Towards Using Reo for Compliance-aware Business Process Modeling

Farhad Arbab, Natallia Kokash, and Sun Meng

CWI, Kruislaan 413, Amsterdam, The Netherlands
firstName.lastName@cwi.nl

Abstract. Business process modeling and implementation of process supporting infrastructures are two challenging tasks which are not fully aligned. On the one hand, languages such as Business Process Modeling Notation (BPMN) exist to capture business processes at the level of domain analysis. On the other hand, programming paradigms and technologies such as Service-Oriented Computing (SOC) and web services have emerged to simplify the development of distributed web systems that control underlying business processes. BPMN is the most recognized language for specifying process workflows at the early design steps. However, it is rather declarative and may lead to the executable models which are incomplete or semantically erroneous. Therefore, an approach for expressing and analyzing BPMN models in a formal way is required. In this paper we describe how BPMN diagrams can be represented by means of a semantically precise channel-based coordination language called Reo which admits formal analysis using model checking and bisimulation techniques. Moreover, since additional requirements may come from various regulatory/legislative documents, we discuss the opportunities of Reo and its mathematical abstractions in expressing process-related constraints such as Quality of Service (QoS) or time-aware conditions on process states.

1 Introduction

The Service-Oriented Computing (SOC) paradigm supports the idea of building distributed applications by composing self-contained and loosely-coupled services. Service-Oriented Architecture (SOA) is the main architectural concept within this paradigm designed to support the realization of cross-organizational business processes. In this kind of architecture, services are employed to accomplish certain activities within a process. Several specifications coordinate the collaboration of individual services. In the simplest case, known as orchestration, one business partner manages the order in which required services are executed. In a more complex scenario, called choreography, each partner is responsible for executing services that realize its own business logic as well as interacting with other partners to achieve a common goal.

A stack of protocols that defines how web services collaborate is currently established. In particular, WS-BPEL [1] and WS-CDL [2] are the most commonly

recognized languages dealing with orchestration and choreography, respectively. However, these languages are implementation-level languages while business processes incorporate various aspects both functional and non-functional that may be difficult to capture and convert directly into executable code. Therefore, additional tools are used at the level of domain analysis and abstract process design. The Business Process Modeling Notation (BPMN) [3] is a standard and widely-accepted graphical notation for this purpose. According to this notation, the process can be represented in the form of activities produced either by humans or software applications, important events occurring in the process and control flow on the involved activities. One of the reasons why BPMN stands out among other notations for business process modeling is its ability to define concurrent tasks and sub-processes with exception handling and compensation associations, which have been proven to be useful even at the stage of early design. As a trade-off to its expressive power, BPMN lacks semantic precision.

Two attempts have been made at defining formal semantics for BPMN subsets [4, 5]. In the first approach [4], a core subset of BPMN is mapped into Petri nets. However, this approach encounters problems with reflecting the behavior of multiple concurrent activities in a presence of exception handling. The second approach [5] formalizes the BPMN semantics (including time-aware semantics [6]) in a more consistent way using Communicating Sequential Processes (CSP). The main drawback of this model is that it does not preserve the structure of BPMN diagrams which makes the mapping difficult to follow. Additionally to these approaches, a number of works provide insights and tools for automated translation of BPMN into BPEL processes [7, 8]. Such translations bridge the gap between the process modeling and their implementation using web services technology. However, they pose significant restrictions on admissible BPMN patterns and do not prevent developers from implementing erroneous processes. Later on, BPEL processes can be verified using a wide range of formal techniques [9–11] and model checking tools [12], but this scenario shifts the process verification to the implementation phase and thus slows down the incremental process design.

A number of challenging issues need to be addressed before the SOC becomes a mature approach for developing flexible business applications. Among such issues is the SOA adaptation to the ever changing business/legislative requirements and process evolution. Multitude of regulations constantly emerge to shape the businesses and incorporate the best practices into corresponding software applications. The aim of the recently started COMPAS (Compliance driven Models, Languages, and Architectures for Services) project[1] is to develop an infrastructure that would ensure dynamic and ongoing compliance of services-oriented applications to business regulations. These regulations come out from legislative documents such as Basel II1, IFRS2, MiFID3, LSF4, HIPAA, Tabaksblat5, and the Sarbanes-Oxley6 Act, just to name a few. In addition to external regulations, there are internal movements towards Quality of Service (QoS) which result in similar requirements. Currently there are no well-established practices for representing and tracking compliance-related controls. At the early

[1] http://www.compas-ict.eu/

process development stages they can be expressed by means of modeling languages like BPMN or UML with textual annotations. However, such specifications are often ambiguous and may result in erroneous process implementation. Therefore, a modeling notation with precise semantics is required. This notation should be powerful enough to represent the major structural and control/data flow elements of processes, as well as to express various compliance concerns.

In this paper, our objective is two-fold. Firstly, we consider the main business process modeling primitives as defined in BPMN and show how they can be represented using a semantically precise coordination language Reo. Secondly, we discuss the potential of our formalism in expressing compliance concerns.

