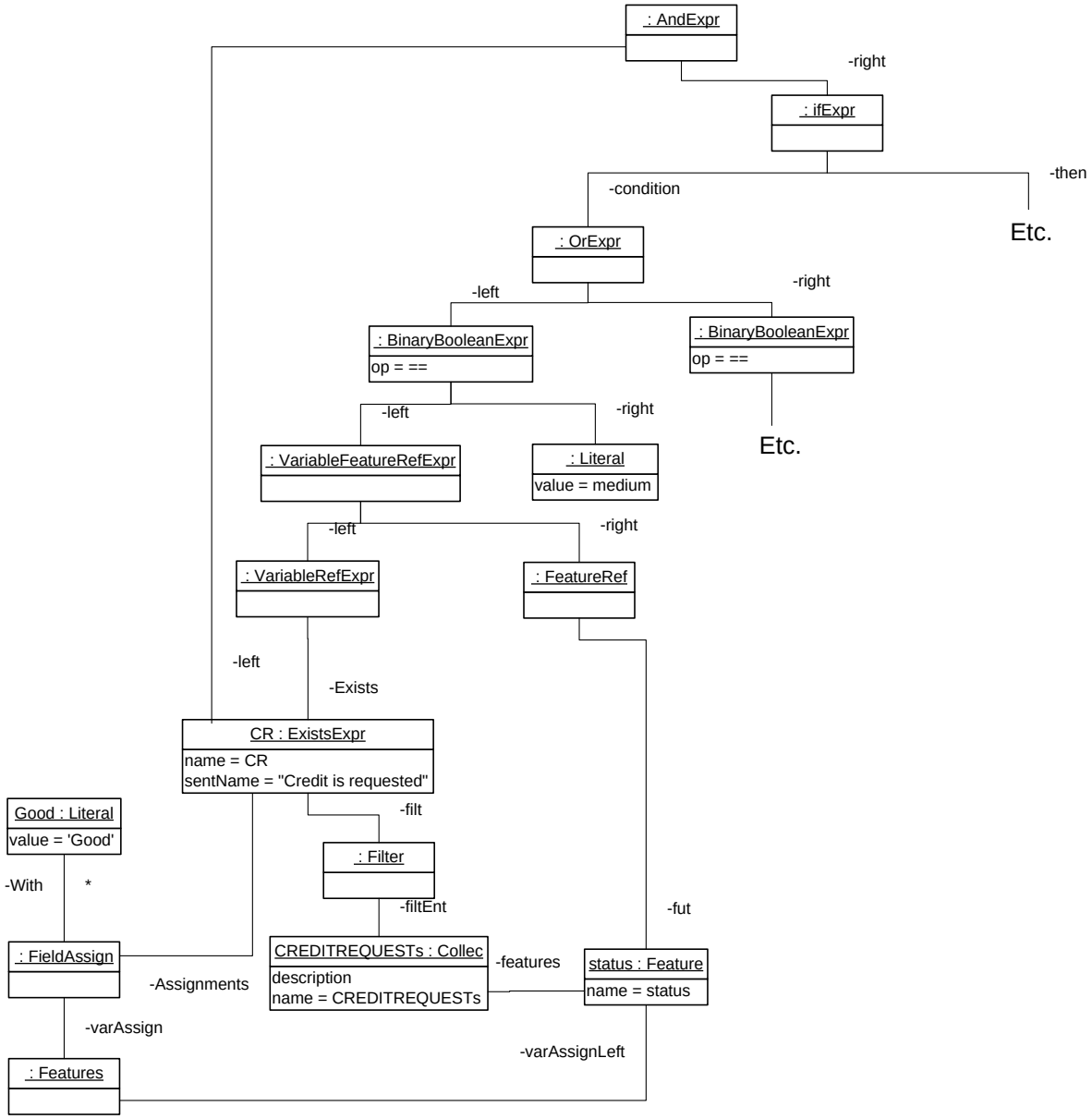

Figure 44: Object diagrams showing how the example of Listing 13 is represented in a PAM

5.1.5 Primary expressions

Figure 45 shows the PA-metamodel elements related to primary expressions.

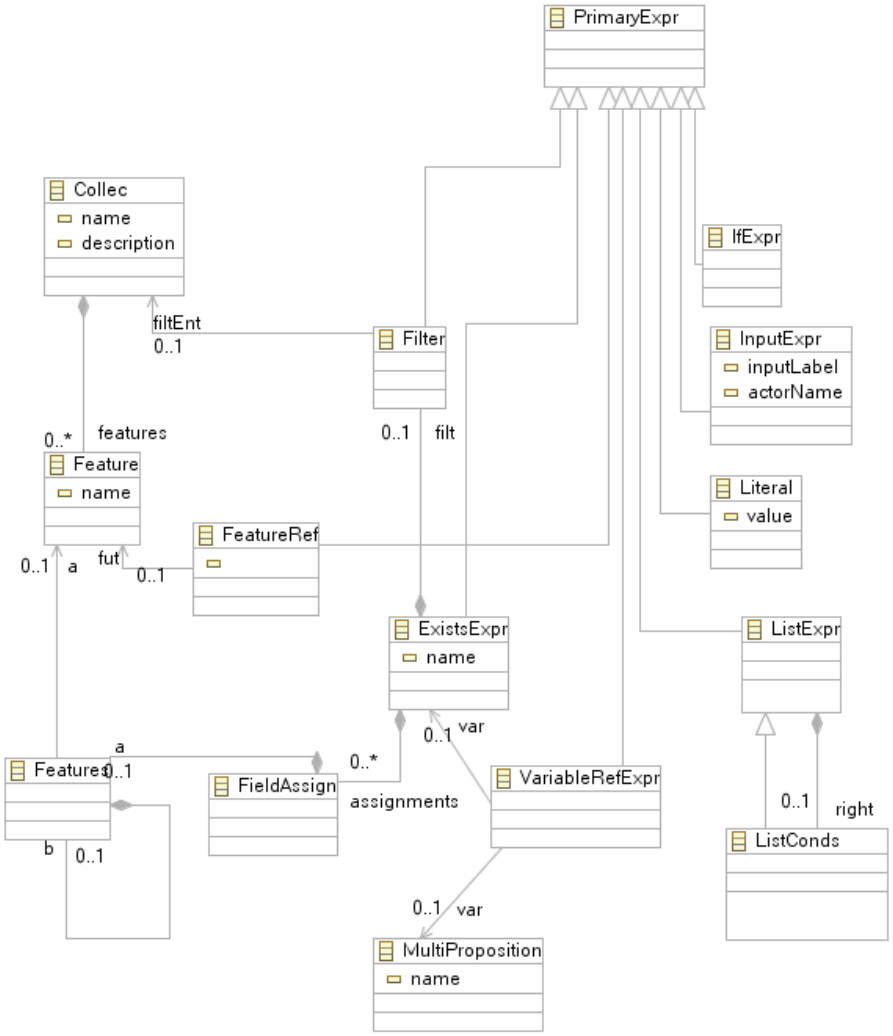

Figure 45: PA-notation primary expressions

There are eight types of primary expressions:

1. *IfExpr*, which is used to represent if then else statements, e.g., if CR.amount > 500 then “deny credit” else “approve credit”. Three other expressions are used in respectively the condition, then and else parts;
2. *InputExpr*, which is used to indicate that some input needs to be delivered. Two other expressions are used in the ChosenFrom and BasedOn part of the input expression;
3. *ExistsExpr*, which is used to represent that some element needs to be created. Expressions are used to assign a value to a collection element similar to an assign expression;
4. *Literal*, which is either a String, an Integer, a Boolean or the keyword NOW;
5. *Filter*, which is used to apply a filter on a collection, e.g., PERSONS (name=’John’). In a Filter an expression is used to represent the filterCondition;

6. *ListExpr and ListConds*, which are used to represent a list, e.g., [‘approved’, ‘denied’]. Expressions represent the elements in the list;
7. *VariableRefExpr*, which is used to refer to a variable introduced by an Exists expression or Multiproposition statement;
8. *FeatureRef*, which is used to refer to certain features defined in a collection.

Not all expressions are allowed to be used in each PrimaryExpression. Table 15 gives an overview of the expressions allowed to be used in PrimaryExpressions.

Table 15: Allowed expressions to be used in Primary expressions

Expression	IfExpr		InputExpr		ExistsExpr	Filter	ListExpr
	condition	then/else	chosenFrom	BasedOn	with		ListCond
UnaryExpr							
UnaryNumberExpr					*		
UnaryBooleanExpr	v				*		
BinaryExpr							
AssignExpr	v	v					
OrExpr	v					v	
AndExpr	v	v				v	
BinaryBooleanExpr	v				*	v	
BinaryResultExpr					*		
VariableFeatureRefExpr			v	v	*		
PrimaryExpr							
IfExpr		v					
InputExpr	*						
ExistsExpr	v	v			*		
Literal					*		v
Filter			v	v	*		
ListExpr			v		*		
VariableRefExpr				v	*		
FeatureRef							

To explain the *’s in the table above:

- In the right part of an ExistsExpr the limitations are the same as in an AssignExpr. Thus the allowed expression depends on the type of the attribute that is being assigned.
- The condition part of an IfExpr can only hold an InputExpr if the InputExpr returns a Boolean.

Figure 42 shows how the example of Listing 13 containing all primary expressions are represented as an instance of the PA-metamodel.

```
If CR.amount == 500 then ……

“Credit is requested” =
       One CR one CR exists in CREDITREQUESTs with:
              amount = input from ‘REQUESTER’ chosen from [500,1000]
```

Listing 14: PA-notation primary expressions example

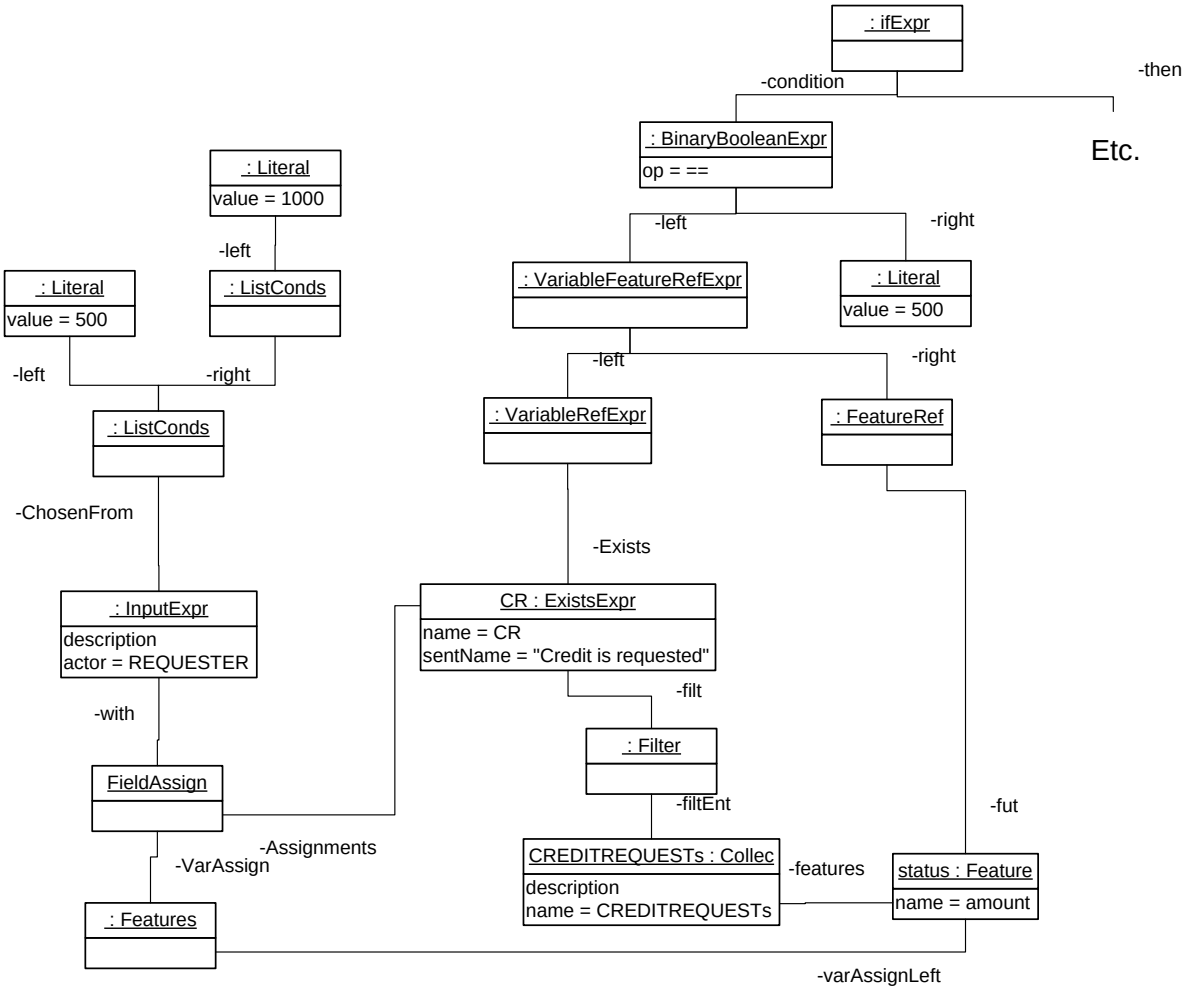

Figure 46: Object diagrams showing how the example of Listing 14 is represented in a PAM

5.1.6 Transformation between PA-notation metamodels

This section discusses the syntax changes we had to make to the PA-Notation, differences between the PA-generated metamodel and the PA-metamodel and the transformation process between the two PA-notation metamodels.

PA-grammar

In our approach we have defined the grammar of the PA-notation in Xtext. Listing 15 gives an example of such a specification. However, in order to obtain a valid Xtext specification we had to make small changes to the PA-notation. These changed are related to the identification of relationships between grammar elements. In the PA-Notation several language elements are identified using capitals:

- Collections, e.g., *PERSONS* description 'Natural persons'.
- Relation attributes, e.g., *MARRIEDTO* = PERSONS (1).
- Exists expressions, e.g., There exist a *PS* in PERSONS with.
- Actors in input expressions, e.g., Input from *SECRETARY.*
- Multi propositions, e.g., For each *P* in PERSONS applies.

Xtext only allows for maximal one identifier type, e.g., all capitals, all numbers, etc, to be used for maximal one grammar element. Therefore we decided to use the following identifiers:

- Collections are identified by a term with all capitals ending with a lowercase s, e.g., *PERSONs.*
- Relation attributes (as well as calculated and normal attributes) are identified by a lowercase term, e.g., *marriedto.*
- Exists expressions are identified by an uppercase term, e.g., *PS.*
- Actors do not have an identifier, they are just strings, e.g., Input from *'SECRETARY'.*
- Multi propositions are identified with any combination of capitals, numbers, and lowercase letters so that it cannot be identified as one of the above identifiers, e.g., *Pk* is allowed but *Ps* is not because it can then be identified as a collection.

```
SingleProposition:
"The" "following" "applies" ":"
      proExpr=Expr
;
```

Listing 15: Example Xtext specification

Differences between metamodels

There are several differences (apart from the structure) between the constructs used in the metamodel generated by oAW (PAGMM) and the PA-metamodel we defined (PAMM). These differences are listed below.

The constructs of sentences and sentence definitions that can be found in the PAGMM are removed in the PAMM. Each sentence reference found in the PAGMM is replaced by the expression in the sentence definition, and the name of the sentence definition is attached to that expression. To show the difference, the example in Listing 16 is parsed to the PA-generated model (PAGM) that can be seen in Figure 47 and Figure 48 shows how that model is represented as a PA-model (PAM).

```
Then "Credit is approved"
"Credit is approved" =
      CR.requeststatus = 'approved'
```

Listing 16: Example of sentence

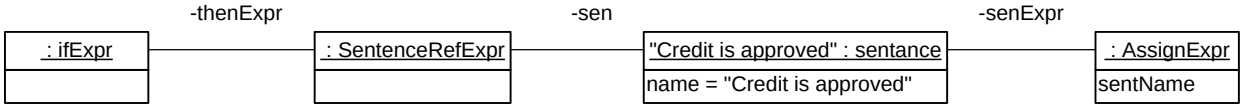

Figure 47: Simplified Object diagram showing how the example of Listing 14 is parsed to a PAGM

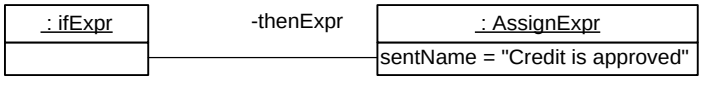

Figure 48: Object diagram showing how the PAGM of Figure 47 is represented in a PAM

The same process is performed for the capsulate expression. This language element is only relevant in textual specifications and is used to distinguish one expression from another. For example, in the specification given below it might become unclear for the reader if the “interest is determined” part belongs to the then statement or to the overall rule specification. We solved this by allowing the use of brackets.

```
if “Credit CR is requested” then “Credit is judged” and “Interest is
determined”
```

With brackets:

```
 (if “Credit CR is requested” then “Credit is judged”)
and
“Interest is determined”
```

In a model we do not have this problem so the capsulate construct is removed. For example, the PA-specification given above is parsed to the PAGM as shown in Figure 49 and Figure 50 shows how that model is represented as a PAM.

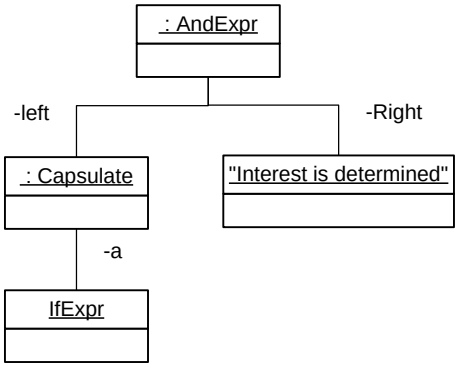

Figure 49: Object diagram showing the encapsulate concept

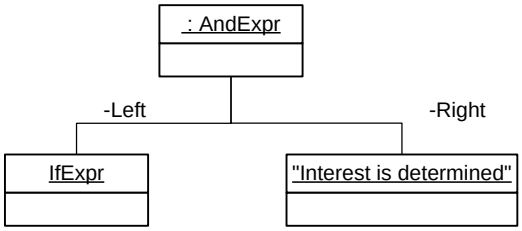

Figure 50: Object diagram showing the result of the removal of the encapsulate construct

In the PAGMM there is a difference between an UnaryRight expression and an UnaryLeft expression. This difference is the result of the operator being at the left or at the right of the expression. For example, the expression “PERSONs *is empty*” is an UnaryLeft expression and the expression “*not* (3<5)” is an UnaryRight expression. This is a syntax difference that is not relevant for the model, since in both cases it is one operator with one expression. Since all UnaryLeft expressions return a value of the type Boolean we decided to transform all UnaryLeft expressions to an UnaryBoolean expression.

Transformation process

Next to the transformations discussed above the main goal of transforming the PAGMM to the PAMM is to change the structure of the model. The structure change is only related to Expressions. Collections and propositions remain the same. Figure 51 shows the structure of Expressions in the PAGMM.

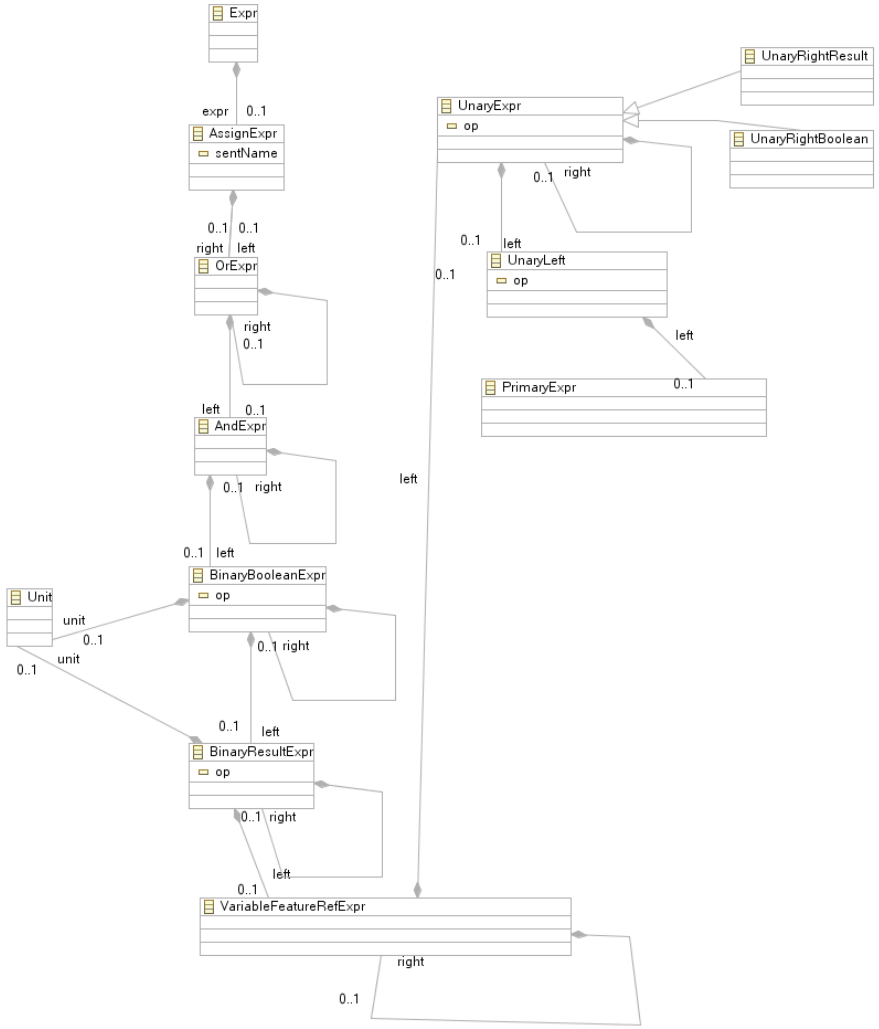

Figure 51: Structure of expression in the PAGMM

The transformation starts with processing the expression in the (Single or Multi) Proposition. For that expression we generate the corresponding PAMM element. However, each expression in the PAGM is an AssignExpr, at first sight. In order to process the expression we have to find the real expression that is being used. This is done in the following way:

- For each BinaryExpr (assign, or, and, etc) we check if the attribute right is empty. If right is empty then we have to look a level deeper, so we check the expression type of left. If right is not empty then the real expression is found.
- For an UnaryExpr we check if left is empty; if so then the real expression is an UnaryExpr, if not, we look one level deeper.

- For an UnaryLeftExpr we look if the attribute op is empty, if so then the real type is a PrimaryExpr. If not then the real expression is an UnaryLeftExpr.

If the real expression is known we check if the specific type is allowed to be used. For example, in a proposition the allowed types are AndExpr, IfExpr and ExistsExpr. If the type is allowed we create the PAMM element that corresponds to the PAGM element. This PAGM element may again contain several other expressions so the above process is recursively repeated until there are no more expressions left to process.

5.2 Process metamodel

The process metamodel (PMM) is a self defined metamodel that has two distinct purposes. It captures the dependencies between activities and captures the sequence between activities. Below we discuss the metamodel elements used for both purposes. The complete PMM can be found in Appendix D.

5.2.1 Dependencies

Figure 52 shows the elements of the PMM that are used to capture the dependencies between activities. Each of these elements is described below.

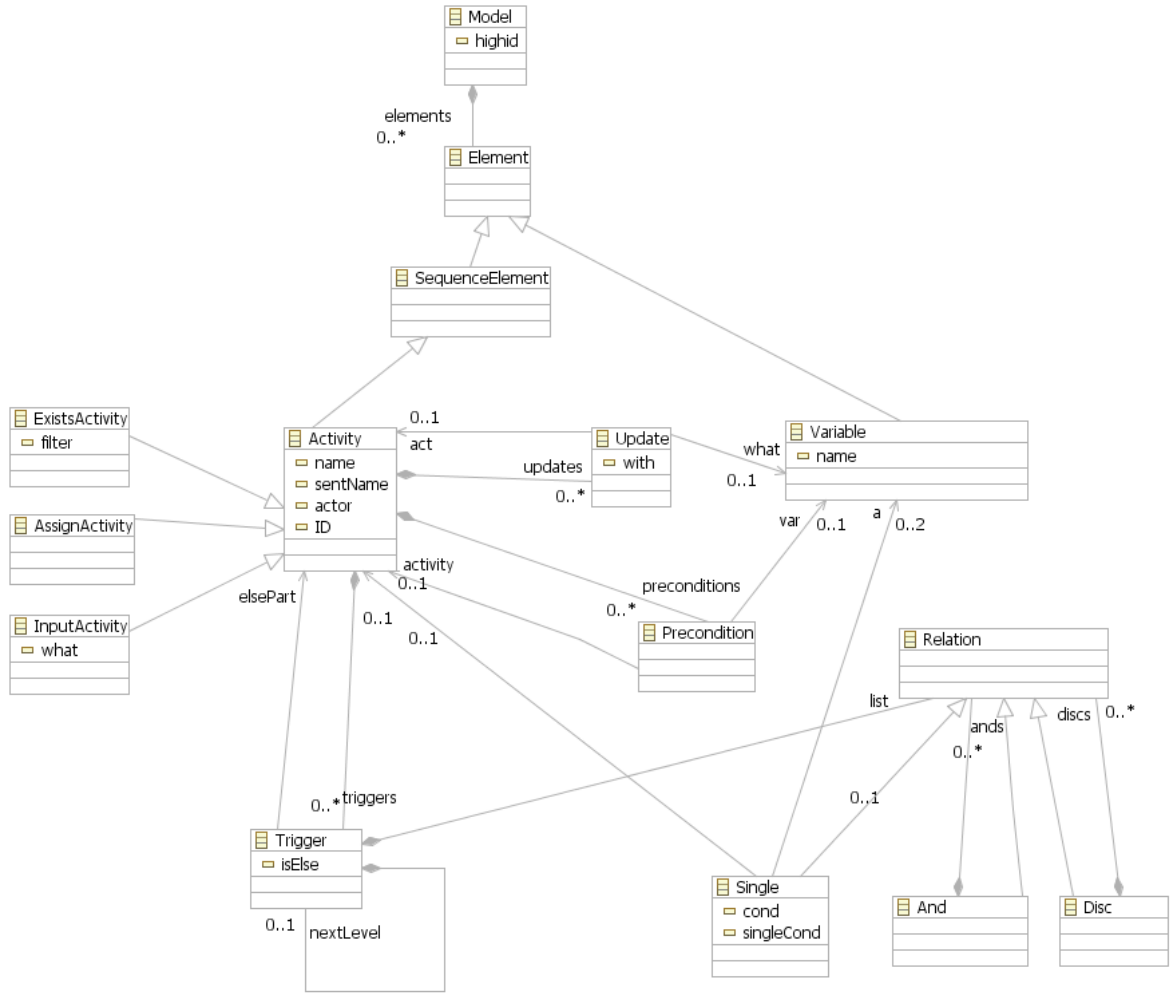

Figure 52: Process metamodel, elements related to dependencies

- *Activity* has a name, an Id and an actor indicating who should perform the activity. Each activity can have several triggers and preconditions and can update several variables.
- *Trigger* is used to capture certain conditions that, when true, enable the activity the trigger is related to. Each trigger has a relation, an elsePart to indicate the activity that must be performed if the trigger is false and an IsElse Boolean to indicate that the trigger is the ElsePart of another trigger.
- *Relation* is used to capture the different structures that triggers can have. These structures are Single, Disc and And:
    - *Single* represents the condition that must be true in order for the trigger to be true. A single relation can refer to activities, variables or it can be a single condition meaning that is does not need variables or activities in order to be evaluated, e.g. , "PERSONs (age==65) is Empty".
    - *Discriminator (Disc)* means that one of several other relations must be true in order for the trigger to be true.
    - *And* means that several relations must be true in order for the trigger to be true.
- *Update* is used to indicate that a variable is updated by a specific activity. The value that the variable is updated with is stored in the attribute what, or in the relation attribute act if the variable is updated with the outcome of some activity, such as an input activity.
- *Precondition* is used to specify dependencies between activities that are not explicitly defined with triggers. An example of such an implicit relationship is that one activity uses a data element created by another activity.

