Modeling Service Orchestrations with a Rule-enhanced Business Process Language

Milan Milanović¹, Dragan Gašević², Gerd Wagner³, Vladan Devedžić¹

¹University of Belgrade, Serbia
²Athabasca University, Canada
³Brandenburg Technical University at Cottbus, Germany

Abstract

Business process modeling has been a promising direction in developing service compositions, including both service orchestrations and choreographies. This paper fully focuses on the problem of modeling service orchestrations. Despite many promising aspects of using business process modeling (BPM) languages for modeling service orchestrations, this paper aims to demonstrate that: i) best practices (workflow patterns) for control flows (primary concern of service orchestrations) are not fully covered in present languages; ii) complete service compositions cannot be completely generated from business process models; and iii) BPM languages have limited support for representing logical expressions, business vocabularies, and business rules, which severely limits their flexibility and expressivity. To address these challenges, we have integrated business rule modeling constructs of the REWERSE Rule Markup Language (R2ML) with the Business Process Modeling Notation (BPMN), resulting in our rBPMN proposal.

1 Introduction

Various application areas (e.g., supply chain systems) with rapidly growing demands for collaboration motivated a need for developing more flexible software architectural styles. Service-oriented architectures (SOAs) are considered a main contender in this race by offering many important characteristics such as stateless, loose-coupling, reusability, and interoperability. Publishing and discovery of services are the key prerequisites for exchanging and integration of business logic of potential collaborating parties. Once discovered, the key challenge is to compose services in an organized whole which can accomplish requested business goals and which follows business rules and policies of collaborating parties.

Another key issue is the perspective to the service composition. Service orchestrations and choreographies are two widely recognized types of service compositions. Service orchestrations are mainly looking at the problem from the perspective of one of the collaborating parties by focusing on the flow of control of the business process at. Workflow patterns are best practices and evaluation framework for comparison of orchestration languages. Service choreographies, on the other hand, look at the composition from a global perspective through the lenses of interaction of services. Similar to workflow patterns, service-interaction patterns are used as best practices for service choreographies and comparison of choreography languages. In this paper, we primarily focus on the problem of service orchestration, while about the problem of service choreographies we reported on elsewhere [12].

The research problem, which this paper looks at, is related to the development of service orchestrations. The use of business process modeling as a basis for service composition, in general, and orchestrations, in particular, is the most promising research direction [31]. Modeling of service compositions is primarily based on the use of process-oriented languages such as Business Process
Modeling Notation (BPMN). While the research community provided many promising research results in the context of business process-oriented modeling of service orchestrations, there are still some key challenges open, which this paper tries to address, namely: i) separation of business process models from domain models and business vocabularies. This causes the problems related to the generation of complete service descriptions and service compositions from business process models. Basically, we need a way to connect business vocabulary the business process is based on, and definitions of message types used in a business process; ii) lack of a formal language for defining business conditions. Languages such as BPMN allow for defining conditions (e.g., for branching decision points), but they do not define a executable condition language based on the underlying business vocabulary; and iii) limited support for dynamic changes of business processes. By employing rules in a process allows us to dynamically change the control flow of a process, and to enable defining processes in a declarative way, that can be changed by changing only rules.

Research on integration of business rules with business process modeling demonstrates many benefits [4][7][15], which seem to be promising to address the abovementioned challenges. Although there have been many attempts to integrate business rules and languages, there has not been a comprehensive solution that integrates four main types of rules (reaction, production, integration and derivation) systematically by following the principles of software language engineering. To address this problem, we here propose an integration of a rule modeling language (R2ML – REWERSE Modeling Language) with BPMN at the level of their metamodels (Sect. 3). Once defined, we will focus on how such a rule-enhanced business process modeling language can be used in modeling service orchestrations by analyzing workflow patterns (Sect. 4) and concrete business process scenarios (Sect. 5). Before concluding the paper, we compare our proposal with the related work in Sect. 6.

2 Background

In this section, we describe basic concepts of service orchestrations, BPMN and R2ML languages, as well as some aspects of integration of rules and processes.

2.1 Service Orchestrations

There are presently two types of service compositions: orchestrations and choreographies. In this paper, our focus is on orchestrations. According to the W3C’s Web service glossary: “An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.” Orchestration is the interaction between services at a message level controlled from one party. That is, unlike choreographies which look at the service interaction from a global perspective, orchestrations represent a perspective of one of the collaborating parties at a service composition at hand. Such a business process can result in a “long-lived, transactional, multistep process model” [1]. There is presently a standard for service orchestrations called Web Services Business Process Execution Language (WS-BPEL) language [1] or for short BPEL. In the context of orchestration modeling, BPMN [13] and UML activity diagrams are two typically approach. In fact, associated with the actual BPMN specification at OMG, there is a partial mapping defined between BPMN and BPEL.