The rest of the paper is organized as follows. In Section 2, we sketch the main steps of our approach to compliance-aware business process modeling. Section 3 contains an overview of BPMN and in Section 4, we introduce Reo. In Section 5, we present the mapping from BPMN to Reo. In Section 6, we discuss compliance rule modeling from the perspective of Reo. Finally, in Section 7, we outline conclusions and future work.

2 Overview

The overall vision of our framework is shown in Fig. 1. Business analysts may use traditional notations for creating business process models such as BPMN or UML Activity Diagrams (ADs) as well as more specific ones, e.g., BPEL Graphical Modeling Tools (GMT)[2]. At this level, compliance concerns can be expressed using Domain Specific Languages (DSL) or GMT extensions, see [13] and [14] for examples of both approaches. One of the goals of the COMPAS project is to develop DSLs capable of expressing major categories of compliance concerns. These models will not necessarily guarantee the level of precision sufficient for the direct process implementation. Therefore, we propose to introduce an intermediate layer on which the high-level models will be verified and refined. The basic semantics of this layer is defined by Reo.

Reo is a channel-based exogenous coordination language supported by a graphical tool, an animation engine and a model checker[3]. These tools allow us to use Reo both for graphical process modeling and for formal process verification before its actual implementation. There are several reasons why Reo seems appropriate in the context of the COMPAS project. First, using Reo connectors it is possible to represent both choreography and orchestration of process activities as well as internal and external behavior of involved services in unified formalism [15]. Moreover, Reo patterns can be automatically translated into Constraint Automata (CA) which are suitable for representing service compositions with QoS guarantees [16] and time-aware processes [17]. CA are essentially variants of labeled transition systems where transitions are augmented with pairs $\langle N, g \rangle$ rather than action labels. The states of a CA stand for the network configurations (e.g., contents of the buffers) while transition labels $\langle N, g \rangle$

[2] http://www.eclipse.org/bpel/
[3] http://homepages.cwi.nl/~koehler/ect/index.htm

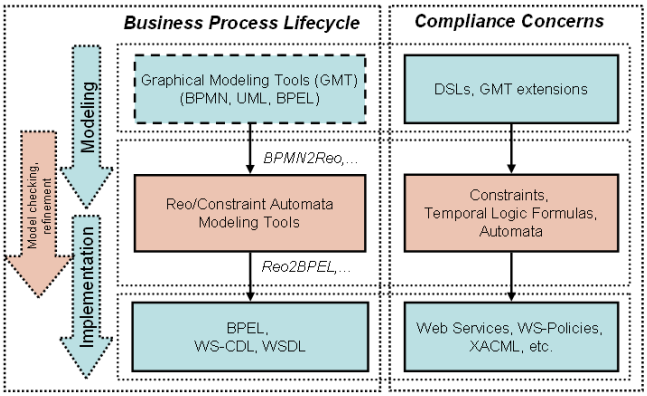


Fig. 1. Reo for business process modeling with compliance concerns

can be viewed as I/O operations performed in parallel (more precisely, sets of nodes where data flow is observed in parallel and boolean conditions on the data items observed on those models). Moreover, CA can be extended by associating various properties with states and transitions (e.g., QoS characteristics). We assume that at this step, compliance rules are converted into automata transition constraints, temporal logic formulae, or result into automata state reachability checking. After model checking and refinement, the Reo/CA process models can be automatically translated into executable SOC languages such as WS-BPEL, as well as to Java code.

This paper focuses on the first step of the proposed framework, namely, on the BPMN to Reo conversion. The choice of BPMN as a modeling notation is justified by the fact that it comes with a number of useful process concepts such as events, exception handling, transactions and message flow. BPMN is a de-facto standard for business process modeling supported by a number of software tools. Moreover, such a mapping is interesting from the research perspective since no efficient semantic model for BPMN currently exists. Nonetheless, generally the COMPAS project is not bounded to this notation and we are planning to develop similar mapping tools for translating other design languages, in particular, UML ADs and BPEL GMTs, into Reo models.

3 Business Process Modeling Notation (BPMN)

In this section we overview the main structural elements of BPMN.

The basic BPMN concepts are *flow objects*, *connecting objects*, *swimlanes* and *artifacts*. Flow objects are the main graphical elements defining the behavior of a business process. BPMN distinguishes three types of flow objects, namely, *events*, *activities* and *gateways*. These elements are linked according to