5.2.2 Sequence

Figure 53 shows the elements of the PMM that are used to capture the sequence between activities. Each element is described below.

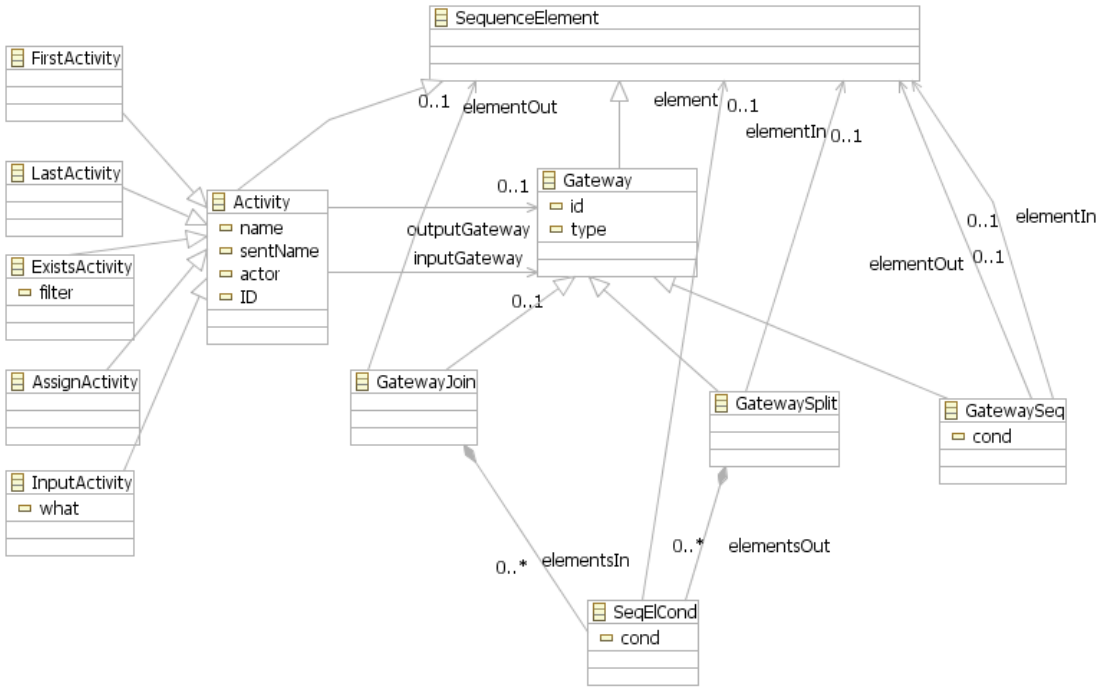

Figure 53: PMM elements related to sequence of activities

- *FirstActivity* and *LastActivity* are dummy activities that are added to ensure that each process has exactly one start and one end activity.
- *SeqGateway* is used to connect one sequence element to another sequence element.
- *GatewaySplit* is used to connect one sequence element to several other sequence elements. This gateway can be of three types:
    1. AND split, which is used in order to represent the Parallel split pattern as discussed in section 6.1.2;
    2. OR split, which is used in order to represent the multiple choice pattern as discussed in section 6.1.6;
    3. XOR split, which is used in order to represent the exclusive choice pattern as discussed in section 6.1.5.
- *GatewayJoin,* is used to connect several sequence elements to another sequence element. This gateway can be of three types:
    1. AND join, which is used in order to represent the synchronization pattern as discussed in section 6.1.4;
    2. XOR join, which is used in order to represent the merge pattern as discussed in section 6.1.3;
    3. DISC join, which is used in order to represent the discriminator pattern as discussed in section 6.1.7 .

5.3 BPMN metamodel

The BPMN metamodel we use in this research is the BPMN metamodel, provided by the Eclipse BPMN modeler (Intalio Inc). The complete BPMN metamodel can be seen in Appendix E. This section discusses

a simplified version of the BPMN metamodel as can be seen in Figure 54. Each element is described below.

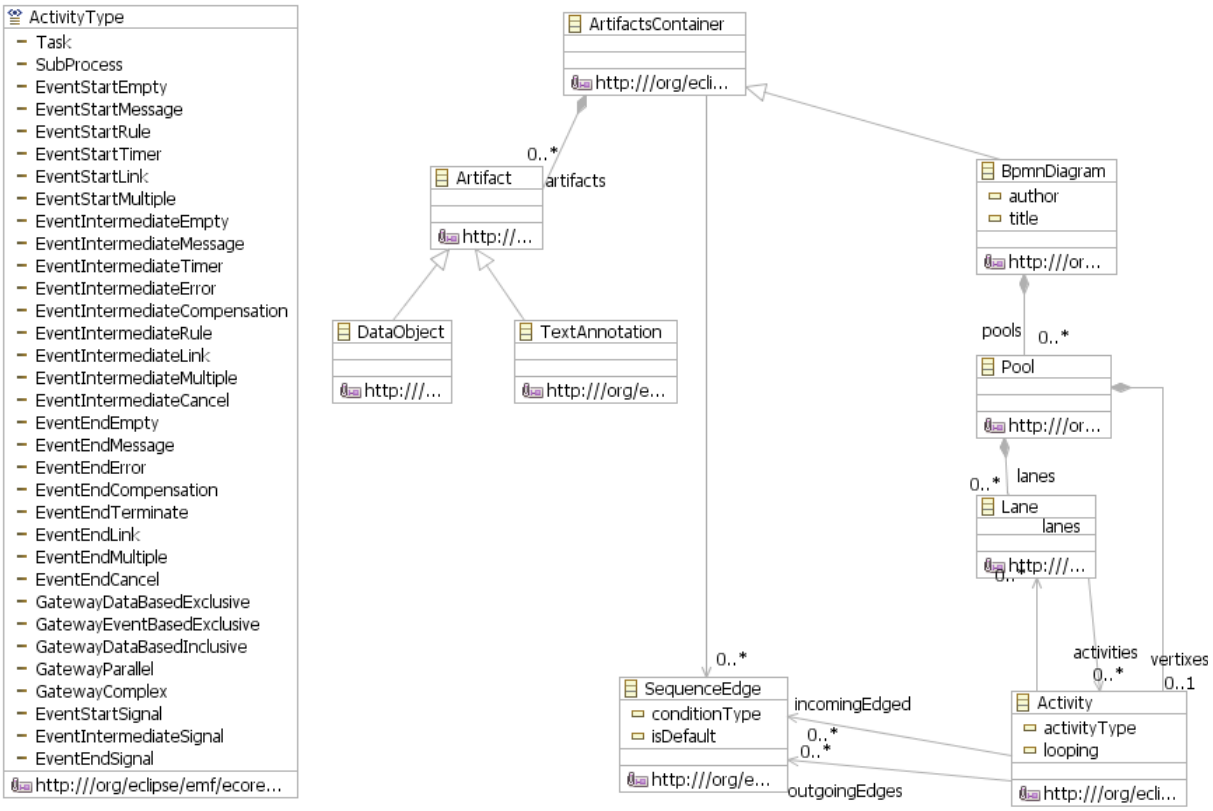

Figure 54: Simplified BPMN metamodel

- *Activity,* is used in the BPMN metamodel to represent the different Flow objects. The Activity type of the Activity determines the flow object that is used. Each Activity must be contained in a Pool and possibly a Lane and is connected to other Activities with Sequence edges. Activities are related to Data Objects and Text annotations using the artifact relation.
- *SequenceEdge,* is used in the BPMN metamodel to represent the sequence flow arrow.
- *DataObject, TextAnnotation, Pool* and *Lane* respectively represent the Data Object, Annotation and Swimlane concepts as described in section 3.2.1.

6 Transformation method

This chapter discusses the transformation method that we defined to transform a PA-model to a BPMN model. This transformation method is based on the following assumptions about the business rule specification:

- The specification is complete, i.e., if, after the transformation, rules are added to the business rule specification we cannot ensure that the generated business process still complies with the specification.
- The specification is consistent, i.e., it contains no contradictory statements. For example, the rules given below are contradictory rules

```
If CR.requester.status == 'good' then CR.status = ’approved’
And
If CR.requester.status == 'good' then CR.status = ’denied’
```

We also decided to make a semantic change to the PA-notation related to the ‘condition’ part of if expressions. In the original semantics of the PA-notation expressions mentioned in the ‘condition’ part of if expressions must be evaluated in an active way, e.g., if the expression is not true, try to make it true. Following the original semantics of the PA-notation, the example given below would lead to a valid process model.

```
The following applies:
if “Credit CR is requested” and then “Credit is approved”
```

Listing 17: Example PA-specification following original semantics of the PA-notation

This is however, to our knowledge, not common practice in business rule specifications. Therefore, we decided not to threat expressions in the ‘condition’ part in an active way, e.g., in order for an expression in the ‘condition’ part to be true it must be made true by other business rules. Listing 18 shows how the example of Listing 17 is expressed following our semantics of the PA-notation.

```
The following applies:
“Credit CR is requested” and
(if “Credit CR is requested” then “Credit is approved”)
```

Listing 18: Example PA-specification following our semantics of the PA-notation

In order to understand how we can transform business rules to business processes, we use a reverse engineering approach by first relating business processes to business rules. Based on these relations and the work presented in the previous chapters we defined the transformation method as described in the following sections.

6.1 Relating control flow patterns to business rules

This section relates a selection of control flow patterns to business rules. For the control flow patterns, the classification of (Börger, 2007) is used, in which the patterns by (Russel, Hofstede, Aalst, & Mulya, 2006) were analyzed, providing a more general classification of the patterns. Each of the sections below:

- Describes the pattern.
- Gives a graphical representation of the patterns in BPMN, following the BPMN semantics as described in (Wynn, Verbeek, Aalst, Hofstede, & Edmond, 2007).
- Identifies which structure of PA-rules leads to the discussed pattern.
- Gives an example of a PA-specification that includes this pattern. The PA-specifications are later used as an input to test our transformation method.

The term 'result' is used below in both the 'condition' part and 'then' and 'else' part of business rules. If the word 'result' is used in the 'condition' part then it actually means that an activity has produced some result or an outcome of an activity has some value (in the PA-notation, CR.amount <500). If the term 'result' is used in the 'then' or 'else' part it actually denotes a "trigger activity" (in PA-terms produce result). The term 'needs' is also used below. This is not a syntax element of the PA-notation but a simple way to express certain preconditions.

Here we should also mention that there are often two ways two represent a certain pattern. (1) Through explicit trigger relations and (2) via implicit preconditions relations. Although the PA-specification examples used in this section and throughout this thesis might suggest otherwise it is a best practice, in PA-specifications, to represent a certain dependency with precondition relations as much as possible.

6.1.1 Sequence

Description: An activity in a process is enabled by the completion of a preceding activity. This pattern can be seen in Figure 55.

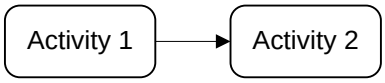

Figure 55: BPMN representation of the sequence pattern

The following rule leads to this pattern:

- *If* Result 1 *then* Result 2*,* with the constraint that there are no other rules with Result 1 in their condition part.

This pattern can also be expressed using precondition rules in the following way:

- Result 1 *and* Result 2
- Result 2 *needs* Result 1 (or vice versa)

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
"Credit CR is requested"
and (if "Credit CR is requested" then "Credit is approved")
and (if "Credit is approved" then "Interest is determined")

"Credit CR is requested =
```

```
       one CR exists in CREDITREQUESTs with:
              amount = 500
;
"Credit is approved" =
        CR.status = 'approved'
;

"Interest is determined" =
       CR.interest = '5'
;
CREDITREQUESTs =
       amount:int
       interest:int
       status:String
```

In this example first a credit request must be made, if the credit request has been made then the credit request must be approved. If the credit request is approved then the interest rate must be determined.

6.1.2 Parallel split

Description: Two or more activities in a process are enabled concurrently by the completion of a preceding activity. This pattern can be seen in Figure 56.

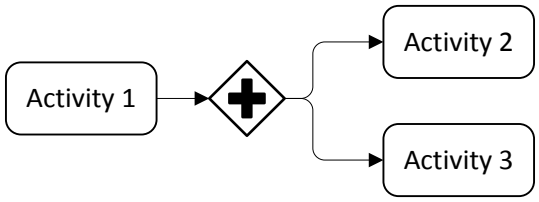

Figure 56: BPMN representation of the Parallel split pattern

The following rule leads to this pattern:

- *If* Result 1 *then* Result 2 *and* Result 3*,* with the constraints that there are no other rules with Result1 in their condition part and that Result 2 and Result 3 are mutually exclusive.

Another way to express this pattern is to use two separate rules:

1. *If* Result 1 *then* Result 2.
2. *if* Result 1 *then* Result 3.

This pattern can also be expressed using precondition rules in the following way:

- Result1 *and* Result2 *and* Result3
- Result 2 *needs* Result 1
- Result 3 *needs* Result 1

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
"Credit CR is requested" and
if "Credit CR is requested" then "Credit is approved" and "Interest is
determined"


"Credit CR is requested" =
       one CR exists in CREDITREQUESTs with:
              amount = 500
;
"Credit is approved" =
        CR.status = 'approved'
;
"Interest is determined" =
        CR.interest = 5
;


CREDITREQUESTs =
       amount:int
       status:String
       interest:int
```

In this example first a credit request must be made. If a credit request has been made then the credit request can be approved and the interest can be determined.

6.1.3 Merge

Description: An activity in a process is enabled by each completion of several other preceding activities. This pattern can be seen in Figure 57.

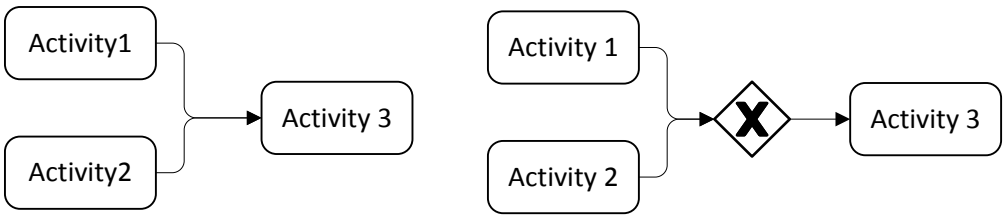

Figure 57: BPMN representations of the merge pattern

The following combination of rules leads to this pattern:

- *If* Result 1 *then* Result 3.
- *If* Result 2 *then* Result 3.

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
“Credit CR is requested” and
(if “Credit CR is requested” then “Status is determined” and “Interest is determined”)
and
(if “Status is determined” then “Value is checked”)
and
(if “Interest is determined” then “Value is checked”)

“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount = 500
;

“Value is checked” =
       CR.valueschecked = Input from ‘MANAGER’
;

“Interest is determined” =
       CR.interest = 5
;

“Status is determined” =
       CR.requesters.status = 'medium'
;

CREDITREQUESTs =
       requesters:REQUESTERs(1)
       amount:int
       valueschecked:Boolean
       interest:int

REQUESTERs =
status: String
```

In this example first a credit request must be made. If this is done then the status of the requester can be determined and the interest can be determined. Both the completion of those activities requires that a manager checks the values that have been entered.

6.1.4 Synchronization

Description: An activity in a process is enabled only after all preceding activities are completed. This pattern can be seen in Figure 58.

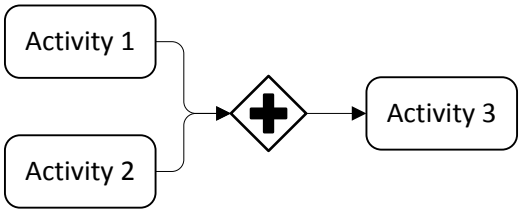

Figure 58: BPMN representation of the Synchronization pattern

One rule leads to this pattern:

- *if* Result 1 *and* Result 2 *then* Result 3 with the restrictions that are no other rules that have Result 1 or Result 2 in their condition part and that Result 1 and Result 2 are mutually exclusive.

This pattern can also be expressed using precondition in the following way:

- Result 1 *and* Result 2 *and* Result 3
- Result 3 *needs* Result 1
- Result 3 *needs* Result 2

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
“Credit CR is requested” and “Interest is determined” and
(if “Credit CR is requested” and “Interest is determined” then “Credit is approved”)

“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount = 500
;
“Credit is approved” =
        CR.status = 'approved'
;
“Interest is determined” =
       one R exists in INTERESTs with:
              interest=4
;

CREDITREQUESTs =
       amount:int
```

```
       status:String
       interest:int

INTERESTs =
       interest:int
```

In this example first a credit request must be made and the interest must be determined. If both are done then the credit must be approved.

A variation of this pattern is the Synchronization merge. In this case all active incoming paths must be enabled in order for the process to continue to the subsequent activity. With the language constructs available in the PA-notation this behavior cannot be expressed. A possible solution would be to add a new language construct, next to 'or' and 'and', to the PA-notation.

6.1.5 Exclusive choice

Description: The completion of an activity enables one of several possible other activities based on some condition related to the activity. This pattern can be seen in Figure 59.

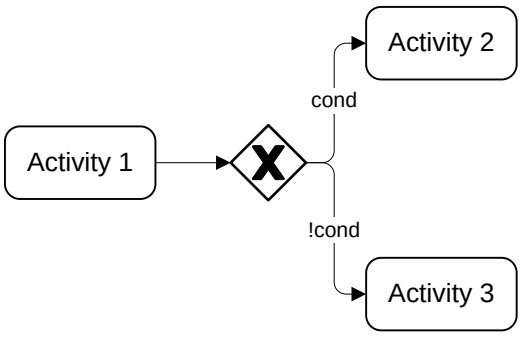

Figure 59: BPMN representation of the Exclusive split pattern

The following rule leads to this pattern:

- If Result 1 then Result 2 *else* Result 3.

Another way to express this pattern is to use two separate rules:

- If Result 1 then Result 3.
- If Result 2 then Result 4.

The constraints that must hold when using two separate rules is that both Result 1 and Result 2 can be evaluated after at most one activity, that they are mutually exclusive and that there are no other activities that are enabled based on the same results.

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
"Credit CR is requested" and
if "Credit CR is requested" then
```

```
(if CR.amount>500 then “Credit is denied” else “Credit is approved”)

“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount = 500
;
“Credit is approved” =
        CR.status = 'approved'
;
“Credit is denied” =
       CR.status = 'denied'

;

CREDITREQUESTs =
       amount:int
       status:String

REQUESTERs =
       status: String
```

In this example first a credit request must be made. If this is done then the credit is either approved or denied based on the credit amount

6.1.6 Multiple choice

Description: The completion of an activity enables one or more of several possible other activities based on some conditions. This pattern can be seen in Figure 60.

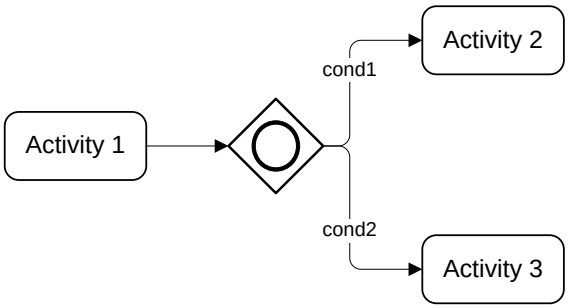

Figure 60: BPMN representation of the multiple choice pattern

The following combinations of rules lead to this pattern:

- If Result 1 *then* Result 3.
- If Result 2 *then* Result 4.

The constraint that must hold is that the moment that Result 1 and Result 2 can be evaluated is the same.

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
“Credit CR is requested” and
(if “Credit CR is requested” then
 (if CR.amount < 500 then “Credit is approved”) and
 (if CR.requesters.status == 'good' then “Interest is determined”))

“Credit is approved” =
        CR.status = 'approved'
;
“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount= input from 'REQUESTER'
              requesters = input from 'REQUESTER' chosen from REQUESTERs
;
“Interest is determined” =
       CR.interest = input from 'MANAGER'
;
CREDITREQUESTs =
       requesters:REQUESTERs(1)
       amount:int
       status:string
       interest:int

REQUESTERs =
       status: string
```

In this example first a credit request must be made. If this is done then the credit is approved if the credit amount is lower than 500 and the interest is determined if the status of the requester is good.

6.1.7 Discriminator

Description: An activity in a process is enabled by the completion of just one of several preceding activities. After the first preceding activity is completed the completion of other activities is ignored. This pattern can be seen in Figure 61.

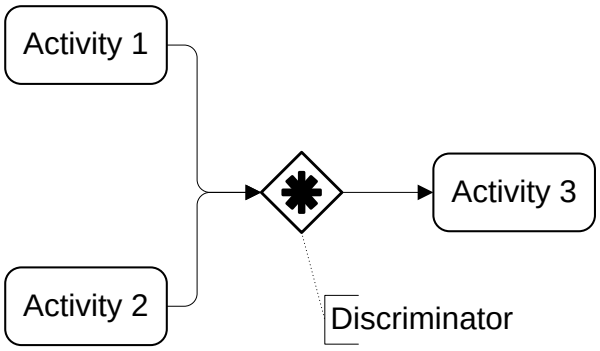

Figure 61: BPMN representation of the Discriminator pattern

BPMN does not have a standard symbol for the discriminator. We solved this by using a complex gateway combined with an annotation.

The following rule leads to this pattern:

- if Result 1 or Result 2 then Result 3 with the restrictions that are no other rules that have Result 1 or Result 2 in their condition part and that Result 1 and Result 2 are mutually exclusive.

An example of a PA-specification that includes this pattern is given below:

```
The following applies:
"Credit CR is requested"
and
(if "Credit CR is requested" then "Credit is judged" and "Interest is determined")
and
(if "Credit is judged" or "Interest is determined" then "Credit is approved")

"Credit is approved" =
       CR.status = 'approved'
;
"Credit CR is requested" =
       one CR exists in CREDITREQUESTs with:
              amount = 500
;
"Interest is determined" =
        CR.interest = 5
;
"Credit is judged" =
       CR.status = 'denied'
;
```

```
CREDITREQUESTs =
        amount:int
        status:String
        interest:int
```

In this example first a credit request must be made. After this the credit request can be judged and the interest can be determined. After the first of these activities completes then the credit must be approved.

6.1.8 Cycles

Description: A point in a workflow process where one or more activities can be done repeatedly. There are three types of cycles, (1) structured cycles that have one entry and exit point, (2) arbitrary cycles that have several entry or exit points and (3) recursion where an activity triggers itself. These patterns can be seen in Figure 62.

