Automatic generation of optimal business processes from business rules

Bas Steen, Luís Ferreira Pires and Maria-Eugenia Iacob
Centre for Telematics and Information Technology
University of Twente
Enschede, the Netherlands
b.steen@alumnus.utwente.nl, l.ferreirapires@ewi.utwente.nl, m.e.iacob@utwente.nl

Abstract— **In recent years, business process models are increasingly being used as a means for business process improvement. Business rules can be seen as requirements for business processes, in that they describe the constraints that must hold for business processes that implement these business rules. Therefore, in principle one could devise (automated) transformations from business rules to business processes. These transformations should improve the quality (correctness) of business processes, by imposing their conformance to the applicable business rules, and should also allow business processes to be optimized for some additional requirements, like, for example, resource allocation, performance and costs. The objectives of this paper are twofold: to investigate the automated support to transform business rules into optimized processes, and to assess the suitability of model-driven technologies (metamodeling and transformations) and tools to implement these transformations.**

SBVR; BPMN; UML; Model transformation; MDA; MDE; business processes; business rules

I. INTRODUCTION

In recent years, business process models are increasingly being used as a means for business process improvement, allowing different stakeholders possibly with different background (for example, from business and technology) to communicate. Business process models can also serve as an input for a workflow management system (WFMS), which can be used to support and enforce the modelled business processes [1]. Business process models are often expressed in an imperative language, so that process models tend to contain information on how things should be done, without describing why things should be done in that specific way. In this sense, a business process is already a solution to some business requirements. This creates a problem when requirements change, since then the whole business process model has to be reconsidered to make sure that it is still the most suitable solution to the renewed set of the requirements.

Business rule languages are declarative languages that allow the specification of the constraints that apply to business objects during business activities [2], without prescribing how and by whom these constraints are imposed. Therefore, business rules can be seen as requirements for business processes. Since business processes are supposed to implement business rules, we may expect that it should be possible to devise (automated) transformations that facilitate the generation of business processes from business rule specifications. Therefore, in an automated approach, the mapping decisions made during the transformation process are documented and repeatable. However, it is reasonable to expect that different business processes (with different non-functional characteristics) may exist, which all implement the same set of rules. Nevertheless, only one (or a few) of these processes may be optimal for some optimization criteria. This means that given some particular optimization criteria, automated transformations enable rules to be transformed into processes consistently and in the same way. Thus, instead of generating and analyzing the business process model manually, by using automated transformations one can just rerun the transformations whenever the rules, the optimization criteria or rule data change.

The objectives of this paper are twofold: (1) to develop automated support to transform business rules into optimized processes and (2) to assess the suitability of model-driven technologies (metamodeling and transformations) and tools to implement these transformations. Since the gap from business rules to business processes can be significant, in this work we address the challenges of developing a transformation chain that bridges this gap, and finding proper tool support to implement this transformation chain. This demonstrates the feasibility of our approach in realistic situations.

This paper reports on the development of automated transformations from business rules to optimized business processes. Business rules are normally defined in terms of business objects and their properties, which characterize the business vocabulary of these rules. This business vocabulary is also used in the business process model, which operates on the same objects and properties. Our transformation approach makes the business vocabulary explicit by extracting it from the business rule specifications and representing it as a domain model. Our approach has taken into consideration that many languages exist to express business rules and processes, so that the approach can be adapted relatively easily to support different languages. In this paper we discuss how our approach has been applied to transform rules conforming with SBVR [3] into business processes described using BPMN [4]. The domain model extracted from the SBVR rules is represented as UML class diagrams [5].

The remainder of this paper is structured as follows: Section II gives an overview of our transformation approach. Section III presents the metamodels we have used in this

work. Section IV discusses the metamodel transformations we have developed. Section V discusses our optimization transformation. Section VI discusses the steps we have taken to validate our approach. Section VII discusses some related work. Finally, Section **Error! Reference source not found.** includes our final remarks and outlines directions for future work.

II. TRANSFORMATION APPROACH

The goal of our approach is to transform business rules into optimized business processes using model-driven technologies. Because of the significant gap between business rules and business processes, performing the transformation in one step would lead to a large and complex transformation specification. Therefore, we decided to apply a transformation approach that consists of a sequence of four transformation steps. Following the principle of separation of concerns, we isolated the process optimisation step from the other transformation steps between business rules and business processes. In this way we made it possible to change any of the steps without affecting the others. Furthermore, the transformation approach we developed can be applied with minimal adaptation to other source and target languages (or optimization criteria). The only constraint imposed by our approach is that these languages have a metamodel represented in some metametamodel, such as MOF [6] or Ecore [7]. In this paper we show evidence that this approach is sound by discussing how business rules represented using SBVR can be translated to business processes represented using BPMN and a domain model represented using UML class diagrams. In [8] we reported on the generation of BPMN process specifications from business rule specifications written in a proprietary language developed by Logica (http://logica.nl) called PA-notation, by using the same transformation approach.

Fig. 1 shows the steps of our transformation approach. Since we start our transformation chain with a business rules specification written in textual form, our initial step consists of parsing this business rules specification to generate a business rule model. This text-to-model transformation can be considered as a preparation step in our transformation chain, and it is denoted in Fig. 1 as transformation T0.

