Modeling Service Choreographies with Rule-enhanced Business Processes

Milan Milanović
FON-School of Business Administration
University of Belgrade
Belgrade, Serbia
e-mail: milan@milanovic.org

Dragan Gašević
School of Computing and Information Systems
Athabasca University
Athabasca, AB, Canada
e-mail: dgasevic@acm.org

Abstract— **The research community has so far mainly focused on the problem of modeling of service orchestrations in the domain of service composition, while modeling of service choreographies has attracted less attention. The following challenges in choreography modeling are tackled in this paper: *i)* choreography models are not well-connected with the underlying business vocabulary models. *ii)* there is limited support for decoupling parts of business logic from complete choreography models. This reduces dynamic changes of choreographies; *iii)* choreography models contain redundant elements of shared business logic, which might lead to an inconsistent implementation and incompatible behavior. Our proposal – rBPMN – is an extension of a business process modeling language with rule and choreography modeling support. rBPMN is defined by weaving the metamodels of the Business Process Modeling Notation and REWERSE Rule Markup Language. To evaluate our proposal, we use service-interaction patterns and compare our approach with related solutions.**

Keywords - BPMN, business rules, business processes

I. INTRODUCTION

Responding to the increasing demands for developing advanced solutions to the integration of business processes in collaborative information systems, service-oriented architectures (SOAs) emerged as a promising approach. Offering features such as loose-coupling, statelessness, reusability, and interoperability, the key SOA issues are service publishing, discovery, and composition. Considering the present state of SOA, we can witness a need for the development of new software engineering approaches suitable for this new development context. Here, we consider the challenge of software engineering languages for service composition.

Following proven principles of business process modeling, service engineers have prevalently based their approaches on languages such as Business Process Modeling Notation (BPMN) [19]. Such languages offer a suitable way for requirements elicitation from stakeholders, which can (semi-)automatically be bound to the existing services and transformed onto the executable service compositions (i.e., languages such as Business Process Execution Language, BPEL[18]). In the service composition task, we generally have two main approaches [30]: i) service orchestration – composition of service from the perspective of one of the participants. Orchestrations are typically modeled w.r.t. control flows, while workflow patterns (http://www.workflowpatterns.com) are used as best practices and evaluation framework for comparison of orchestration languages; ii) service choreographies – composition of services from a global perspective where service interaction is the primary focus. Similar to workflow patterns, service-interaction patterns (SIPs) are used as best practices for service choreographies and comparison of choreography languages.

The research community has so far mainly focused on the problem of modeling of service orchestrations, while choreographies have attracted less attention [17]. In this paper, we exactly focus on the problem of modeling choreographies in order to address to the following challenges: *i)* service choreography models are not well-connected with the underlying vocabulary/domain models. This reduces type safety; *ii)* there is limited support for decoupling parts of business logic (e.g., constraints and process decisions) from complete choreography models. This reduces dynamic changes of choreography models; *iii)* service choreography models typically contain redundant elements of the shared business logic, which might lead to the inconsistent implementation and potentially incompatible behavior.

To address the above challenges, in this paper, we propose a rule-based approach to modeling of service choreographies. Our solution, rBPMN, is based on the integration of two proven languages – BPMN and REWERSE Rule Markup Language (R2ML) at the level of their metamodels. In the rest of the paper, we first introduce background knowledge, including, business process, service choreographies, metamodels of BPMN and R2ML, and challenges for integration of business rules and processes. Next, we introduce our proposal through a detailed description of the metamodel (Sect. III), which is followed by the discussion about their support for SIPs (Sect. IV) and a case study (Sect. V). Before concluding the paper, we discuss the related work (Sect. VI).

II. BACKGROUND

A. Business processes

According to [30] "a *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations." BPMN is the OMG's standard for business process modeling [19]. BPMN has a graphical notation, but has no standardized metamodel.

B. Service Choreographies

According to the Web service glossary, "a choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order

to perform a task to achieve a goal state." This definition is further specialized in the Web Service Choreography Description Language (WS-CDL) candidature recommendation [12]: "a *choreography* defines re-usable common rules that govern the ordering of exchanged messages, and the provisioning patterns of collaborative behavior, as agreed upon between two or more interacting participants." The key aspect is messages exchanged among collaborating parties, which agree on rules for ordering of messages.

There are two approaches to modeling of choreographies [7]: interaction models and interconnected interface behavior models (interconnection models). Interaction models are built up of basic interactions (message exchanges), while interconnected interface behavior models define control flows of each participant a choreography. The representatives for the interaction model approach are WS-CDL [12], Let's Dance [31] and iBPMN [7]. In BPMN 2.0 [20] these models are known as choreography models. Interconnected interface behavior models can be represented in BPMN [19] and BPEL4Chor [6][8]. In BPMN 2.0 these models are known as collaboration models. As rBPMN can be used for both modeling approaches, we here show basic interaction patterns expressed in rBPMN as both interaction and interconnected interface behavior models. Interaction models avoid the problems of redundancy and inconsistent behavior, while interconnection models are suitable for each individual party in their implementation of choreographies.

Similar to software patterns, the area of business process modeling has also discovered several types of patterns in different perspectives of business processes. The most known example is workflow patterns, which are used in consideration of control flow-oriented business process models (i.e., orchestrations). Service-interaction patterns (SIPs) are used in the case of service choreographies. Presently, there are 13 SIPs [1] divided into the four groups based on the three criteria: i) number of parties involved (i.e., bilateral and multilateral interactions); ii) maximum number of messages exchanged in an interaction (i.e., single or multi-transmission interactions) and iii) in the case of two-way interactions, whether the receiver of the response is in the same time as the sender of request (round-trip and routed interactions).

C. Business Process Modeling Notation (BPMN)

BPMN represents an OMG adopted specification [19] whose intent is to model business processes. Business process models are expressed in business process diagrams. Each business process diagram consists of a set of modeling elements. BPMN includes three types of flow objects which defines behavior: Events, Activities and Gateways. *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 [20] [30]; and end events signal the termination of processes. A BPMN *activity* can be atomic or non-atomic. 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 1.2 specification [19] does not propose a metamodel for BPMN, we analyzed the existing proposals for the BPMN metamodel [11] [20] [21]. For choosing an appropriate business process metamodel, we decided to use the criteria defined in [15], which we extended with the concepts defined in [26]. We selected the BPMN metamodel proposal given in [20], which covers the highest number of our selection criteria, and we used it as a basis for extensions and modifications. 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 [21] 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 allowing BPMN adopters to extend the specified metamodel in a way that allows them to be still BPMN-compliant.