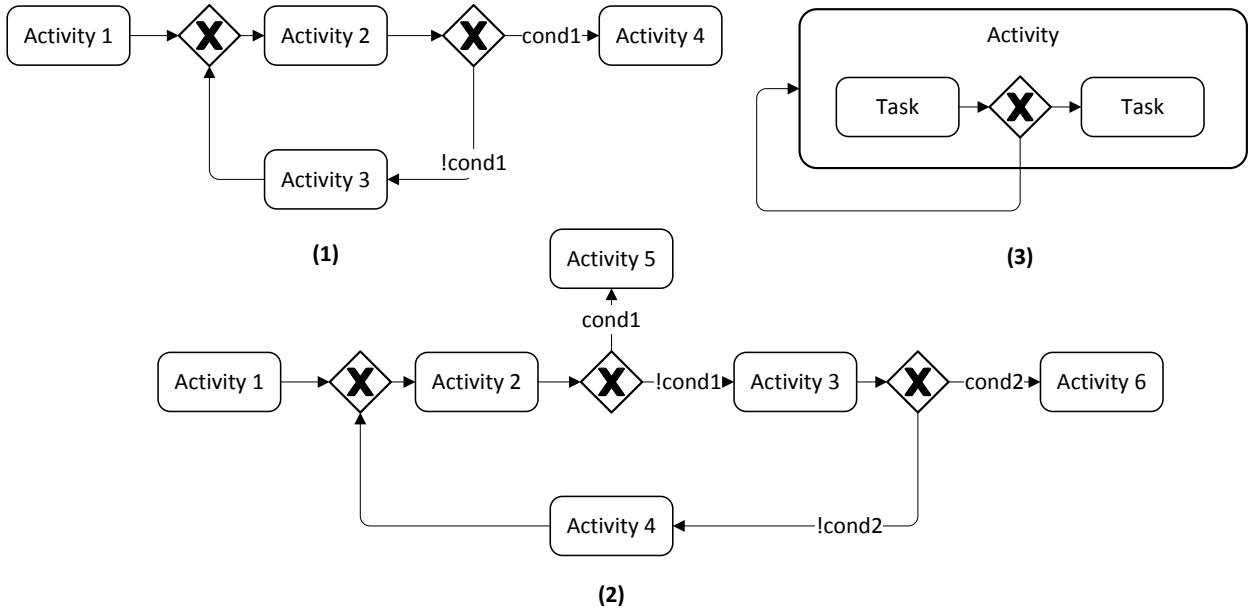

Figure 62: BPMN representation of (1) the structured cycle, (2) the arbitrary cycle and (3) the recursion patterns

The general rule for an Activity x to be in a loop is that there is a subsequent Activity y, that can only be enabled if Activity x is completed and that triggers Activity x.

An example of a set of rules that would lead to a structured cycle is:

- *If* Result 1 *then* Result 2.
- *If* Result 3 *then* Result 1 *else* Result 4.

Here Result 3 must be some condition that can be evaluated based on Result 2.

An example of a PA-specification that represents this pattern is given below:

```
The following applies:
```

```
“Credit CR is requested” and
(if “Credit CR is requested” then “Credit request evaluation”) and
(if “Credit request evaluation” then (if CR.status == 'More Arguments needed' then “More info” else “Assign interest”))
and
(if “More info” then “Credit request evaluation”)



“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount = input from 'REQUESTER'
              requesters = input from 'REQUESTER' chosen from REQUESTERs
;

“Credit request evaluation” =
        CR.status = input from 'manager'
;

“Assign interest” =
        CR.interest = input from 'boss'
;

“More info” =
       CR.moreinfo = input from 'USER'
;

CREDITREQUESTs=
       requesters:REQUESTERs(1)
       amount:int
       status:String
       interest:int
       moreinfo:String

REQUESTERs =
       status: String
```

In this example first a credit request must be made. If this is done then the credit request must be evaluated. If the result of the evaluation is that the manager needs more info then the requester is asked to give more information. In all other casus the interest must be determined. If the user has given more info then the credit request must be evaluated again.

6.2 From PA model to dependencies model

This section discusses the transformation from a PA-model (PAM) to a dependencies process model (DPM). This transformation starts with processing the Proposition. A Proposition can either hold an IfExpr, ExistsExpr or an AndExpr. If an AndExpr is used in a Proposition it can only hold expressions of the type ExistsExpr, AssignExpr or IfExpr. We discuss how these elements are transformed in the next sections.

6.2.1 ExistsExpr

Figure 63 shows how an ExistsExpr is mapped to an ExistsActivity. The name and sentenceName of the ExistActivity are the same as in the ExistsExpr, the actor field is not filled in at this point. Each Feature that is updated by the ExistsExpr is mapped to a variable contained in an Update. The mapping of the Expr that determines the value of the variable depends on the type of the expression:

- If the Expr is an InputExpr then the InputExpr is mapped to an inputActivity, contained in the act attribute.
- If the Expr is a VariableFeatureRefExpr then the VariableFeatureRefExpr is mapped to a variable, contained in Precondition.
- In all other cases the Expr is mapped to a String and stored in the with attribute.

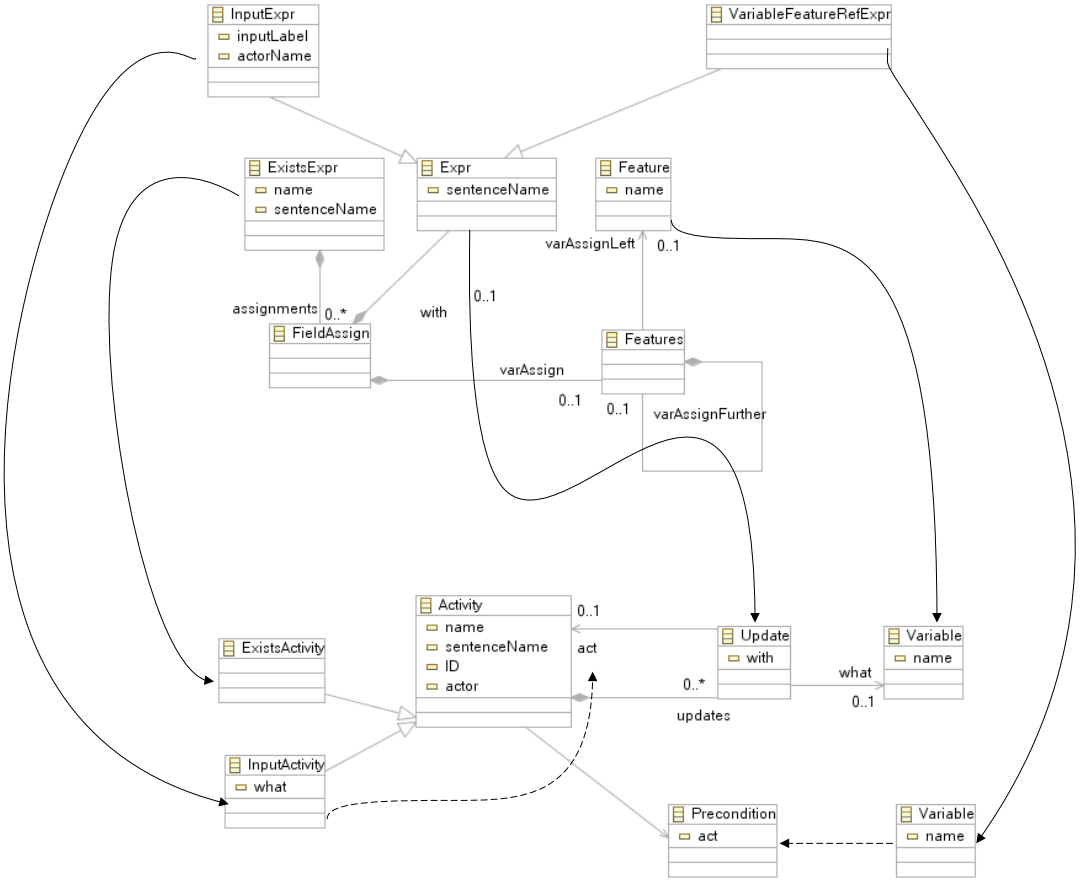

Figure 63: Basic mapping between ExistsExpr and ExistsActivity, the arrows represent direct transformation and the dotted arrows represent relations that are created.

To show an example of transformation using the above mapping, consider the following PA-specification:

```
"Credit is requested" =
      one CR exists in CREDITREQUESTs with:
            amount = input from 'REQUESTER'
            status = 'good'

CREDITREQUESTs =
      amount:int
      status:string
```

This specification is represented as a PAM as shown in Figure 64.

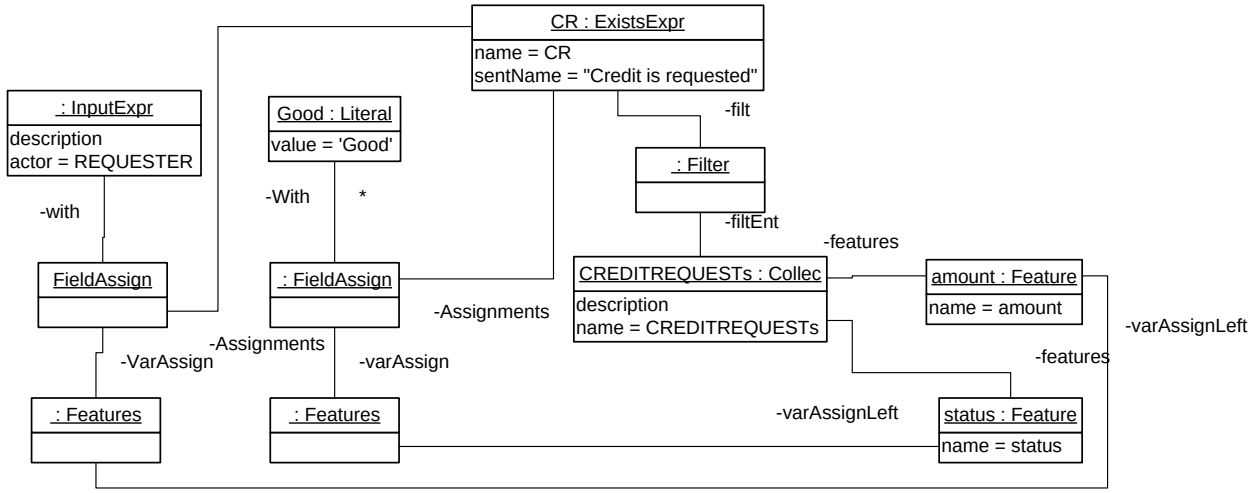

Figure 64: Example PAM of ExistsExpr

Figure 65 shows the result of the transformation of the above PAM to a DPM.

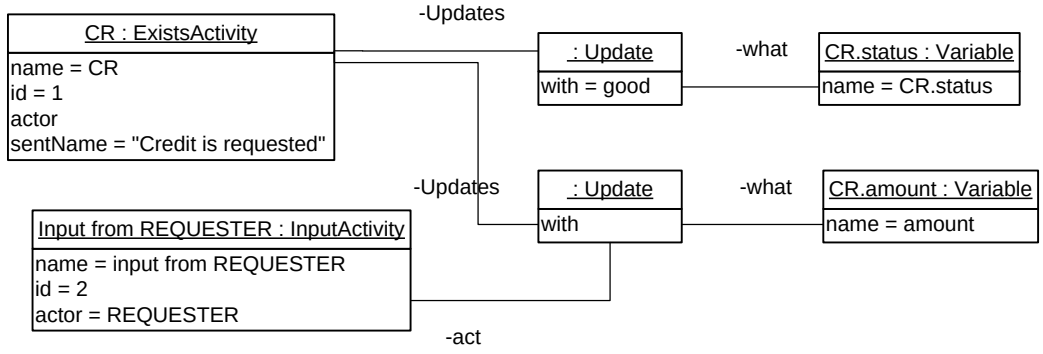

Figure 65: Example DPM of ExistsActivity

6.2.2 AssignExpression

Figure 66 shows how an AssignExpr is mapped to an AssignActivity. The name of the AssignActivity is a combination of the word “assign” and the variable that it updates, the sentenceName of the AssignActivity is the same as the sentenceName of the AssignExpr, the actor field is not filled in at this point. The feature updated by the AssignExpr (stored in VariableFeatureRefExpr) is mapped to a Variable. If the AssignExpr refers to an ExistsExpr (through a VariableRefExpr) then the ExistsExpr is mapped to an ExistsActivity contained in a Precondition of the AssignActivity. The mapping of the Expr that determines the value of the Variable is the same as for an ExistExpr.

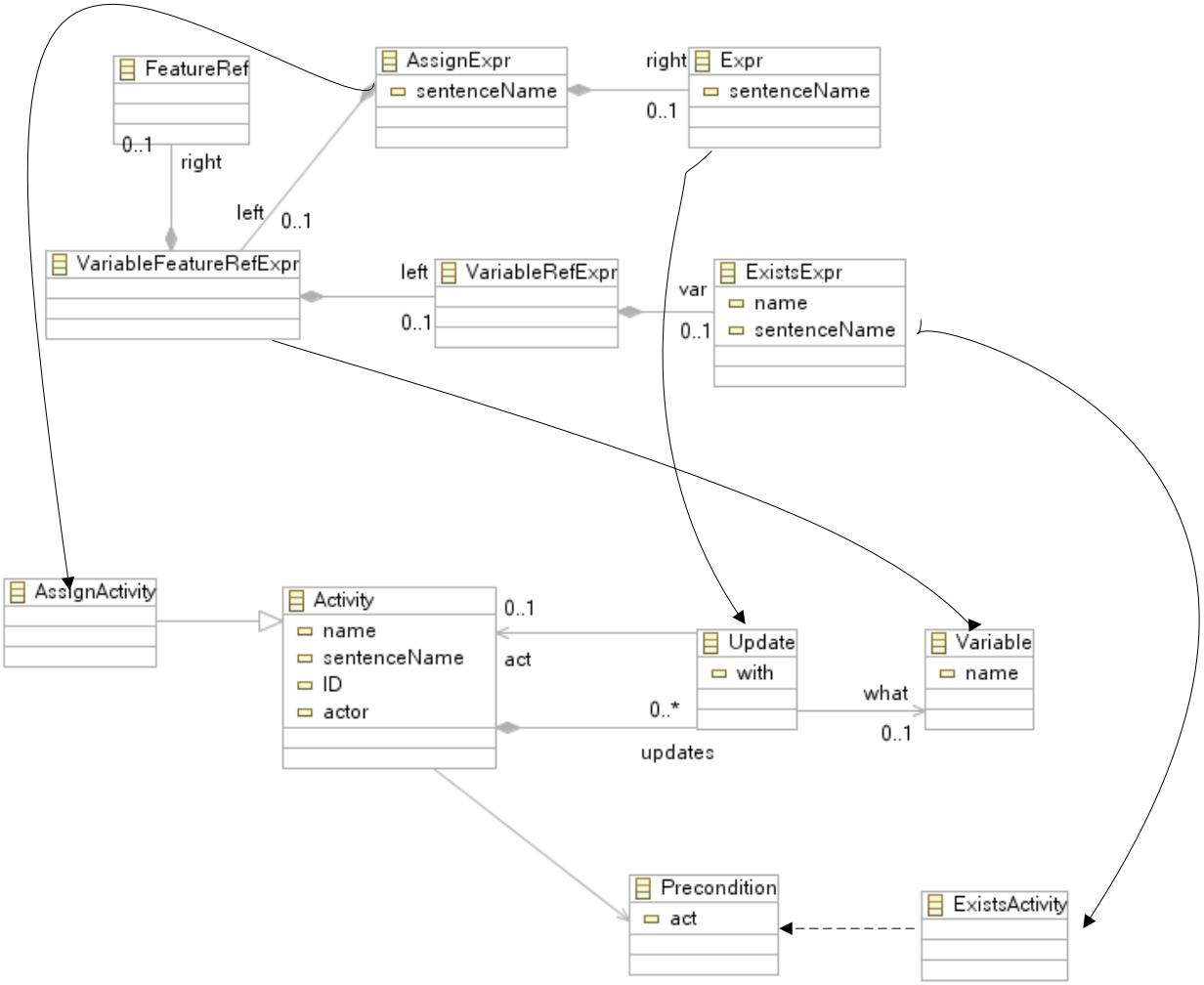

Figure 66: Basic mapping between AssignExpr and AssignActivity, the arrows represent direct transformation and the dotted arrows represent relations that are created

To show an example of transformation using the above mapping, consider the following PA-specification:

```
 “Assign Status” =
       CR.status = ’Medium’
;
CREDITREQUESTs =
```

```
        amount:int

        status:string
```

This specification is represented as a PAM as shown in Figure 67

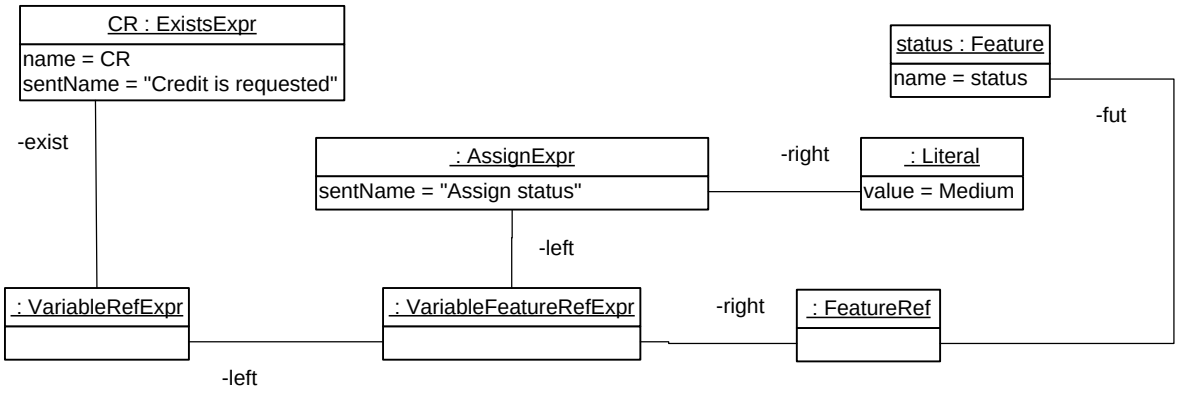

Figure 67: Example PAM of AssignExpr

Figure 68 shows the result of the transformation of the above PAM to a DPM.

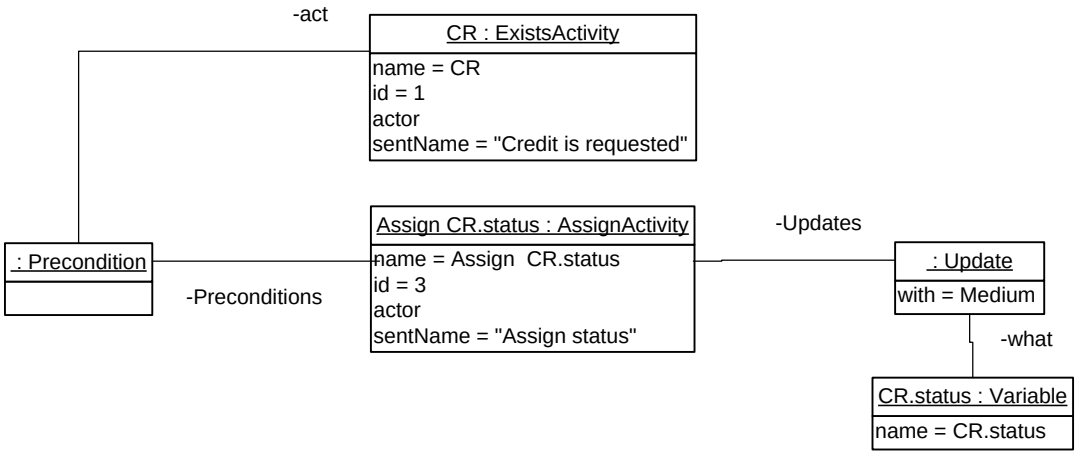

Figure 68: Example DPM of Assign Activity

6.2.3 IfExpr

If Expressions determine the trigger relations between activities. An IfExpr consists of three other Expressions: (1) the condition (2) the thenExpr and (3) the ElseExpr. Figure 69 shows how the different elements in an IfExpr are mapped to elements of the PMM.

The condition part is mapped to a Trigger. The expression type of the expression determines the Relation type of the trigger. If the type is And then the relation type is And, if the type is Or then the relation type is Disc and in all other cases the Relation type is Single. And and Or Relations itself consist of Relations that can be of any type. However, at the lowest level all Relations are of the type Single.

The trigger created by the condition is added to all Activities created in the thenExpr. If the IfExpr has an ElseExpr then an Empty trigger is created where the Boolean IsElse is set to true. This trigger is then added to all activities that are created in the ElseExpr.

If the ThenExpr or ElseExpr contains yet another IfExpr then the trigger created by that ifExpr is set as a NextLevel trigger of the current trigger. For example in the following rules:

```
if "Credit CR is requested" then
(if CR.amount>500 then "Credit is denied"
```

The first trigger is "Credit is requested" and the NextLevel trigger is CR.amount>500. This construction is used to capture that the first trigger has to be true before another trigger becomes relevant.

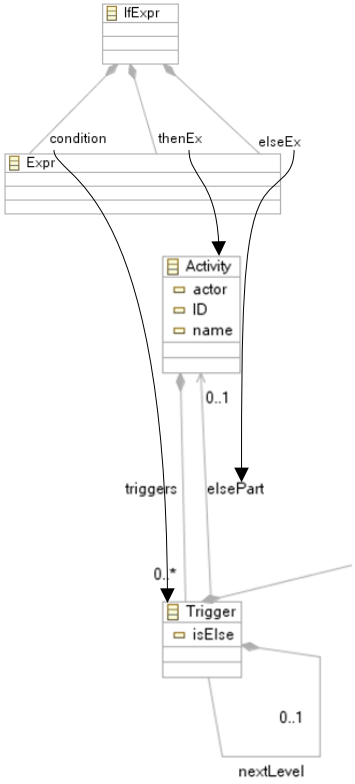

Figure 69: Basic mapping between IfExpr and Process dependencies

To show an example of transformation using the above mapping, consider the following PA-specification:

```
if "Credit CR is requested" then "Assign Status"
```

This specification is represented as a PAM as shown in Figure 70

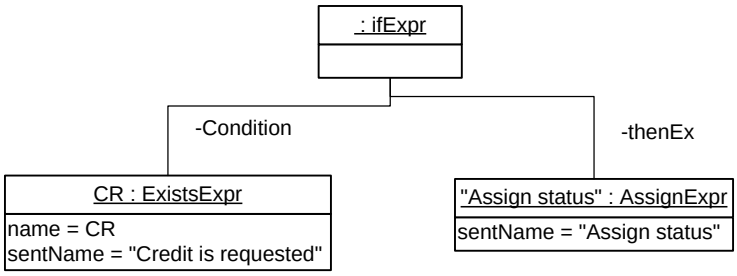

Figure 70: Example PAM of IfExpr

Figure 71 shows the result of the transformation of the above PAM to a DPM.

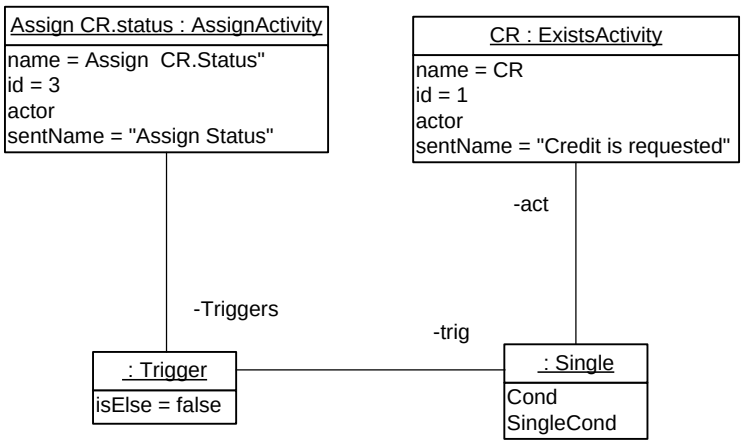

Figure 71: Example DPM of Trigger relation

6.3 Optimize dependencies model

The optimization of the dependencies process model (DPM) to the optimized process model (OPM) is performed in two steps. First, the optimal sequence of activities is determined and second the optimal allocation of resources is determined.

6.3.1 Determining optimal sequence of activities

In order to determine the sequence of activities we first discuss what we consider to be the optimal sequence of activities. As we do not know the knockout ratio of the different activities, we can only influence the time optimization by the sequence of activities. For example, the following rules:

```
The following applies:
"Credit CR is requested" and
if "Credit CR is requested" then "Credit is approved" and "Interest is
determined"
```

Could be transformed to the three process models as shown in Figure 72.

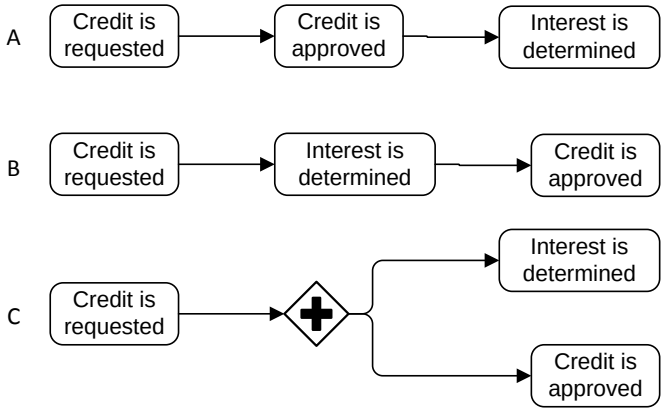

Figure 72: Example process models

All three process models comply with the rules, however, only alternative C is optimal in terms of time. Therefore we define the optimal sequence of activities as the sequence where each activity is enabled as soon as it can be enabled. Here the best practice of parallelism is used (Reijers & Mansar, 2005).

Figure 73 shows the method we have defined to determine the optimal sequence of activities. Each step is further explained below.