2.2 Business processes: BPMN

BPMN is the OMG’s standard language for modeling business processes [13]. Business process models expressed in BPMN consist of a set of modeling elements. BPMN includes three types of ‘flow objects’ defining behavior: Events, Activities and Gateways (see Figure 1). Events can be partitioned into three types, based on their position in the business process: start events are used to trigger processes; intermediate events can delay processes, or they can occur during processes [29]; and end events signal the termination of processes. BPMN supports three types of activities: Process, Sub-Process and Task, where the latest two have a graphical representation. Activities are represented by rectangles (with rounded corners). Gateways are used for guiding, splitting and merging control flow. The diamond shaped gateways represent decisions, merges, forks, and joins in the control flow. A gateway can be thought of as a question that is asked at a point in the process. BPMN has data- and event-based XOR gateways, as well as, inclusive, parallel and
complex gateways. BPMN also has connecting objects (represented by a solid arrow), which are used to connect flow objects in order to create a basic skeletal structure of a process. BPMN has a concept called Pool, which represents a participant in a business process.

As BPMN specification does not propose a metamodel for BPMN, we analyzed the existing proposals for the BPMN metamodel [6], [14], [15]. For choosing an appropriate business process metamodel, we decided to use the criteria defined in [9], which we extended with the concepts defined in [24]. We selected the BPMN metamodel proposal given in [14], which covers the highest number of our selection criteria, and we used it as a basis for extensions and modifications (see Sect 3). This proposal uses an explicit BPMN terminology (e.g., the BPMN sequence flow element is represented with the BPMN concept “Sequence-Flow”); it is a much simpler than the BPDM [15] proposal (e.g., much less abstract classes); and its mapping relations to BPEL are clearer than in the case of other proposals. For extensions, this BPMN metamodel has the BPMN Extensibility Model that allows BPMN adopters to extend the specified metamodel in a way that allows them to be still BPMN-compliant.

R2ML is completely built by using model-driven engineering (MDE) principles, which means that the R2ML language definition consists of the three main parts: i) metamodel – an abstract syntax in the Meta-Object Facility (MOF) language; ii) textual concrete syntax – an XML based syntax that facilitates rule interchange; and iii) graphical concrete syntax – a graphical notation suitable for modeling rules in a style similar to software modeling languages, which is defined as an extension of UML and named UML-based Rule Modeling Language (URML) [26]. Validity of R2ML to model and interchange rules has been proven by transformations with many rule-based languages.

Here, we just briefly illustrate the R2ML metamodel for the sake of better understanding of the rBPMN metamodel, while its full reference can be found in [22]. All R2ML rule definitions (e.g., ReactionRule in Figure 2) are inherited from the Rule class. Each type of rule is defined over the R2ML vocabulary, where elements of the vocabulary are used in logical formulas (e.g., Logical-Formula – with no free variables) through the use of Atoms and Terms. An important aspect of R2ML is that it distinguishes between object and data atoms.

R2ML graphical concrete syntax, URML, is an extension of UML class models through the so-called heavyweight UML profiling [26]. All rules, but integrity, are represented by a circle, while integrity rules are represented as OCL invariants. UML class models are used to represent graphically the R2ML vocabulary elements. URML rule circles are connected with UML classes to create logical expressions of rule such as conditions or conclusions. Conditions are also defined through OCL filters, which are based on a part of OCL that models logical expressions, which can be translated to R2ML logical formulas. A complete reference to URML can be found in [26].

2.4 Rules and Processes

Integration of rules in processes and modeling processes by using rules, can be divided into two major research categories: i) fully rule-based; and ii) integration of business rules into process-oriented models (so-called hybrid approaches).

The first group of approaches aims to model business processes fully by using production and reaction business rules. An important work in this group is presented in Knolmayer et al. [7], where the authors demonstrated how workflow patterns
could be fully modeled by means of business rules. There are a few issues with this approach: comprehension of the overall process and relations among its constitutive parts is tedious given that business rules only focus on small parts of business logic; business process execution is fully driven by reasoning algorithms (such as Rete), which might lead to some unexpected behavior hard to determine upfront; there is no effective and unified modeling support of different types of rules; and rules are typically represented in implementation languages, without features for their use in high-level business process modeling languages.

In the second category of so-called hybrid approaches, the above problems are recognized and different methods are proposed for integration of business rules and business process modeling languages. Eijndhoven et al. [1] propose a methodology for identification of variability points in business processes, which can be implemented by production rules. However, that approach is related to implementation of business processes, rather than to the full rule-driven development using all types of rules. Graml et al. [4] focus on identification of three groups of patterns based on business (mainly integrity and derivation) rules for: control flow decisions; data constraints; and process composition. However, neither of these two solutions proposes a systematic definition of a rule-based business modeling language and none of them analyzed that problem in the context of modeling of choreographies.

3 rBPMN metamodel

As rBPMN is a product of integration between BPMN and R2ML, it is defined by importing the elements from the BPMN and R2ML abstract syntaxes (metamodels). BPMN has been selected due to its broad user adoption, comprehensive coverage of business process concepts, and rich experience in use. The selection of R2ML was driven by: need to make use of a proven and rich rule modeling language; previous experience in integrating with software modeling languages; and objective to follow proven principles and standards for engineering software modeling languages.