D. Business Rules: REWERSE I1 Rule Markup Language

REWERSE I1 Rule Markup Language (R2ML) is a general rule language [22]. It is originally designed to support rule interchange (thus, markup in its name), but it is also a comprehensive rule modeling language. R2ML is completely built by using model-driven engineering 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. In fact, its graphical syntax is defined as an extension of UML and named UML-based Rule Modeling Language (URML) [27]. Validity of R2ML to model and interchange rules has been proven by transformations with many rule-based languages[22].

A business rule is a statement that aims to influence or guide behavior and information in an organization [25]. There are different categories of business rules such as [28] integrity, derivation, reaction, and production. R2ML defines all of these four types of rules and provides modeling concepts for defining vocabularies. Here, we just briefly illustrate the R2ML metamodel for the sake of better understanding of the rBPMN metamodel. The full reference of R2ML can be found in [22]. All R2ML rule definitions (e.g., ReactionRule in Figure 1) 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., LogicalFormula – with no fee variables) through the use of Atoms and Terms. An important aspect of R2ML is that it distinguishes between object and data atoms.

URML, graphical syntax of R2ML, is an extension of UML class models through the so-called heavyweight UML profiling [27]. All rules, but integrity, are represented by a

circle, while integrity rules are represented as the Object Constraint Language (OCL) invariants. UML class models are used to represent graphically the vocabulary elements. URML rule circles are connected with UML classes to create logical expressions of rule like conditions or conclusions. Conditions are also defined via 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 [27].

E. Integration of rules and processes

Integration of rules and processes research can be divided into two major 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 business rules. This is usually done with production and reaction rules. An important work is presented in Knolmayer et al. [13], where the authors demonstrated how workflow patterns could be fully modeled by business rules. Similarly, JBoss' Drools are used for business process modeling, and consequently for regulating service compositions. However, 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 (e.g., 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, and weakly in high-level process modeling languages.

The second category (i.e., hybrid approaches) recognizes the above problems and propose methods for integration of business rules and business process modeling languages. Eijndhoven et al. [4] propose a methodology for identification of variability points in business processes, which can be implemented by business (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. [10] 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.

III. rBPMN Metamodel: Rule and Choreography Modeling

rBPMN is a result of integration of BPMN and R2ML. 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.

A. rBPMN Metamodel: Rule Modeling Support

The rBPMN metamodel is defined by importing the elements from the BPMN and R2ML metamodels. In Figure 1, we show extension to the Process package of the rBPMN metamodel. *RuleGateway* is an element, which we added in the Process package of the BPMN metamodel and which actually relates to R2ML *Rule*s. 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 1, we can see that *RuleGateway* as a *Gateway* can be connected by using *SequenceFlow* with other *FlowElements* such as *Tasks*, *Events* and *Gateways*. This enables us to use rules in different places in rBPMN process models, as shown in Sections 4 and 5. We also added a *RuleCondition* concept, which is used to show rule condition directly attached to the *RuleGateway* in a business process diagram.

We should note that, in the standard BPMN [19], there is Conditional Event Definition, which can be used to attach some expression defined in a rule language, but this event type models only the behavior of production rules. In addition, it is possible to use expression attached to the outgoing conditional sequence flow when the source is gateway, however, in the standard BPMN, there is no concrete proposal for a rule language that could handle such expressions.

Although rBPMN supports all four types of rules, we only discus production and reaction rules due to the space limit. In Figure 1, we show extensions of the BPMN *Activity* package. In this package, we have supported BPMN tasks to be triggering or triggered task of the R2ML Reaction rules (*R2MLTriggeringTask* or *R2MLTriggeredTask* class, respectively). This package also contains support for subprocess as a production rule actions for R2ML Reaction and Production rules (*R2MLTriggeredSubProcess* class). Along with support for tasks and subprocesses, we added support for events and gateways by introducing classes *R2MLTriggeringEvent*, *R2MLTriggeredEvent*, *R2MLTriggeringGateway* and *R2MLTriggeredGateway*. By introducing these concepts we enabled that an R2ML rule attached to the rule gateway can be connected to the BPMN process elements. The main advantage of this solution is that we can model parts of business processes by rule gateways.

We can have a rule as a valid element in a business process, but we should also have a way to connect underlying data models to business rules. In rBPMN, we use R2ML Vocabulary as an underlying data model, so that any BPMN message can be represented with an R2ML *AtomicEventExpression* (see Figure 2). We enabled this by introducing *R2MLMessageType* class, which connects to the *AtomicEventExpression* class, and the *R2MLMessageType* 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*, via connection with an *AtomicEventExpression*. Here we have a *MessageEventAnnotation* class, which is used to connect an Event with a *MessageFlow* in an