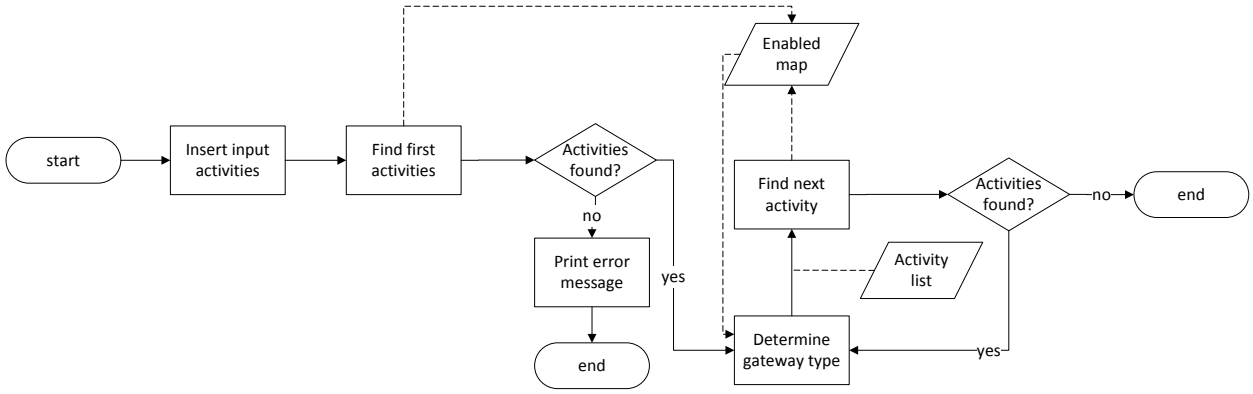

Figure 73: Method to determine the optimal sequence of activities

Insert input activities

The first step in determining the sequence of activities is to insert the input activities in the DPM. In this step each update of each assign and exists activity is searched for input activities. If an input activity is found then the triggers and preconditions of the activity it belongs to are attached to the input activity and the input activity is set as a precondition for that activity. In the case of an exist activity multiple input activities can be found. If these input activities have the same actor attribute they are combined into one input activity. To show an example of this step, consider the following PA-specification:

```
if "Credit CR is requested" then "Assign Interest"

"Credit CR is requested" =
       Amount = input from 'REQUESTER'
       Status = input from 'REQUESTER' chosen from ['good', 'medium']

"Assign Interest" =
       CR.interest = input from 'Manager'
```

This specification is transformed to the DPM as shown in Figure 74.

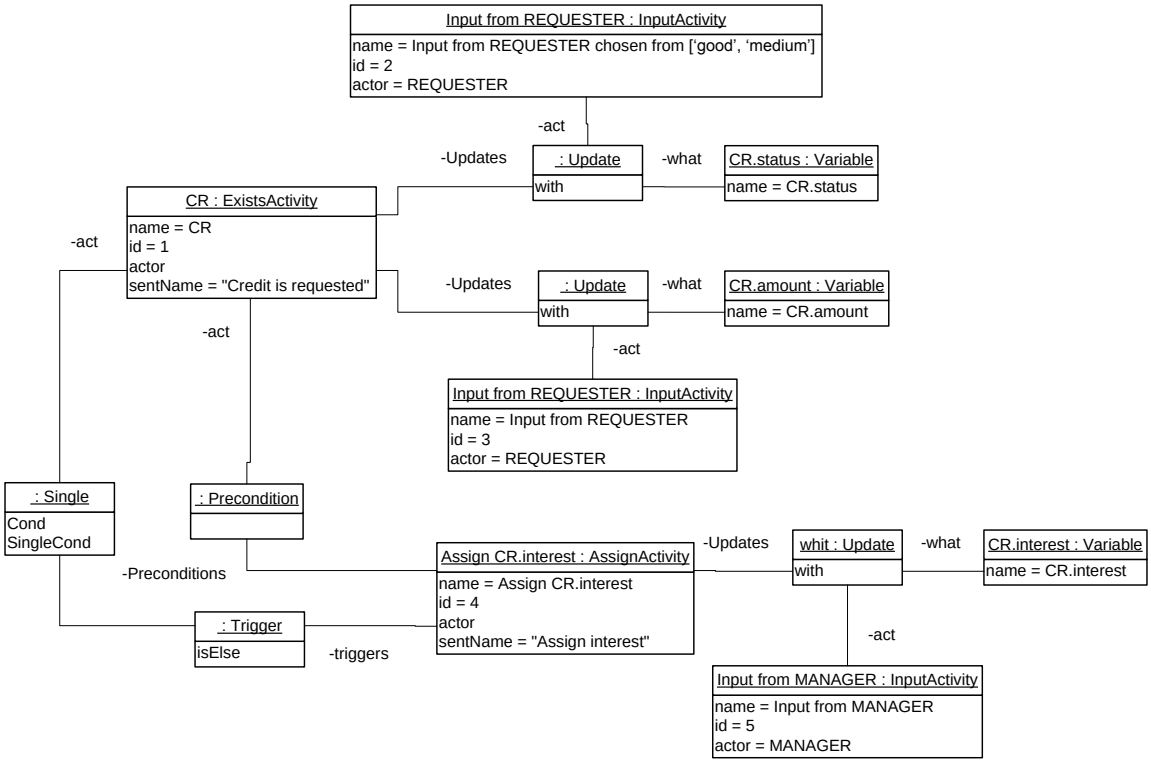

Figure 74: DPM example

Figure 75 shows the result of performing the insert input activities step for the DPM of Figure 74.

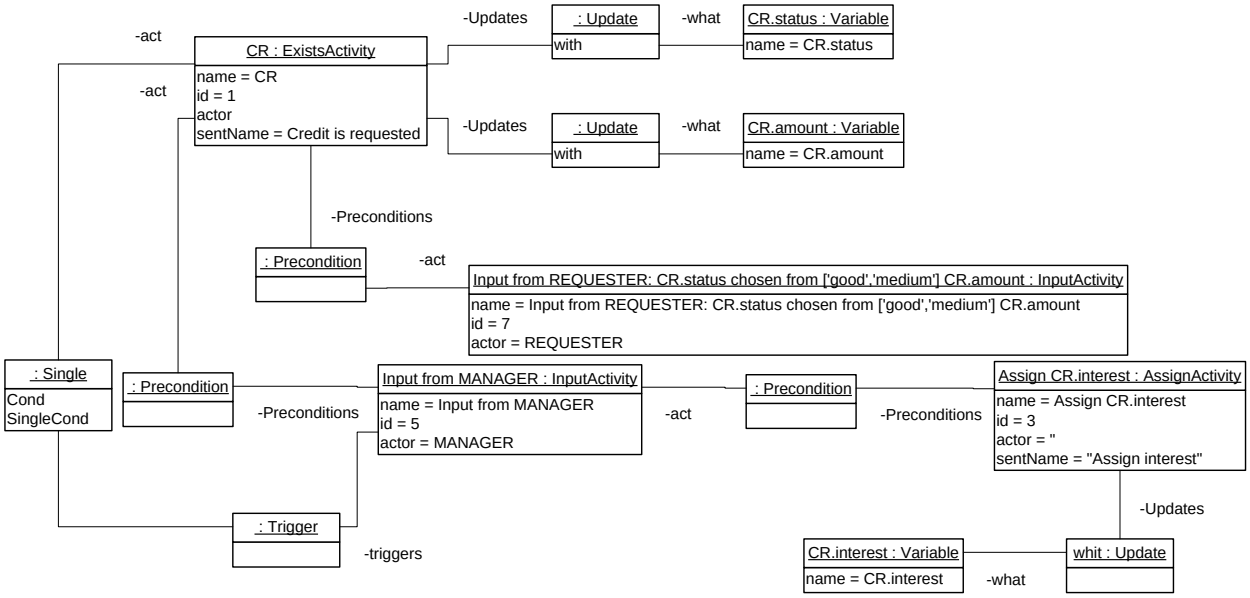

Figure 75: Result of the insert input activities step given DPM of Figure 74.

Find first activities

In this step all activities are checked if they are a first activity. An activity is a first activity if one of the following conditions is true:

- The activity has no triggers and no preconditions.
- The activity has no preconditions and all attached triggers do not need other activities in order to be evaluated.

This step together with the determine gateway type step can have three different results:

1. No first activity is found, which means there is no mapping to a business process possible. In this case an error message is printed and the transformation process is ended.
2. One first activity if found. In this case a dummy first activity is created which is the starting point of the process, followed by a SeqGateway that is connected to the found activity.
3. Several firsts' activities are found. In this case a dummy first activity is created which is the starting point of the process, followed by a SplitGateway of the type AND that is connected to each found activity. If one of the found activities has a cond or SingleCond related to it then the split is of type OR.

Find next activities

In this recursive step all activities that can be enabled based on a single activity and all previously enabled activities are found. Figure 76 shows the method to find these activities that is further explained below.

The first aspect that is determined is if it is necessary that the source activity (the activity that previously is enabled) is checked. It is not necessary to check a source activity if:

- The source activity already is in the diagram, i.e., it triggered other activities before and since the last check nothing has changed. If something has changed it means that the source activity can trigger another activity which has not been triggered before, so it must be checked again.

If the source activity has to be checked then, for each (target) activity, it is determined if it can be enabled. A target activity is enabled by the source activity if:

- The source activity makes all preconditions of the target activity fulfilled and the target activity has no unmet triggers.
- All preconditions of the target activity are met and the source activity aids in enabling one of the target activity triggers.

The result of this process is a map containing all enabled activities, and activities needed to enable these enabled activities. Listing 19 shows the structure of the resulting map.

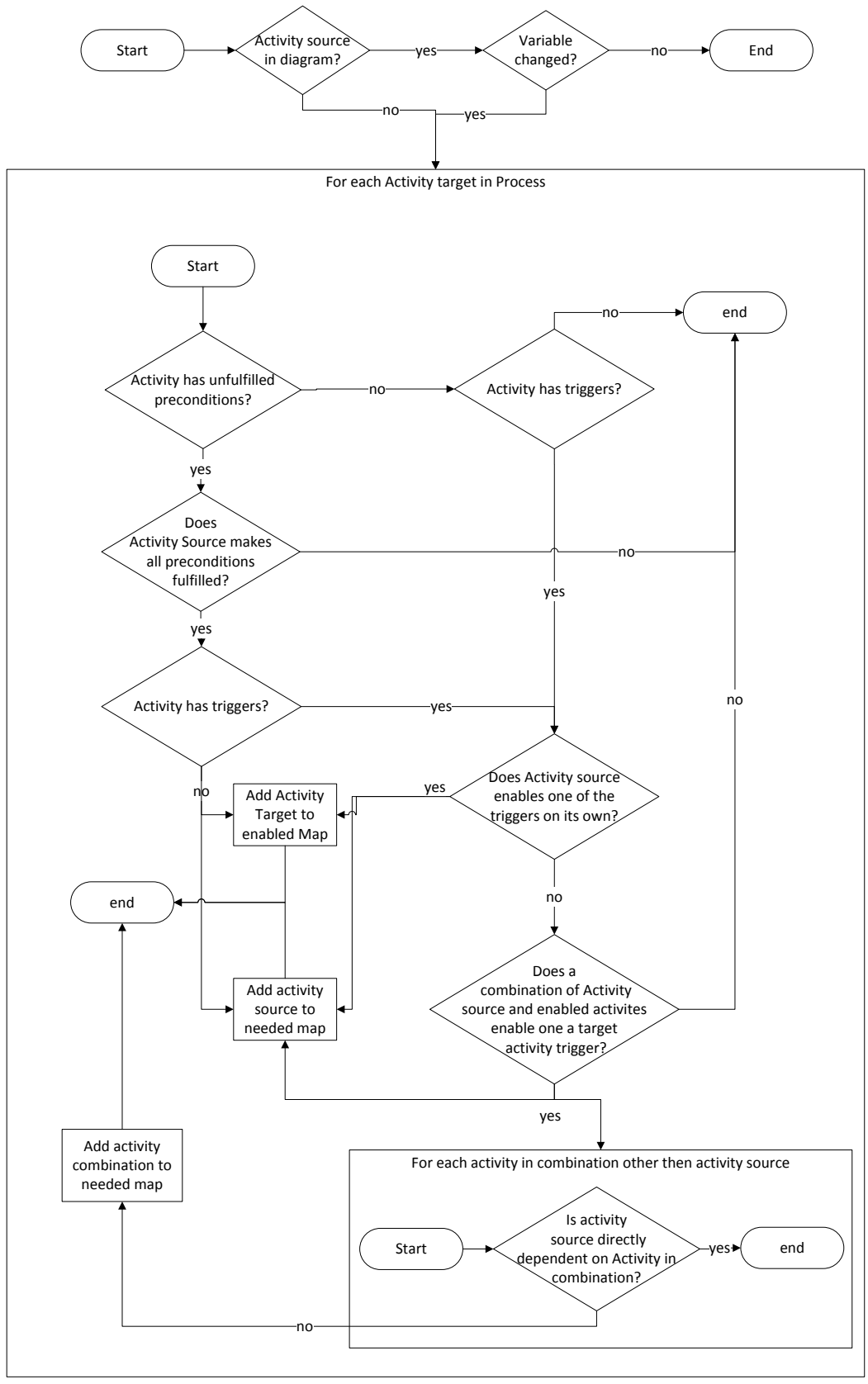

Figure 76: General method of finding new activities

```
<enabled map>
<ActivityTargetEnabled>
       <needed map type=’relation type’>
              <Activity source>
                     <cond>
              <Other Activity>
                     <cond>
              etc
       </needed map >
< ActivityTargetEnabled>
       <needed map type=’relation type’>
              <Activity source>
                     <cond>
              <Other Activity>
                     <cond>
              etc
       </needed map >
Etc.
</enabled map>
```

Listing 19: Enabled map structure

Determine gateway type

If the activity source enables other activities then we determine which gateway type must be used to connect the activities to each other.

Four situations can be distinguished:

1. One source activity enables one target activity;

   In this case the source activity and the target activity are connected to each other using a sequence gateway.

2. One source activity enables several target activities;

   In this case the source activity and the target activities are connected to each other using a split gateway. The gateway type is determined by the conditions that are in the needed map.

   - If all conditions are the same then the gateway type is AND.
   - If there are only two target activities and one condition is else (indicating that the trigger was an else trigger) and the other is not empty then the gateway type is XOR.
   - In all other cases the gateway type is OR.

3. Several source activities enable one target activity;

In this case the source activities and the target activities are connected to each other using a join gateway. The gateway type is the same as the trigger type of the needed map.

4. Several source activities enable several target activities.

This is a difficult situation that is, due to time constraints, not supported in our transformation method. In this case the source and target activities should be connected to each other using a combination of join and split gateways. The specific combination depends on which combinations of source activities precisely enable which combination of target activities under which conditions.

6.3.2 Determining optimal allocation of resources

Based on the sequence of activities determined in the previous step and a comma separated values(CSV) file, the optimal allocation of resources is determined. Listing 20 shows the structure of the CSV file.

```
Activity performed by resource, Resource Name, Time, Costs
1, System x, 6, 5
1, System y, 5, 7
```

Listing 20: Structure CSV file

The first column contains the ID of the activity that could be performed by the resource, the second column contains the name of the resource and the third and fourth column contains in which time the resource performs the activity against which costs. Due to time limitations we have chosen not to use a multi objective algorithm. Figure 77 shows the method that we have taken to find the optimal allocation of resources. This method is further explained below.

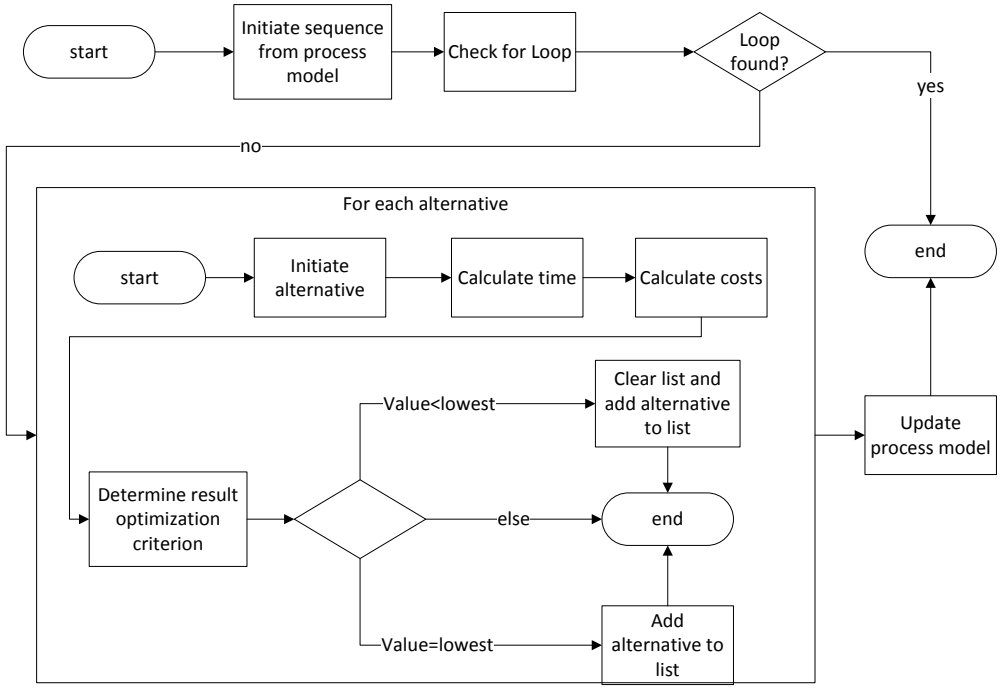

Figure 77: Method to determine allocation of resources

In the first step the Java classes, as can be found in Appendix F, are initiated based on the OPM. Then the process is checked if it contains any loops. If it contains loops then this step will end. If it does not contain loops then the CSV file is processed, which means that a list of alternatives is build and that the highest and lowest costs and time that are in CSV file are determined for each activity. The lowest and highest values are determined in order to be able to apply the weighted summation technique[3] (Herwijnen, 2006). This is a technique that can be used to make variables that have a different measurement unit and scale, comparable.

Then for each alternative in the list the completion time and total costs are calculated and compared to the, at that point, lowest value for the optimization criterion. Below we explain how the completion time and costs are calculated for one alternative.

Completion time

Determining the total completion time for one alternative is based on the approach of (Vergidis, Tiwari, & Majeed, 2006) and is done as follows:

- First, the first sequence element (an activity or a gateway) is notified that it can start calculating its end time. That element then calculates its own end time and notifies all directly connected outgoing sequence elements that they can start calculating there end time too.
- If a sequence elements receives such a call it calculates its end time in the following way:
    - For activities, Or, And, and XOR splits the end time is calculated as the end time of the sequence elements that sends the call+ its own completion time.
    - For Joins the end time is calculated as the highest incoming end time of all the sequence elements that are joined.
- When a last activity, which is defined as an activity that has no outgoing sequence elements, has updated its end time it sends its update to the main process.
- When the main process has received results of all last activities the total completion time is returned, which is the highest end time of all last activities.

Costs

The costs are calculated using a pull mechanism where each sequence elements calculates the costs as the costs of itself + the costs of all its outgoing sequence elements. For each sequence element the costs are calculated as follows:

- For an activity the costs are the costs of the activity itself + the costs of the outgoing sequence element.
- For Sequence, And join, XOR join and Disc elements the costs are the costs of the outgoing sequence element.
- For an And split the costs are the sum of the costs of all outgoing sequence elements.

[3] There are other techniques available, it is not the focus of this research to go into a detailed discussion about whether this technique is more suitable then another. We have chosen this particular technique because it is relatively simple to apply.

- For an Or split the costs are the sum of the costs of all outgoing sequence elements. In an Or split the process can continue down to one or more paths. Because we do not have information about the chance that a process continues on one or more paths and we consider worst case scenario we have to assume that the process continues on all possible paths. In terms of costs calculation the behavior of the And split and the Or split are thus exactly the same.
- For an XOR split the costs are the highest costs of all outgoing sequence elements.

Determining optimal allocation of resources

When the total costs (Tc) and completion time(Ct) of an alternative is determined then those values are made comparable using the weighted summation technique (Herwijnen, 2006) that uses the following formula:

$$comparablevalue = \frac{originalVdue - \min value}{\max value - \min value}$$

The value of the optimization criterion is then determined, using the comparable values for completion time and total costs (CvCt and CvTc), and is compared to the, at that point, lowest value of the optimization criterion:

$$(a * CvCt) + (b * CvTc) = \min$$
$$a + b = 1$$
$$0 \le a \le 1$$
$$0 \le b \le 1$$

At this moment a and b are both set on 0,5. All alternatives that have the lowest outcome for that formula are, at the end of the calculation, in the list of optimal alternatives. From this list the first alternative is chosen, and based on that alternative the actor field of all activities (in the OPM model) that have a resource attached to it is updated with the resource that performs that specific activity.

6.4 Transformation to BPMN model

The final step in our transformation method is to transform the OPM to a BPMN model. The mapping between our process metamodel and the BPMN metamodel is relatively simply:

- Sequence elements, except for sequence gateways, are mapped to BPMN activities. The type of sequence element determines the BPMN activityType.
    - Activities are mapped to the BPMN activityType task, and the actor of the activity is mapped to a BPMN text annotation containing the name of the actor. Ideally one would like to transform the actor to a BPMN lane. However, the Eclipse BPMN modeler does not handle a lane in such a way that the actor who performs the Activity directly becomes clear. In fact, the generated diagram becomes a bit unclear when one uses lanes in a BPMN model and generate a BPMN diagram on the basis of that model.
    - Gateway splits and joins of the type AND are mapped to the BPMN activityType GatewayParallel.

    - Gateway splits and joins of the type XOR are mapped to the BPMN activityType GatewayDataBasedExclusive.
    - Gateway splits of the type OR are mapped to the BPMN activityType GatewayDataBasedInclusive.
    - Gateway joins of the type Disc are mapped to the BPMN ActivityType ComplexGateway with an annotation attached to it with the text 'Discriminator'.
- Gateway Sequences are mapped to a BPMN edge.
- Each connection between sequence elements is mapped to a BPMN edge. The condition, if there is any, that belongs to the connection between two sequence elements is mapped to the name of the edge.

6.4.1 Verification

The BPMN model produced by our transformation method can be verified using the analysis available in the YAWL editor as described in (Wynn, Verbeek, Aalst, Hofstede, & Edmond, 2007) using the bpmn2yawl conversion tool as described in (Decker, Dijkman, Dumas, & García-Bañuelos, 2008).

In this analysis the process model is checked of having the following properties:

- Soundness.
- Weak soundness.
- Irreducible cancellation regions.
- Immutable OR-joins.

The latter three are properties that process models with OR-joins and cancellation regions should have. Our process models do not contain OR-joins and cancellation regions. Thus, the only property that is relevant for process models created by our transformation method is soundness.

A process model is sound if:

- The process always completes when started.
- The process does not have any other tasks still running for that process if the process is completed.
- The process does not contain tasks that will never be executed.

The bpmn2yawl conversion tool does not support the conversion of the Discriminator join to its YAWL counterpart. To solve this we decided to convert the Discriminator join to a XOR join. The difference between Discriminator joins and XOR joins is that, in XOR joins, all enabled incoming paths lead to the enabling of the subsequent path and. In a Discriminator join, only the first enabling incoming path leads to the enabling of the subsequent path. However, the correctness of the subsequent path does not depend on the amount of times (higher then 1) it is enabled, thus in terms of overall correctness there is no difference between using a discriminator join or an XOR join.

To give an example of the analysis, the figures below show an unsound BPMN diagram (an XOR Split is followed by an AND join, thus the process can never complete), its counterpart in YAWL and the result of the analysis. The highlighted row shows that the process model is unsound.

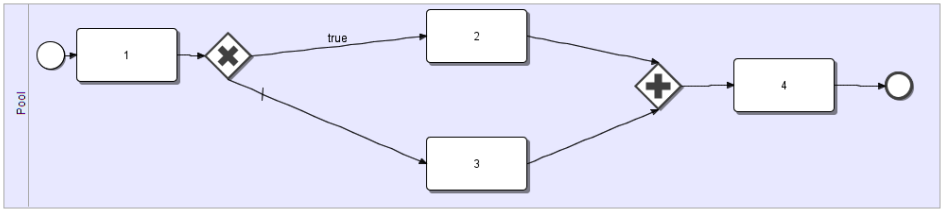

Figure 78: BPMN example of an unsound process model

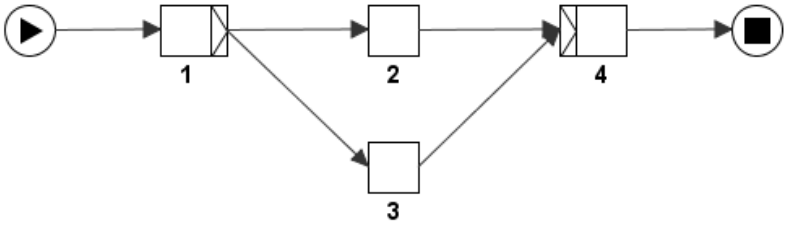

Figure 79: YAWL counterpart of BPMN model of Figure 78.

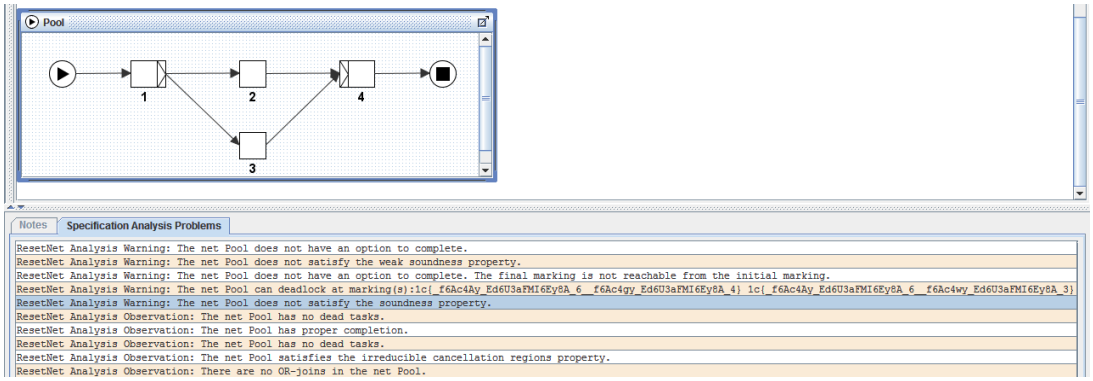

Figure 80: Result of analysis of YAWL model of Figure 79.

7 Verification

This chapter discusses the verification of the transformation method discussed in the previous chapter. First we describe how to perform a transformation using the developed prototype, followed by the verification of the transformation method using a case study.