The next step in our approach (transformation T1) is to transform the business rules model into a process model that represents the dependencies between activities and a set of domain concepts. This intermediary process model conforms to an internal (process) metamodel we called *process metamodel* (PMM). The PMM only captures domain concepts and basic relations between activities (dependencies and sequences), so that its instances are not yet suitable to be directly transformed to, for example, a BPMN process model. In the next step (transformation T2), the intermediary process model undergoes an optimization algorithm. In the last step (transformation T3), the optimized process model is converted to a business process model and a domain model that conform to the metamodels of the business process language and domain modeling language being applied. By using the PMM as an intermediate metamodel, transformations can be defined from any specific rule language metamodel to the PMM, and from the PMM to any specific business process language metamodel and any specific domain model language metamodel, making the approach amenable to adaptation to other languages or evolving metamodels. Since the PMM is an intermediary metamodel in our transformation approach, it plays the role of an abstract platform, as introduced in [9].

III. METAMODELS

Our transformation approach requires a metamodel for the business rule language, the business process language and the domain model language. Furthermore, in our transformation approach we generate an intermediary model that complies with the PMM (process metamodel). These metamodels are briefly discussed in the sequel.

A. SBVR metamodel

The SBVR specification [3] defines a MOF-based SBVR metamodel, so the starting point of our implementation of the transformations chain defined in Fig. 1 should be an instance of this metamodel. To make the project manageable in the time available, we defined a simplified version of the SBVR metamodel (sSBVRMM), which covers the most relevant elements of the original SBVR metamodel.

Fig. 2 shows an excerpt of this simplified metamodel, which has been used for the representation of rules in the beginning of our transformation chain. We refrain from explaining this metamodel in detail here, since this metamodel is extensively discussed in the SBVR specification.

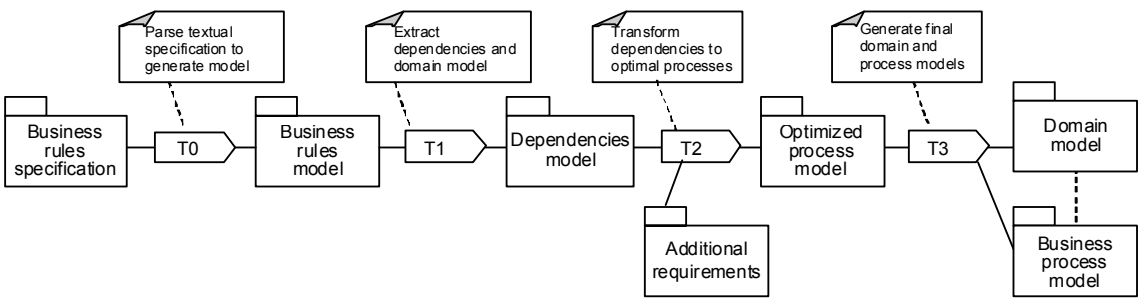


Figure 1. Our transformation approach.

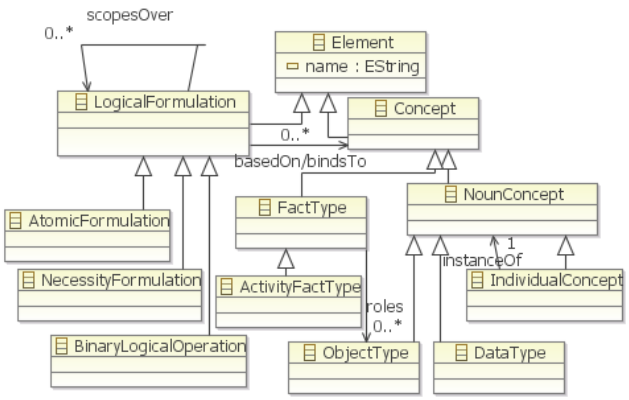


Figure 2. Excerpt of the simplified SBVR metamodel.

We defined two elements in the sSBVRMM that are not in the SBVR specification, namely DataType and ActivityFactType.

The SBVR metamodel has elements to represent the instances of the Set, Number, and Text primitive types (Section 8.7 in [3]). However, there is no element to express the fact that a certain instance of an ObjectType is a Number or some Text. For example, it is not possible to indicate that an instance of the ObjectType Name is some Text. Therefore, as suggested in [10], we have introduced the DataType element. Using this element we can express that an instance of a certain ObjectType is always of a certain DataType by specifying that the ObjectType specializes that DataType. We have also decided not to use the elements of the SBVR metamodel to represent instances of the primitive types Text and Number, but to represent them as IndividualConcepts that have an instanceOf relation with DataType Text and DataType Number, respectively.

In our transformation approach we must be able to determine whether a rule refers to a business activity. In related work this problem has been solved by (1) mapping all BinaryFactTypes to activities [11], or (2) mapping all AssociationFact-Types that use a transitive verb to activities [12]. Solution 1 has the serious drawback that it results in loss of expressiveness with respect to the representation of the domain model. Solution 2 is reasonable in terms of expressiveness, but it requires a sentence analyzer to work. Since we did not have the time necessary to implement a sentence analyzer in our project, this solution could not be applied. Therefore, we decided to introduce our own Binary-FactType element (ActivityFactType).

B. Process metamodel

Fig. 3 shows an excerpt of our process metamodel (PMM). The PMM is an auxiliary metamodel that has elements to represent domain concepts and the dependencies and sequences between activities. In the sequel we briefly discuss the most important elements of this metamodel.

Class is used to represent a business object. A Class can specialise and/or be a categorisation of another Class. A categorization can be distinguished from a specialization because it can change over the lifetime of the object. If a Class is the categorisation of another Class then it can also have a category condition (catCond), which is a collection of relations that indicate when a certain Class is the categorization Class. *Attribute* is used to represent properties of business objects. *Instance* is used to represent that a certain element is an instance of a Class.