In Figure 3, we show extension to the Process package of the rBPMN metamodel. RuleGateway is a key element, which we added in the Process package of the BPMN metamodel and which actually relates to R2ML Rules. In this way, we enabled that R2ML Rule (i.e., Reaction, Derivation, Production or Integrity rule) can be placed into a process as a Gateway, but in the same time not to break the R2ML Rule syntax and semantics. We should note here that one rule gateway could have one or more rules attached to it. This is quite important, as in some cases, we need to first derive or constrain some part of the business logic, before being able to perform some other rules such as reaction or production. In Figure 3, we can see that RuleGateway as a Gateway can be connected by using SequenceFlow with other
FlowElements such as Tasks, Events andGateways. This enables us to use rules in differentplaces in rBPMN process models, as shown forcontrol flow patterns in Sect. 4. We also added aRuleCondition concept to show that a rulecondition is directly attached to a RuleGateway in a business process model.

We should note that, in the standard BPMN[13], exists Conditional Event Definition, whichcan be used to attach some expression defined in arule language, but this event type models onlythe behavior of production rules. In addition, it ispossible to use expression attached to the out-going conditional sequence flow when the source is gateway, however, in the standard BPMN, thereis no concrete proposal for a rule language thatcould handle such expressions.

Although rBPMN supports all four types ofrules, we only discuss production and reactionrules due to the space limit. Figure 2 shows extensions of the BPMN Activity package. This packagessupports BPMN tasks to be a triggering or triggeredtask of reaction rules (R2MLTriggeringTask or R2MLTriggeredTask,respectively). This package also contains supportfor sub-processes as a production rule actions forreaction and production rules (R2MLTriggeredSubProcess). Along with support for tasks and subprocesses, we also support eventsand gateways by introducing classesR2MLTriggeringEvent, R2MLTriggeredEvent,R2MLTriggeringGateway, and R2MLTriggeredGateway. By introducing theseconcepts we enabled that an R2ML rule attachedto the rule gateway can be connected to theBPMN process elements. The main advantage ofthis solution is that we can model parts of thebusiness processes by using rule gateways.

As we can have a rule as a valid element in a business process, we also need to have a way toconnect underlying data models to business rules.In rBPMN, we use R2ML Vocabulary as anunderlying data model, so that any BPMN messagecan be represented with an R2ML AtomicEventExpression (see Figure 4). We enabled this byintroducing R2MLMessageType class, which connectsto the AtomicEventExpression class, and theR2MLMessageType class is a subclass of the
StructureDefinition class, used to define an actual structure of a message. The AtomicEventExpression element is connected to the MessageType, which is used to define actual message type. We additionally attached an OCL constraint to the R2MLMessageType class, so that we have the same MessageType connected to the same R2MLMessageType, through connection with an AtomicEventExpression. Here, we have a MessageEventAnnotation class to connect an Event with a MessageFlow in an interaction model.

The StructureDefinition element is used to specify a Message structure. The Message is connected to the StructureDefinition through structure relation, i.e., one Message can have exactly one StructureDefinition. We extended these BPMN elements by inheriting StructureDefinition and we defined its subclass R2MLMessageType that can have one R2ML EventExpression, through relation hasVocEntry. In this way, rBPMN messages can be directly mapped to a ReactionRule’s triggering event or triggered event expression, and later into WSDL descriptions [8].

4 Workflow Patterns

The control flow patterns represent a set of 21 workflow patterns created to show expressivity of workflow management systems [21]. These patterns can be used in business process modeling, and to compare the expressiveness of process languages. This is the reason why we use them to evaluate the expressivity of rBPMN.

As these patterns can be presented in the standard BPMN without using rules, by adding rules we enrich the BPMN models in a way that such business processes can be more precisely described in a declarative way and changed in a design-time or in a real-time by changing only rules without a need to redesign the whole process. By employing rules in a process allows us to dynamically change the control flow of a process. We also use these patterns to evaluate where in a workflow process our rules can be used. As BPMN [13] has a weak support for rule-based gateways, where conditions are usually written in a natural language [18], by adding formal rules, we enable execution of such processes possible on some execution platform, such as BPEL [5].

In the rest of the section, we follow the organization of the patterns in the six groups as proposed by the authors [21]. Due to the space limitation, we only discuss one pattern from each group.

4.1 Basic Patterns

The patterns in this group include elementary aspects of process control. Here, we have five patterns: Sequence, Parallel Split, Synchronization, Exclusive choice, and Simple Merge.

Figure 4. rBPMN data model
The pattern “Exclusive Choice” chooses one of several activities for performing based on a control data. In an example of Figure 6, after the end of an activity1, if the condition specified on a reaction rule attached to the rule gateway R by the predicate and condition is true, activity3 is started to execute. Otherwise, activity2 is started. The negative choice is denoted with a crossed line between R and activity2. Here we introduce activities as events. It should be noted that we used the crossed BPMN outgoing sequence flow from a gateway, which is chosen when the condition of the attached rule(s) evaluates to true. This means that we must have exactly two reaction rules attached to the rule gateway, where the condition is the same, only negated on the second rule. When the first rule condition evaluates to true, the crossed outgoing sequence flow is chosen, and when it evaluates to false, the other sequence flow is chosen (activity2 in this case). This pattern show the main advantage of using rules to choose a flow in a process, as the rule’s condition can be changed in run-time, but also in design-time. If this pattern is realized in the standard BPMN, with an AND-gateway, through sub-Activities or in a context [30], this advantage would not exist.