This chapter is further structured as follows: Section 7.1 presents the prototype and section 7.2 discusses the verification of the transformation method.

7.1 Prototype

In order to run prototype several tools are needed. The tools used in this prototype are:

- openArchitectureWare. To be able to install openArchitectureWare one needs an existing Eclipse application, the version we used can be found at: http://www.eclipse.org/downloads/packages/release/ganymede/sr1[4]. openArchitectureWare can then be downloaded by using the following update site in Eclipse: http://www.openarchitectureware.org/updatesite/milestone/site.xml.
- The SOA tools BPMN modeler, which can be downloaded using the following update site in Eclipse: http://download.eclipse.org/stp/updates/site.xml[4].
- The BPMN2YAWL plugin, which can be downloaded at: http://sourceforge.net/projects/yawl/.
- The YAWL editor that can be downloaded at: http://www.yawl-system.com/.

This section explains how to perform a transformation and how to validate the transformation.

7.1.1 Step 1, creating a PA-specification

In step 1 one must create a PA-specification. This can be done using two methods:

1. One can use a plain textual editor such as notepad, write the specification and store it as a file with the file extension .pa. Figure 81 shows a credit request example we created.
2. One can use the PA-editor created by oAW. A PA-specification can be created by using file->new->other->Xtext DSL Wizards->Panotation project. The PA file can be found in the src folder of the created project. Figure 82 shows our example in the PA-editor.

[4] There are newer versions of Eclipse (sr2) and the bpmn modeler. We however do not use these versions because at this moment (15-04-2009) they are incompatible (https://bugs.eclipse.org/bugs/show_bug.cgi?id=266959)

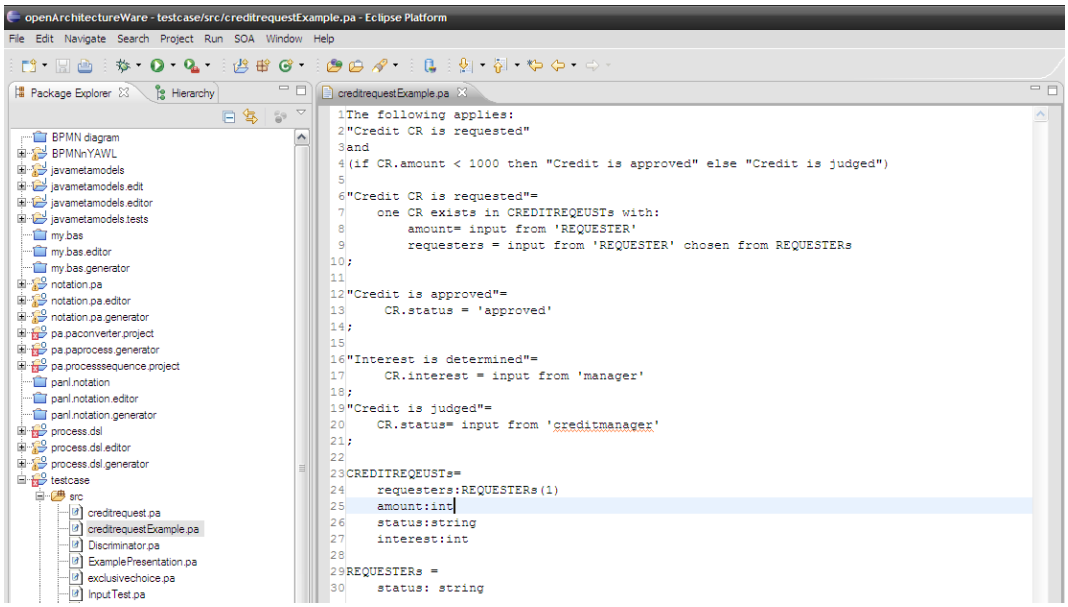

Figure 81: Prototype, creating a PA-specification using a plain textual editor

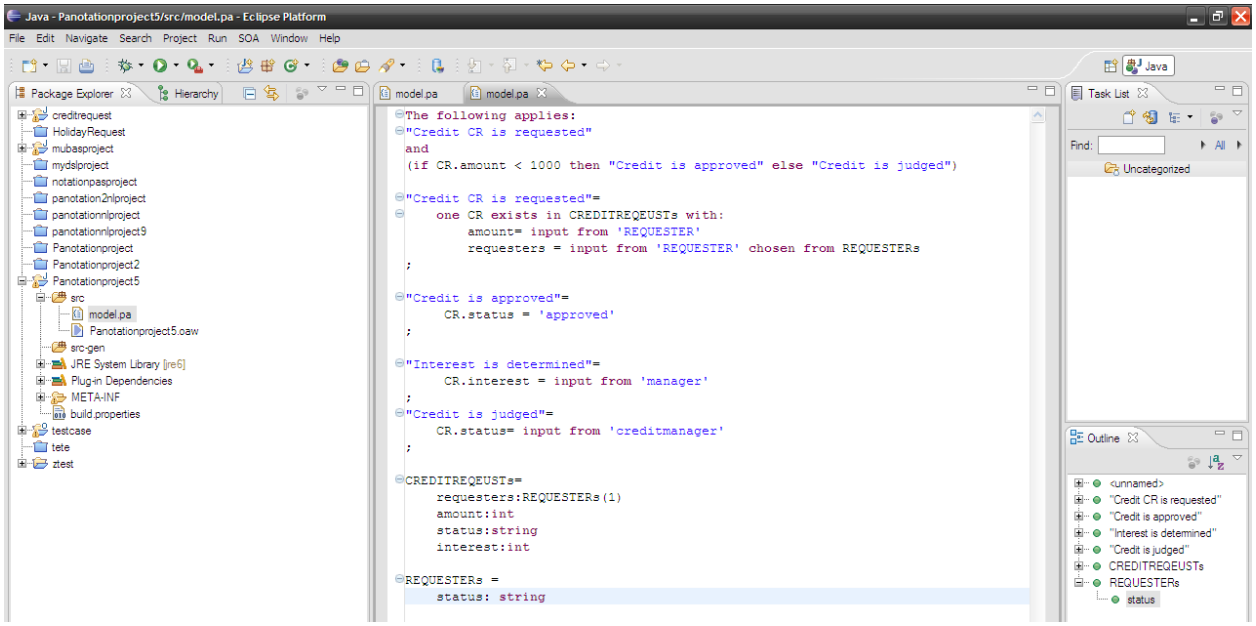

Figure 82: Prototype, example in the PA-editor

7.1.2 Step 2, the transformation from PA-specification to DPM

In this step one first has to create an oAW workflow file. This process contains of three steps:

1. Right click on notation.pa project -> new -> other (see Figure 83);
2. Choose to create a Workflow file (see Figure 84);
3. Give the file a name and create it in the src folder (see Figure 85).

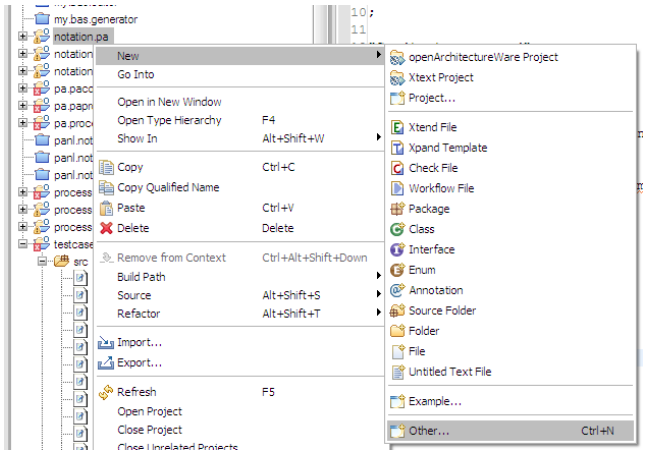

Figure 83: Prototype, creating a workflow file, step 1

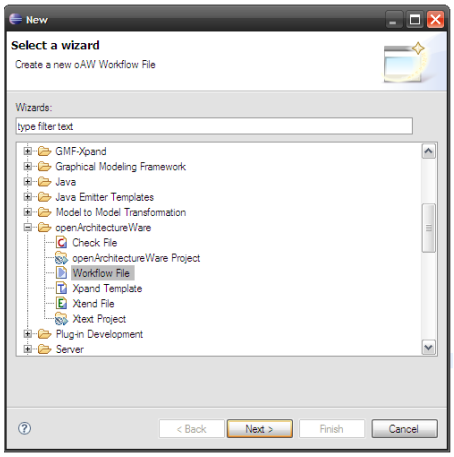

Figure 84: Prototype Creating workflow file step 2

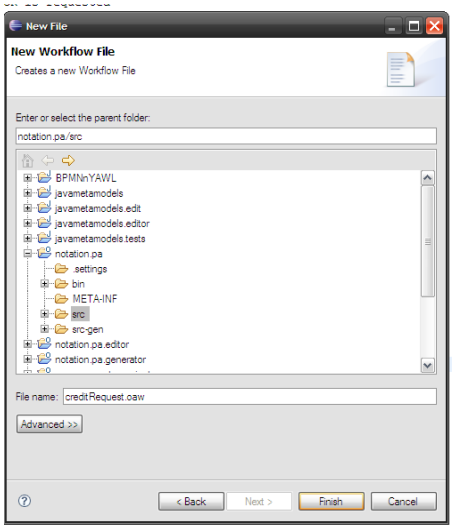

Figure 85: Prototype, creating a workflow file step 3

Next the contents of the workflow file must be written. The template for the content can be found in Appendix G. Figure 86 shows the workflow file of our example.

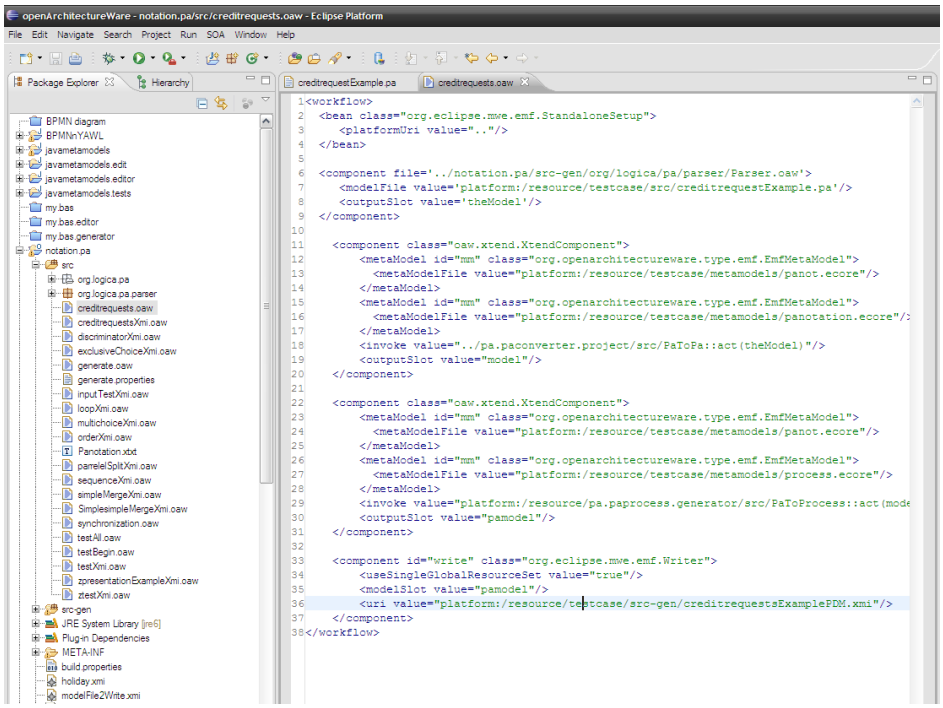

Figure 86: Prototype, workflow file contents

This step is completed by running the workflow file. This is done by right clicking in the workflow file and selecting Run As->oAW Workflow, as can be seen in Figure 87.

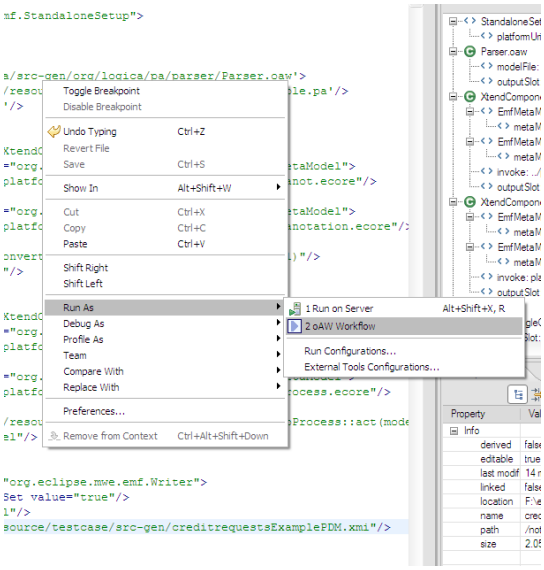

Figure 87: Prototype, running a workflow

7.1.3 Step 3, the transformation from DPM to OPM

This step consists of creating a CSV file and a workflow file in the pa.processsequence.project. The CSV file must have the structure as can be seen in Figure 90, must have the name resources.csv and must be stored in the root of the project. The activity ID's that can be used in the CSV file can be seen by opening

the result of the previous example. The ID of the activities can be found in the properties, as can be seen in Figure 88.

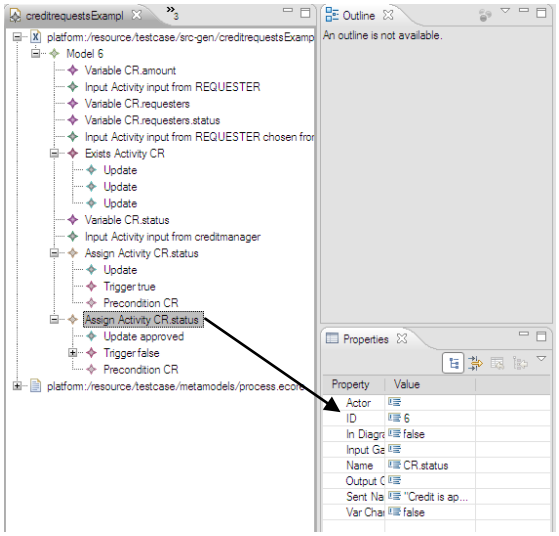

Figure 88: Prototype, checking activity ID's

The workflow file must be created the same way as in the previous step, but now according to the template that can be found in Appendix H.

In the workflow file one also has to choose which transformation one wants to execute. There are three choices:

1. ProToBpmn, in which the complete BPMN diagram will be created;
2. ProToBpmnSimple, in which a BPMN diagram will be created without artifacts;
3. ProToBpmnYawl, in which a BPMN diagram will be created that can be transformed to Yawl and be used in the verification.

Figure 89 shows the workflow file of our example and Figure 90 shows the CSV file we used.

Figure 89: Prototype, workflow file for second transformation

A	B	C	D
Activity	Name	Time	Costs
1	System x	1	1
1	System y	2	2
4	System a	2	2
4	System b	4	5
6	System x	4	4
6	System b	5	7

Figure 90: Prototype, CSV file

Running the workflow file will result in a BPMN file that can be converted to a BPMN diagram. The only intermediate step that has to be performed is to open the generated BPMN file in the Generic editor(right click on the file-> Open with->Generic editor) and to save the file again. If this is done the BPMN diagram can be created by right clicking on the file and choosing the option “Initialize bpmn_diagram file” as shown in Figure 91.

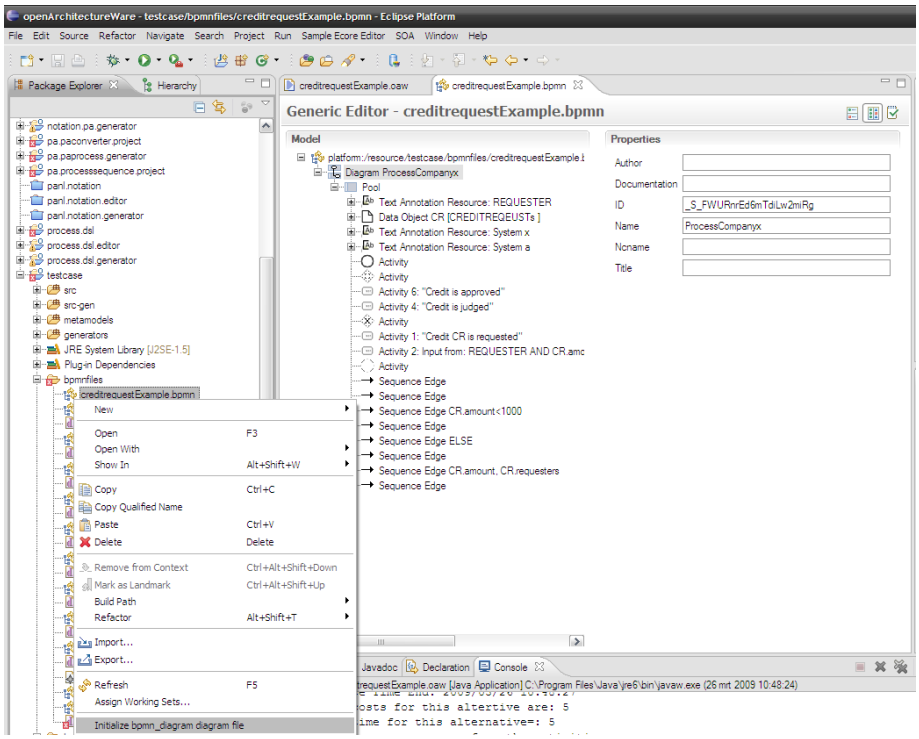

Figure 91: Prototype, creating BPMN diagram

Figure 92 shows the result of creating a diagram of our example.

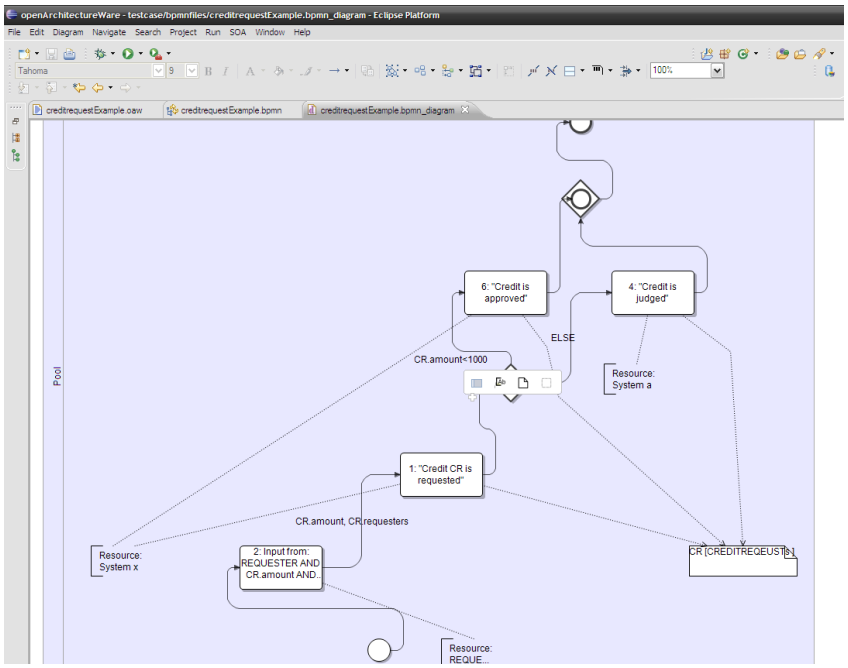

Figure 92: Prototype, result

7.1.4 Verification of result

Verifying the BPMN diagram is done by running the workflow file of step 2 with the transformation ProToBpmnYawl. The resulting BPMN file can then be transformed to YAWL by right clicking on the file and choosing the "BPMN->YAWL option" as shown in Figure 93. The resulting YAWL file can then be found in the xmlfiles folder.

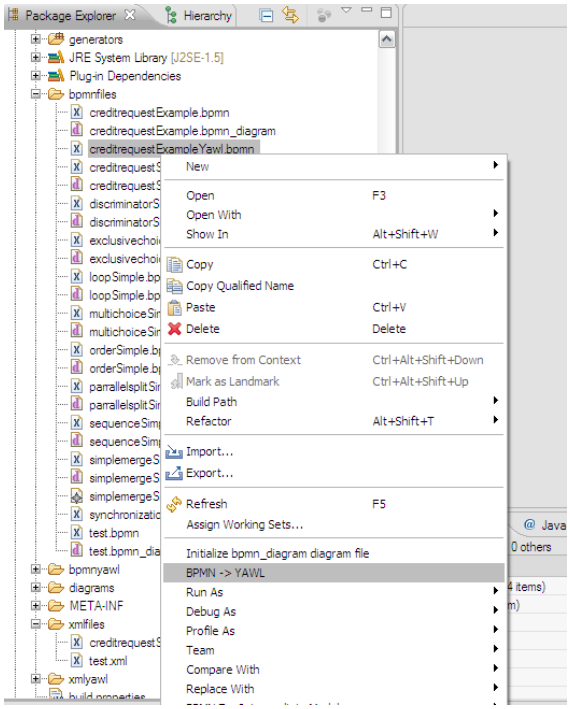

Figure 93: Prototype, BPMN to YAWL conversion

That file can then be imported into the YAWL editor by using the specification->import YAWL engine file option.

If the import succeeds the model can be validated by choosing Specification->Analyze Specification, as shown in Figure 94. Figure 95 shows the result of the analysis of our example.

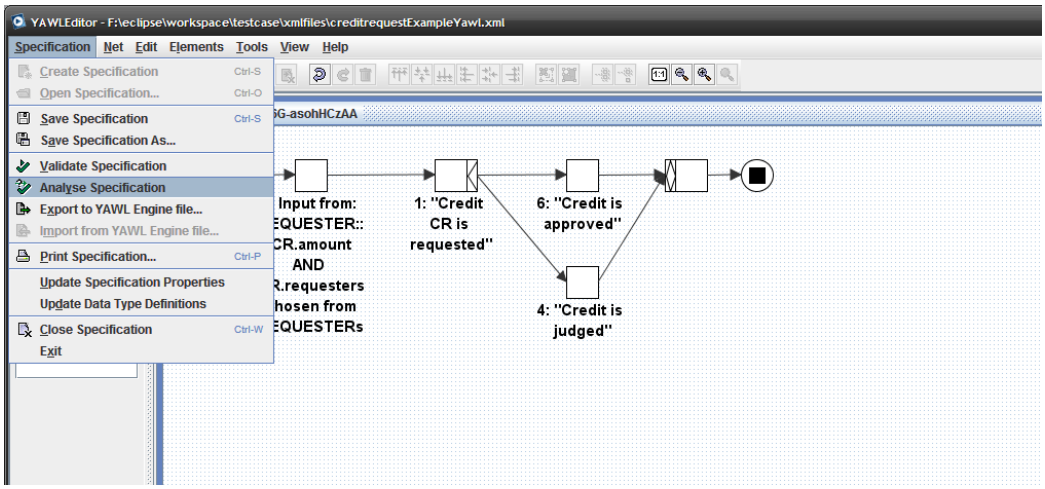

Figure 94: Prototype, analyzing model

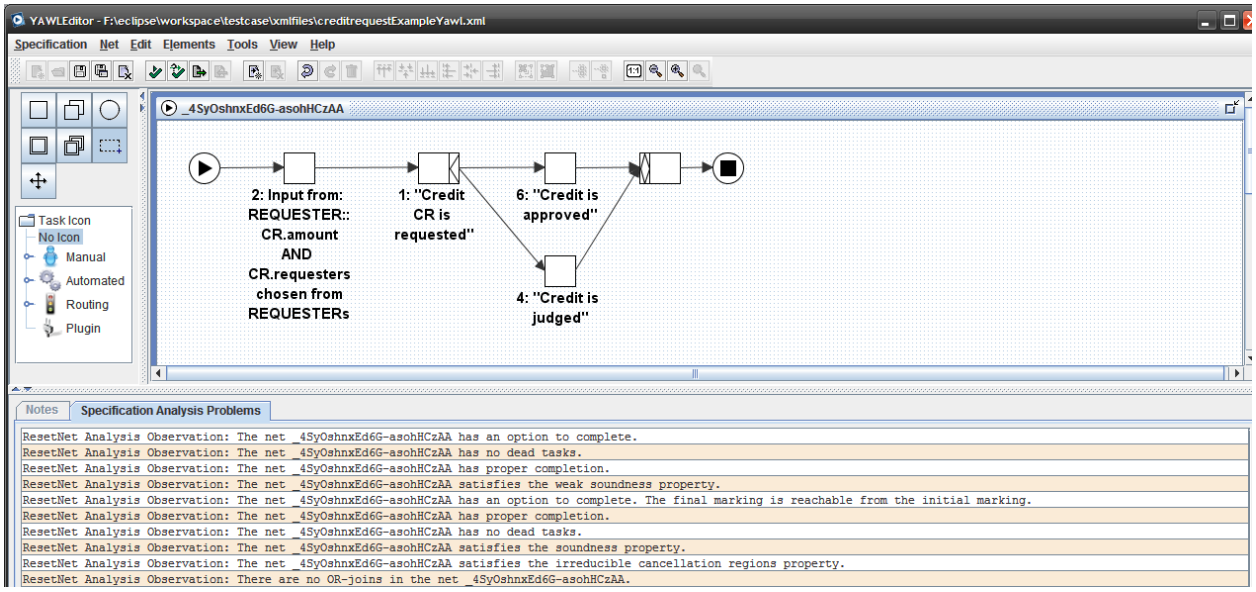

Figure 95: Prototype, analysis result

7.2 Method verification

Our method has already been tested with the eight test casus as discussed in section 6.1. Appendix I discusses the results of these tests. To verify that our method also produces correct results for a combination of patterns section 7.2.1 presents the results of a case study. Furthermore, section 7.2.2 evaluates the transformation method based on the results of the verification and other experiences.