S*equenceElement* is used in a process and can have several InputGateways and several OutputGateways. A SequenceElement is either an Activity or a Gateway. *Activity* has a Boolean meta-attribute indicating if the Activity is potentially an initial activity, and an actor indicating who should perform the activity. Each activity can have several triggers and preconditions and can update several variables. *Gateway* has a type (split, join, And, Or, Xor, Disc) and an associated condition.

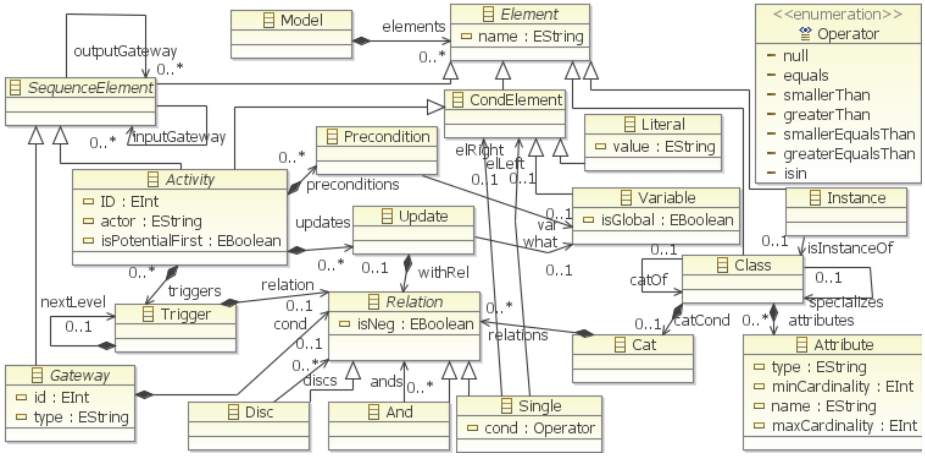


Figure 3. Excerpt of our process metamodel.

CondElement can be used to specify conditions and is either a Variable, a Literal or an Activity. *Variable* is used to capture the information that is used or created in the process. The isGlobal Boolean meta-attribute is used to indicate that a certain Variable is available at the beginning of the process. *Literal* is used to represent values of the primitive types String and Int.

Trigger is used to capture certain conditions that, when true, enable the activity to which the trigger is related. Each trigger has a relation. *Relation* is used to capture the different structures of triggers. *Single* represents the condition that must be true in order for the trigger to be true, *Discriminator (Disc)* means that one of several other relations must be true in order for the trigger to be true, and *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 with which the variable is updated is stored in the withRel attribute.

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 variable updated by another activity.

C. BPMN and UML metamodels

In our transformations we used the BPMN metamodel provided by the Eclipse BPMN modeler [13]. Fig. 4 shows an excerpt of this metamodel.

Activity is used to represent the different process flow objects. Each Activity must be contained in a *Pool* and possibly a *Lane*. The *ActivityType* of an Activity indicates the type of the flow object (Task, Gateway, etc.). Activities are connected to each other through the *SequenceEdge* element. The *Association* element is used to relate Activities to *Artifacts*. *DataObject* and *TextAnnotation* are Artifacts that represent the Data Object and Annotation [4], respectively.

We used the UML2 metamodel provided by the Eclipse Model Development Tools project [14] to represent the domain model. We refer to [5] for details about this metamodel.

IV. METAMODEL TRANSFORMATIONS

In this paper we report on our efforts to transform SBVR business rules to optimized BPMN process models. The SBVR specification [3] defines the SBVR Structured English (SBVRSE) notation that allows rules to be written in structured English. Although SBVRSE is informative, it plays the role of a concrete syntax for SBVR. We decided then to start from a textual representation of SBVR rules in SBVRSE in our transformation chain. A strong argument for this decision is that business people are more likely to define and understand rules written in a nearly natural language such as SBVRSE.

Therefore, in order to validate our approach we start with SBVRSE specifications and generate BPMN process models and a domain model represented as a UML class diagram, by applying the transformation chain depicted in Fig. 1.

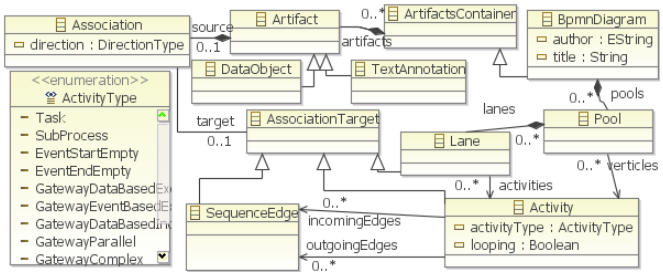


Figure 4. Excerpt of the BPMN metamodel.

Below we introduce a simple running example, and discuss the transformations that we have designed and implemented.

A. Running example

Our running example concerns rules that could apply for a car rental business. It consists of a small set of rules that state that a car renter must make a reservation, and if a reservation is made the booking clerk must assign a status to the reservation. Below we show how these rules are expressed using SBVRSE.

```
Renter;
Booking_clerk;
Reservation;
Status specializes Text;
Reservation has Status;
Renter is_served_by Booking_clerk;
Booking_clerk sets Status;
Renter hasreserved Reservation Type:Activity;
Booking_clerk sets Status Type:Activity;
It is necessary that Renter hasreserved Reservation;
It is necessary that if Renter hasreserved Reservation
then there is a Reservation;
It is necessary that if Renter hasreserved Reservation
then Booking_clerk sets Status;
It is necessary that if Booking_clerk sets Status then
Reservation has Status;
```

B. Parsing the SBVRSE specification (T0)

Our transformations have been developed based on the following assumptions regarding the business rule specification:

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

```
If Status of Renter is 'Good' then Rental isapproved
If Status of Renter is 'Good' then Rental isdenied
```

In order to be able to parse SBVRSE specifications we used the xText framework [15] to generate an SBVRSE

metamodel from the SBVRSE grammar described using the EBNF-like notation supported by xText. Unfortunately, the SBVRSE metamodel instance generated from parsing the rules written with SBVRSE could not be directly mapped to the elements of the sSBVRMM, so that an additional transformation was necessary to perform this mapping.