![Figure 5. The Arbitrary cycles pattern in rBPMN](image)

For this pattern, we have defined the following OCL constraints:

1. The rule gateway must have exactly two reaction rules attached to it.
   
   ```
   context RuleGateway
   inv: rule->size() = 2
   ```

2. One of the outgoing sequence flows must have “Default” value for its `ConditionType`. We use “Default” mark on a crossed sequence flow from a gateway.
   
   ```
   context RuleGateway
   inv: let OutSequenceFlow : SequenceFlow.allInstances()->select(c | c.sourceRef = this)->asSequence()->first() in OutSequenceFlow->exists(e | e.ConditionType = "Default")
   ```

3. The `triggeringEventExpr` and `triggeredEventExpr` of both reaction rules attached to the rule gateway must be of the `Task` type.
   
   ```
   context RuleGateway
   inv: rule->first().triggeringEventExpr.oclIsKindOf(Task) and rule->first().triggeredEventExpr.oclIsKindOf(Task) and rule->last().triggeringEventExpr.oclIsKindOf(Task) and rule->last().triggeredEventExpr.oclIsKindOf(Task)
   ```

4.2 Advanced Branching and Synchronization Patterns

In this group, there are four patterns: Multi-Choice, Multi-Merge, Discriminator, and Synchronizing Merge patterns.

The pattern “Discriminator” models a point in a business process that waits for one of the preceding, possibly parallel activities to complete before starting the subsequent activity. From that moment on, it waits for all remaining preceding activities to complete and “ignores” them. Once all preceding activities have been completed, it
reset itself, so that it can be started again. The pattern “Discriminator” generalizes to the pattern “N-out-of-M-join” where N threads from M incoming transitions are synchronized. This generalized pattern “Discriminator” can be modeled as a reaction rule attached to the rule gateway with a counter counting the number of the rule’s triggering events of the activity type and that have occurred. A model of this pattern is shown in Figure 7. The value of the counter variable is increased when every preceding activity (activity1 ... activityN) ends. The precondition of this pattern is based on the two variables: counter set to 0 (as the input parameter) and count set to the number of the wanted preceding activities. The triggering event in this pattern is a composite choice event expression, composed from one or more activities.

Without the rule gateway, this pattern in the standard BPMN can partially be supported, because this pattern can be supported only in the case of a multiple instance task, and because it is not clear how the IncomingCondition expression on the COMPLEX-join gateway is defined [30].

Figure 7. The Discriminator pattern in rBPMN

For this pattern, we have defined following OCL constraint:

[1] The reaction rule attached to the rule gateway, for its triggeringEventExpr, must have a ChoiceEventExpression; for its triggeredEventExpr must have a Task; and for its postcondition must have a DatatypePredicateAtom.

class RuleGateway
context RuleGateway
inv: rule-->first().triggeringEventExpr.
    oclIsKindOf(ChoiceEventExpression) and
    rule-->first().triggeredEventExpr.
    oclIsKindOf(Task) and
    rule-->first().postcondition.
    oclIsKindOf(DatatypePredicateAtom)

4.3 Structural Patterns

Patterns in this group characterize design restrictions that specific workflow languages may have on the form of a process model that they are able to represent and how these models behave at runtime. They include two main aspects: loops and termination of a process instance. In this group, there are two patterns: Arbitrary cycles pattern and Implicit termination pattern.

The Arbitrary Cycle pattern is a mechanism for allowing sections of a process to be repeated (as a loop). This pattern allows looping that is unstructured or not block structured. The looping part of the process may allow more than one entry or exit point. This pattern is important for the visualization of valid, but complex, looping situations in a single diagram. An rBPMN model of this pattern is shown in Figure 5. This pattern in rBPMN is represented by two reaction rules attached to two rule gateways: the rule gateway (R1) used to split the sequence flow and the rule gateway (R2) used to handle the loop exit condition in a process. The rule gateway (R2) based on its condition decides whether to return the sequence flow to activity3 (loop) or to activity5 (end loop). The triggering event for the rule gateway (R1) is the start event, while for the rule gateway (R2) is the activity4.

This pattern can be supported in the standard BPMN by using Exclusive gateways, but the advantage of rBPMN is that it can more precisely define entry and exit looping conditions, as well as those conditions could be changed at both run-design-time. Instead of a reaction rule attached to the rule gateway (R1), we could use also a production rule, which would be invoked on some predefined condition. Based on that condition, the whole loop process can begin as well without a need to have a triggering event.

The OCL constraints for this pattern are similar to the constraints in “Exclusive Choice”.