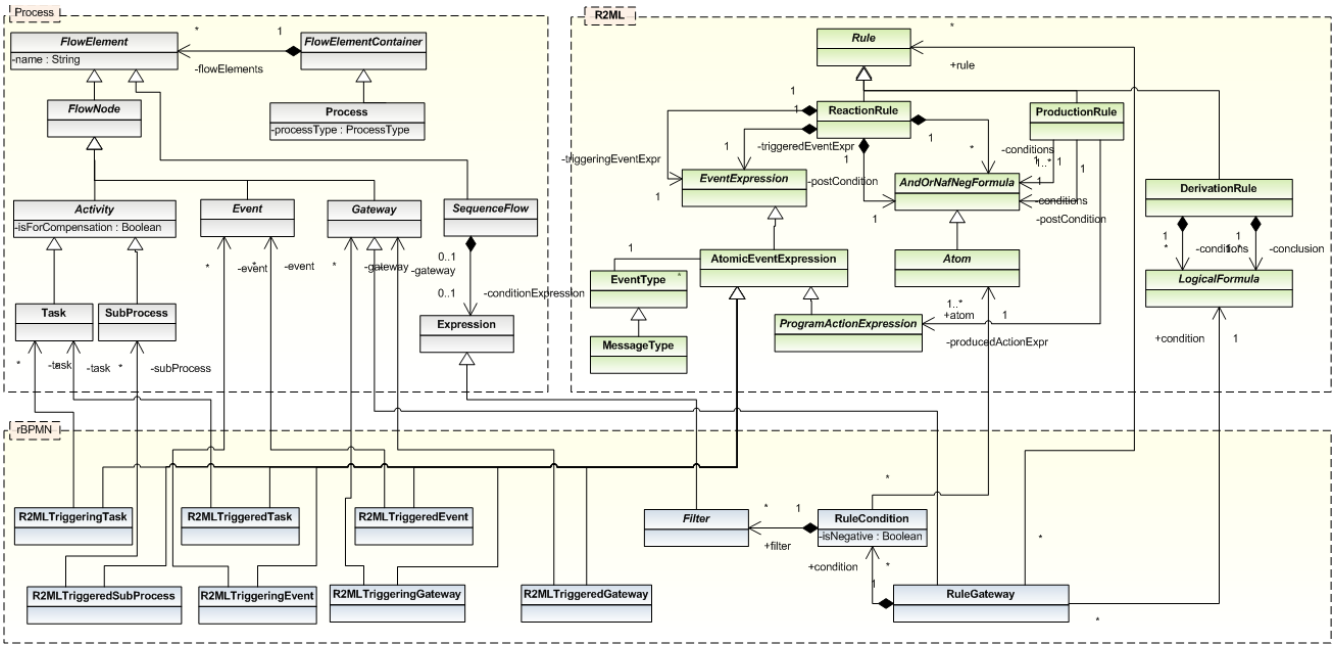

Figure 1. Activity extensions in rBPMN metamodel

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 *StructureDefintion* and we defined its subclass *R2MLMessageType* that can have one R2ML *EventExpression,* through relation *hasVocEntry*. This way, rBPMN messages can be directly mapped to a *ReactionRule*'s triggering event or triggered event expression, and also in service WSDL descriptions [14].

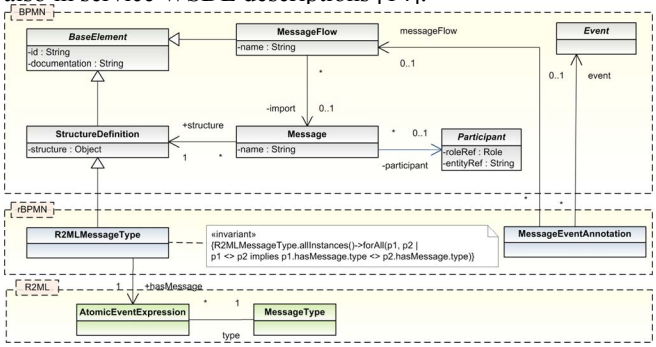

Figure 2. rBPMN data model

B. rBPMN Metamodel: Choreography Modeling Support

The standard BPMN cannot capture several choreography aspects, as recognized in [7]. In order to fully support these aspects we need to integrate these aspects into rBPMN. Those aspects are as follows.

1) Multiplicity of Participants

*Problem***:** In business process models, we often need to model multiple participants of the same type (i.e. Pool) such as multiple participants involved in a conversation.

*Solution***:** For distinguishing multiple participants from each other in the same pool, we will use "Multiple-instance participant" marker (denoted with ||| in bottom of the pool) which is introduced in [20]. This marker means that a pool represent not one, but one or more participants.

2) References

Problem: From the previous problem, it is often needed to distinguish one participant from multiple participants, as we need to know, for example, which participants did some action in a process.

Solution: As introduced in [7], the main challenge with multiple participants is that we need to distinguish individual participants out of this set. Authors in [7] introduce references and reference sets as a special data object enhanced with <ref>, where reference can be connected to a flow object via an association. Reference sets cover those cases where there might be a need to select a subset of participants involved in one conversation. We base our rBPMN extension on reference sets, where we integrate both reference sets and references in one mutual concept called participant set, which may contain zero or more references to participants. To support Participant sets, used in interaction modeling to handle references to participants, we added a *ParticipantSet* class (see Figure 3). We defined *ParticipantSet* as an R2ML Class, so that we can use it in Rule's conditions, when needed. Every participant set could optionally have a name, below <par> annotation.

Figure 4a shows an association from a receiving flow object to a participant set object. This association denotes that a message will be stored in the associated set. The actual participant reference in the set is represented by the participant reference object associated with the flow object. Figure 4b illustrates that an association emanating from a participant set leading to a task denotes that a message is sent to this participant. If an association leads to a receiving flow object (message event or task), a message from participant(s) in this participant set is expected. When a participant set is associated with a multiple instance task or sub-process denotes

that the loop will iterate over that participant set.

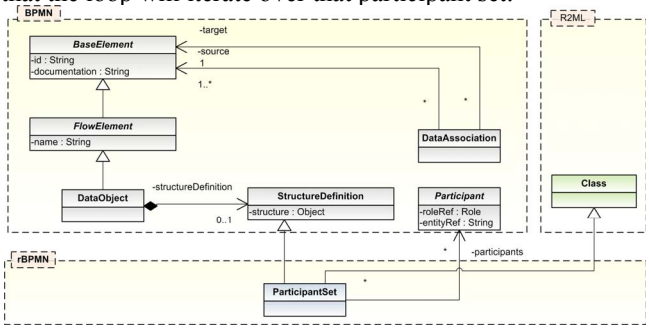

Figure 3. Participant sets in rBPMN metamodel

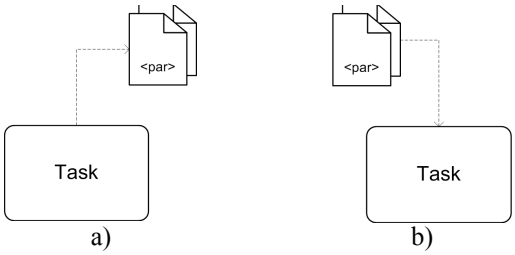

Figure 4. Participant sets – graphical representation

3) Correlation information

Problem: It is not always known which participant has sent a message, so that the returning message can be replied back to that participant.

Solution: Building further on the abovementioned References, where an interaction partner in a participant set is known, we propose an extension of the BPMN message flow. We represented this extension by introducing an association from *Message* to *Participant* (see Figure 2). In the case when an interaction partner is not known, the request message sent from one requester could carry a requester identifier, which is then also contained inside the response message. The *ParticipantSet* object can be associated with the message flows as presented in Figure 5, or with the pool in case of an interaction modeling. This realizes link passing mobility [2]: The associated participant objects are referenced over the message flow.