The main discrepancies between the generated SBVRSE metamodel and the sSBVRMM concern how AtomicFormulations are defined in the sSBVRMM. In an AtomicFormulation, one can refer to ObjectTypes, IndividualConcepts and FactTypes. However, in SBVRSE the words in the fact types can be split into different parts in several ways, as indicated below:

- By combining them with keywords used to identify other types of LogicalFormulations, like, for example, in Person *has* **at most** 5 Children.
- By combining them with keywords that have no direct meaning, like, for example, in Amount *of* **the** CreditRequest.
- By using a chain of fact types, like, for example, in End_date *of* RentalPeriod *of* Rental.

If a fact type is split, the parser cannot determine the fact type that should be used. Therefore, in the SBVRSE metamodel an AtomicFormulation is just a sentence that can contain keywords (a, an, the), verbs, references to ObjectTypes, and instances of the DataTypes Number and Text. In order to retrieve an instance of the sSBVRMM from an instance of the generated SBVRSE metamodel, we have defined a transformation that maps the sentences of the SBVRSE specification to AtomicFormulations in the SBVR model. Fig. 5 shows how our running example is expressed in sSBVRMM.

C. *Extracting domain concepts and dependencies from rules (T1)*

In transformation T1 of our approach (see Fig. 1) we extract dependencies and the domain concepts from business rules. Below we discuss how we can extract dependencies and domain concepts from a SBVR model and represent them as a process model that complies with the PMM. This transformation consists of two parts: (1) the ObjectTypes, FactTypes and NecessityFormulations that range over UniversalQuantifications are transformed into domain concepts and (2) all the other LogicalFormulations are transformed into process dependencies.

1) Domain concepts: For extracting the domain concepts, the mapping of the sSBVRMM element to PMM elements is performed as follows:

- *ObjectType* is mapped to a Class, except if the ObjectType specialises a DataType, and *specialization* relations are mapped to the specializes attribute, except if the specialization relation is an ObjectType that specialises a DataType.
- *UnaryFactType* is mapped to a relation between an Attribute and a Class, where the name of the Attribute is the verb name of the UnaryFactType, the type is Boolean and the Class is created using the ObjectType referred to by role1.
- *IsOfPropertyFactType* is mapped to a relation between an Attribute and a Class where the Class is created using the ObjectType referred to by role1 and the name and type of the Attribute is the name of the ObjectType referred to by role2. If the ObjectType referred by role2 specializes a DataType then the type is the name of that DataType.

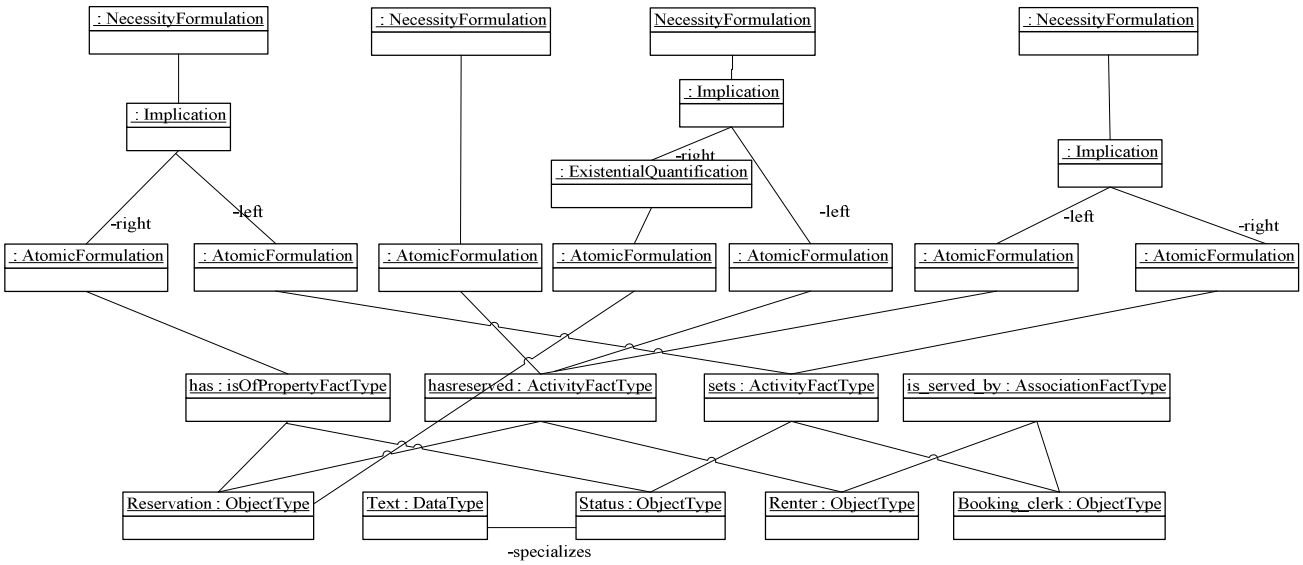


Figure 5. Running example in terms of sSBVRMM elements.