Figure 8. The Multiple Instances with no a Priori Runtime Knowledge pattern in rBPMN

4.4 Multiple Instance Patterns

These patterns are used in a process where there are multiple instances of an activity active at the same time for the same process instance. Multiple instances can arise when an activity is able to create multiple instances of itself once the activity
is triggered. That is, a given activity is initiated multiple times as a consequence of receiving several independent triggers, or when two or more activities in the process share the same implementation definition. In this group, there are four patterns: Multiple Instances without Synchronization, Multiple Instances with a Priori Design-time Knowledge, Multiple Instances with a Priori Runtime Knowledge, and Multiple Instances with no a Priori Runtime Knowledge patterns.

In Multiple Instances with no a Priori Runtime Knowledge, the number of instances of a given activity is not known at neither design or run time, until immediately before the instances of that activity type need to be created. Once all instances are completed, an activity of some other type needs to be started. The difference from the pattern “Multiple Instances with a Priori Runtime Knowledge” is that even while some of activity instances are being executed or have already completed new ones can be created.

An rBPMN model of this pattern is shown in Figure 8. The Subprocess is started after the end of activity1. In the Subprocess, activity2 is invoked multiple times, and activity2 along with the reaction rule attached to the rule gateway is used to determine if more instances of activity3 are needed. activity2 can also increase an Entity’s attribute count, so that the number of instances increases. Upon the start of Subprocess, if the condition attached to the rule gateway (and its associated reaction rules, as in the previous pattern) \( counter = count \) is true, the reaction rule with non-negated condition passes to the parallel gateway that executes activity2 and activity3 instances in parallel. The Entity counter attribute is pre set to 0 before entering Subprocess. activity2 checks if more instances of activity3 are needed and records the decision by increasing the count attribute of the type Entity. The completion of activity2 again invokes the rule gateway. This loop continues until all instances of activity3 are finished. Synchronization of multiple instances of activity3 is achieved through the implicit termination of Subprocess when all its instances have completed. This actually means that when all activity3 instances have ended, Subprocess also ends, and the next activity4 is started (or the business process ends).

By employing reaction rules in this pattern, we have been able to support this pattern, which is not the case with the standard BPMN. BPMN lacks features to create new task instances dynamically [30].

### 4.5 State-based Patterns

These patterns define situations for which solutions are most easily accomplished in process languages that support the notion of state. A state of a process instance includes a broad collection of data associated with the current execution including the status of various activities. The patterns in this group reflect the possibility that the business processes could be affected by factors outside of the business process engine. In this group there are three patterns: Deferred choice, Interleaved Parallel Routing and Milestone pattern.

We only discuss the “Milestone” pattern, which enables an activity until a milestone (specific state) is reached. When the milestone is reached, the nominated task can be enabled. Only one activity is selected to be performed, while all activities are completed and the end of the process is reached. According to the example of the pattern “Milestone” in Figure 9, after the start of a process, the sequence flow splits by using the pa-
rallel gateway. The first branch invokes activity1 and the flow reaches the rule gateway (R1), which enables the Entity enabled attribute. The triggering event for the rule gateway (R1) is the activity1. The second branch invokes activity2 and reaches the rule gateway (R2), which then use the Entity enabled attribute for a condition to decide which flow will be selected. In this case, the triggering event for the rule gateway (R2) is the activity2. If the enabled attribute is true, then activity3 is invoked and the sequence flow is returned to the start of the branch. If the enabled attribute is false, activity4 is selected and the second branch ends. After invocation of the rule gateway (R1) in the first branch, activity5 is invoked, and the rule gateway (R3) is reached. The rule gateway (R3) then sets Entity enabled attribute to false, and the first branch ends. When this happens, activity3 in the second branch cannot be invoked anymore, because the condition enabled=true on the rule gateway (R2) is not true. The triggering event for the rule gateway (R3) is the activity5. Each of the three rule gateways has a reaction rule attached to them.

This pattern is not supported in the standard BPMN, as BPM lacks support for states [30], which addressed by reaction rules in rBPMN.

4.6 Cancellation Patterns

These patterns characterize the concept of activity or process cancellation where enabled or active activity instances are withdrawn.

We only discuss the pattern “Cancel Case.” An rBPMN model of the pattern is shown in Figure 10, where the whole business process instance is cancelled. An intermediate event is attached to the boundary of a Subprocess that contains other activities, rather than an activity (see Figure 10). We have a Subprocess that is initiated by a start event and followed by a rule gateway with two reaction rules attached to it. The triggering event for the rule gateway is the start event in the Subprocess. The reaction rules attached to the rule gateway then decides based on their condition, whether to invoke activity1 or to cancel the Subprocess. The Subprocess can also be cancelled when activity1 generates an error during its execution. When the exception is generated, the Fault handler attached to the Subprocess boundary passes the sequence flow to the Exception handler, which is then responsible to handle the exception. In this simple case, the Exception handler is used just to send the message.

This pattern is supported in the standard BPMN [30]. However, the rule gateway enabled us to cancel case not only in the case when activity1 fails, but also if the predefined rule is not satisfied (i.e., due to some expectations of the underlying business logic).

