Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems

Natallia Kokash and Farhad Arbab

CWI, Science Park 123, Amsterdam, The Netherlands
firstName.lastName@cwi.nl

Abstract. In this paper, we present a framework for formal modeling and verification of service-based business processes with focus on their compliance to external regulations such as Segregation of Duties (SoD) or privacy protection policies. In our framework, control/data flow is modeled using the exogenous coordination language Reo. Reo process models are designed from scratch or (semi-)automatically obtained from BPMN, UML or WS-BPEL specifications. Constraint automata (CA), a semantic model for Reo, provide state-based representations of process workflows and enable their verification by means of model checking technology. Various extensions of CA make it possible to analyze time-, resource- and Quality-of-Service (QoS) process models.

1 Introduction

One if the key ideas of Service-Oriented Computing (SOC) is to enable the development of cross-organizational software systems by composing pre-existing services. Services are self-contained and loosely-coupled applications that advertise their interfaces and/or observable behavior specifications. Given such specifications, one can compose appropriate services to realize a certain business logic. This paradigm helps designers to abstract from low-level details and implementation issues, reduces time and cost of software development and increases its reusability and adaptability to changing process requirements.

Despite this promise, implementation of business processes by composing services remains a challenging task. The problem of ensuring that the composed behavior is compliant to the process specification and related business requirements is one of the key issues here. Formal approaches to process behavior specification such as Petri-nets, automata-based models or process algebras together with logic-based formalisms for specifying system properties provide rigorous tools for compliance analysis. However, complexity, the absence of visual notations and difficulties to obtain these models from widely-recognized high-level specification formats such as Unified Modeling Language (UML) or Business Process Modeling Notation (BPMN) limit their utility in practice. Another problem is the absence of actually implemented software tools that use theoretical approaches in this area to support automated process analysis and generate executable code to run corresponding service compositions. Finally,

business requirements may affect various aspects of the corresponding process model such as its control, data or time- flow, impose constraints on access control or performance. All these issues entail the need for an extensible formal model for service-based business process design suitable for reasoning about various types of functional and non-functional properties.

In this paper, we introduce a framework to benefit compliance-aware business process development with formal analysis and automated code generation. This work is part of the EU project COMPAS (Compliance-driven Models, Languages, and Architectures for Services) which aims at designing and implementing novel models, languages, and an architectural framework to ensure dynamic and ongoing compliance of software services to business regulations and stated user-service requirements. We understand *compliance* as any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process. Such compliance rules come from internal sources, e.g., technical instructions, regulations aimed at improving Quality-of-Service (QoS) delivered to end users, Service-level Agreements (SLAs), or external sources such as user privacy protection policies, fraud prevention regulations and laws. *Compliance policy* is a logical grouping of a set of coherent rules that realizes a specific goal, e.g., fraud prevention by limiting access to vulnerable data. By context-aware analysis and stepwise decomposition of organizational high-level goals such as "comply to Sarbanes-Oxley Act or Basel II" to a set of relevant policies and, finally, to concrete compliance rules, we can come up with a number of formally-expressed constraints that must be satisfied to guarantee the compliance of a particular process to the initial requirements. We aim at developing a unified extensible behavioral model that is able to incorporate various types of information relevant to automated design-time compliance analysis. Our solution is based on Constraint Automata (CA) which offers an operational model for specifying composite service behavior. CA are essentially a variant of a labeled transition system where transitions are augmented with pairs of synchronization and data constraints. The states of a CA stand for the process configurations while transition labels can be viewed as input/output 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). This model is fully compositional, and can express arbitrary mixes of synchronous and asynchronous communication. CA were developed as a semantic model for the coordination language Reo [1] (although several other semantic models for Reo are available) and later have been extended to express time dependent behavior, probabilistic, and stochastic systems.