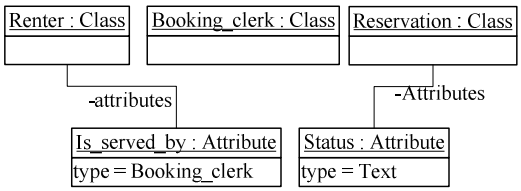


Figure 6. Mapping of the running example onto PMM elements.

- *AssociationFactType* is mapped to a relation between an Attribute and a Class where the Class is created using the ObjectType referred to by role1 and the name of the Attribute is the verbName of the AssociationFactType and the type is the name of the ObjectType referred to by role2.
- *IsOfCategoryFactType* is mapped to a Class that has a catOf attribute, where the category Class is created using the ObjectType referred to by role1, and the Class that is in the catOf attribute is created using the ObjectType referred to by role2.
- *UniversalQuantification* is mapped to a cardinality of the Attribute that is created from the mapping of the FactType that is based on the AtomicFormulation that is scoped over by this UniversalQuantification.

Fig. 6 shows how our running example is expressed in PMM elements related to domain concepts.

2) Process dependencies: For transforming Logical-Formulations of the SBVR model to process dependencies, we identified two alternative mappings, which depend on where the LogicalFormulation is defined: (1) in the condition part of implications, or (2) in the consequent part of implications or in a NecessityFormulation.

- LogicalFormulations used in the *condition* part are mapped to a Relation. A Conjunction is mapped to an And Relation, a Disjunction is mapped to a Disc Relation and all other types of LogicalFormulations are mapped to a Single Relation;
- LogicalFormulations used in the *consequent* part of Implications (or in NecessityFormulations) are basically mapped to an Activity. If one of the LogicalFormulations scopes over another LogicalFormulation that is an AtomicFormulation, which is mapped to a Variable, then that Variable is set as either the Variable that is updated by the Activity or as a Precondition of the Activity, depending on the context of the LogicalFormulation.

AtomicFormulation is mapped to an Activity, Variable or a Literal depending on the FactType it is based on. If the FactType is a *Unary*-, *Association*- or *IsOfPropertyFactType* then the AtomicFormulation is mapped to a Variable. If the FactType is an *ActivityFactType* then the Atomic-Formulation is mapped to an Activity, where the actor of the Activity is indicated by the ObjectType referred to by role1 and the name of the Activity is a combination of the verb name and the name of the ObjectType referred to by role2. If the LogicalFormulation that contains the AtomicFormulation is a NecessityFormulation then there are no conditions related to this Activity, and the Activity is potentially an initial Activity. Therefore, in this case the IsPotentialFirst attribute is set to true. If the AtomicFormulation is not based on a *FactType* and has only one binding NounConcept then the AtomicFormulation is mapped to a Literal in case the NounConcept is an IndividualConcept that is an instance of a DataType, and it is mapped to a Variable in all the other cases.

Fig. 7 shows how our running example is expressed in the PMM elements related to process dependencies.

D. Generating the final process model and the domain model (T3)

The final step in our transformation method is to transform the optimized process model to a BPMN model and the domain concepts to an UML model. The mapping between our process metamodel and the BPMN metamodel is relatively simple.

- *SequenceElement* is mapped to BPMN Activity, where the type of the sequence element determines the BPMN activityType. For example, an Activity is mapped to the BPMN activityType Task and a Gateway of the type And is mapped to the BPMN activityType GatewayParallel.
- *Connection* between sequence elements, i.e., the input- and outputGateway, is mapped to a BPMN SequenceEdge. If there is a condition that belongs to the connection between two sequence elements, it is mapped to the name of the edge.

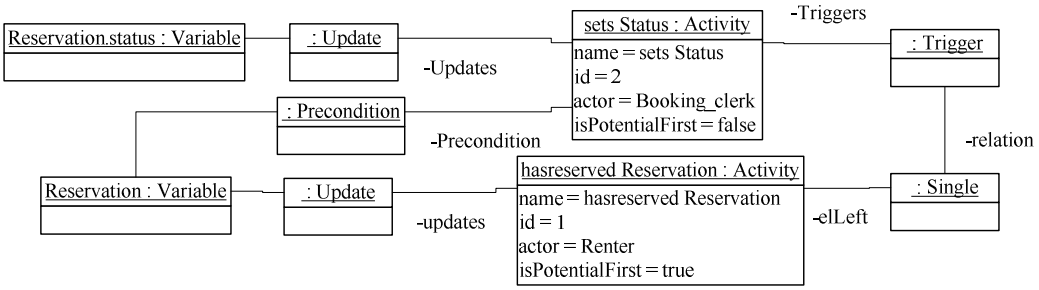


Figure 7. Mapping of the running example onto PMM elements.

- *Variable* is mapped to a DataObject, and a connection between Activity and Variable, i.e., the precondition and update attribute, is mapped to an Association.

The mapping between our process metamodel and the UML metamodel is also relatively simple:

- *Class* is mapped to a UML Class. *Instance* is mapped to an UML Instance specification and *Attribute* is mapped to a UML Association or a UML Property depending on the type and the name of the attribute. *Cardinality* of an attribute is mapped to a cardinality between UML classes.
- A *categorization* or *specialization* relation is mapped to an UML generalization.

Fig. 8 shows the BPMN diagram and the UML class diagram generated in our running example.

V. Optimisation Transformation (T2)

The transformation of the dependencies process model (DPM) to an optimized process model (OPM) depends on the optimization criterion that is used, such that for each optimization criterion a different optimization transformation must be implemented. We have tested our approach with an optimization transformation that is based on the criterion that optimizes the tradeoff between cost and performance of resources. Given which activities can be performed by which resources at which costs and with which performance (completion time), the OPM is the process model with the minimal weighted sum of the expected total costs and completion time.