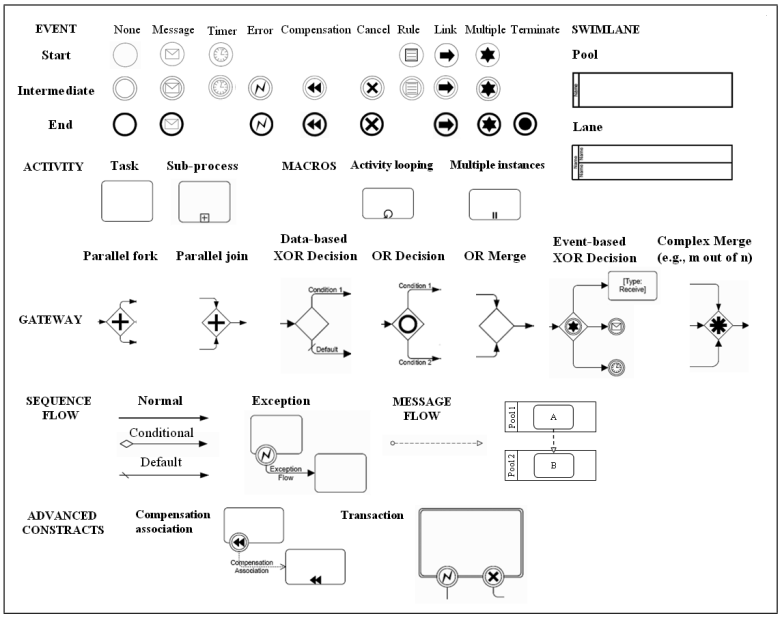


Fig. 2. Selected BPMN elements

well-defined syntactic rules by two *connecting objects*: *sequence flow* and *message flow*. A third connecting object is *association* and it is used to connect flow objects with text and non-flow elements. Two types of *swimlanes*, *pools* and *lanes*, arrange the main BPMN elements into groups. Finally, *artifacts* are introduced to provide additional information about a process. This concept is extendable and besides three standard artifacts, that is, *data objects*, *groups* and *annotations*, designers can introduce their own artifacts.

Figure 2 shows the selected BPMN elements that are essential for modeling process behavior. BPMN identifies three types of events: a *start event* signals the start of a process, an *end event* signals the end of a process and an *intermediate event* is an event occurring during a process. Different triggers such as *message, timer, rule, link, error, cancel, compensation, terminate and multiple trigger* can be associated with events. The detailed description of the triggers and their usage rules can be found in the BPMN specification [3].

An *activity* can be an *atomic task* or a *sub-process*. To each task a *type* can be assigned. Among the specific task types are *service*, *receive* and *send* tasks. A sub-process is a compound of other activities and a sequence flow on them. BPMN introduces two attributes that are commonly used to identify special types of activities (both tasks and sub-processes), namely, *looping* activities and *multiple concurrent instances* of the same activity.

A *gateway* is a construct used to control divergence and convergence of the sequence flow. A *parallel fork* gateway is used to split an incoming sequence flow into several concurrent branches, while a *parallel join* gateway synchronizes

several concurrent sequence flows. A *data/event-based XOR decision* gateways select one out of a set of mutually exclusive sequence flows according to some data-based condition or external event. An *OR decision* behaves similarly but it admits more than one alternative to be selected. An *OR merge* shows the convergence of several sequence flows into one sequence flow. Finally, *complex decision/merge* gateways are used to cover the advanced sequence flow control constructs which cannot be easily handled using other gateways. One such example is the so called *m out of n* choice when m arrived tokens out of n initiated parallel sequence flows are required to continue the process.

BPMN distinguishes two basic types of flow. The *sequence flow* prescribes the order of activities performed by one entity while the *message flow* regulates the flow between two communicating entities represented by separate *pools*. The sequence flow consists of a *normal flow* to which a transition guard can be assigned (uncontrolled, conditional or default flow) and *exception flow* that originates from some event and is used to handle exceptions. Finally, BPMN defines a number of advanced constructs such as *compensation association* and *transaction*. Due to space limits, we will not consider these constructs in this paper.

4 Reo

Reo [18] is a channel-based exogenous coordination model wherein complex coordinators, called *connectors*, are compositionally constructed from simpler ones. We summarize only the main concepts in Reo here. Further details about Reo and its semantics can be found in [17–20].

Complex connectors in Reo are formed as a network of primitive connectors, called *channels*, that serve to provide the protocol which controls and organizes the communication, synchronization and cooperation among the components/services that they interconnect. Each channel has two *channel ends* which can be of two types: *source* and *sink*. A source end accepts data into its channel, and a sink end dispenses data out of its channel. It is possible for the ends of a channel to be both sinks or both sources. Reo places no restriction on the behavior of a channel and thus allows an open-ended set of different channel types to be used simultaneously together. Figure 3 shows the graphical representation of basic channel types in Reo. A *FIFO1 channel* represents an asynchronous channel with one buffer cell which is empty if no data item is shown in the box (this is the case in Fig. 3). If a data element d is contained in the buffer of a FIFO1 channel then d is shown inside the box in its graphical representation. A *synchronous channel* has a source and a sink end and no buffer. It accepts a data item through its source end iff it can simultaneously dispense it through its sink. A *lossy synchronous channel* is similar to synchronous channel except that it always accepts all data items through its source end. The data item is transferred if it is possible for the data item to be dispensed through the sink end, otherwise the data item is lost. For a *filter channel*, its pattern $P \subseteq Data$ specifies the type of data items that can be transmitted through the channel. Any value $d \in P$ is accepted through its source end iff its sink end can simultaneously dispense

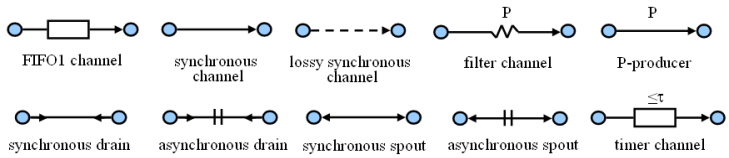
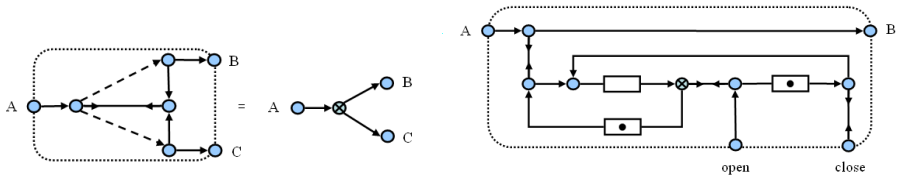