For this pattern, we have defined following OCL constraints:

[1] The rule gateway must have exactly two reaction rules attached to it.
context RuleGateway
inv: rule->size() = 2

[2] The one of the outgoing sequence flows must have the “Default” value for its ConditionType.
context RuleGateway
inv: let OutSequenceFlow : SequenceFlow.allInstances()->select(c | c.sourceRef = this)
- >asSequenceFlow()->first() in OutSequenceFlow->exists(e | e. ConditionType = “Default”)

5 Book request scenario

In order to show how rBPMN can be used to model a service orchestration, we present a case study with a simple book buy request scenario. In this scenario, a Customer requests a certain book from a Bookstore by supplying the requested quantity along with the book name. When such a request is received, the Bookstore returns the information about a book with a price to the Customer whether the book is available. Once the Customer receives this information, the Customer decides if (s)he wants to buy the book based on the book’s offered price. This scenario described is shown as an rBPMN model in Figure 12.
In Figure 12, the scenario begins when the Customer requests information about the book that (s)he wants to buy from a Bookstore. This request is realized by using the Book request task that sends a message from the Customer to the Bookstore (represented by the Bookstore pool). This message request is annotated by the CustomerBookRequest message type, which specifies the required quantity and bookName attributes. When such a request is received by the Bookstore, it is handled by a start message event and a rule gateway. This message type is represented with the cbr name as a variable in the rule gateway’s condition. Then, a reaction rule (see Figure 11) attached to the rule gateway (R1), based on its predefined condition evaluates whether the requested book in the requested quantity is available in the store. We actually have two reaction rules attached to the rule gateway (R1); one with the positive condition, and another with the negated condition. This is done due to the nature of reasoning formalisms of rule languages, where the use of the ELSE statement in rules may lead to the reasoning problems [22][27]. If the book is not available in the Bookstore or there is no requested quantity, the reaction rule with the negated condition is activated and the Send book not available task is preformed and followed by the message sent to the customer. The actual message send is the FaultOrderResponse, which annotates the message flow. This message contains requested book price and quantity. When the customer receives such a message the process ends, and the Customer could have another rule gateway that can be used to ask Customer to repeat request with less quantity, and if he agrees, the request can be repeated.

An important implication of this model is how we can relate it to service orchestrations. The basic message exchange between a Web services is called Message Exchange Patterns (MEP) [28] and MEPs define types (including faults) and order of messages that can be exchanged between services and service requestors. Here we can see that we actually have an In-Out MEP, because this pattern consists of exactly two messages: a message received by a service from some other node, followed by a message sent to the other node. This kind of MEP could return a fault message, and in our case this message is FaultOrderResponse.

If the condition of the reaction rules attached the rule gateway (R1) evaluates to true, another sequence flow is selected, and it goes to the Send book available task, which is used to send the message to the customer that the requested book is available. This message flow is annotated with the ApproveOrderResponse message, which in this case holds price of required book. When this message is received by the customer, using the intermediate message event, the sequence flow goes to the rule gateway (R2) in order to decide whether it will buy the book with returned price. The rule gateway (R2) also have two reaction rules attached to it, one with the positive and another one with the negated condition (resp.price<100). This means that the rule gateway’s outgoing sequence flow will be chosen based on the decision whether the received price is less than 100 units. If this condition is satisfied, the Buy book task is used to send a message to the Bookstore to inform that the customer wants to buy the book. The actual message CustomerBookRequest is sent, but we omit it from Figure 12 for the sake of clarity and space constraint of this paper. The Bookstore, by sending the message after the Send book available task, uses an Event-based gateway to wait for the customer’s decision for 24h (by using intermediate message timer event). If the message confirming that the customer wants to buy the book is not received within 24h, the timeout happens and the process ends. However, if condition defined on the rule gateway (R3) is satisfied, the Buy book task is used to send a message to the Bookstore, and then we again use the reaction rule attached to the rule gateway (R3), to decrease the book quantity for the requested book in underlying data model. After that, the Send book task is used to send the book to the Customer, who receives this message by using the Receive book task, and the

![Figure 11. Reaction rules attached to the rule gateway (R1) in Figure 12 represented in URML](image-url)
process ends.

In this scenario from Figure 12, we can identify three workflow patterns from section 4, where rule gateways are used. The first pattern is Exclusive choice in the Bookstore pool. As described in section 4, we used this pattern to select one of two available activities based on predefined condition. The advantage of our solution is a shared vocabulary, and so the two rule gateways R1 and R3 can access the same BookItem, to check or decrease its quantityInStock attribute value. Additionally, bookName and quantityInStock values used in rules’ condition are not fixed, but can dynamically be changed in each Customer request, by using the information from the CustomerBookRequest message type and the AvailableBookItem class, respectively.