To create choreographies by means of interaction models, one can use the *Process* concept from the rBPMN *metamodel* or the *Choreography* concept [20]. We prefer to use the plain *Process* concept, as the *Choreography* concept represents a narrower type of a workflow, which contain only three elements (activities): *ChoreographyReference*, *ChoreographyTask*, and *ChoreographySubProcess*. As per findings of [5], choreography-based languages such as WS-CDL [12] need a distinction between explicit choices and racing choices. The first type of choices needs a data-driven XOR-gateway, and we opt for using a rule gateway in this place, where the second type of choices needs an event-based XOR gateway. It is also needed to associate gateways and one of the pools in order to define who actually carries out the choice [5] (Figure 2), as well as event annotations on messages, in order to represent start, intermediate or end of an interaction (Figure 2). Pools are empty in interaction modeling and they are left for concrete orchestrations to implement. These extensions are exemplified in Figure 5.

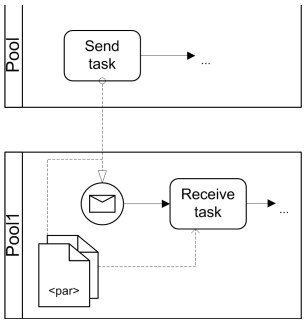

Figure 5. Passing a participant reference over the message flow

IV. SERVICE INTERACTION PATTERNS

In order to evaluate rBPMN for modeling choreographies, we experimented with the proposed rule and choreography modeling by modeling all SIPs. In this paper, we only have enough space to discuss two SIPs – One-to-Many Send/Receive and Racing incoming messages.

A. One-to-Many Send/Receive

In this pattern, a participant sends out several requests to other different participants and waits for their responses. Typically, not all responses need to be waited for. The requester rather waits for a certain amount of time or stops waiting as soon as enough responses have arrived. An rBPMN interconnected behavioral interface model of this pattern is shown in Figure 6. The correlation between send/receive messages is done by pointing to the participants that should be included from the associated participant set. We can see that a multiple-instance subprocess with the "Send" task is used to send messages to the other partners (Pool 2), by using information about partners from the participant set (<par>). The reason for such a design is because the number of partners may or may not be known at design time. When such a message gets to the partner (Pool 2), this partner uses its own logics modeled with a rule gateway (R_2) to decide whether it should return a message to the sender or not by invoking the own "Send" task. When a response from a partner (Pool 2) is received, the reference to the partner who has sent the message is stored in the participant set (<par1>), and a reaction rule attached to the rule gateway (R_1) is invoked to evaluate if the requested number of messages is received (i.e.,. this is the *stop condition*). It is possible that no response is received. In this process, a timeout can occur. In this case, we use another reaction rule attached to R_3 to decide if the *success condition* is achieved or not. If not, an end exception event happens. If yes, the sequence flow goes to the start to wait for a new set of interactions.

The "One-to-Many Send/Receive" pattern represented as an interaction model in rBPMN is shown in Figure 7. This model introduces a repeating Subprocess, which has Send and Receive message flows annotated with message events. Messages used by those message flows, are sent and received by participants referenced in the participant set (<par>). After the "Send" message from Pool 1 to Pool 2, a reaction rule attached to the rule gateway (R_2) is used to decide whether the response from Pool 2 should be received. When the message is received from Pool 2, another reaction

rule attached to the rule gateway (R_1) is used to evaluate whether the *stop condition* is satisfied (i.e., wanted number of messages is received). If so, the interaction completes. During this conversion, a timeout may occur, and then the third rule gateway (R_3) is used to evaluate whether the *success condition* is achieved, and in that case exception occurs. If not, sequence flow is returned to the start to wait for a new set of interactions.

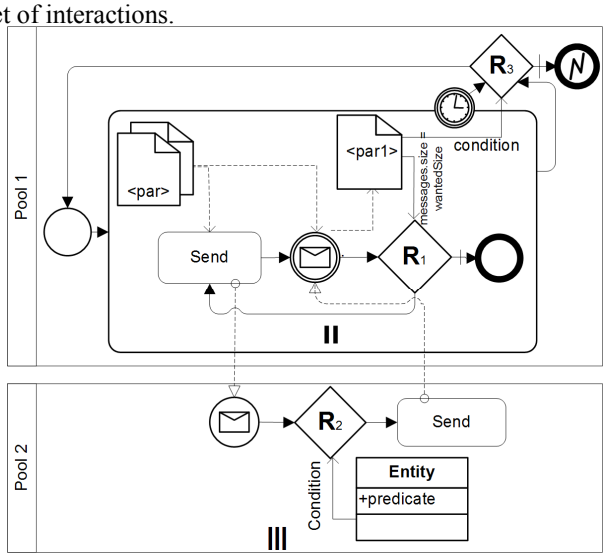

Figure 6. The One-from-Many Receive pattern in rBPMN

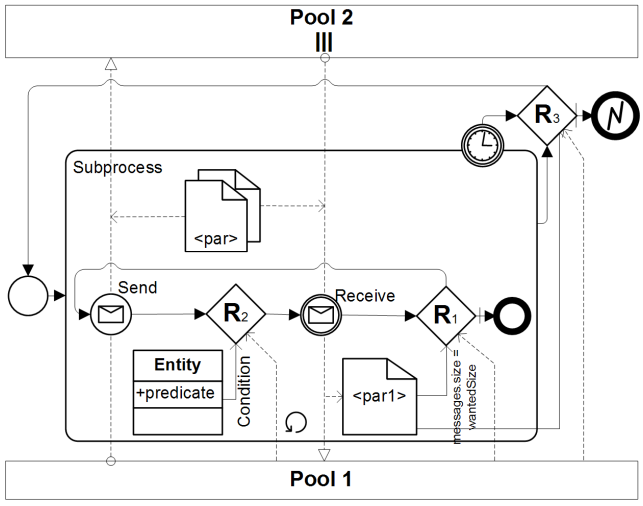

Figure 7. The "One to Many Send/Receive" pattern (interaction model)

B. *Racing incoming messages*

In the racing incoming messages pattern, a participant waits for a message to arrive, while the other participants have a chance to send a message. These messages by different participants "race" with each other. The message that arrives first will be processed. The type of the message sent or the category, to which the sending participant belongs, can be used to determine how the receiver processes the message. The remaining messages may be discarded or kept for later consumption. We model this aspect of message racing by using an event-based XOR gateway and a rule gateway as shown in Figure 8.