Fig. 3. Some basic channels in Reo



(a) Exclusive router (b) Valve

Fig. 4. Examples of Reo connectors

d; all data items $d \notin P$ are always accepted through the source end but are immediately lost. The P-producer is a variant of a synchronous channel whose source accepts any data item, but the value dispensed through its sink is always a data element $d \in P$.

There are some more exotic channels permitted in Reo: *(A)synchronous drains* have two source ends and no sink end. A synchronous drain can accept a data item through one of its ends iff a data item is also available for it to simultaneously accept through its other end as well and all data accepted by the channel are lost. An asynchronous drain accepts data items through its source ends and loses them, but never simultaneously. *(A)synchronous Spouts* are duals to the drain channels, as they have two sink ends. A *timer channel with early expiration* allows the timer to produce its timeout signal through its sink end and reset itself when it consumes a special "expire" value through its source [17]. Complex connectors are constructed by composing simpler ones via the *join* and *hiding* operations, see [17] for more details.

Example 1. Figure 4(a) shows an implementation of an exclusive router by composing five synchronous channels, one synchronous drain and two lossy synchronous channels together. The connector provides three nodes A, B and C for other entities (connectors or component instances) to write to or take from. A data item arriving at the input port A flows through to only one of the output ports B or C, depending on which one is ready to consume it. The input data is never replicated to more than one of the output ports. If both output ports are ready to consume a data item, then one is selected non-deterministically. To avoid writing an exclusive router every time it is used, we introduce a notation similar to a node to represent this connector. We will also use XOR-nodes with

more than two outputs. Such a connector can be defined by combining several exclusive routers with two outputs.

Additionally, it is useful to define a priority on the outputs of an exclusive router in such a way that the data item will always flow into the prioritized output if more than one output is available. Such a deterministic prioritized exclusive router can be implemented by connecting the input of an exclusive router with its non-prioritized outputs through valve connectors (see Fig. 4(b)). A valve connector is able to close and reopen the flow from A and B. Initially, the circuit is in the "open" state, i.e., a data item arriving at the input port A flows to the output B until the close command arrives. After that the circuit goes into the "close" state, i.e., the flow remains blocked until the open command arrives. If the prioritized output of the exclusive router becomes ready to accept data, it can simultaneously close the valves thus making other outputs unavailable.

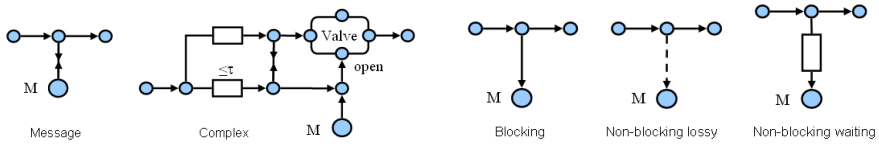
5 Mapping BPMN to Reo

In this section we use Reo to represent a comprehensive set of BPMN modeling primitives and common constructs.

5.1 Basic Objects: Tasks, Events, Gateways and Message Flow

Generally, BPMN tasks and sub-processes correspond to external components or black-boxes whose collaboration is coordinated by Reo. However, it is still possible to simulate the behavior of certain activities using Reo channels. For example, an atomic task with one input and one output can be represented by a simple FIFO1 or a timer channel while a sub-process can be modeled by a Reo connector that preserves the number of its incoming and outgoing flows.

An event with no trigger (start, end or intermediate) or an end event with a terminate trigger can be shown as a Reo node (source, sink or mixed). Other event triggers can be modeled using the basic Reo channels. Thus, (i) a timer event can be represented with the help of a timer channel, (ii) an incoming message event can be simulated by a synchronous drain whose first end is an input port and the second end is an internal process node (see Fig. 5(a)) while (iii) an event with a rule trigger corresponds to a filter channel with an appropriate transition condition. Other BPMN events such as outgoing messages, error, compensate, cancel or link events occurred as a part of the sequence flow correspond to the immediate transitions into required places of the process where they will be triggered and can be represented by means of synchronous channels. However, if a process or a subprocess that must react to such an event is not ready to accept it, the current sequence flow will be blocked. This problem can be resolved either by using a lossy synchronous channel that indicates that if an event is not picked up at the destination point it will be lost, or a FIFO1 channel that indicates that a message generated by an event will be waiting until it can be processed. Figure 5(b) shows the Reo connectors corresponding to these three message sending protocols. The composite conditions such as the case when the