There are several reasons that motivate our choice of Reo and CA for specifying the behavioral composition of business processes and web services. First, Reo has a simple graphical notation which makes it easy to use in practical applications by process designers without any prior experience in formal methods. A small number of Reo modeling primitives (channels) are sufficient for representing rather complex behavioral protocols. Second, precise semantics of Reo in terms of CA enables automated process verification. Reo process models can

be automatically translated into CA which are suitable for representing service compositions with QoS guarantees [2], and time- and resource-aware processes [3, 4]. Moreover, there is a solid set of software tools supporting process modeling, verification and code generation based on Reo process models.

The rest of this paper is organized as follows: Section 2 contains an overview of domain-level compliance-aware business process design. In Section 3, we introduce the coordination language Reo and illustrates its application to business process modeling. In Section 4, we discuss several extensions of constraint automata used for automated workflow analysis. Section 5 exemplifies the application of Reo/CA for detecting errors in process workflow. Section 6 illustrates how our framework can be applied to deal with advanced process requirements such as separation-of-duty and privacy constraints. Section 7 is a survey of related work. Finally, Section 8 provides our conclusions and an outline of our future work.

2 Business Process Modeling

In SOC, *business process* is defined as a collaborative service that is closely linked to a business purpose[1]. A collaborative service is a service implemented through the composition of other services. This definition poses no restrictions on the nature of the composed services. We can distinguish *functional services* which perform self-contained business operations, and *coordination services*, which implement so called "glue code". In our approach, we assume that observable behavior of functional services is described using CA, while "glue code" is modeled by means of the Reo coordination language.

Traditional graphical notations for business process modeling such as BPMN and UML Activity Diagrams (ADs) represent business processes in the form of abstract tasks (activities) with a control flow over them. Additionally, BPMN provides modeling primitives for specifying important events occurring in the system, exception handling, compensation associations and transactional subprocesses, which make it possible to depict most common features of real-world business processes. However, the specification of this notation does not assume a formal semantics. As a result, BPMN process diagrams can be misunderstood and require preprocessing and refinement before they become suitable for rigorous analysis or software implementation.

Figure 1 shows a BPMN diagram for a sample purchase order process that appeared in [5]. It consists of three basic activities, *checkCreditCard*, *prepareProducts* and *shipItems*. When a purchase request is received, the client's credit card is checked and the requested products are prepared simultaneously. After that the prepared products are shipped to the customer.

UML Sequence Diagrams (SDs) present a conceptually different approach to system modeling. The goal of UML SDs is to model dynamic system behavior in terms of entities, components/services or objects that exchange messages

[1] http://www.nexof-ra.eu/?q=node/187

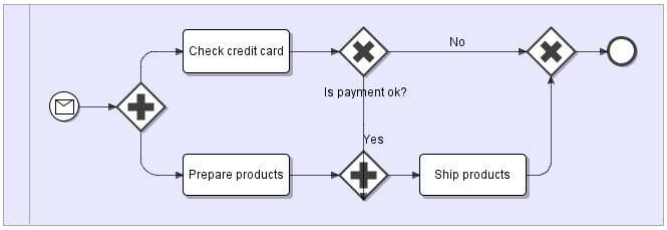


Fig. 1. BPMN diagram for the purchase order scenario [5]

or functional calls. The diagram conveys the information along the horizontal and vertical dimensions: the vertical dimension shows the time sequence of messages/calls as they occur, and the horizontal dimension shows with the help of lifelines, the object instances that the messages are sent to. In the context of business process modeling UML SDs are convenient to represent scenarios involving several interacting entities such as auctions or service contract negotiation.

WS-BPEL is a language for describing executable business processes on top of WSDL service specifications. Due to the lack of graphical notation and the need to deal with implementation-level details, WS-BPEL is not suitable for domain analysis and conceptual business process modeling, although some efforts exist to adopt WS-BPEL for this purpose. Nevertheless, modern business processes are rarely developed from scratch. We assume they can be composed from reusable business process fragments with existing behavioral specifications in WS-BPEL.