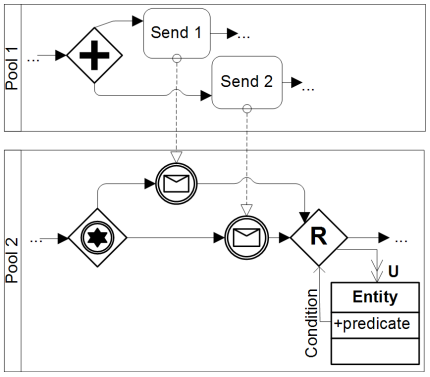

Figure 8. The "Racing incoming messages" pattern

The figure contains the interconnected behavioral interface model of the pattern. In this case we use multiple tasks from Pool 1 to send multiple messages, but also, multiple pools that would send a message could be also used. Figure 8 shows a scenario where Pool 2 has done some activities and now waits for the Pool 1 message. If the Send 1 task sends the message, the intermediate message event in Pool 2 receives the message and then the reaction rule attached to the rule gateway is fired and continues a sequence flow if its condition is satisfied. At the same time, when this happens, the reaction rule attached to the rule gateway updates the Entity's predicate used for its condition, so that any other message that arrives after this first message will not be consumed anymore. When this happens, if the Send 2 task sends the message, the rule gateway will hold execution of this sequence flow. This is the case where a reaction rule attached to the rule gateway can be used in this pattern. We used a reaction rule as we have event for an input, and also a produced action (Entity's update), which could be modeled by a reaction rule. Without the rule gateway, it would be hard to implement this pattern in standard BPMN, as it is not possible to define such an (runtime-updatable) condition on a rule.

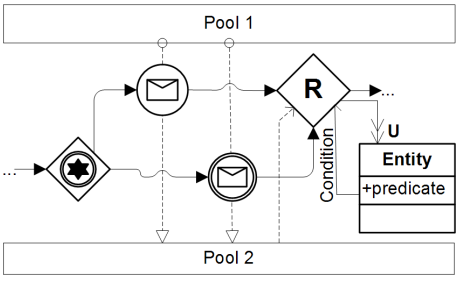

Figure 9. The "Racing incoming messages" pattern (interaction model)

The "Racing incoming messages" pattern represented as an interaction model in rBPMN is shown in Figure 9. This model only shows interactions between partners involved in the scenario. Every message event modeled as interaction is attached to a message flow. A rule gateway is connected to Pool 2 with a dashed line. The decision logic is the same as in Figure 8. In this interaction model in difference to the interconnection model (Figure 8), we can only see interactions between participants, but not their internal logic.

V. FLIGHT REQUEST: A CASE STUDY

In this section, we give an example of a service choreography modeled by the rBPMN language. The example is about of a flight booking process (Figure 10) where a traveler uses an agent to book a flight with a *Travel agency* via the *Trip request* task. When the *Travel agency* receives the request from the Traveler agent, it starts the "Request price" subprocess. In this subprocess, the *FlightRequest* message is sent to each *Airliner* from the *Airlines* participant set, by using the "*Request flight price*" task. The message request is annotated by the *FlightRequest* message type, which specifies the requested *departureDate, arrivingAirport* and *seats* attributes. When this request is received by the Airliner through the start message event, the *Calculate price* task is used to calculate the price for the requested *departureDate*, *arrivingAirport* and number of *seats*. Then the process flow goest o the rule gateway (R_1). The *FlightRequest* message type is represented by the *fReq* variable in the rule gateway's (R_1) condition. The reaction rules (Figure 11) attached to the rule gateway (R_1), based on its pre-defined condition, evaluates whether there are any free seats for the flight at requested *departureDate*.

We have two reaction rules attached to the rule gateway (R_1): one with a 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 ELSE statements in rules may lead to the reasoning problems [28]. If the number of *seats* left for the *Flight* on the requested departure date is less than the number of seats requested, then the reaction rule with the negated condition is activated and the *Send airline not found* task is invoked; this is followed by the message sent *FaultFlightResponse* to the *Travel agency*. This message contains a fault description. If the condition on the reaction rule attached to the rule gateway (R_1) evaluates to true, the sequence flow which goes to the *Send Flight Price* task is selected. In this case, the message flow is annotated by the *FlightResponse* message, which contains *flightNumber* and *price*. When the *FlightResponse* message is received by the *Travel agency*, it writes reference to the *Airliner* which sends the message to the *found* participant set. When all of the *Airlines* from the *Airlines* participant set are contacted, the *Request price* subprocess ends and the sequence flow goes to the R_2 rule gateway. We use this rule gateway to check whether we have any *Airliners* in the *found* participant set. If so, the *Select Airline* task is invoked; otherwise the *Send airline not found* task is invoked to inform the *Traveler agent* that no flights can be found for the requested departure date, arriving airport and requested number of seats. We also have two reaction rules attached to the R_2 rule gateway, but we omit their definition here due to the lack of space. In the *Select airline* task, the *Airliner* is selected and the message is sent to the *Airliner* to reserve the