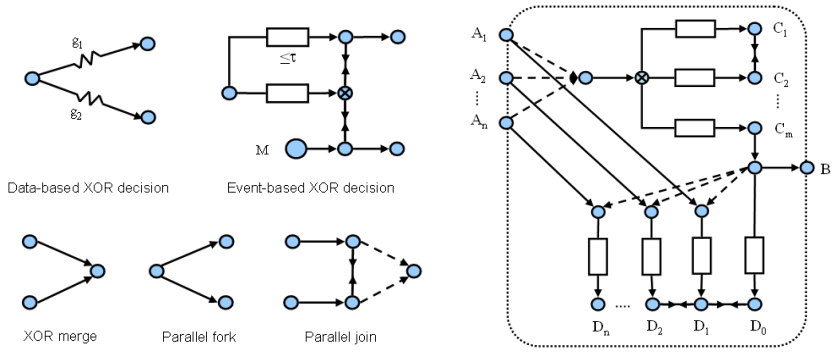
(a) Events (b) Messages

Fig. 5. Modeling BPMN events and messages in Reo

process execution continues when a required message has arrived or a certain time-date has been reached, can be modeled by the combination of several Reo channels. The Reo pattern for the aforementioned complex event is shown in Fig. 5(a). It uses a valve connector introduced in Fig. 4(b) to control the data flow from the start to the end node. We assume that initially in this circuit the valve is closed and reopened upon a timer event or a message arrival.

Figure 6(a) shows the Reo connectors for the basic BPMN gateways, namely, data-based XOR decision, event-based XOR decision, XOR merge, parallel fork and parallel join. A data-based XOR decision is modeled using a synchronous channel which represents the incoming flow and two (or more) filter channels with a common source that represent the alternative outgoing flows. Filter transition conditions (guards) are defined by boolean expressions g_1 and g_2. The representation of an event-based XOR decision mainly depends on the semantics of the events that affect the decision. In our case, the lower branch is selected if a message arrives in a predefined period of time, and the higher branch is preferred otherwise. An XOR merge consists of two (or more) synchronous channels with a common sink. A parallel fork is composed of two (or more) diverging synchronous channels. A parallel join consists of two (or more) synchronous channels representing the incoming parallel sequence flows that are further synchronized with the help of a synchronous drain channel. Several lossy synchronous channels with a common sink then are used to get a single outgoing token. An OR decision gateway can be modeled in Reo similarly to the data-based XOR decision whose guards are not necessarily mutually exclusive. Additionally, using Reo, the designer can define various complex control gateways. For example, Fig. 6(b) shows a connector for an *m out of n* synchronizer pattern. This is a *lossy* version of the pattern, that is, the circuit loses its extra inputs before the next cycle. Alternatively, by substituting n lossy synchronous channels introducing the input data with n simple synchronous channels one can create a *sparing m out of n* pattern that delays to spare its extra inputs for the next cycle.

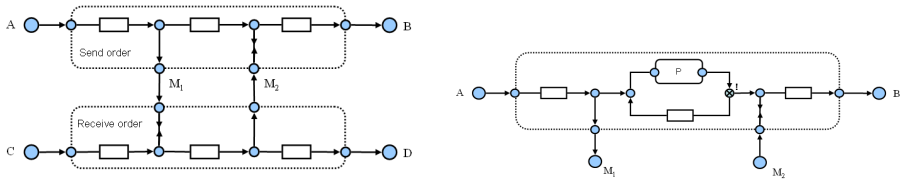
In BPMN a message flow is used to show the flow of messages between two entities that are prepared to send and receive them. Therefore, by default we can represent the BPMN message flow using synchronous channels. However, BPMN does not aim at specifying any further details about entity communication except for textual annotations. In contrast, the Reo syntax enables the process designers to model this aspect at a high level of detail. Thus, one can



(a) Basic gateways

(b) Complex gateway: lossy m out of n join

Fig. 6. Modeling BPMN gateways in Reo



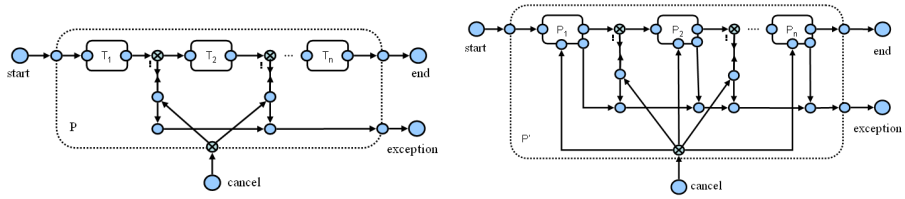
(a) Synchronous message exchange: Send/Receive Order Scenario

(b) Asynchronous message exchange

Fig. 7. Modeling BPMN message flows in Reo

differentiate synchronous and asynchronous message exchanges. In the former case, the sequence flow is blocked until the reply message is received. In the latter case, other activities can be performed while waiting for a reply message. Figure 7(a) shows a synchronous version of a Send/Receive Order scenario while Fig. 7(b) demonstrates how the asynchronous messaging can be represented in Reo: after sending a message M_1 the entity can perform activities of the sub-process P until a reply message M_2 is received. Here we assume that the output of the exclusive router being opened by the message M_2 has a priority and a token will successfully leave the cycle. We use a small exclamation mark to show a prioritized output on the figure.

Despite the behavioral simplicity of the basic Reo channels, the issue of building Reo connectors with a desired behavior is not a trivial task. Therefore, in the following subsections we provide Reo connectors for the most tricky BPMN constructs, namely, sub-processes with exception handling and transactions.