The aforementioned notations lack support for compliance. For example, they provide no standard ways to express Segregation of Duties (SoD) requirements, show link dependencies, or specify QoS constraints on sub-processes rather than using textual annotations or additional domain-specific languages. Both industry and academia have proposed numerous extensions for process compliance support on top of these notations. For instance, in [6] BPMN processes are annotated with QoS information, in [7] additional textual annotations expressing task authorization constraints are introduced, while in [8] a language for specifying regulatory compliance on top of WS-BPEL is proposed. Due to the higher level of expressiveness of Reo, we can explicitly model some of these requirements in a formal way. In Section 6, for instance, we demonstrate how SoD can be enforced using our framework.

3 Process Modeling with Eclipse Coordination Tools

In this section, we summarize the main concepts of Reo. Further details about Reo and its semantics can be found in [9, 1, 10].

Reo [1] is a channel-based exogenous coordination model wherein complex coordinators, called *connectors*, are compositionally constructed from simpler ones. Complex connectors in Reo are formed as a network of primitive connectors, called *channels*, that serve to provide the protocol which controls and

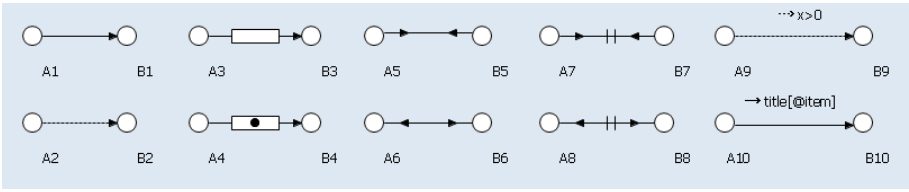


Fig. 2. Basic Reo channels

organizes the communication, synchronization and cooperation among the services that they interconnect. Each channel has two channel ends which can be of two types: source or sink. A source end accepts data into its channel, and a sink end dispenses data out of its channel. It is possible for both ends of a channel to be either sinks or 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 2 shows the graphical representation of basic channel types in Reo. A *synchronous channel* SYNC(A1,B1) 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* LOSSY(A2,B2) 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. A *FIFO1 channel* FIFO1(A3,B3) represents an asynchronous channel with one buffer cell which is empty if no data item is shown in the box. If a data element d is contained in the buffer of a FIFO1 channel, it looks like a channel FIFO1_FULL(A4, B4) in this figure.

A *synchronous drain* SYNC_DRAIN(A5,B5) has 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 this channel are lost. An *asynchronous drain* ASYNC_DRAIN(A7,B7) accepts data items through its source ends and loses them, but never simultaneously. *Synchronous* and *asynchronous spouts* SYNC_SPOUT(A6,B6) and ASYNC_SPOUT(A8,B8) are duals to the drain channels, as they have two sink ends. For a *filter channel* FILTER(A9,B9), 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 d; all data items $d \notin P$ are always accepted through the source end but are immediately lost. Finally, a *transformer* channel TRANSFORMER(A10, B10) accepts a data item and rewrites it according to the transform expression of the channel (e.g., xPath expression), as the data item passes through.

The aforementioned channels are supported by Eclipse Coordination Tools (ECTs), a tool suite consisting of Eclipse plug-ins for designing, testing and verification of connectors, as well as runtime engines for executing coordination

protocols on multiple platforms [11]. This set can be extended with new channels. For example, *timer* channels, namely, *t-timer*, *t-timer with off- and reset-option* and *t-timer with early expiration* have been introduced to deal with time-aware service coordination [3]. Essentially, these channels accept data items at their input ports and dispose them at their output ports after t units of time, thus, enabling modeling of process timeouts and delays. Additionally, *(a)synchronous drains with filter conditions* appear to be useful for business process modeling when conditional synchronization of two flows is required.