7.2.1 Case study

Listing 21 described an example of a fictional order process that we use to verify our transformation method. This case is an extension of the example of Listing 2.

> Company x has an only store, and every day the store receives several orders. The standard process for handing an order is as follows:
>
> > The first aspects that are checked are if the specific product is in stock and if the customer is not on the blacklist. If either of the aspect are not true then the order process is ended. If they are true then the customer must enter the way he would like to pay for the product. There are two options, Pin or Credit. If the customer selects pin then the product is send to the customer who then pays when receiving the product. If the customer select credit then a credit request is send to the internal credit department.
> >
> > The standard process for handling such a credit request is as follows:
> >
> > > Based on the credibility status of the customer and the size of the requested credit (which is the same as the order amount) the credit request is either directly approved or sent to the credit manager for evaluation. The request is directly approved if:
> > >
> > > > The creditability status of the customer is medium and the requested amount is lower than 500
> > > > The creditability status of the customer is good and the requested amount is lower than 1000
> > >
> > > In all other situations the credit manager has to determine whether to approve or deny the credit request.
> >
> > When the credit request has been handled the process continues as follows:

If the credit request is approved then the product is send to the customer and if the credit request is denied then the order is denied.

In the current situation checking if the product is in stock, checking if a customer is blacklisted and sending the product to the customer is done manually. Company x wants to automate these tasks. They have issued a tender and have received the following offers:

For automatically checking if a product is in stock:

Stock4You	costs 700	time 1
Istock	costs 415	time 4
microStock	costs 500	time 3

For automatically checking if someone is on the blacklist

Blacklistchecker	costs 650	time 1
CheckBlack	costs 500	time 3

For automatically sending out the products

TNT	costs 450	time 2
UPS	costs 470	time 2
FedEx	costs 420	time 1

Listing 21: Order approval process

Our transformation method needs a PA-specification as input. Listing 22 gives the PA-specification that reflects the example described above.

```
The following applies:
"Order is received"
and
(if "Order is received" then "Check availability" and "Check blacklist")
and
(if "Check blacklist" == 'true' and "Check availability" == 'true' then
"Payment method is selected" else "Order is denied")
and
(if ODR.paymentmethod == 'CREDIT' then "Credit CR is requested")
and
(if "Credit CR is requested" and (CR.customer.status == 'medium' and
CR.amount < 1000 or CR.customer.status == 'good' and CR.amount < 500)
then "Credit is approved" else "Credit is judged")
and
(if ODR.paymentmethod == 'PIN' then "Product is send")
and
```

```
(if CR.status == 'denied' then “Order is denied”)
and
(if CR.status == 'approved' then “Product is send”)

“Product is send” =
       ODR.status= input from 'DELIVERY' chosen from 'send'
;

“Order is denied” =
       ODR.status = 'denied'
;

“Credit CR is requested” =
       one CR exists in CREDITREQUESTs with:
              amount = ODR.product.price
              customer = ODR.customer
;

“Check blacklist”=
       input from 'EMPLOYEE' chosen from [true, false] based on
       ODR.customer
;

“Credit is approved”=
        CR.status = 'approved'
;

“Interest is determined”=
        CR.interest = input from 'manager'
;
“Credit is judged”=
       CR.status= input from 'creditmanager'
;

“Payment method is selected”=
       ODR.paymentmethod = input from 'REQUESTER' chosen from ['PIN',
       'CREDIT']
;
```

```
"Order is received" =
	one ODR exists in ORDERs with:
		product = input from 'REQUESTER' chosen from PRODUCTs
		customer = input from 'REQUESTER' chosen from CUSTOMERs
;

"Check availability" =
	input from 'STOCKEMPLOYEE' chosen from [true, false] based on
	ODR.product
;

BLACKLISTs =
	blacklistedcustomer:CUSTOMERs(1)

CREDITREQUESTs =
	customer:CUSTOMERs(1)
	amount:int
	status:string
	interest:int

CUSTOMERs =
	status: string

ORDERs =
	orderid: int required identifies
	product:PRODUCTs
	paymentmethod: string
	status:string
	message:string
	customer:CUSTOMERs(1)

PRODUCTs=
	productid: int required identifies
	name:string
	price:int
```

Listing 22: PA-specification, reflecting the order case

The first part of our transformation method (from PA-specification to DPM) performs without any problems. From this part of the transformation we have extracted the activity ID's and created the CSV file as shown in Figure 96.

A	B	C	D
Activity	Name	Time	Costs
4	Stock4You	1	700
4	Istock	4	415
4	microStoc	3	500
5	Blacklistch	1	650
5	CheckBlac	3	500
14	TNT	2	450
14	UPS	2	470
14	FedEx	1	420

Figure 96: Case CSV file

Figure 97 shows the result of running the second transformation. This is not a correct diagram as a part of the process is missing. This is caused by a case of multiple activities enabling multiple other activities, i.e., after checking the stock status and if the person is blacklisted two activities can occur, namely the selection of the payment method, or denying the order. As mentioned in section 6.3.1 this case is not supported in our transformation method.

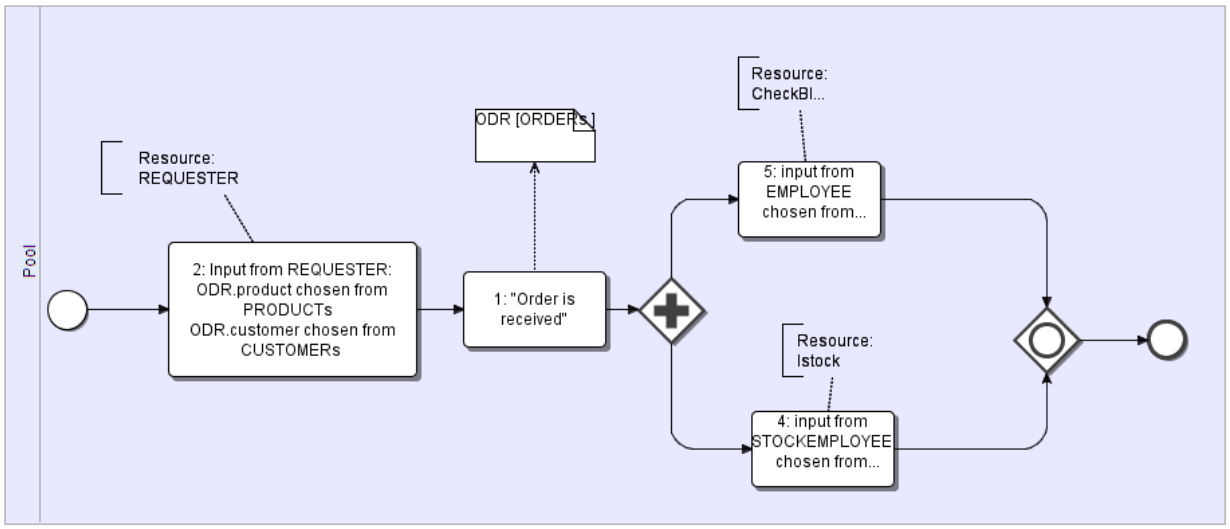

Figure 97: Verification generated BPMN diagram 1

To still be able to verify our method we have decided to apply a workaround for this problem by adding an activity in the business rules specification. We have changed the following statement:

```
(if "Check blacklist" == 'true' and "Check availability" == 'true' then
"Payment method is selected" else "Order is denied")
```

To

```
(if "Check blacklist" == 'true' and "Check availability" == 'true' then
"Preconditions are true")

and

(if ODR.precondition == 'true' then "Payment method is selected" else
"Order is denied")
```

The sentence "Preconditions are true" is defined as

```
"Preconditions are true" =
       ODR.precondition = 'true'
;
```

Figure 98 shows the result of the transformation using this workaround. Both the sequence of activities as the allocation of resources is as expected. The costs and time calculated using the sequence of activities and the allocation of resources is as expected but is not according to real life. For example, the completion time using the resources CheckBlack, Istock and FedEx is 415+500+420=1335. The result of our transformation method is that the costs are 1755. This difference is caused by the OR gateway after the activity "Payment method is selected" which in real life is a Xor gateway as the payment method cannot be both PIN and CREDIT.

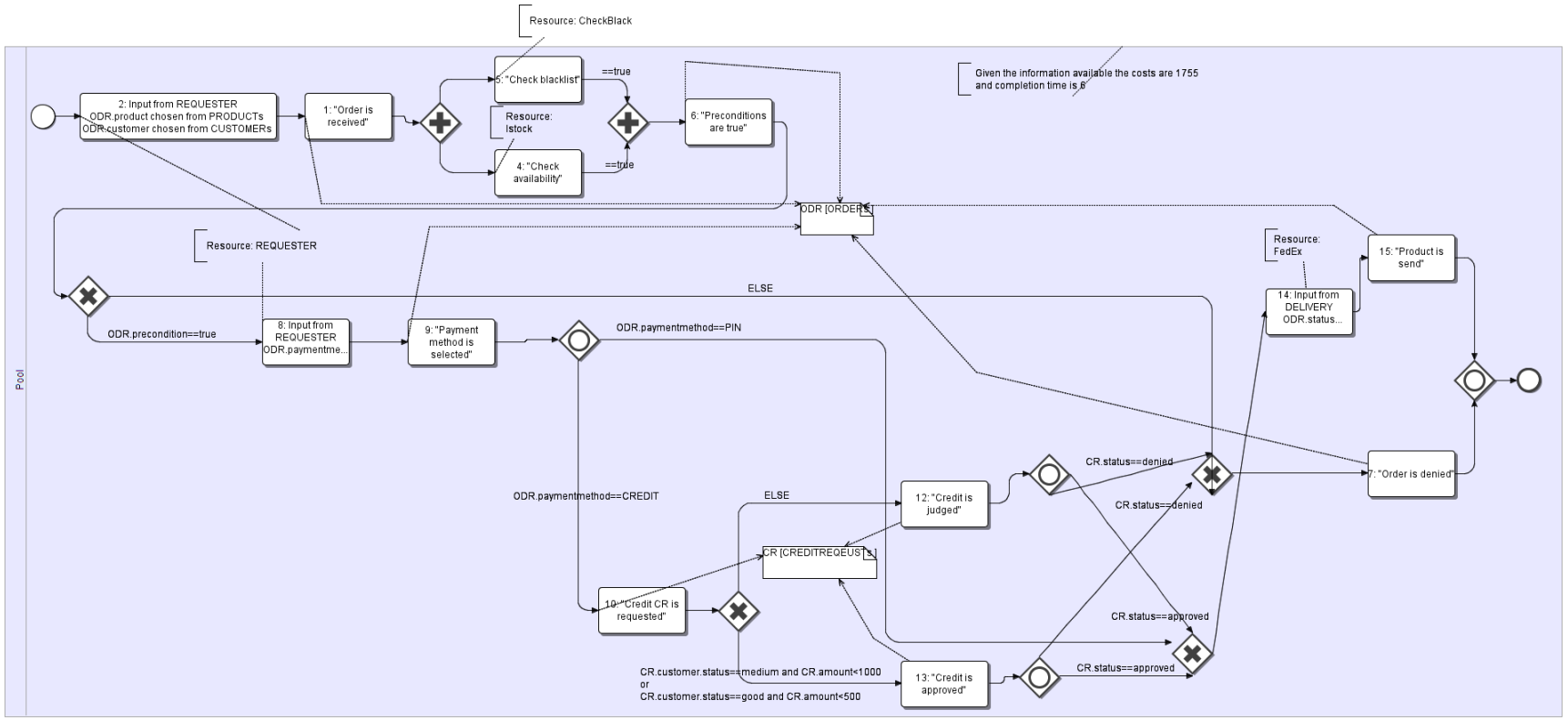

Figure 98: Verification, generated BPMN diagram

7.2.2 Method evaluation

The previous section has shown that our method can, to a large extend, automatically transform business rules to an optimal business process model. There are however some limitations which are listed below.

Determining sequence of activities

- As the case study has shown our transformation method is not capable of dealing with situations where a combination of activities enabled another combination of activities. A possible solution for this limitation is to first find out, for each source activity, the output gateway (if the source activity aids in enabling one activity then the output gateway is of type SEQ, if it aids in enabling multiple activities then the output gateway is of type AND). Then to determine the input gateway for each enabled activity, and then to connect the output gateways to the input gateways. The previous section also showed a workaround that can be used for this issue and that is to use dummy activities.
- Another situation that cannot be handled is when a single activity enables another activity in different ways. An example of a specification that would lead to this problem is given below:

```
If CR.amount > 500 then "Credit is approved"
       And
If CR.requester.status == 'good' then "Credit is approved"
```

A possible solution for this limitation is to change the structure of the enabled map. Listing 23 shows the structure of the enabled map as it is now. It only allows for one set of activities to be connected to one enabled activity.

```
<"Credit is approved">
   <needed map type='Single'>
          <CR>
                 <condition1> CR.amount > 500</condition1>
          </CR>
   </needed map>
</"Credit is approved">
```

Listing 23: Structure needed map

Listing 24 shows a structure of the enabled map that could solve this limitation, now multiple sets of activities can be connected to enabled activities.

```
 <"Credit is approved">
   <list>
          <element1>
                 <needed map type='Single'>
                        <CR>
```

```
                              <condition>
                                     CR.amount > 500
                              </condition>
                        </CR>
                  </needed map>
            </element1>
            <element2>
                  <needed map type='Single'>
                        <CR>
                              <condition>
                                     CR.requester.status==good
                              </condition>
                        </CR>
                  </needed map>
            </element2>
      </list>
</"Credit is approved">
```

Listing 24: Example structure needed map

- The way in which first activities are determined is also not entirely correct. For example, in the following specification:

```
"Credit CR is requested"
And
"Credit CR is judged"
And
If judgement == disapproved then "Credit CR is requested" else …..
```

Producing the result "Credit CR is requested" is a start activity. However, in our current transformation method this activity would get a Trigger meaning that it is not recognized as a first activity. A possible solution for this problem is to add a new type of trigger, with a Boolean value indicating that the trigger is an empty trigger. And adapt the step "find first activities" so that it does not search for activities with no triggers but for activities with empty triggers.

- Another limitation is that our method cannot reason about conditions used to trigger different conditions. For example, in the following specification:

```
"Credit is requested"
If CR.amount > 500 then "Credit is disapproved"
And
If CR.amount <=500 then "Credit is approved"
```

The conditions used are mutually exclusive and thus when connecting the activities that produce the result "Credit is requested" , "Credit is disapproved" and "Credit is approved" a Xor gateway should

be used. However, in our transformation method this is not recognized and an Or gateway is used. This limitation is difficult to solve and we cannot come up with an instant solution for this problem. A possible workaround for this problem is to use the else part of the if statement if one knows that the conditions are mutually exclusive. Thus instead of using the rules as given above using the rules as given below.

```
"Credit is requested"
and
If CR.amount > 500 then "Credit is disapproved" else "Credit is approved"
```

Determining allocation of resources

Our method cannot calculate the correct costs and completion time when there are loops in the process. This limitation is difficult to solve and we cannot come up with an instant solution our workaround for this problem.

Untransformed PA-notation elements

- Business rules that are used to specify the concepts and their relation in the business domain ,e.g., the Collections in the PA-notation, are not transformed at all, since these rules do not have a counterpart in BPMN. A possible solution for this problem would be to not transform these rules to BPMN but to, for example, UML class diagrams or ERD diagrams.
- Some conditions used in business rules can be interpreted and transformed to time triggers used in business processes. For example, the condition used in the following statement could be seen as a time trigger.

```
If currentdate == somedate then "Result"
```

Our method does not incorporate this. A possible solution to this problem is to add a new trigger type called time trigger and to transform each BinaryBooleanExpr which incorporates a date value to that trigger. This solution would also require a new activity type to be added to the process metamodel that represents the time trigger. The trigger can then be transformed to that activity type and the activity type can be transformed to the BPMN symbol for time triggers.

8 Final remarks

This chapter gives the conclusions of this thesis and identifies some future work. Section 8.1 presents our general conclusions, and section 8.2 identifies some future work.

8.1 Conclusions

The goal of this project was to develop a method to automatically transform business rules to business processes. To reach this goal, we first performed a literature study on the topics of business rules, business processes and business process optimization. Among other things this study resulted in a choice for a business rule language, a choice for a business process language and the optimization problem used to determine the optimal business process.

The choice for a business rule language was based on the expressiveness, determined by an ontology analysis, and the support of the language. Three business rule languages have been evaluated, SBVRSE, EM-BrA^{2}Ce and the PA-notation. Based on expressiveness and support we concluded that SBVRSE is not suitable as a business rule language, and that EM-BrA²CE and the PA-notation are comparable. Since the PA-notation is developed by the company where our research has been performed we decided to use the PA-notation as the business rule language.

The choice for a business process language was also based on the expressiveness and the support of the language, however in this case the expressiveness was evaluated using a workflow pattern analysis. Two business process languages where evaluated, BPMN and YAWL, and based on this evaluation BPMN was chosen as the business process language.

The optimization problem is defined by looking at the metrics that are available for the time, quality, costs and flexibility dimensions and optimization techniques. Based on these aspects the optimization problem was defined as follows:

Given:

A set of activities.

The dependencies between the activities.

A set of entities that each performs one activity against a certain costs in a certain time.

The optimal solution is the solution where:

$$(a * total\,costs) + (b * completion\,time) = \min$$

$$a + b = 1$$

$$0 \le a \le 1$$

$$0 \le b \le 1$$

In order to understand how we could transform business rules to business process, we used a reverse engineering approach by first relating business processes to business rules. Based on these relations we have developed a method to automatically transform a PA-specification to a BPMN diagram. The general approach taken in this method can be seen in Figure 99.

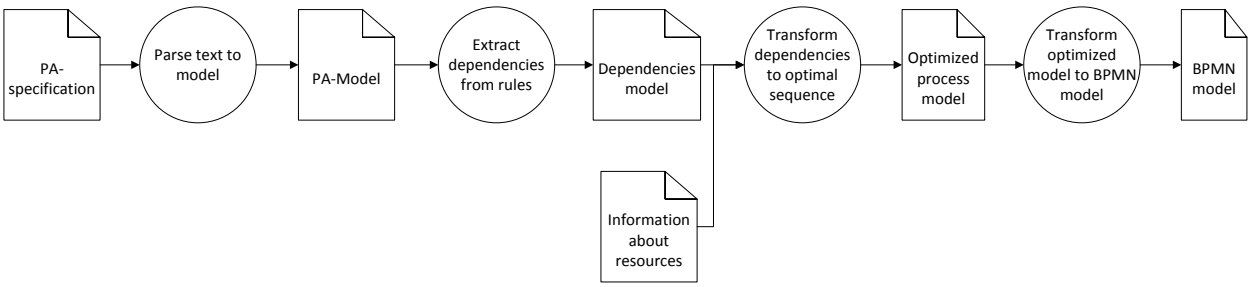

Figure 99: From PA-specification to BPMN

This method has been implemented in a prototype and verified using a case study. Given that the business rule specification is complete and consistent, our method can automatically transform a PA-specification to a BPMN model. There are some issues we were not able to address given the available time. These issues are listed in section 7.2.2. The most noticeable are:

- Making it possible to deal with the situation where a combination of activities enables another combination of activities.
- Making it possible to reason about conditions used to trigger different activities.
- Calculating the correct costs and completion time when there are loops in the process.
- The transformation of business rules that are used to specify the concepts and their relation in the business domain, e.g., the Collections in the PA-notation.

8.2 Future work

Next to addressing the limitations mentioned in section 7.2.2. We suggest the following aspect to be looked at:

- *Further verification*. Our transformation method has only been verified with one large and several small examples. To ensure that our transformation always results in the right business process a more rigorous verification using large real life examples is advisable.
- *More formal verification*. In our research we verify the results of the transformation method in an informal way. There are however also several approaches available to use a formal verification for model transformation (Varro & Pataricza) (Narayanan & Karsai, 2007). It will be interesting to look if these techniques can be used to perform a formal verification of our transformation method.
- *Tracking inconsistencies.* Our transformation method needs a business rule specification that is complete and consistent. To improve the applicability of our method it might be useful to look at the possibility to relax the input requirements and to incorporate a mechanism in our transformation method that can detect and inform the user of inconsistencies and incompleteness in the business rules.
- *Further automation.* The transformation method implemented in the prototype is not fully automatic, during the transformation several automated steps need to be started by the user in a certain order. In the future the transformation method should be implemented in such a way that the transformation can be performed in one go. This might have as a consequence that an own verification tool must be build to verify the business process model, instead of using the

analysis techniques available in the YAWL editor. Furthermore, the Eclipse BPMN modeler does not create a great looking diagram. If the prototype is going to be used in practice this issue should be addressed.

- *Other input and output formats.* Although our method is developed for the specific case of the transformation from a PA-specification to a BPMN model we do believe that, with only small changes to the process metamodel and the optimization transformation step, it can be applied to any case of business rule to business process transformation. This however should be verified by actually building the transformation using other inputs, e.g. SBVRSE and other outputs, e.g. YAWL or EPC.
- *Splitting the dependencies and process metamodel.* For the sake of separation of concerns, it might be better to split the process metamodel into two separate metamodels. One metamodel should capture the dependencies between activities and another metamodel should capture the sequence of activities. In the current transformation method we have chosen not to do this mainly for implementation convenience
- *Support of more patterns:* Our transformation supports all the basic control flow patterns. There are however also other more complex control flow patterns. To further extend our transformation method research could be done on how these control flow patterns relate to concepts available in business rules.
- *Using Multi-objective algorithms.* Our transformation method uses a brute force approach to calculate the optimal allocation of resources. For each possible alternative the total amount of costs and total time are calculated. However, if there are many activities and many different resources per activity then this method does not perform well. For example, depending on computer configuration the calculation of the optimal alternative for 10 activities and 10 different alternatives per activity would take over 40 minutes. To improve the performance it can be useful to research if and how multi objective algorithms can be applied.
- *Extending allocation of resources algorithm.* Our transformation method cannot incorporate knockout ratio's or information about the chance that a process continues on one or more paths. In the future it might be useful to extend our method by allowing the user to enter this information in the process model that is generated and adapting the optimization algorithm so that it can incorporate this information.
- *Further transformation.* Our transformation method only generates a business process model that is used for documentation. To recall, in our introduction we said that ideally the business rules should be transformed to an executable business process model. Extending our method with an transformation to an executable process model is interesting topic for future research. Some tools that can be looked at in that research are the tools available to convert BPMN to BPEL[5] and the prototype build by Logica that can convert a PA-specification, using a limited version of the PA-notation, to an executable process language (Klimstra & Bergsma, 2008).

[5] For example, http://code.google.com/p/bpmn2bpel/downloads/list and http://www.oracle.com/technology/pub/articles/dikmans-bpm.html

- *More (detailed) metrics.* Our transformation method uses global metrics for costs and time. In the future is might be useful to extend our method so that it can determine the optimal process using more detailed metrics, such as variable and fixed costs, and other metric categories such as flexibility.

Next to future work on the transformation method itself a number of other research topics can be identified that might be interesting:

- *Business rule engines.* There are several business rule engines on the market that can execute business rules. It might be interesting to look at, if, and to what extend the execution of business rules is similar to what we achieve with our transformation process.
- *Ontology for business rules.* During the course of the research we found that there is not really an established ontology available to compare different business rule languages in terms of expressiveness. Creating this ontology is an interesting topic for future research.
- *Structured research on transformation languages.* During the course of this research we had to make a choice between different transformation languages. This choice was not easy as there is little information available on how the different languages compare and why one should choose one transformation language over the other. A structured research on how the languages compare could be an interesting topic for future research.

Bibliography

Aalst, W. v., Hofstede, A. t., Kiepuszewski, B., & Barros, A. (2003). Workflow Patterns. *Distributed and parallel databases* , 5-51.

ATLAS Group. (n.d.). *ATL*. Retrieved 12 21, 2008, from Eclipse: http://www.eclipse.org/m2m/atl/

Börger, E. (2007). A critical analysis of workflow patterns.

Brand, N., & Kolk, H. v. (1995). *Workflow analysis and design.* Deventer: Kluwer bedrijfswetenschappen.

Business Rules Group. (n.d.). *defining Business Rules...* Retrieved oktober 20, 2008, from Business Rules Group: http://www.businessrulesgroup.org/defnbrg.shtml

Cardoso, J., Vanderfeesten, I., Reijers, H., Mendling, J., & Aalst, W. v. (2008). On a Quest for Good Process Models: The Cross-Connectivity Metric. In *Advanced Information Systems Engineering* (pp. 480-494). Heidelberg: Springer Berlin.

Charfi, A., & Mezini, M. (2008). Hybrid Web Service Composition: Business Processes Meet Business Rules. *International Conference On Service Oriented Computing* (pp. 30 - 38). New York, USA : ACM.

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal , 45* (3), 621-645.

Davenport, T. H. (1993). *Process Innovation: Reengineering Work Through Information.* Boston: Harvard Business School Press.

Deb, K. (2001). *Multiobjective Optimization using Evolutionary Algorithms.* John Wiley and Son Ltd.

Decker, G., Dijkman, R., Dumas, M., & García-Bañuelos, L. (2008). Transforming BPMN Diagrams into YAWL Nets. *Proceedings of the 6th International Conference on Business Process Management* (pp. 386 - 389). Milan, Italy : Springer-Verlag.

Dijkman, M., & Dumas, M. O. (n.d.).

Doshi, P., Goodwin, R., Akkiraju, R., & Verma, K. (2005). Dynamic Workflow Composition using Markov Decision Processes. *International Journal of Web Services Research,* , 1-17.

Eder, R., Filieri, A., Kurz, T., Heistracher, T., & Pezzuto, M. (2008). Model-transformation-based Software Generation utilizing Natural language notations. *2nd IEEE International Conference on Digital Ecosystems and Technologies*, (pp. 306-312). Phitsanuloke, Thailand, .