As we already mentioned, this pattern is related to the In-Out MEP, and this is important as we have developed a proof of a concept that transforms such reaction rule based models into complete Web service (WSDL) descriptions [19]. In this approach, a (set of) R2ML reaction rule(s) model(s) a MEP. Triggering events model input messages, while triggered events are output messages. As all event expressions in R2ML have an event (and, thus message) type assigned, we can generate the complete message types (i.e., complexTypes) from our reaction rules. Another important implication of our model is that for each reaction rule in R2ML, we can also generate its implementation in a concrete rule-based language.

In our experiments, we provide full definition several languages (e.g., Drools or Jess) by simulating semantics of reaction rules on production rule engines. We call such rules “how-to-use” rules, as they specify conditions under which a service can be used. We could also assign another rule to the rule gateways from Figure 12 and Figure 11. For example, this could be a derivation rule that derives if a Customer is a registered user based on attributes assigned to the Customer profile in a Bookstore (i.e., subsumption reasoning), and consequently allow an access to the further tasks or not. Figure 11 also indicates the fully established traceability between the elements of rBPMN and R2ML. Namely, all BPMN tasks (Send book available and Send book not available) and messages (ApproveOrderResponse and FaultOrderResponse) have their counterparts in R2ML. The traceability is established through the rBPMN metamodel and the OCL constraints given in the section 4. Yet, the combination of BPMN and R2ML complements each other (e.g., message type definition). More on benefits of using rBPMN in modeling MEPs can be found in [11].

In the Customer pool (Figure 12), we can also recognize the Deferred choice pattern. The Book request task is followed by an Event-based gateway, from which two possible branches could be selected based on the received message event. When the message about the available book is received by an intermediate message event from the Bookstore, we used reaction rules attached to the rule gateway to additionally constraint sequence flow that follows this decision. By doing this, we

Figure 12. Book buy request scenario in rBPMN
introduced a new logic to the branch that is selected by the environment, which is in our case the Bookstore pool.

Finally, in the Bookstore pool, we can see the Sequence pattern. When the message is received from the Customer pool and handled by the intermediate message event, we used the reaction rule attached to the rule gateway to decrease the book quantity in the underlying data model (which is based on the shared vocabulary), and to invoke the Send book task. By using a rule gateway, we also additionally constrained the message flow here. An example of a practical implication here is that the rule gateway could be used to check if Customer has a valid credit card, and to inform the Customer if there is enough money on a credit card or a credit card number is wrong.

6 Discussion

In Table I, we compare rBPMN with languages for workflow and business process modeling. If the language directly supports a pattern through one of its constructs, it is rated with +. If the pattern is not directly supported but can be “mimicked”, it is marked with +/- . Any solution which results in complicated (“spaghetti”) diagrams or coding to compensate semantics is considered as giving no support and is rated with -. The table includes the following languages: Extended AORML [25], UML Activity Diagrams [16], BPEL (Business Process Execution Language for Web Services) [5], and BPMN (Business Process Modeling Notation) [13]. We used the information about BPEL workflow pattern support from [1], for UML and AORML we used analysis from [25], and for BPMN we used analysis from [30].

From Table I, we can see that all languages supported all patterns in the first group Basic control flow patterns. In this group, we introduced rules combined with a rule gateway to more precisely define conditions under which an activity or a sequence of activities will be invoked. By using rules in this group, we enabled for defining and for changing those conditions at both run- and design-time.

The benefits stemming from the usage of rules in the second group of Advanced branching and synchronization patterns are even more obvious. One of the key criteria to support of the Multiple Choice pattern is a need to define the logical condition(s) on outgoing branches, which can be changed in run-time. This is exactly the advantage of rBPMN, because reaction rules are connected to the data model (built on a shared vocabulary) that can be changed at design- or runtime. Additionally, we have supported the Discriminator pattern by using a reaction rule attached to the rule gateway to define conditions under which subsequent activities should be invoked. This pattern is not fully supported in the standard BPMN, because the meaning of the Complex-join gateway, i.e., its IncomingCondition expression, is not clearly supported to be used for this pattern.

In the third group, Structural patterns, we used reaction rules to define entry and exit points of loops, where conditions of these rules can be changed dynamically. rBPMN also enables us to make an ordering in activity invocation by using a shared vocabulary. Regarding the Arbitrary Cycles pattern, rBPMN business process diagrams do not impose any restrictions on the structure of cycles; a business process model may have multiple entry and exit points which are represented as reaction rules.

In Multiple Instances patterns group, rBPMN is crucial especially for patterns with and without a priori runtime knowledge. Reaction rules can dynamically change a number of required instances, as well as to define complex conditions on which an activity should be invoked. For the Multiple instance with a priori design time knowledge pattern, we enabled to precisely define a number of created instances, and to define a condition when all of those instances have completed. The standard BPMN [13] does not have a language for defining conditions. This now added by our rule extension of BPMN.

In fifth group, State-based patterns, by using rules, we supported the Milestone pattern. This pattern is not supported in the standard BPMN, as BPMN lacks the support for states. In this pattern, by using a shared domain model (based on a shared vocabulary) between multiple rules, we allowed that some tasks could be invoked only when they are enabled by some precisely defined condition. Interleaved Parallel Routing is not supported completely in BPMN, nor is it supported in rBPMN. This reason lies in the fact that BPMN does not support interleaving groups and sequences of tasks [21] [30].