Complex connectors are constructed by composing simpler ones via the *join* and *hiding* operations. Join plugs two channel-ends together creating a node at the point of connection. To this node one can connect more channels via join afterwards. If more than one accepting channel end is connected to a node every incoming message is simultaneously written to all outgoing channels whenever all outgoing channels at the node are ready to accept data. Whenever more than one channel-end offers data at a node a non-deterministic choice decides which data item is taken and written to all outgoing channels. The hiding operation hides away one node which means that the data-flow occurring at this node cannot be observed from outside and no new channel-end can be connected to this node.

Figure 3 shows a Reo connector that simulates the purchase order scenario introduced in Figure 1. We represent BPMN activities as simple FIFO1 channels meaning that data flow in the source end of each channel corresponds to the start of the activity, data flow in the sink end of the channel corresponds to the end of the activity, while the data token residing in the channel buffer implies that the activity is being executed. Special components, *Writer* and *Reader*, are used to introduce and consume data flow at the beginning and the end of the process. Nodes obtained by joining both source and sink ends of Reo channels are called *mixed* and considered to be *internal* for the connector. In contrast, nodes where only source or only sink channel ends are merged are considered to be *external* and can be attached to writers or readers, respectively. The Reo editor in the ECT environment automatically highlights the internal nodes with grey color. Two parallel flows are initiated by joining a sink end of a Reo channel with start ends of two other channels (see, e.g., node *start*), and synchronized using a synchronous drain (see, e.g., SYNC_DRAIN(*paymentIsOk*, *paymentAck*) which requires tokens on both sides to fire). As explained in [12], data-driven conditional choice can be realized using Reo filter channels. However, in this model, for simplicity, we abstract from data issues and use a non-deterministic exclusive router connector to direct a token from the node *isPaymentOk* either to the node *paymentIsNotOk* or to the node *paymentIsOk*, thus, obtaining a process model with two possible execution paths.

Preliminary business process analysis and simulation can be accomplished using a tool that generates flash animated simulations of Reo connectors. The plug-in depicts the connector shown in the editor in the animation view and generates a list of possible execution scenarios. The parts of the connector highlighted blue represent synchronous data flow. Tokens move along these synchronous regions.

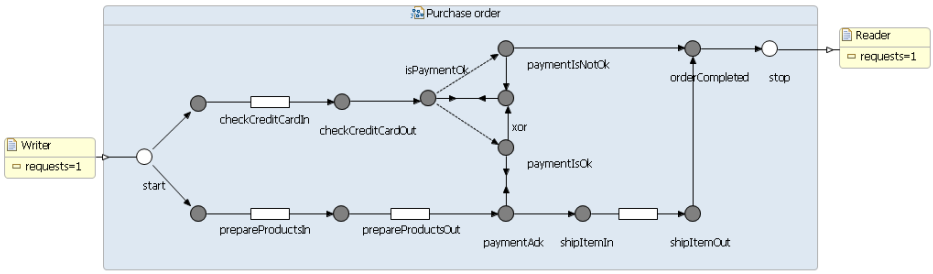


Fig. 3. Reo circuit for the purchase order scenario

Two simulation modes are supported: a *plain mode*, which demonstrates all possible execution alternatives for the whole process, and a *guided* or *stepwise mode*, which shows each execution step separately, including all possible alternatives for a current step.

Reo process models can be automatically obtained from BPMN diagrams using the *BPMN2Reo* converter available as part of ECT. In our previous work [12], we defined rules for mapping all major BPMN modeling primitives to Reo. In this way, we can also refine ambiguous BPMN diagrams by giving them precise semantics. The mapping of UML ADs to Reo is similar to the mapping of a subset of BPMN to Reo and will be integrated into ECT as well. The theoretical basis for the converter from UML SDs to Reo is given in [13]. Such conversion is automatically performed by the *UMLSDs2Reo* mapping tool which is currently being integrated into ECT. Finally, *BPEL2Reo* converter is a tool provided by the University of Tehran for converting BPEL process specifications to Reo [14].