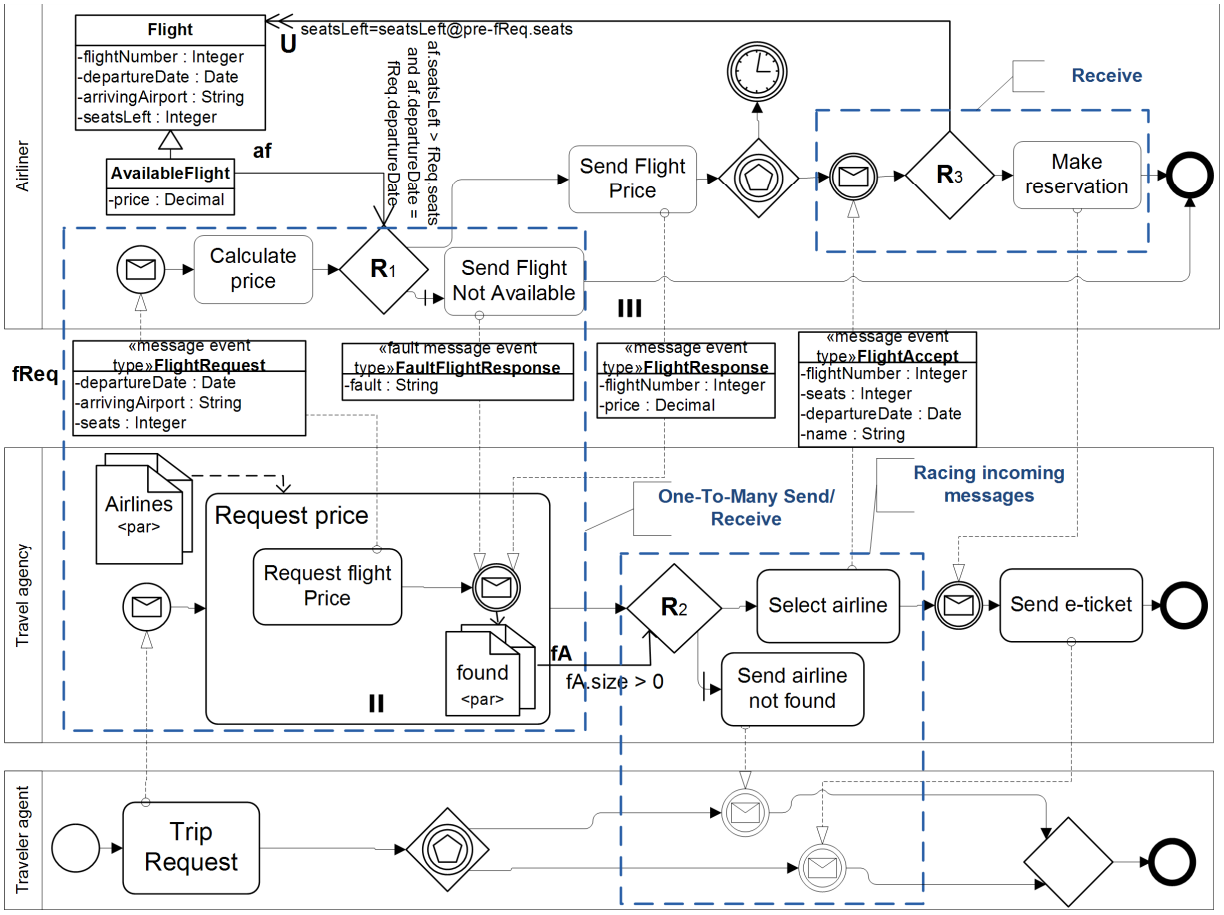

Figure 10. Interconnected behavioral choreography diagram for the Flight request process in rBPMN

seats on the flight. When the message is received by the *Airliner*, by using the intermediate message event, the rule gateway R_3 is invoked to update the number of the *seatsLeft* on the *Flight* by using the update action (U) on the attached reaction rule. Then, the *Make reservation* task is invoked, which creates the reservation and sends it to the *Travel agency*. When the *Travel agency* receives the message, it prepares and sends the e-ticket to the *Traveler agent*.

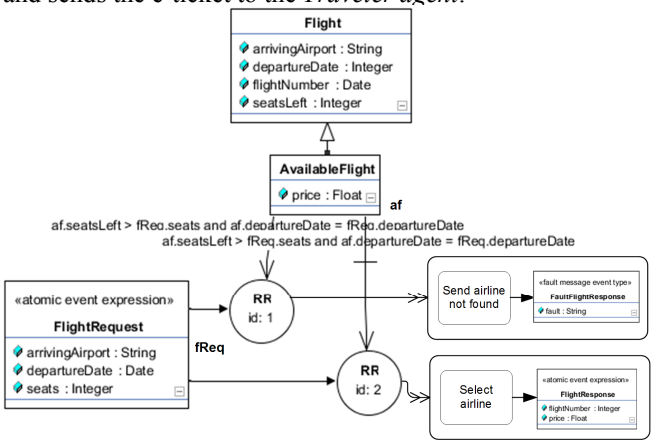

Figure 11. Reaction rules attached to the rule gateway (R1) in Figure 10 represented in URML

In the process shown in Figure 10, we identified three service interaction patterns. The first identified pattern is the *One-To-Many Send/Receive* pattern. In this pattern, a participant sends out several requests to other different participants and waits for their responses. Here, we have the multiple-instance *Request price* subprocess with the *Request flight price* task that is used to send messages to the *Airliners*, by using the information about partners from the participant set (*Airlines*). The reason for such a design decision is because the number of partners may or may not be known at design time and this represents an advantage of our approach. Another advantage of our approach is the shared vocabulary based on which two rule gateways R1 and R3 can access the same *Flight* instance to check or decrease the value of their *seatsLeft* attribute. Also, *seats* and *departureDate* values used in the rules' condition are not fixed, and can be dynamically changed in each request of the *Travel Agency*, by using the information from the *FlightRequest* message type and *Flight* class, respectively.

One another important aspect of this pattern is that is related to the In-Out Web service Message Exchange Pattern (MEP) [29]. In our previous work, we created a transformation that transforms such reaction rules into complete Web service descriptions [23]. That allows for transforming a (set of) R2ML reaction rule(s) model(s) into 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 of several languages (e.g., Drools or Jess) by simulating semantics of reaction rules on production rule engines. We call such rules "how-to-use", as they specify conditions under which a service can be used.

Additionally, Figure 11 shows how we enabled for a full traceability between elements of the rBPMN and R2ML. Namely, all BPMN tasks (*Select airline* and *Send airline not found*) and messages (*FaultFlightResponse* and *FlightResponse*) have their counterparts in R2ML. This traceability is established via the rBPMN metamodel and OCL constraints.

In the *Airliner* pool, we can see the *Receive* pattern. When the message is received from the *Traveler agency*, and handled by using the intermediate event message, we used the reaction rule attached to the rule gateway to decrease the number of seats left on the flight, in the shared vocabulary. Another implication of using rules here is that we attach another rule to the rule gateway to check if the user is a regular user and if so, to give him a discount. Given the nature of rules, if the discounting business logic changes, this can be dynamically reflected by the change of the rule.

Finally, in the *Traveler agent* pool, we can recognize one variant of the Racing incoming messages pattern. The *Trip request* task is followed by an Event-based gateway, from which two possible branches could be selected based on the received message. Here, we can use rules to introduce additional logic, such as to check if the *Traveler* has enough money on the account to pay the ticket.

In Figure 10, we show an interconnected behavioral model of the choreography for the flight request process. However, such models are individual views on the choreography from the perspective of one of the participants. Modeling choreographies in this way have two major drawbacks, as reported in [5]: redundancy (where parallelism, branching, loops and timeouts are duplicated in the model) and potentially incompatible behavior (errors in the model in the case of event-based XOR-gateways). Interaction models do not have these problems, so we translated our model to the interaction model and shown in Figure 12. In the interaction model, we attached a message event to each message interaction, while the pools are empty. We actually have a complete set of transformation rules between the two types of choreography models supported in rBPMN, but due to the space limitation, we cannot report on them in this paper.