Our optimization transformation is performed in two steps: (1) determine the optimal sequence of activities and (2) determine the optimal allocation of resources.

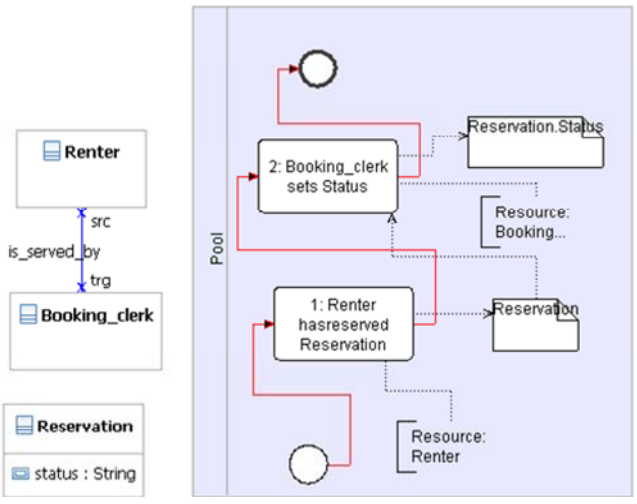


Figure 8. UML and BPMN diagrams generated for the running example.

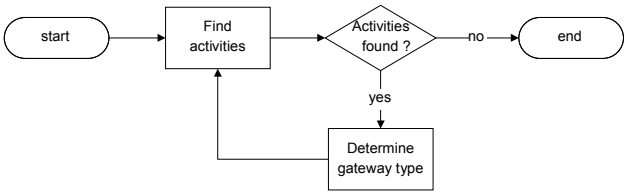


Figure 9. Algorithm to extract optimal sequence from dependencies.

A. Determine the optimal sequence of activities

Fig. 9 shows the algorithm we have defined to determine the optimal sequence of activities. In an optimal sequence, each activity is enabled as soon as the business rules allow this. Each step of the algorithm is further explained below.

1) Find activities: In this step all activities that can be enabled based on all previously enabled activities are determined.

In the beginning of a process there are no previously enabled activities, but some activities (initial activities) are unconditionally enabled, allowing the process to start. Therefore, the algorithm starts by determining the initial activities of the process. An activity is an initial activity if the attribute potentialFirstActivity of the activity is set to true and the activity has no preconditions, or if the activity has no preconditions and none of the attached triggers need other activities in order to be evaluated.

In case there are previously enabled activities, for each previously enabled (source) activity the algorithm determines if another (target) activity can be enabled. A source activity enables a target activity if the source activity is the last activity that is needed to enable all the preconditions of the target activity, and the target activity has no unmet triggers, or if the source activity contributes to enabling one of the target activity triggers, and the target activity does not have any unfulfilled precondition. A precondition is fulfilled if all the variables needed in the precondition are updated by previously enabled activities, and a trigger is enabled if all the condition elements used in the trigger are available. For example, in the case of a variable, it must be global or previously updated by an activity, and in case of an activity, it must be a part of all previously enabled activities.

The result of this algorithm step is a function that relates each enabled activity to the set of activities needed to enable this activity. If the function is empty no activities can be enabled and the algorithm ends. If the function is not empty we have to determine which gateway type must be used to connect the activities to each other.

2) Determine gateway type: The gateway type to be used depends on the amount of necessary and enabled activities.

If one source activity enables one target activity then we connect them to each other through a sequence gateway. If one source activity enables several target activities we connect them to each other through a split gateway, and the gateway type is determined by the

conditions that are used to enable the target activities. If several source activities enable one target activity we connect them to each other through a Join gateway, and the gateway type is determined by the relation type of the relation connected to the trigger. If several source activities enable several target activities, we split the function into several cases of several source activities that enable one target activity. If there are target activities but no source activities, the target activities are initial activities. In this case we create an empty activity (start node) as the source activity and connect the start node to the initial activities using the rules mentioned above in order to determine the gateway type. This start node is added to the sequence to ensure that the process has exactly one start point, which is required for determining the total costs. After the gateway is determined, the algorithm continues with the step to find activities.

Fig. 10 shows how our running example is expressed in PMM elements after the optimal sequence of activities is determined.

B. Determine the optimal allocation of resources

The optimal allocation of resources is determined based on the sequence of activities obtained in the previous step and the library of resources. The optimal allocation of resources is the allocation in which the value of the optimization criterion is minimal. We start this step by identifying all possible alternative resource configurations. After that, for each alternative the completion time and total costs are calculated.

The *total completion time* is calculated using a push mechanism in which the contribution of each activity is considered by emulating the activities. First, the initial activities are notified that they can start calculating their end time. These activities calculate their own end time and notify all directly connected outgoing sequence elements that they can start calculating their end time as well; this process continues recursively. For activities, Or, And, and Xor splits the end time is calculated as the end time of the sequence elements that sends the call plus its own completion time, and for joins the end time is calculated as the highest incoming end time of all the sequence elements that are joined. When all activities have updated their end time, the total completion time can be determined as it is the highest end time of all activities.

The *total costs* are calculated using a pull mechanism that starts with the start node. The start node calculates its costs as the costs of the outgoing sequence element. If the sequence element is an activity then its costs are the costs of this activity plus the costs of the outgoing sequence element. If the sequence element is a Sequence, And join, Xor join, or Disc then its costs are the costs of the outgoing sequence element. If the sequence element is an And or Or split, then its costs are the sum of the costs of all outgoing sequence elements. If the sequence element is a Xor split, its costs are the highest costs of all outgoing sequence elements. The total costs are determined by the costs given by the start node calculated recursively in this way.