(a) Atomic tasks

(b) Sub-processes with possible internal exceptions

Fig. 8. Exception handling in processes consisting of sequential activities

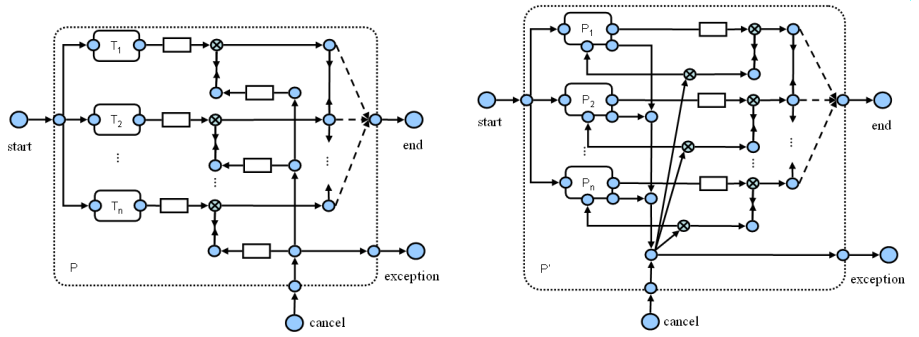
5.2 Sub-processes and Exception Handling

Each sub-process can be seen as a separate BPMN process. The translation of BPMN processes without exception handling into Reo circuits is rather straightforward. However, the occurrence of an exception event within a sub-process interrupts the execution of the sequence flow and spawns the exception flow that often affects other sub-processes and must be appropriately handled. There are two major issues here, namely, (i) to be able to interrupt a sub-process at any point of its execution and (ii) to clean all tokens/data in the circuit including those used to propagate exception events. The composition of Reo connectors implementing these issues depends on the structural aspects of sub-processes. We consider four basic constructs, namely, (i) sequential execution of atomic tasks, (ii) sequential execution of sub-processes, (iii) parallel execution of atomic tasks, and (iv) parallel execution of sub-processes.

Figure 8(a) depicts a Reo circuit that simulates the execution of a process P consisting of n serial atomic tasks. The normal flow traverses tasks $(T_1, T_2, ..., T_n)$ from the start to the end. Each two neighbor tasks are interconnected using an exclusive router with priority that is used to interrupt the process. Another exclusive router is used to direct a cancel message into the point where the execution token currently resides. The cancel message opens the output of the prioritized exclusive router and two tokens fire in the corresponding synchronous drain. Simultaneously, the cancel message is directed to the exception output which signals that the process has been interrupted.

In the above circuit we assumed that an atomic task, once invoked, always completes successfully. This may not be the case for some activities. Figure 8(b) depicts a Reo circuit that simulates the execution of a process P' consisting of n serial sub-processes $(P_1, P_2, ..., P_n)$. Each sub-process $P_i, 1 \leq i \leq n$, can be interrupted from outside by a cancel message or can generate an internal exception. The exception handling in the former case is analogous to the case of atomic tasks. In the latter case, the exception flow originating from a sub-process is redirected to the exception output of the process P'.

Figure 9(a) shows a process consisting of n parallel atomic tasks. This Reo circuit is essentially composed of a parallel fork and a parallel join gateways with n outgoing and n incoming branches, respectively. When a task $T_i, 1 \leq i \leq n$,



(a) Atomic tasks

(b) Sub-processes with possible internal exceptions

Fig. 9. Exception handling in processes consisting of parallel activities

is completed, the corresponding token waits in the FIFO1 channel until other tasks are completed as well. After that, the token flows to the circuit output. For interrupting the process, the cancel message is directed to each of the prioritized exclusive routers. FIFO1 channels are used to avoid synchronization of task cancelations. Indeed, the fact that some tasks were not completed when a cancel message has arrived should not prevent the interruption of the tasks in other branches. Additionally, the cancel message is directed to the exception output to signal the interruption of the process P.

If an internal exception occurs in a sub-process that is executed in parallel with other sub-processes within a process, this exception should be propagated to all other branches in order to interrupt them as well. Figure 9(b) shows how a Reo connector looks in this case. In each branch, an additional exclusive router is employed to propagate a cancel message (originating either from an internal or external event) to a sub-process being executed or to the point where the token waits for a synchronization with other sub-processes.

In the Reo connectors for processes with sequential activities we assumed that once a process has been invoked it will not be invoked again until the first invocation has completed. Such mutual exclusion behavior can be ensured by a FIFO1 channel whose source end coincides with the start state of the process and whose sink end is connected using a synchronous drain with the end state of the process. When a process is invoked, one token flows into the FIFO1 channel and waits until the execution reaches the end state, thus, preventing other tokens from entering the circuit through the process input port. It is easy to see that this assumption can be released without significant changes in the circuit behavior. The main difference is that in this case a cancel message will choose one of the executions non-deterministically and stop it without affecting the others. Please also observe that if the previous invocation has been interrupted, the valve connectors (see Fig. 4(b)) used to implement prioritized exclusive routers

may be closed. Before the next execution cycle, they must be reopened by means of messages sent to their open ports.