Edmond, D., Russell, N., Hofstede, A. t., & Aalst, W. v. (2004). *Workflow Data Patterns.* QUT Technical report FIT-TR-2004-01, Queensland University of Technology, Brisbane.

Eertink, H., Janssen, W., Luttighuis, P. O., Teeuw, W., & Vissers, C. (1999). A Business Process Design Language. *Lecture notes in computer science , 1709*, 76-95.

Forgy, C. L. (1991). Rete: a fast algorithm for the many pattern/many object pattern match problem. In *Expert systems: a software methodology for modern applications* (pp. 324-341). Los Alamitos, CA: IEEE Computer Society Press.

Goedertier, S., Heason, R., & Vanthienen, J. (2007). *EM-BrA2Ce v0.1; A Vocabulary and Execution Model for Declarative Business Process Modeling.*

Grossmann, G., Schrefl, M., & Stumptner, M. (2008). Modelling inter-process dependencies with high-level business process modelling languages. *Proceedings of the fifth on Asia-Pacific conference on conceptual modelling* (pp. 89-102). Wollongong, Australia : Australian Computer Society, Inc.

Guizzardi, G. (2005). *ONTOLOGICAL FOUNDATIONS FOR STRUCTURAL CONCEPTUAL MODELS* (Vol. no. 15). Enschede: Telematica Instituut Fundamental Research Series, Universal Press.

Hammer, M., & Champy, J. (1993). *Reengineering the Corporation: A Manifesto for Business Revolution.* Londen: Brealey.

Herwijnen, M. v. (2006). *Weighted summation.*

Hirtle, D., Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., et al. (n.d.). *Schema Specification of RuleML 0.91.* Retrieved november 10, 2008, from ruleml: http://www.ruleml.org/0.91/

Hofstede, A. t., Russel, N., Edmond, D., & Aalst, W. v. (2004). *Workflow resource patterns.* Tech. Rep. WP 127, Eindhoven University of Technology , Eindhoven.

Iacob, M., & Jonkers, H. (2009). Performance and Cost Analysis of Service-Oriented Enterprise Architectures. In A. Gunasekaran, *Global Implications of Modern Enterprise Information Systems: Technologies and Applications* (pp. 49-73). Hershey: Information Science Referenc.

IKV++ technologies. (n.d.). *medini QVT released*. Retrieved 1 10, 2009, from IKV++ technologies: http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77&lang=en

Initiative, W. P. (2008). *Workflow Patterns* . Retrieved 9 17, 2008, from http://www.workflowpatterns.nl

Intalio Inc. (n.d.). *STP BPMN Modeler: BPMN object model*. Retrieved 2 26, 2009, from http://www.eclipse.org/bpmn/model/index.php

Jansen-Vullers, M. H., Kleingeld, P. A., Loosschilder, M. W., & Reijers, H. A. (2007). Performance Measures to evaluate the impact of Best Practices. *Proceedings of Workshops and Doctoral Consortium of the 19th International Conference on Advanced Information Systems Engineering (BPMDS workshop)* (pp. 359–368). Trondheim: Tapir Academic Press.

Jonkers, H., & Franken, H. (1996). QUANTITATIVE MODELLING AND ANALYSIS OF BUSINESS PROCESSES. *Proceedings of the 8th European Simulation Symposium.* Genoa, Italy: Society for Computer Simulation Europe.

Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., et al. (2007). *Web Services Business Process Execution Language Version 2.0.* oasis.

Keller, G., & Nüttgens, M. S. (1992). *Semantische Prozebmodellierung auf der Grundlage.* Institut für Wirtschaftsinformatik. Saarbrucken: University of Saarland.

Klimstra, M., & Bergsma, N. (2008). *Implementatierapport PA-Compiler.* Logica & Saxion Hogescholen.

Kurtev, I. (2008). State of the Art of QVT: A Model Transformation Language Standard. In *Applications of Graph Transformations with Industrial Relevance* (pp. 377-393). Springer Berlin / Heidelberg.

Linehan, M.H. (2008, March 11). *Date-Time Foundation Vocabulary RFP Proposal.* Retrieved November 20, 2008, from OMG: http://www.omg.org/docs/ad/08-03-07.pdf

Lonneke Dikmans, Oracle. (n.d.). *Transforming BPMN into BPEL: Why and How* . Retrieved 12 19, 2008, from Oracle: http://www.oracle.com/technology/pub/articles/dikmans-bpm.html

Mitoussis, N., & Macos, D. (2008, 12 11). *MDA Transformation Languages.* Retrieved 1 4, 2009, from Special Interest Group "Model-Driven Software Engineering": www.sig-mdse.org/web/export/sites/sig-mdse/informationen/dokumente/20081211_Mitoussis_Macos.ppt

Muehlen, M. z., Indulska, M., & Kamp, G. (2008). Business Process and Business Rule Modeling:A Representational Analysis. *2007 Eleventh International IEEE EDOC Conference Workshop.*

Narayanan, A., & Karsai, G. (2007). *Verifying Model Transformations by Structural Correspondence.* Institute for Software Integrated Systems. Nashville: Vanderbilt University.

Object Management Group. (2008). *Business Process Modeling Notation V1.1.*

Object management Group. (2008, 4). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification.* Retrieved 1 5, 2009, from http://www.omg.org/docs/formal/08-04-03.pdf

Object Management Group. (2006, 5 1). *Object Constraint Language.* Retrieved 4 20, 2009, from http://www.omg.org/docs/formal/06-05-01.pdf

Object Management Group. (2008). *Semantics of Business Vocabulary and Business Rules (SBVR) v1.0.*

Object Management Group. (2005, 8 1). *Unified Modeling Language: Superstructure.* Retrieved 4 20, 2009, from http://www.omg.org/docs/formal/05-07-04.pdf

openArchitectureWare.org. (n.d.). *openArchitectureWare*. Retrieved 12 28, 2008, from openArchitectureWare: http://www.openarchitectureware.org

Parr, T. (n.d.). *ANTLR*. Retrieved Oktober 9, 2008, from ANTLR parser generator: http://www.antlr.org/

Process mining group. (2008, June 26). *Release notes for ProM Version 5.0*. Retrieved 12 19, 2008, from ProM - the leading process mining toolkit: http://is.tm.tue.nl/trac/prom/wiki/ReleaseNotes_5_0

Raj, A., Prabhakar, T., & Hendryx, S. (2008). Transformation of SBVR business design to UML models. *Proceedings of the 1st conference on India software engineering conference* , (pp. 29-38). Hyderabad, India .

Reijers, H., & Aalst, W. v. (2005). The effectiveness of workflow management systems: Predictions and lessons learned. *International Journal of Information Management 25* , 458–472.

Reijers, H., & Mansar, S. L. (2005). Best Practises in business process redesign: an overbiew and qaulitative evaluation of succesful redesign heuristics. *Omega 33* , 283-306.

Russel, N., Hofstede, A. t., Aalst, W. v., & Mulya, N. (2006). WORKFLOW CONTROL-FLOW PATTERNS:A Revised View. *BPM Center Report , 06* (22).

Russell, N. H., Aalst, W. v., & Edmond, D. (2007). *newYAWL: Achieving Comprehensive Patterns Support in Workflow for the Control-Flow, Data and Resource Perspectives.* BPM Center Report.

Sharp, A., & McDermott, P. (2001). *Workflow Modeling: Tools for Process Improvement and Application Development.* Artech House.

Sinderen, M. v., Ferreira Pires, L., & Guizzardi, G. (2005). An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages. In *Model Driven Engineering Languages and Systems* (pp. 691-705). Springer Berlin / Heidelberg.

SmartQVT. (n.d.). *SmartQVT*. Retrieved 1 10, 2009, from SmartQVT: http://smartqvt.elibel.tm.fr/

Söderström, E., Andersson, B., Johannesson, P., Perjons, E., & Wangler, B. (2002). Towards a Framework for Comparing Process Modelling Languages . In *Lecture Notes in Computer Science* (pp. 600-611). Springer Berlin / Heidelberg.

Taveter, K., & Wagner, G. (2001). Agent-Oriented Enterprise Modeling Based on Business Rules. In *Conceptual Modeling — ER 2001* (pp. 527-540). Berlin: Springer Berlin / Heidelberg.

The Model Development Tools project. (n.d.). *Model Development Tools (MDT)*. Retrieved December 22, 2008, from Eclipse: http://www.eclipse.org/modeling/mdt/?project=sbvr

The World Wide Web Consortium. (n.d.). *XQuery 1.0: An XML Query Language*. Retrieved 12 28, 2008, from The World Wide Web Consortium: http://www.w3.org/TR/xquery/

Tiwari, A., Vergidis, K., Majeed, B., & Roy, R. (2007). Optimisation of Business Process Designs:An algorithmic approach with multiple objectives. *International Journal of production Economics , 109* (1-2), 105-121.

University of Zurich. (n.d.). *Attempto Project*. Retrieved 11 12, 2008, from Attempto Project: http://attempto.ifi.uzh.ch/site/

van der Aalst, W., & Hofstede, A. t. (2005). YAWL: Yet Another Workflow Language. *Information Systems , 30* (4), 245-275.

Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H., & Aalst, W. v. (2007). Quality Metrics for Business Process Models. In L. Fischer, *BPM and Workflow Handbook 2007* (pp. 179-190). Future Strategies USA.

Varro, D., & Pataricza, A. Automated Formal Verification of Model Tranformations. *Critical Systems Development in UML; Proceedings of the UML'03 Workshop.* 2003.

Verbeek, E., Hattem, M. v., Reijers, H., & Munk, W. (2005). Protos 7.0: Simulation made accessible. In *Lecture Notes in Computer Science* (pp. 465–474). Miami, USA: Springer-Verlag.

Vergadis, K., Tiwari, A., & Majeed, B. (2007). Composite Business Processes: An Evolutionary Multi-objective Optimization Approach. *IEEE congress on Evolutionart Computation*, (pp. 2672-2678).

Vergidis, K., Tiwari, A., & Majeed, B. (2006). Business process improvement using multi-object optimisation. *BT technology Journal , 24* (2), 229-235.

Verschuren, P., & Doorewaard, H. (2007). *Het ontwerpen van een onderzoek.* Den Haag: Lemma.

Weber, R. (1997). *Ontological Foundations of Information systems.* Coopers and Lybrand.

Weiden, M., Hermans, L., Schreiber, G., & Zee, S. v. (2002). Classification and Representation of Business Rules. *European Business Rules Conference, 2002.*

Wohed, P., Aalst, W. v., Dumas, M., Hofstede, A. t., & Russell, N. (2006). On the Suitability of BPMN for Business Process Modelling. *Proceedings 4th International Conference on Business Process Management.* Vienna, Austria: Springer Verlag.

Workflow Managent Coalition. (2008, oktober 10). *WFMC-TC-1025-Oct-10-08-A (Final XPDL 2.1 Specification).* Retrieved december 28, 2008, from http://wfmc.org/View-document-details/WFMC-TC-1025-Oct-10-08-A-Final-XPDL-2.1-Specification.html

Wynn, M., Verbeek, H., Aalst, W. d., Hofstede, A. t., & Edmond, D. (2007). Business Process Verification - Finally a Reality!. *Business Process Management Journal* .

List of Abbreviations

AceRules:	Attempto Controlled English Rules
ANTLR:	ANother Tool for Language Recognition
ATL:	Atlas Transformation Language
BPMN:	Business Process Modeling Notation
BPD:	Business Process Diagram
BWW:	Bunge Wand Weber
CSV:	Comma Separated Values
DPM:	Dependencies process model
EBNF:	Extended Backus-Naur Form
EM-BrA^{2}Ce:	Enterprise Modeling using business Rules, Agents, Activities, Concepts and Events
EMF:	Eclipse Modeling Framework
EPC:	Event-driven Process Chains
m2m:	Model to Model
m2t:	Model to Text
oAW:	openArchitectureWare
OCL:	Object Constraint Language
OMG:	Object Management Group
OPM:	Optimized process model
PAGM:	Pa generated model
PAGMM:	Pa generated metamodel
PAM:	PA-model
PAMM:	PA-metamodel
PMM:	Process metamodel
QVT:	Query/View/Transformation
RuleML:	Rule Markup Language
SBVR:	Semantics of Business Vocabularies and Business Rules
SBVRSE:	Semantics of Business Vocabularies and Business Rules Structured English
t2m:	Text to Model
UML:	Unified Modeling Language
WS-BPEL:	Web Services Business Process Execution Language
XML:	Extensible Markup Language
YAWL:	Yet Another Workflow Language

Appendix A Results of expressiveness analysis BPMN and YAWL

Table 16 shows the results of the analysis of BPMN and YAWL for the control flow patterns. The results for BPMN are taken from and (Russel, Hofstede, Aalst, & Mulya, 2006) the result of YAWL is taken from (Russell, Aalst, & Edmond, 2007).

Table 16: Control flow pattern analysis BPMN and YAWL

Control Flow Pattern	BPMN	YAWL	Control Flow Pattern	BPMN	YAWL
1 Sequence	+	+	23 Transient Trigger	-	-
2 Parallel Split	+	+	24 Persistent Trigger	+	-
3 Synchronization	+	+	25 Cancel Region	+/-	+
4 Exclusive Choice	+	+	26 Cancel Multiple Instance Activity	+	+
5 Simple Merge	+	+	27 Complete Multiple Instance Activity	-	-
6 Multi Choice	+	+	28 Blocking Discriminator	+/-	-
7 Structured Synchronizing Merge	+	+	29 Cancelling Discriminator	+	+
8 Multi Merge	+	+	30 Structured Partial Join	+/-	-
9 Structured Discriminator	+/-	+	31 Blocking Partial Join	+/-	-
10 Arbitrary Cycles	+	+	32 Cancelling Partial Join	+/-	-
11 Implicit Termination	+	+/-	33 Generalised AND Join	+	+
12 Multiple Instances without Synchronization	+	+	34 Static Partial Join for Multiple Instances	+/-	+
13 Multiple Instances with a Priori Design Time Knowledge	+	+	35 Cancelling Partial Join for Multiple Instances	+/-	+
14 Multiple Instances with a Priori Run Time Knowledge	+	+	36 Dynamic Partial Join for Multiple Instances	-	-
15 Multiple Instances without a Priori Run Time Knowledge	-	+	37 Local Synchronizing Merge	-	+
16 Deferred Choice	+	+	38 General Synchronizing Merge	-	+
17 Interleaved Parallel Routing	-	+	39 Critical Section	-	+
18 Milestone	-	+	40 Interleaved Routing	+/-	+
19 Cancel Activity	+	+	41 Thread Merge	+	-
20 Cancel Case	+	+	42 Thread Split	+	-
21 Structured Loop	+	-	43 Explicit Termination	+	-
22 Recursion	-	+	**Total**	**29**	**31**

Table 17 shows the results of the analysis of BPMN and YAWL for the data patterns. The results for BPMN are taken from (Wohed, Aalst, Dumas, Hofstede, & Russell, 2006) and the result of YAWL is taken from (Russell, Aalst, & Edmond, 2007).

Table 17: Data pattern analysis BPMN and YAWL

	Data pattern	BPMN	YAWL		Data pattern	BPMN	YAWL
1	Task Data	+	-	1	Environment to Case - Push-Oriented	-	-
2	Block Data	+	+	2	Case to Environment - Pull-Oriented	-	-
3	Scope Data	-	-	3	Workflow to Environment - Push-Oriented	-	-
4	Multiple Instance Data	+/-	+	4	Environment to Workflow - Pull-Oriented	-	-
5	Case Data	+	-	5	Environment to Workflow - Push-Oriented	-	-
6	Folder Data	-	-	6	Workflow to Environment - Pull-Oriented	-	-
7	Workflow Data	-	-	7	Data Transfer by Value - Incoming	+	+
8	Environment Data	-	-	8	Data Transfer by Value - Outgoing	+	+
9	Task to Task	+	+	9	Data Transfer - Copy In/Copy Out	+/-	-
10	Block Task to SubWorkflow Decomposition	+	+	10	Data Transfer by Reference - Unlocked	-	-
11	SubWorkflow Decomposition to Block Task	+	+	11	Data Transfer by Reference - With Lock	+	-
12	To Multiple Instance Task	-	+	12	Data Transformation - Input	+/-	+
13	From Multiple Instance Task	-	+	13	Data Transformation - Output	+/-	+
14	Case to Case	-	-	14	Task Precondition - Data Existence	+	-
15	Task to Environment - Push-Oriented	+	-	15	Task Precondition - Data Value	-	-
16	Environment to Task - Pull-Oriented	+	-	16	Task Postcondition - Data Existence	+	-
17	Environment to Task - Push-Oriented	+	-	17	Task Postconditon - Data Value	-	-
18	Task to Environment - Pull-Oriented	+	-	18	Event-Based Task Trigger	+	-
19	Case to Environment - Push-Oriented	-	-	19	Data-Based Task Trigger	+	-
20	Environment to Case - Pull-Oriented	-	-	20	Data-Based Routing	+	+
Total		**20**	**12**				

Table 18 shows the results of the analysis of BPMN and YAWL for the resource patterns. The results for BPMN are taken from (Wohed, Aalst, Dumas, Hofstede, & Russell, 2006) and the result of YAWL is taken from (Russell, Aalst, & Edmond, 2007).

Table 18: Resource pattern analysis BPMN and YAWL

Resource pattern	BPMN	YAWL	Resource pattern	BPMN	YAWL
1 Direct Allocation	+	+	1 Resource-Initiated Execution - Offered Work Item	-	-
2 Role-Based Allocation	+	+	2 System Determined Work Queue Content	-	-
3 Deferred Allocation	-	-	3 Resource-Determined Work Queue Content	-	-
4 Authorisation	-	-	4 Selection Autonomy	-	-
5 Seperation of Duties	-	-	5 Delegation	-	-
6 Case Handling	-	-	6 Escalation	-	-
7 Retain Familiar	-	-	7 Deallocation	-	-
8 Capability Based Allocation	-	-	8 Stateful Reallocation	-	-
9 History Based Allocation	-	-	9 Stateless Reallocation	-	-
10 Organisational Allocation	-	-	10 Suspension/Resumption	-	-
11 Automatic Execution	+	+	11 Skip	-	-
12 Distribution by Offer - Single Resource	-	+	12 Redo	-	-
13 Distribution by Offer - Multiple Resources	-	+	13 Pre-Do	-	-
14 Distribution by Allocation - Single Resource	+	-	14 Commencement on Creation	+	-
15 Random Allocation	-	-	15 Commencement on Allocation	-	-
16 Round Robin Allocation	-	-	16 Piled Execution	-	-
17 Shortest Queue	-	-	17 Chained Execution	+	-
18 Early Distribution	-	-	18 Configurable Unallocated Work Item Visibility	-	-
19 Distribution on Enablement	+	+	19 Configurable Allocated Work Item Visibility	-	-
20 Late Distribution	-	-	20 Simultaneous Execution	+	+
21 Resource-Initiated Allocation	-	+	21 Additional Resources	-	-
22 Resource-Initated Execution - Allocated Work Item	-	+	**Total**	**8**	**9**