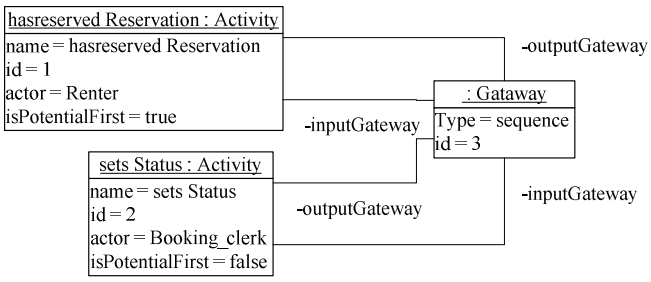


Figure 10. Mapping of the running example onto PMM elements.

Once the completion time and total costs are determined we make those values comparable by using the weighted summation technique [16].

When the value of the optimization criterion is determined for all alternatives, we choose the alternative that has the lowest value for the criterion and add the resources of that alternative to the process model. In our running example this would mean that in Fig. 10 the actor field of each activity is updated to contain the name of the resource used in the optimal alternative.

VI. VALIDATION

In order to check the feasibility of our approach we have implemented it in a transformation tool and validated it with eight small test cases that represent the basic workflow patterns of [17] and two large case studies: (1) the case study used in [8] and (2) the EU-rent case study [2], which is a well known case study in the business rules domain. Below we discuss our prototype and show a fragment of the EU-rent case study.

For the implementation of our approach we relied on tools from the Eclipse Modeling Tools distribution, combined with the BPMN modeler plug-in [13]. We used Ecore [7] to specify the metamodels, and Ecore tools to generate Java classes from Ecore metamodels. We used the Xtext framework [15] in combination with Xtend [18] to implement the parsing and Xtend [18] to implement transformations T1 and T3. We implemented the optimization algorithm in Java and used MWE [19] to define and execute the transformation chain. We used the SOA tools BPMN modeler plugin to create a BPMN diagram from a BPMN model, and the UML plug-in [14] to create a UML class diagram from the UML model.

Fig. 11 shows a fragment of the EU-rent case study expressed in SBVRSE. This specification defines the business objects and their properties, and constraints that state that (1) a renter must make a reservation, (2) if a reservation is refused then the booking clerk should send a rejection and (3) if a reservation is accepted then the booking clerk should send a confirmation. A reservation is refused if the renter of the rental is on the blacklist. The library of resources used in this case study is also shown in Fig. 11.

After running our transformations on the SBVRSE specification we obtained an UML model and a BPMN model. These models are transformed into diagrams using the tools mentioned above.

Fig. 12 shows the UML class diagram and BPMN diagram that have been generated. Both the class diagram and the BPMN diagram correspond to our expectations. However, the BPMN diagram is not completely correct as the Or gateway that is used after the activity Renter reserves Rental should be a Xor gateway, because the outgoing conditions of this Or gateway are mutually exclusive. In terms of execution behavior this is not a problem, as the possible execution paths remain the same. However, it is a problem for the total costs calculation, since the costs calculations for a Xor and Or Split are different. To ensure the correct total cost calculation, the process model has to be adjusted by hand before the optimal allocation of resources is determined. A future improvement of our method could be to make it reason about the conditions used in the gateways in order to determine the correct gateway type, in this way avoiding manual adjustments.

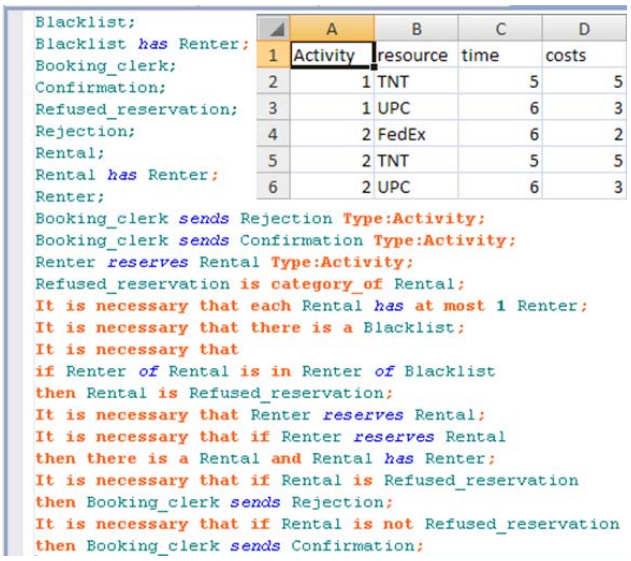
VII. Related work

Raj et al. [12] and Eder. et al. [11] suggest approaches to transform SBVRSE to UML activity diagrams and BPMN diagrams, respectively. Both approaches focus on finding a correct process sequence. Our approach is similar to the approaches in [12] and [11] in that we also use the "if condition then action" construct of business rules to determine the process dependencies. Similarly to the approach of [12], we also derive the sequence of activities by finding the initial activity and then recursively looking for the next activity until there are no more activities left. The novelty of our approach is that we also address optimization and that we are also able to determine the optimal sequence when complex rules are used, in which the condition part is used in multiple rules, or the action part uses a condition that has not been enabled yet. Therefore, our approach is applicable in more realistic situations.

Other related work focuses on automatically transforming business rules into a domain model or vice-versa, such as in [12], [20] and [10]. The mapping we propose between SBVRSE and (eventually) UML class diagrams is inspired by the mappings suggested by these authors. Furthermore, our sSBVRMM is similar to (but not identical) to the simplified SBVR metamodel used in [20]. The main difference is that we have added the ActivityFactType, BinaryLogicalOperation, and all elements that specialize the BinaryLogicalOperation.