6 Reo Perspectives in Compliance Rule Modeling

In this section we outline our initial ideas about using Reo for modeling advanced process requirements.

Legal, regulatory, and business requirements cause a vast number of organizations to make major changes in their business processes and supporting IT infrastructures. Currently, there are no well-established techniques to ensure the compliance of a process with regulations that may be relevant for it. Most of the regulatory/legislative acts are subject to domain-specific and process-specific interpretations. Often, these interpretations are not even properly documented. Moreover, there is no obvious notation or (semi-)formal modeling language for expressing compliance concerns.

Compliance policies are very broad in nature. Clearly, some policies relate to business processes, while others may only partially do or may not relate to them at all. Business process modeling languages and their graphical representations are relevant for capturing, describing, formalizing, executing and enforcing policies that can be expressed in a form of local or global *constraints* or *permissions* and *obligations* on control or data flow. Often, process definition languages are augmented with modal or temporal logic formulae to encode certain kinds of compliance rules such as that some condition will eventually be true or will not be true until another statement becomes true. Several frameworks exploit this approach for modeling legislative/regulatory compliance rules [14, 21]. In particular, Liu et al. [14] introduce the Business Process Specification Language (BPSL) for expressing compliance concerns on top of BPEL processes. Then, BPSL constructs are automatically translated into Linear Temporal Logic (LTL) while BPEL processes are first translated into pi-calculus and finally into Finite State Machines to enable static process verification by means of model-checking techniques. Ghose and Koliadis [21] deal with BPMN processes that are further refined and represented in a form of semantically-annotated digraphs called Semantic Process Networks (SPNets). Compliance rules in this work are modeled using Computation Tree Logic (CTL). Giblin et al. [22] introduce REALM (Regulations Expressed as Logical Models), a metamodel and a method for modeling compliance rules over concept models in UML. Since the UML Object Constraint Language (OCL) does not support temporal predicates, REALM specifically focuses on time-based properties expressed in a specially designed Real-time Temporal Object Logic (RTOL). Several other approaches consider specific categories of compliance rules. For example, Governatori et al. [23] developed a Formal Contract Language (FCL) for representing compliance requirements extracted from service contracts. FCL expresses normative behavior of the contract signing parties by means of chains of permissions, obligations, and violations. Brunel et al. [24] use Labeled Kripke Structures (LKS) which are a state/event extension

of LTL both for specifying system behavior and related security requirements, also defined in a form of permissions, obligations, and violations.

In this context, we see Reo and its underlying mathematical formalisms, in particular extended CA (e.g, quantitative CA [16], timed CA [17], resource-sensitive timed CA [25]), as a common operational semantics for unambiguous modeling of both process workflows and compliance rules. As mentioned in Section 2, Reo and CA have been successfully applied for service composition with end-to-end QoS guarantees [16] and for the construction of systems with real-time properties expressed by means of temporal logics [17]. Existing model-checking and bisimulation tools for Reo are able to automatically verify some important properties of process models such as the absence of deadlocks, reachability of certain states and proper completion, as well as to check the behavioral equivalence of Reo circuits [26]. Moreover, we believe that using the application of graph transformation theory to Reo [27] we will be able to guarantee process compliance with some structural requirements such as that "*any large loan must be approved by at least two authorized bank officers*".

Our intuition that Reo and its formal models can be used for representing and reasoning about (some kinds of) process-related compliance concerns is supported by a recent independent work in this direction. Brandt and Engel [28] apply Reo, Abstract Behavior Types, and algebraic graph transformations in addition to DSLs for secure modeling of distributed IT systems in a real-world banking scenario. The authors as well claim that security requirements can be modeled by graph constraints on the domain specific models. The mentioned formal methods in particular are used to control requirements originating from security compliance frameworks such as ISO 27001:2005, ISO 27002:2007, SOX or CobiT (e.g., firewall placement and secure connection).

7 Conclusions and Future Work

In this paper, we have presented a novel approach to semantically unambiguous modeling of business process workflows. We have used Reo channels as basic building blocks to model a comprehensive set of BPMN objects and advanced constructs such as sequential and parallel sub-processes with exception handling. The mapping of BPMN diagrams into Reo networks helps to unveil some process aspects that otherwise may remain underspecified (e.g. message synchronization). The resulting Reo models make it possible to formally analyze and compare business processes. In addition, we have discussed how Reo can cope with possible business process modeling extensions that aim at enforcing process-related compliance concerns.

Our approach has several advantages over existing efforts to formalize BPMN semantics, most notably [4, 5]. In contrast to the Petri-net-based approaches [4] our model appropriately deals with exception handling and concurrency. In contrast to the CSP-based approaches [5], Reo is compositional and preserves the exact structure of BPMN diagrams by appropriate grouping of basic channels and finer-grained connectors into coarser-grained connectors. Similarly to [6], we

can take into account time-aware aspects of business processes by means of timer connectors. Moreover, in our work we have considered a significantly larger set of BPMN elements. However, part of our results, in particular, representation of compensation associations, transaction modeling and dynamic reconfiguration of Reo connectors to deal with multiple instances of the same activity, remain uncovered in this paper and are subjects for upcoming publications.