In the interaction models, we need to connect the rule gateway to the participant (pool), so that the participant can invoke the rule and decide which branch to take. In the case of Event-based gateways, one of multiple events can happen, by occurring first in the process. Another implication of the interaction models is passing the participant references by using the participant sets (as described in Section III.B). In the process shown in Figure 12, we need to collect *Airliner* participants in the *found* participant set, in order to pass that participant set to the rule gateway (R_2) condition. A participant set (*found*) is attached to the message flow and each time when the flight price message is received, the participant is written in the set.

VI. Related Work

In Table 1, we summarize the results of the comparison of various languages for service interaction and business process modeling. All analyzed languages support the first group of patterns. This is the case as this group of patterns is

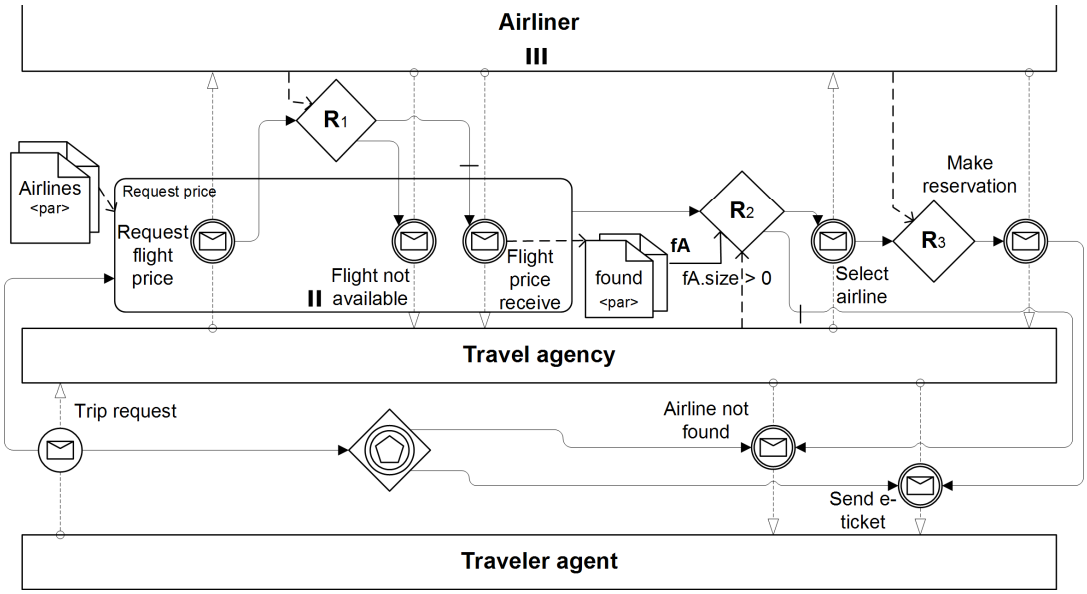

Figure 12. rBPMN interaction choreography model for the flight request process show in Figure 10

based on simple send/receive message interactions between two parties, which are supported in all studied languages. Rules in this group of patterns are used to define precise condition on which the messages could be exchanged, as well as for handling fault messages.

Table 1. Comparison of business process modeling languages per service interaction patterns: 1) Single-transmission bilateral interaction patterns; 2) Single-transmission multilateral interaction patterns; 3) Multi-transmission interaction patterns; and 4) Routing patterns

Pattern group	Pattern	Language				
		Let's Dance	BPMN	WS-CDL	iBPMN	rBPMN
1)	Send	+	+	+	+	+
	Receive	+	+	+	+	+
	Send/Receive	+	+	+	+	+
2)	Racing incoming messages	+	+	+	+	+
	One-to-many send	+	-	+/-	+	+
	One-from-many receive	+	-	+	+	+
	One-to-many send/receive	+	-	+/-	+	+
3)	Multi-responses	+	+	+	+	+
	Contingent requests	+/-	-	+/-	+/-	+
	Atomic multicast notification	-	-	-	-	-
4)	Request with referral	+	-	+	+	+
	Relayed request	+	-	+	+	+
	Dynamic routing	-	-	+/-	-	+/-

In the third group, the Atomic multicast notification pattern is not supported by any of the studied languages. This is the case as none of the languages supports distributed transactions needed for implementation of this pattern. The Multi-responses is supported in all languages, but other languages do not define how the stop condition or fault message should be implemented, which is done through the use of rule gateways in rBPMN. The Contingent requests pattern is just partially supported in Let's Dance, WS-CDL and iBPMN, as these languages cannot accept messages from previous requests that failed due to a timeout. We supported this issue by reaction rules attached to rule gateways, to define a condition when such responses are accepted or not.

In the fourth group, the Request with Referral and Relayed Request patterns are supported in all languages but BPMN. rBPMN supports them by passing participant references between participants. We used a rule gateway in these two patterns to support decision on tasks invoking and whether the participant should return the fault message or not, which is not implemented in other languages than rBPMN. Dynamic Routing is only partially supported in WS-CDL and rBPMN. rBPMN supports the dynamic routing condition by using rule gateways on data contained in the original request or in one of the intermediate steps.

The importance of using rules in the second group of patterns, especially for One from Many Receive and One to Many Send/Receive patterns, is in defining the stop and success conditions. These conditions are precisely defined in a declarative way, by using a rule gateway to decide if interactions are complete or if they are successful or not. Multilateral interaction is enabled via the introduction of participant sets and reference passing in rBPMN along with multiplicity of participants. Such participant sets are similar to reference sets and references in iBPMN [5], which are used to reference particular participants in patterns with multiple participants of the same type.

To the best of our knowledge, rBPMN is a first language which considers integration of choreography modeling with business rules. The present approach in the area of integration of business process and rule modeling are mainly related to languages for orchestrations. rBPMN provides a systematic integration of a sound 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 they can generate full definitions of exactable rules, processes, and services. Previous attempts mainly provide integration on the level of concrete syntax (e.g., RuleML and AGFIL-BM) only with-

out precise language definitions [16]. Alternatively, integration was hard coded into implementation of languages [3][24]. Either of these two groups of approaches makes the verification of well-formedness 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 constrains.