VIII. Final remarks

In this paper we presented an approach to automatically transform business rules to business processes, and reported on the application of our approach for transforming business rule specifications written in SBVRSE into optimized business process models modeled in BPMN, and a domain model represented as an UML class diagram.



	A	B	C	D
1	Activity	resource	time	costs
2	1	TNT	5	5
3	1	UPC	6	3
4	2	FedEx	6	2
5	2	TNT	5	5
6	2	UPC	6	3

Figure 11. EU-rent case study input (SBVRSE specification, library of resources)

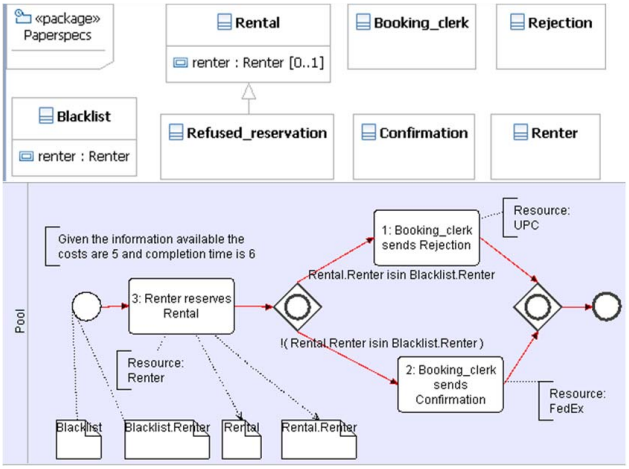


Figure 12. EU-rent case study output(UML Class and BPMN diagram

Because of the significant gap between business rules and business processes, our transformation approach consists of chain of transformation steps that bridge this gap. After the SBVRSE specification is parsed to an SBVR model (an instance of the SBVR metamodel), the dependencies between activities imposed by business rules and the domain concepts are extracted from this SBVR model. The resulting process model subsequently undergoes an optimization algorithm and is finally transformed to BPMN and UML class diagram models.

This specific transformation chain has been implemented in a transformation tool using model-driven technologies provided by the Eclipse community, and has been validated using several test cases and larger case studies. Although our transformation chain still has some limitations, with this experiment we have demonstrated the

suitability of currently available model-driven technologies to implement our transformation approach.

The most noticeable limitations of our approach are that (1) we cannot determine the completion time and total costs if the process contains loops, (2) only one optimization criterion has been implemented so far and (3) we still cannot reason about the conditions used in gateways.

Our transformations are correct by construction but not formally verified. An interesting topic for future work is to extend the approach with a formal verification of the correctness of our transformations. Other possible extension is to increase the expressiveness of the process metamodel and of the related transformations, for example, to make them suitable to express and transform time triggers, which are present in many business processes.

REFERENCES

[1] H. A. Reijers and W. M. P. van der Aalst, "The effectiveness of workflow management systems: Predictions and lessons learned," International Journal of Information Management, vol. 25, pp. 458-472, 2005.

[2] D. Hay and K. A. Healy, "Defining business rules ~ What are they really?," The Business Rules Group 2000.

[3] OMG, "Semantics of Business Vocabulary and Business Rules (SBVR) v1.0," 2008.

[4] OMG, "Business Process Modeling Notation, V1.1," 2008.

[5] OMG, "UML Version 2.2." vol. 2009, 2009.

[6] OMG, "Meta Object Facility (MOF) Core Specification Version 2.0," Object Management Group, 2006.

[7] Eclipse, "Eclipse Modeling Framework Project (EMF)." vol. 2009, 2009.

[8] B. Steen,"Generation of optimal business processes from business rules," M.Sc. Thesis, University of Twente, Enschede, 2009.

[9] J. P. Almeida, R. Dijkman, M. Sinderen van, and L. Ferreira Pires, "On the Notion of Abstract Platform in MDA Development," in Eighth IEEE International Enterprise Distributed Object Computing Conference Monterey, California, USA: IEEE Computer Society Press, 2004, pp. 253-263.

[10] J. Cabot, R. Pau, and R. Raventós, "From UML/OCL to SBVR specifications: A challenging transformation," Information Systems, vol. In Press, Corrected Proof, 2009.

[11] R. Eder, A. Filieri, T. Kurz, T. J. Heistracher, and M. Pezzuto, "Model-transformation-based Software Generation utilizing Natural language notations," in 2nd IEEE International Conference on Digital Ecosystems and Technologies, 2008. DEST 2008. , 2008, pp. 306-312.

[12] A. Raj, T. V. Prabhakar, and S. Hendryx, "Transformation of SBVR business design to UML models," in Proceedings of the 1st conference on India software engineering conference Hyderabad, India: ACM, 2008, pp. 29-38.

[13] Eclipse, "BPMN Modeler," 2009.

[14] Eclipse, "Model Development Tools (MDT) UML2." vol. 11-12-2009, 2009.

[15] Eclipse, "Xtext," 2009.

[16] M. Herwijnen, "Weighted summation," 2006.

[17] N. Russell, A. T. Hofstede, W. M. P. v. d. Aalst, and N. Mulyar, "Workflow control-flow patterns: a revised view," BPMcenter, Technical report 2006.

[18] Eclipse, "Xpand," 2009.

[19] Eclipse, "Modeling Workflow Engine," 2009.

[20] M. Kleiner, P. Albert, and J. Bézivin, "Parsing SBVR-Based Controlled Languages," in Model Driven Engineering Languages and Systems. vol. Volume 5795: Springer Berlin / Heidelberg, 2009, pp. 122-136.