Our future work includes implementation of a BPMN to Reo convertor. We also plan to elaborate our initial ideas on applying Reo and CA for modeling and analyzing compliance-driven processes as discussed in this paper, both theoretically and on a number of practical examples illustrating how the proposed approach can be used to alleviate the problem of erroneous process implementation.

8 Acknowledgements

This work is part of the IST COMPAS project, funded by the European Commission, FP7-ICT-2007-1 contract number 215175, http://www.compas-ict.eu/.

References

1. Curbera, F., Goland, Y., Klein, J., Leymann, F.: Business process execution language for web services. Technical report, IBM, http://www.ibm.com/developerworks/library/ws-bpel/ (2002)
2. Kavantzas, N., Burdett, D., Ritzinger, G.: Web services choreography description language (WS-CDL) version 1.0. Working draft, W3C, http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427 (2004)
3. (OMG), O.M.G.: Business process modeling notation (BPMN) specification. Final adopted specification, OMG, http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf (2006)
4. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of BPMN process models. In: Information and Software Technology (IST). (2008)
5. Wong, P., Gibbons, J.: A process semantics for BPMN. Technical report, Queensland University of Technology, http://www.comlab.ox.ac.uk/publications/publication454-abstract.html (2007)
6. Wong, P., Gibbons, J.: A relative timed semantics for BPMN. Technical report, Queensland University of Technology, http://www.comlab.ox.ac.uk/publications/publication1496-abstract.html (2007)
7. Recker, J., Mendling, J.: On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In: Proc. of the Int. Conf. on Advanced Information Systems Engineering. (2006) 521–532
8. Ouyang, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: Pattern-based translation of BPMN process models to BPEL web services. Int. Journal of Web Services Research (JWSR) **5**(1) (2007) 42–61
9. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Formal semantics and analysis of control flow in WS-BPEL. Science of Computer Programming **67**(2-3) (2007) 162–198

10. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: Proc. of the Int. Workshop on Web Services and Formal Methods. Volume 4937 of LNCS., Springer (2008) 77–91
11. Lucchia, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming **70**(1) (2007) 96–118
12. Nakajima, S.: Model-checking behavioral specification of BPEL applications. Electronic Notes in Theoretical Computer Science (ENTCS) **151** (2006) 89–105
13. McCarty, L.T.: A language for legal discourse. In: Proc. of the Int. Conf. on Artificial Intelligence and Law (ICAIL'89), ACM Press (1989) 180–189
14. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal **46**(2) (2007) 335–361
15. Meng, S., Arbab, F.: Web service choreography and orchestration in Reo and constraint automata. In: Proc. of the ACM Symposium on Applied Computing (SAC'07), ACM Press (2007) 346–353
16. Arbab, F., Chothia, T., Meng, S., Moon, Y.J.: Component connectors with QoS guarantees. In: Proc. of the Int. Conf. on Coordination Languages (Coordination'07). Volume 4467 of LNCS., Springer (2007) 286–304
17. Arbab, F., Baier, C., de Boer, F.S., Rutten, J.J.M.M.: Models and temporal logics for timed component connectors. Int. Journal on Software and Systems Modeling **6**(1) (2007) 59–82
18. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science **14**(3) (2004) 329–366
19. Arbab, F., Rutten, J.: A coinductive calculus of component connectors. In: Recent Trends in Algebraic Development Techniques: Proc. of the Int. Workshop, WADT 2002, Revised Selected Papers. Volume 2755 of LNCS., Springer (2003) 34–55
20. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming **61** (2006) 75–113
21. Ghose, A.K., Koliadis, G.: Auditing business process compliance. In: Proc. of the Int. Conf. on Service-Oriented Architectures (ICSOC'07). Volume 4749 of LNCS., Springer (2007) 169–180
22. Giblin, C., Liu, A.Y., Müller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models (REALM). In: Proc. of the 18th Annual Conf. on Legal Knowledge and Information Systems. (2005) 37–48
23. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: Proc. of the Int. Enterprize Distributed Object Computing Conf. (EDOC'06), IEEE Computer Society (2006) 221–232
24. Brunel, J., Cuppens, F., Cuppens, N., Sans, T., Bodeveix, J.P.: Security policy compliance with violation management. In: Proc. of the Workshop on Formal Methods in Security Engineering (FMSE'07), ACM Press (2007) 31–40
25. Meng, S., F.Arbab: On resource-sensitive timed component connectors. In: Proc. of the Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'07). Volume 4468 of LNCS., Springer (2007) 301–316
26. T. Blechmann, C.B.: Checking equivalence for Reo networks. In: Proc. of the Int. Workshop on Formal Aspects of Component Software (FACS). (2007)
27. Koehler, C., Lazovik, A., Arbab, F.: Connector rewriting with high-level replacement systems. Electronic Notes in Theoretical Computer Science (ENTCS) **194**(4) (2008) 77–92
28. Brandt, C., Engel, T., Braatz, B., Hermann, F., Ehrig, H.: An approach using formally well-founded domain languages for secure coarse-grained IT system modelling in a real-world banking scenario. In: Proc. of the Australasian Conf. on Information Systems (ACIS'07). (2007) 386–395