Some approaches are only developed to provide execution infrastructure [3][24] rather than a modeling and development solution. Other approaches provide transformations from process models to rules [1] or simulation of process models with reaction rules [13]. However, those solutions do not either: provide a graphical representation of a process model, describe positions of rules in process models, or use model-driven engineering principles and standards. Besides rBPMN, the approach presented in [9] is the only one that supports all four types of rules. However, that approach does not use a rule modeling language as rBPMN does, but rather it relies on the use of hardcoded rules or definitions of policies in the PENELOPE language. For rBPMN, the work presented in [10] is very relevant work. That work does provide a set of very useful patterns for integration of business rules in business process driven development of SOAs. However, that work does not provide integration of any formally defined rule language, but rather it uses natural language definitions of rules and defines their place in the business process models. rBPMN provides a full language rule-enhanced process modeling language definition, and complements the work of [10] by supporting all three types of patterns.

VII. Conclusion and Future Work

To the best of our knowledge, the presented work is the first language that integrates business rules with a process-oriented language for modeling choreographies. Our evaluation demonstrated that rBPMN increased the modeling support for service-interaction patterns comparing to the other relevant languages. Another important contribution of our work is that the metamodel-based systematic integration of rules in choreography model also advances the state of the art in the integration of rules into business process modeling. In our on-going work, we have defined and are now testing transformations between two types of rBPMN choreography models (i.e., interaction and interconnected) and are finalizing transformations onto rule-enhanced service composition engines and service languages.

References

[1] A. Barros, et al. "Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection", TR FIT-TR-2005-02, QUT, Australia, 2005.

[2] A. Barros, G. Decker, M. Dumas, F. Weber, "Correlation Patterns in Service-Oriented Architectures", Fundamental Approaches to Software Engineering, Springer Berlin / Heidelberg, 4422, 245-259, 2007.

[3] A. Charfi, M. Mezini, "Hybrid web service composition: business processes meet business rules", *In Proc. 2nd Int'l Conf. on SoC*, pp.30-38, 2004.

[4] T. V. Eijndhoven, et al., "Achieving Business Process Flexibility with Business Rules", *In Proc. 12th Int'l EDOC Conf.*, pp. 95-104, 2008.

[5] G. Decker, A. Barros, "Interaction Modeling using BPMN," In Proceedings of the 1st International Workshop on Collaborative Business Processes, pp. 206-217, 2007.

[6] G. Decker, O. Kopp, F. Leymann, M. Weske, "Interacting services: from specification to execution Data & Knowledge Engineering", Elsevier Science Publishers, 68, 946-972, 2009.

[7] G. Decker, F. Puhlmann, "Extending BPMN for Modeling Complex Choreographies", In Proc. OTM Int'l CoopIS Conf., pp. 24-40, 2007.

[8] G. Decker, et al., "BPEL4Chor: Extending BPEL for Modeling Choreographies", In Proc. of the IEEE Int'l Conf. on Web Services, pp. 296-303, 2007.

[9] S. Goedertier, J. Vanthienen, "Business Rules for Compliant Business Process Models", *In Proc. of the 9th Int'l Conf. on Business Inf. Sys.*, pp. 558-579, 2006.

[10] T. Graml, R. Bracht, and M. Spies, "Patterns of Business Rules to Enable Agile Business Processes," *In Proc. 11th IEEE Int'l EDOC Conf.,* pp. 365-375, 2007.

[11] Intalio, STP BPMN Modeler: BPMN object model, Online, Available: http://www.eclipse.org/stp/bpmn/model/index.php, (2009).

[12] N. Kavantzas, et al. (2005), "Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, , 2005.

[13] G. Knolmayer, et al., "Modeling Processes and Workflows by Business Rules," *In Business Process Manag., Models, Tech., and Empirical Studies*, Springer, pp. 16-29, 2000.

[14] R. Lu, S. Sadiq, G. Governatori, "Compliance Aware Business Process Design", in Business Process Management Workshops, Volume 4928/2008, pp. 120--131, Springer, 2008.

[15] J. Mendling, G. Neumann, M. Nuttgens, "A Comparison of XML Interchange Formats for Business Process Modelling," In Proc. of EMISA 2004 Conf., 2004.

[16] J. Meng, et al., "Achieving dynamic inter-organizational workflow management by integrating business processes, events and rules," *In Proc. 35th HICSS Conf.*, 2002.

[17] M. Milanović, D. Gašević, G. Wagner, M. Hatala, "Rule-enhanced Business Process Modeling Language for Service Choreographies", In Proc. *MODLES 2009*, pp. 337-341.

[18] OASIS, Web services business process execution language (WS-BPEL), v2.0, Online, Available: *http://*www.oasis-open.org/committees/wsbpel, 2007.

[19] OMG, "Business Process Modeling Notation" v1.2 - Final Adopted Specification, Online, Available: http://www.omg.org/docs/formal/09-01-03.pdf, January 2009.

[20] OMG, "Business Process Modeling Notation" v2.0 - FTF Beta 1, Available: http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf, 2009.

[21] OMG, "Business Process Definition MetaModel Vols. I & II", OMG Documents: formal/2008-11-03 & formal/2008-11-04, 2008.

[22] REWERSE Rule Markup Language, http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6.

[23] M. Ribarić, et al., "Model-Driven Engineering of Rules for Web Services," *Gen. and Transf. Tech. II, LNCS 5235*, 2008, pp. 377-395.

[24] F. Rosenberg, S. Dustdar, "Business Rules Integration in BPEL - A Service-Oriented Approach", *In Proc. 7th Int'l IEEE Conf. on E-Comm. Tech.*, 2005.

[25] G. Steinke, C. Nikolette, "Business rules as the basis of an organization's information system," Industrial management + Data Systems, vol. 103, p. 52, 2003.

[26] V. Strahonja, "The Evaluation Criteria of Workflow Metamodels," In Proc. of the ITI 2007 29th Int. Conf. on Inf. Tech. Interfaces 2007.

[27] UML-based Rule Modeling Lang., http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/7.

[28] G. Wagner, A. Giurca, S. Lukichev, "A Usable Inter-change Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL", In Proc. of WWW2006 Reasoning Web WSh., 2006.

[29] Web Services Description Language, Version 2.0 Part 2: Adjuncts, Online, Available: http://www.w3.org/TR/wsdl20-adjuncts.

[30] M. Weske, "Business Process Management", Springer, 2007.

[31] J. Zaha, et al., "Let's dance: A language for service behavior modeling," In Proc. 14th Int'l Conf. on CoopIS, pp. 145-162, 2006.