The final pattern group introduces two cancelling patterns. rBPMN improves modeling of the Cancel Case pattern, as the use of a rule gate-
way enables for cancelling of a subprocess when either one of its activities fails or a condition of the business logic is not satisfied. That is, we can more explicitly specify the conditions of cancelling business subprocesses not only due some internal performance errors, but also due to some business expectations, which may change during the time.

7 Conclusion

rBPMN provides an integration of a rule modeling language with a process modeling language at the level of their formal language descriptions (i.e., metamodels). Not only are tool developers able to use this as a complete definition of the rule-enhanced process modeling tools, but also they could generate definitions of executable rules, processes, and services. Previous attempts mainly provide integration on the level of concrete syntax (e.g., RuleML and AGFIL-BM) only without precise language definitions [10]. Alternatively, integration was hard coded into implementation of languages [1][20]. Either of these two groups of approaches makes the verification of integrated language expressions very hard due to the differences of the language definition formalisms. rBPMN overcomes this problem through the systematic metamodeling complemented by OCL constraints (as in UML).

While the previous work on the integration of business rules in business process modeling demonstrated main promising benefits, to the best of our knowledge, the proposed solution is the first solution that provides a systematic integration of a rule modeling language with a business process modeling language. This is done by following the principles of MDE and integrating two languages at the level of their metamodels. The resulting language, rBPMN integrates the best of two modeling perspectives – rich process-oriented modeling foundation of BPMN and comprehensive rule and vocabulary modeling foundation of R2ML. By comparing the expressivity of rBPMN with

<table>
<thead>
<tr>
<th>Pattern group</th>
<th>Pattern</th>
<th>UML</th>
<th>BPEL</th>
<th>BPMN</th>
<th>AORM</th>
<th>rBPMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic control-flow</td>
<td>Sequence</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Parallel Split</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Advanced branching and synchronization</td>
<td>Synchronization</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Exclusive Choice</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Simple Merge</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Multi Choice</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Multi Merge</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Structural</td>
<td>Discriminator</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>Multiple Instances</td>
<td>Synchronizing Merge</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>State-based</td>
<td>Arbitrary Cycles</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Implicit Termination</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>MI without synchronization</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>MI with a Priori Design Time Knowledge</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>MI with a Priori Runtime Knowledge</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>MI without a Priori Runtime Knowledge</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Deferred Choice</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Interleaved Parallel Routing</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
<td>-</td>
<td>+/-</td>
</tr>
<tr>
<td></td>
<td>Milestone</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Cancellation</td>
<td>Cancel Activity</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Cancel Case</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
other relevant languages for modeling service orchestrations, we have shown that rBPMN increases the level of support of modeling workflow patterns. This is especially indicative on the patterns that are state based and assume the use of multiple instances of the same activities. Another important benefit of the use of rBPMN for modeling service orchestrations is an increase level of dynamism. That is, flexibility to revise parts of the underlying business logic at run-time. Next, the use of advanced logic-expressions and business rule types (e.g., derivation and reaction) allows for defining more precisely different business goals in the definitions of business processes and workflow patterns (e.g., distinguishing between types of the reasons for business process termination). Finally, rBPMN by connecting vocabularies enables for generating complete service descriptions that the standard BPMN.

In the future work, we are going to report on the precise rules of mappings between rBPMN and concrete service orchestration languages such as BPEL. We will also explain how our rule-enhanced orchestration models can be deployed on the orchestration engines by adding additional level of rule support as per the types of rules defined rBPMN. The present rule engines associated with orchestration engines support only production rules, while in our case we need to support other three types of rules. Due to the space limit, we could not describe the full rBPMN metamodel and all workflow patterns, which will be done in the future reports along with our full study on service interaction and message-exchange patterns.

About the Authors

Milan Milanović is a PhD candidate at University of Belgrade. His main research interests are rules, modeling and metamodeling, Service Oriented Architectures and Enterprise systems. His Internet address is http://milan.milanovic.org.

Dragan Gašević is a Canada Research Chair in semantic technologies and an Assistant Professor at the School of Computing and Information Systems, Athabasca University, Canada. His current research interests are in the area of semantic technologies, software language engineering and service oriented architectures. He can be reached at http://dgasevic.athabascau.ca

Gerd Wagner is a Professor of Internet Technology at the Brandenburg University of Technology at Cottbus, Germany. His research interests include agent-oriented modeling and agent-based simulation, foundational ontologies, (business) rule technologies and the Semantic Web. He can be reached at http://www.informatik.tu-cottbus.de/~gwagner/.

Vladan Devedžić is a professor of computer science at the University of Belgrade, Serbia. His main research interests include software engineering, intelligent systems, knowledge representation, ontologies, Semantic Web, intelligent reasoning, and applications of artificial intelligence techniques to education and medicine. He can be reached at http://devedzic.fon.rs.

References