Appendix B PA-notation generated metamodel

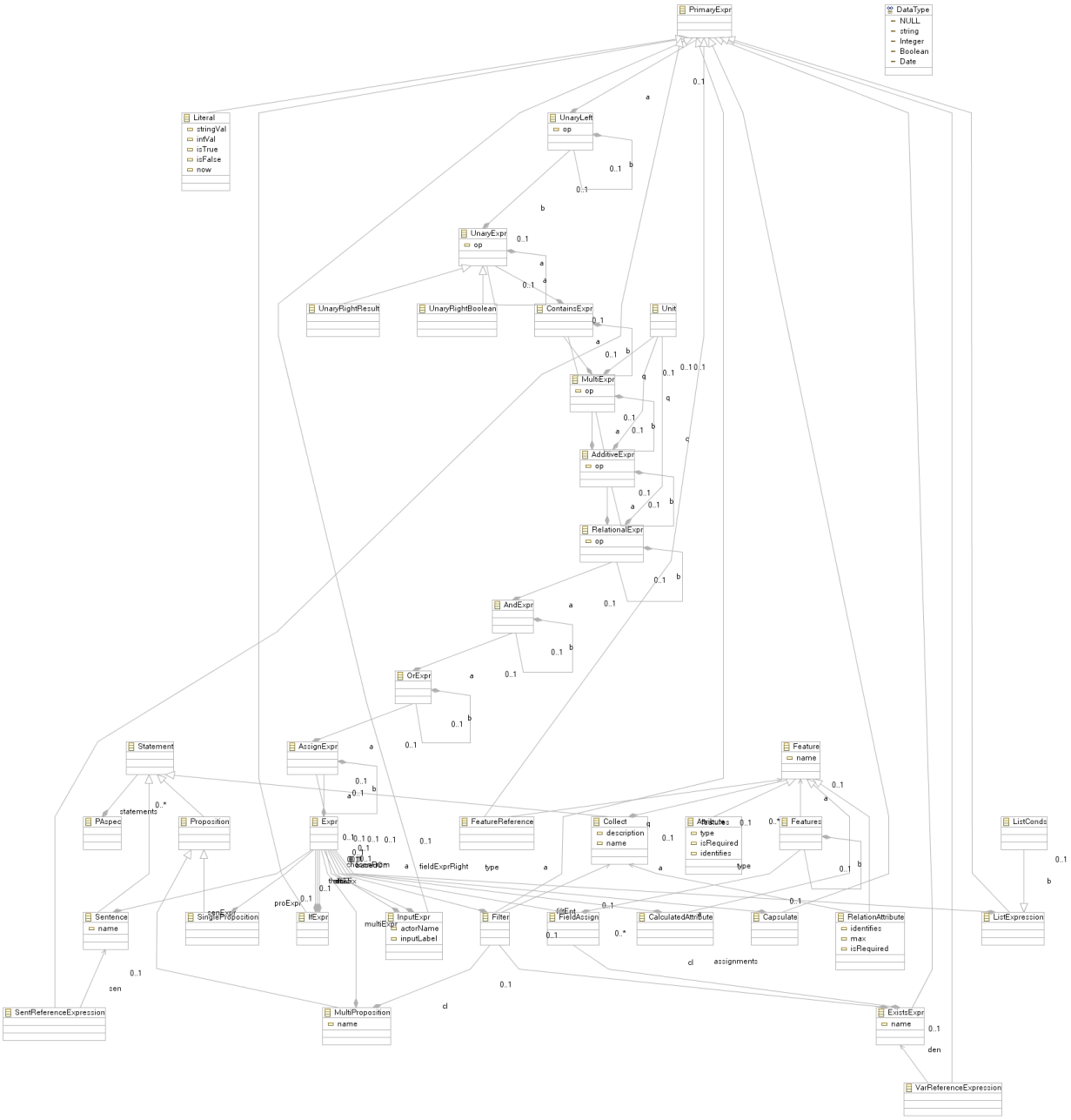

Figure 100: PA-notation generated metamodel

Appendix C PA-notation metamodel

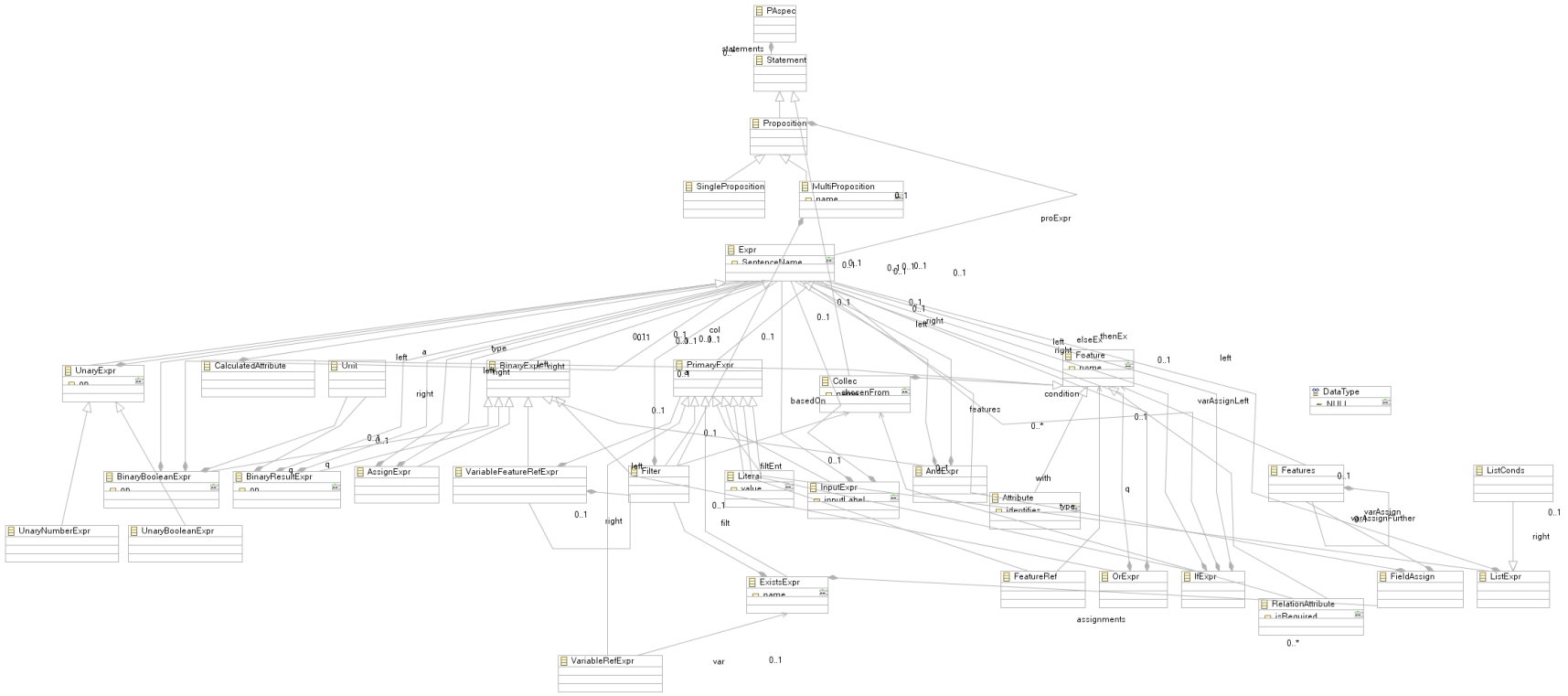

Figure 101: PA-notation metamodel

Appendix D Process metamodel

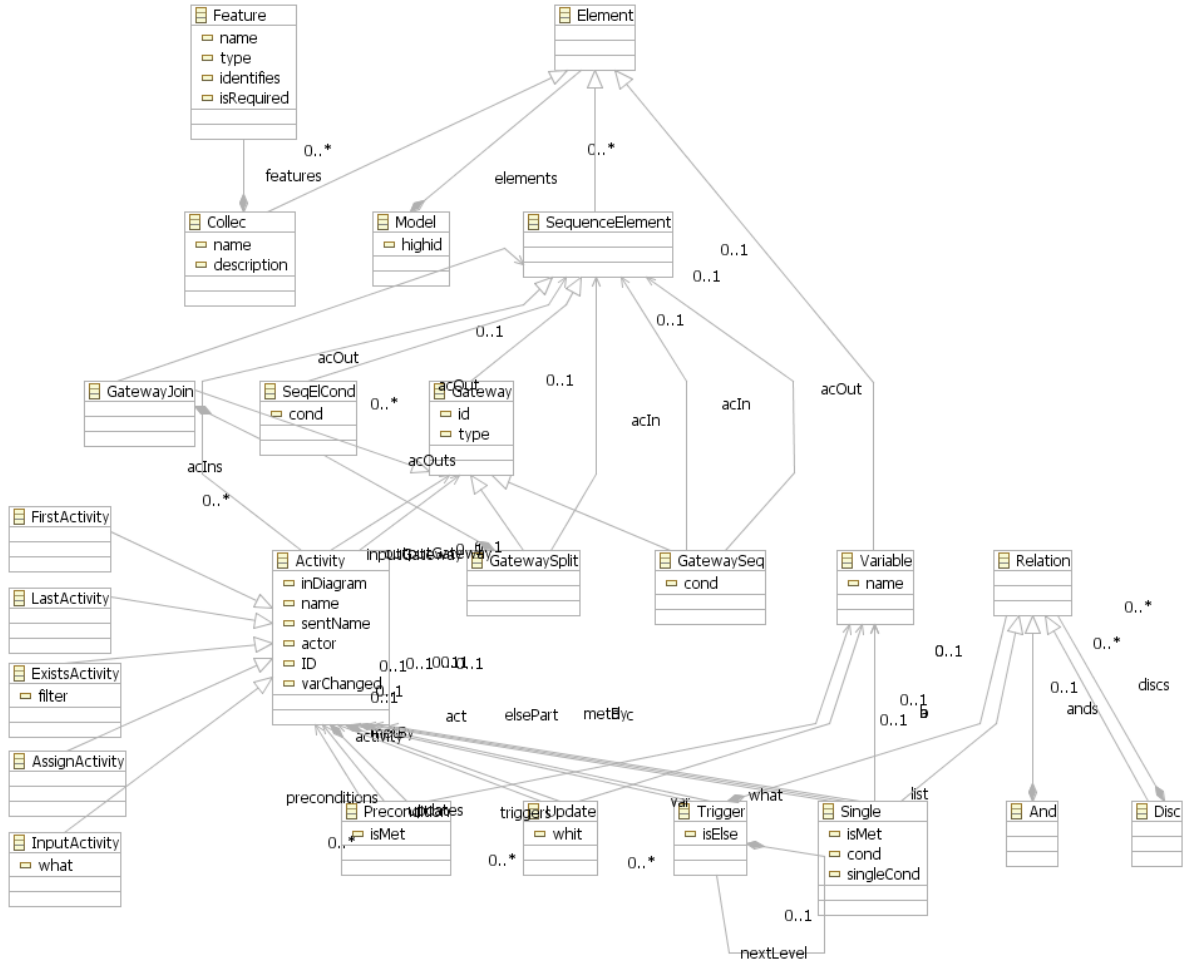

Figure 102: Process metamodel

Appendix E BPMN metamodel

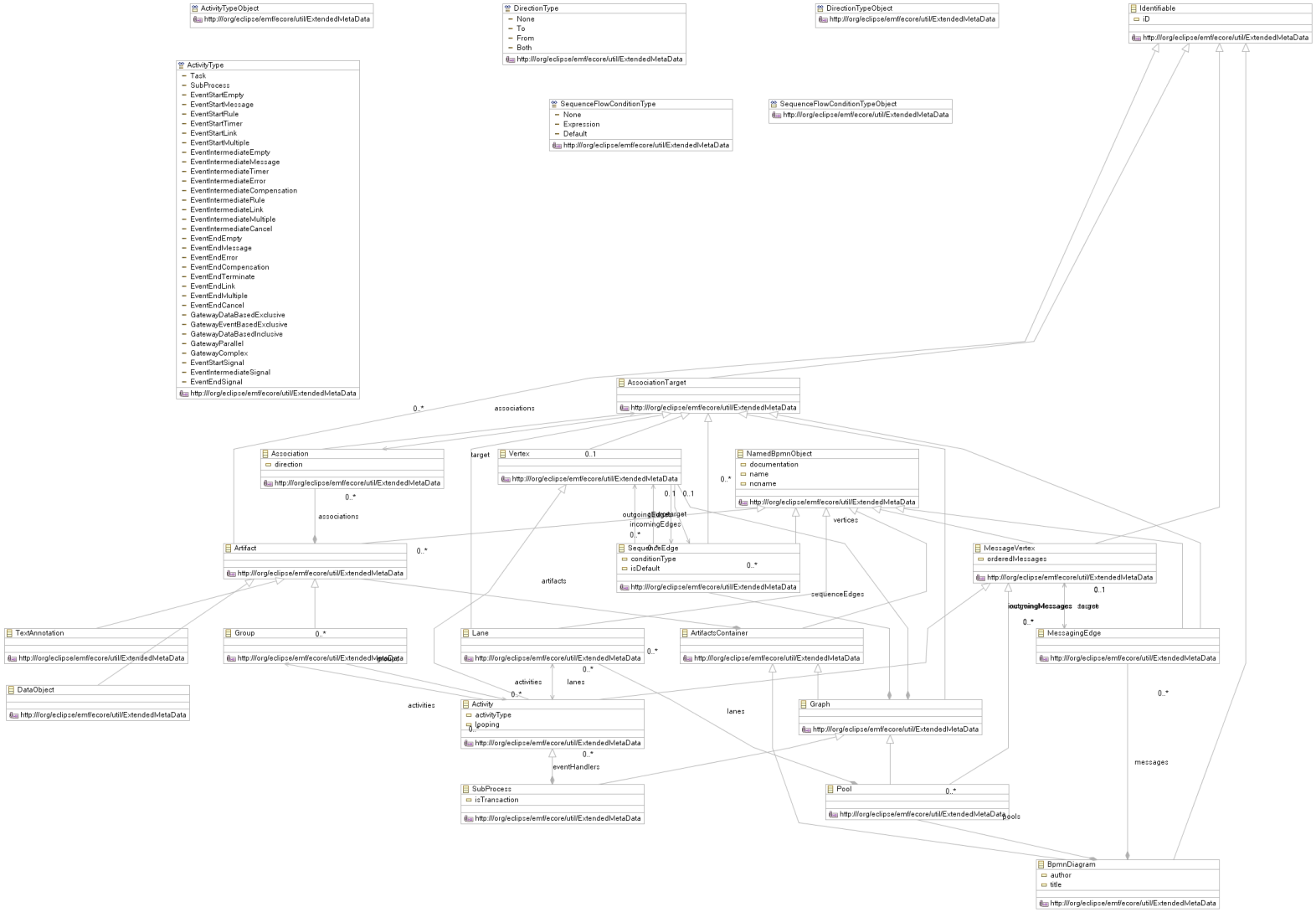

Figure 103: BPMN metamodel

Appendix F Classes used in calculation of allocation of resources

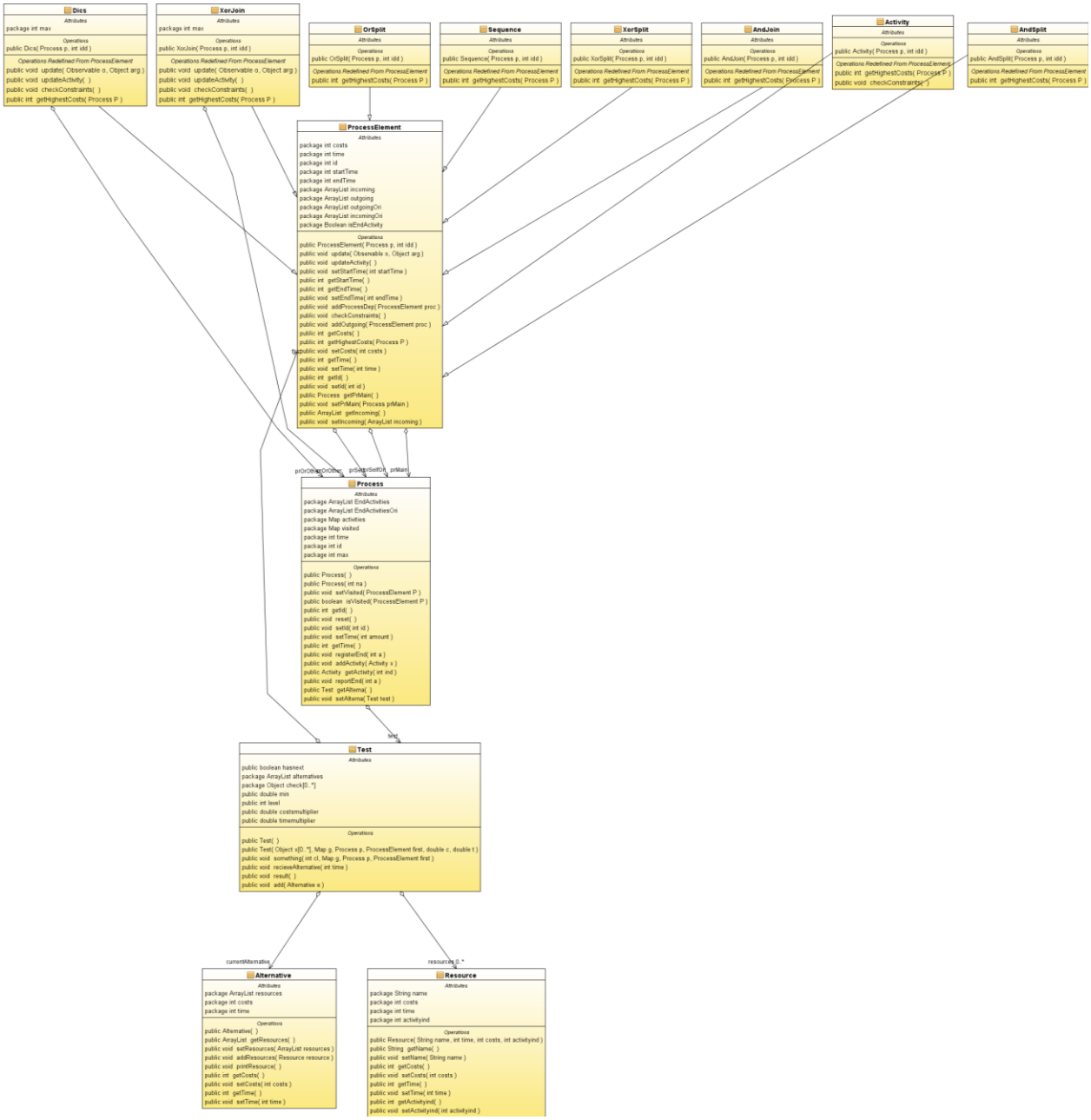

Figure 104: UML class diagram of classes used to calculate optimal allocation of resources

Appendix G Template workflow file first transformation

```
<workflow>
 <bean class="org.eclipse.mwe.emf.StandaloneSetup">
 <platformUri value=".."/>
 </bean>

 <component file='../notation.pa/src-gen/org/logica/pa/parser/Parser.oaw'>
 <modelFile value='platform:/resource/<PATH TO PA-SPEC>'/>
 <outputSlot value='theModel'/>
 </component>

      <component class="oaw.xtend.XtendComponent">
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile value="../testcase/metamodels/panot.ecore"/>
            </metaModel>
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile
value="platform/resource/testcase/metamodels/panotation.ecore"/>
            </metaModel>
            <invoke
value="platform/resource/pa.paconverter.project/src/PaToPa::act(theModel)"/>
            <outputSlot value="model"/>
      </component>

      <component class="oaw.xtend.XtendComponent">
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile
value="platform/resource/testcase/metamodels/panot.ecore"/>
            </metaModel>
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile value="../testcase/metamodels/process.ecore"/>
            </metaModel>
            <invoke
value="platform/resource/pa.paprocess.generator/src/PaToProcess::act(model)"/
>
            <outputSlot value="pamodel"/>
      </component>

      <component id="write" class="org.eclipse.mwe.emf.Writer">
            <useSingleGlobalResourceSet value="true"/>
            <modelSlot value="pamodel"/>
            <uri value="platform:/resource/<WRITE PATH>.xmi"/>
      </component>
</workflow>
```

Appendix H Template workflow file second transformation

```
<workflow>
      <bean class="org.eclipse.mwe.emf.StandaloneSetup">
            <platformUri value=".."/>
      </bean>
      <component id="xmiParser"
            class="org.openarchitectureware.emf.XmiReader">
            <modelFile value='platform:/resource/<PATH TO PREVIOUS RESULT>'/>
                        <metaModelPackage value="process.ProcessPackage"/>
            <outputSlot value="model"/>
            <firstElementOnly value="true"/>
      </component>
            <component class="oaw.xtend.XtendComponent">
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile value="process.ecore"/>
            </metaModel>
            <metaModel id="mm"
class="org.openarchitectureware.type.emf.EmfMetaModel">
             <metaModelFile value="bpmn.ecore"/>
            </metaModel>
            <invoke value="<ProToBpmn>||<ProToBpmnSimple>||<
ProToBpmnYawl>::act(model)"/>
            <outputSlot value="pamodel"/>
      </component>
      <component id="write" class="org.eclipse.mwe.emf.Writer">
            <useSingleGlobalResourceSet value="true"/>
            <modelSlot value="pamodel"/>
            <uri value="platform:/resource/<WRITE PATH>.bpmn"/>    />
      </component>
</workflow>
```

Appendix I Result transformation test casus

This appendix shows the results of the transformation of the PA-specification that are used in section 6.1. The highlighted activities in the diagrams represent the workflow pattern. All the results are as expected.

I.1 Sequence

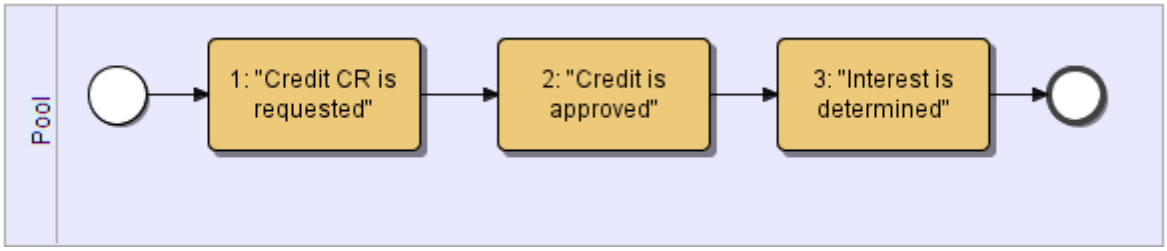

Figure 105: Result of the transformation of the PA-specification of section 6.1.1

I.2 Parallel split

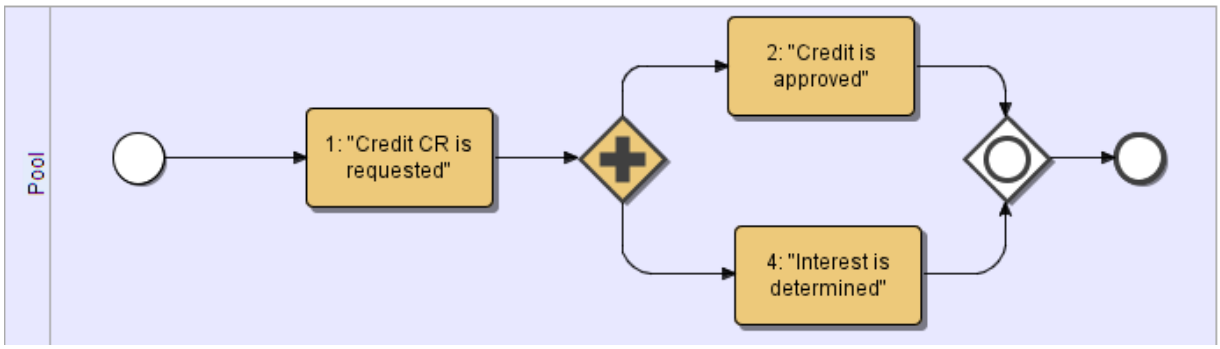

Figure 106: Result of the transformation of the PA-specification of section 6.1.2

I.3 Merge

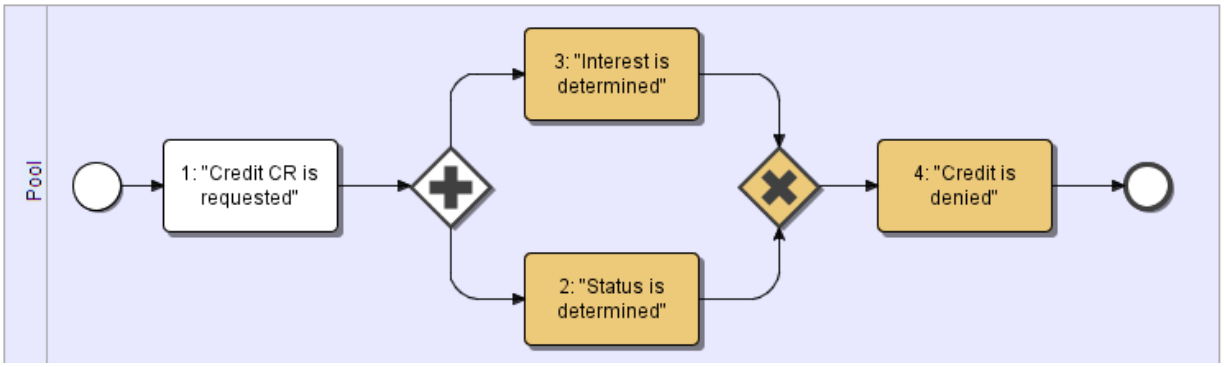

Figure 107: Result of the transformation of the PA-specification of section 0

I.4 Synchronization

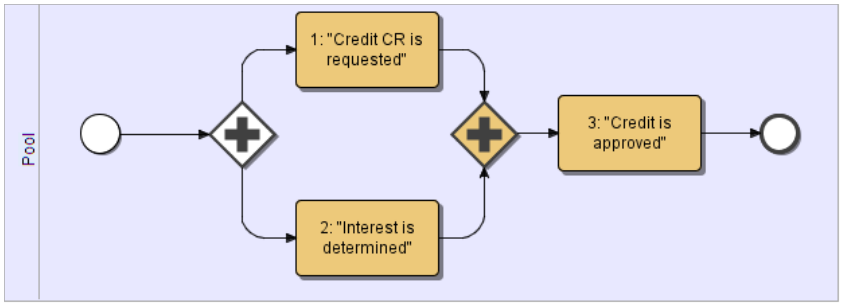

Figure 108: Result of the transformation of the PA-specification of section 0

I.5 Exclusive choice

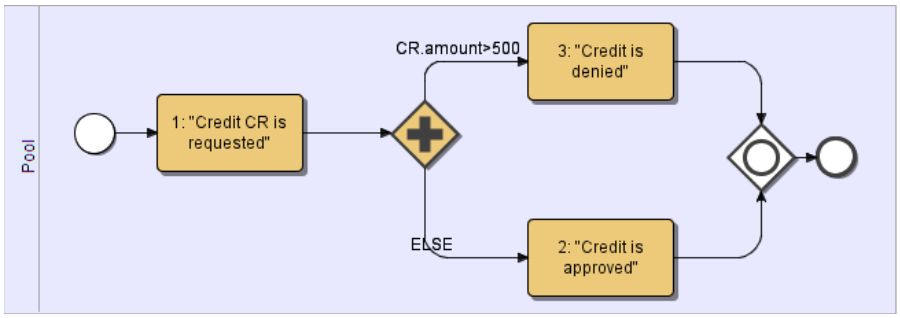

Figure 109: Result of the transformation of the PA-specification of section 6.1.5

I.6 Multiple choice

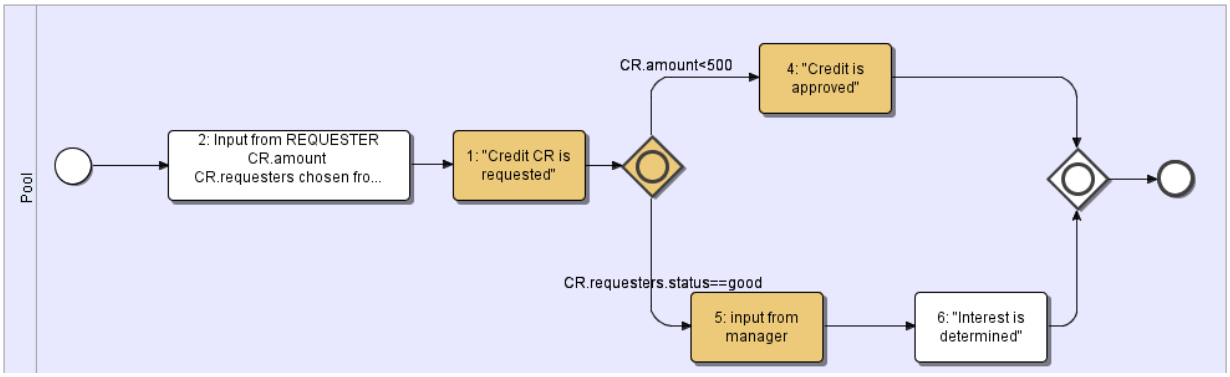

Figure 110: Result of the transformation of the PA-specification of section 6.1.6

I.7 Discriminator

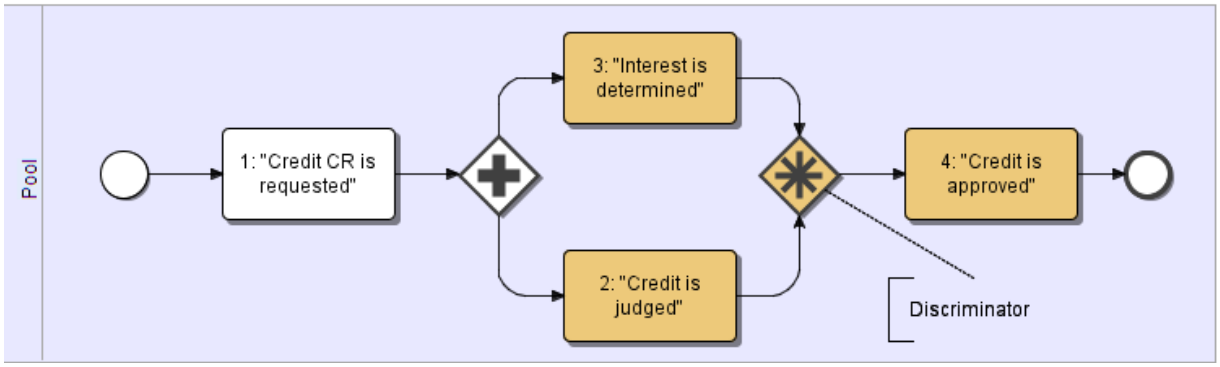

Figure 111: Result of the transformation of the PA-specification of section 0

I.8 Cycle

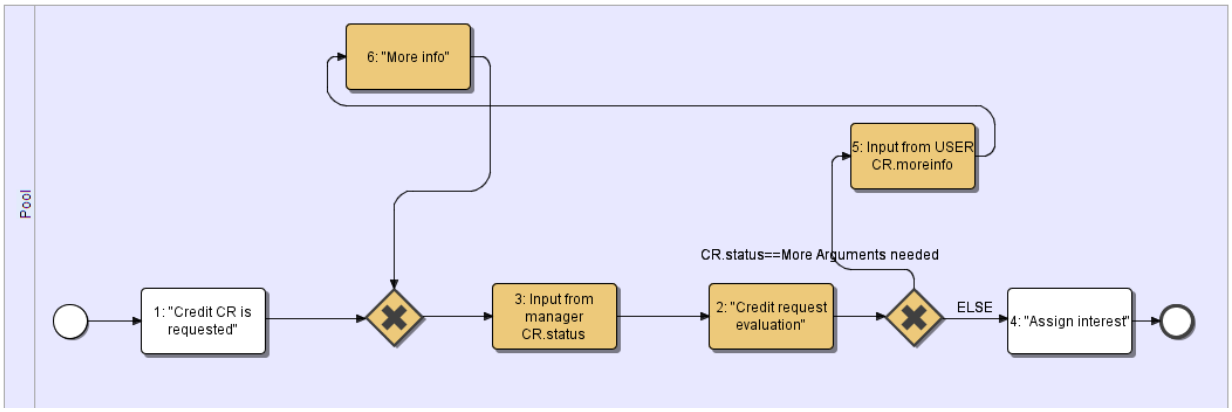

Figure 112: Result of the transformation of the PA-specification of section 6.1.8