4 Formal Behavioral Model for Service Composition: Extended Constrained Automata

There are several extensions of CA that can be useful for automated analysis of time-, resource- and QoS-aware Reo process models.

Let $\mathcal{N}$ be a finite set of nodes, $Data$ a fixed, non-empty set of data that can be sent and received via Reo channels, and define a function $\delta : N \rightarrow Data$, $N \in \mathcal{N}$ as a data assignment. CA use a symbolic representation of data assignments by data constraints which are propositional formulas built from the atoms $d_A \in P$, $d_A = d_B$, and $d_A = d$ with standard boolean connectors, where $A, B \in \mathcal{N}$, d_A is a symbol for the observed data item at node A and $d \in Data$, $P \in Data$. We write $DA(N)$ to refer to the set of all data assignments for the node-set N, $DC(N)$ to denote the set of data constraints that at most refer to the observed data items d_A at node $A \in N$, and DC for $DC(\mathcal{N})$.

Definition 1 (Constraint Automaton (CA) [10]). *A CA is a tuple* $\mathcal{A} = (S, S_0, \mathcal{N}, E)$ *where*

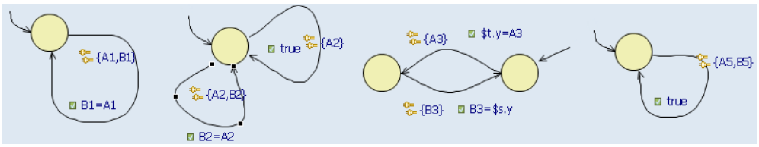


Fig. 4. Constraint automata for basic Reo channels

- S *is a finite set of control states,*
- S_0 *is is a set of initial states,*
- $\mathcal{N}$ *is a finite set of node names (e.g., I/O ports of components/services),*
- E *is a finite subset of* $S \times 2^N \times DC \times S$ *called the transition relation of* $\mathcal{A}$*,*
- DC *is a data constraint that plays the role of the guard for a transition.*

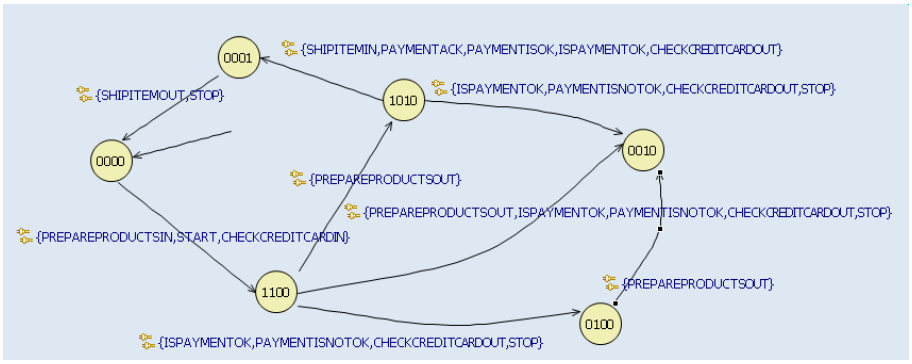
Figure 4 shows the CA for the basic Reo channels. The behavior of any Reo process model can be obtained by computing the product of these automata.

Definition 2 (Product of CA [10]). *The product for two constraint automata* $\mathcal{A}_1 = (S_1, S_{0,1}, \mathcal{N}_1, E_1)$ *and* $\mathcal{A}_2 = (S_2, S_{0,2}, \mathcal{N}_2, E_2)$ *is defined as a constraint automaton* $\mathcal{A}_1 \bowtie \mathcal{A}_2$ *with the components* $(S_1 \times S_2, S_{0,1} \times S_{0,2}, \mathcal{N}_1 \cup \mathcal{N}_2, E)$ *where* E *is the set of transitions* e *given by the following rules, where* $e_1 \in E_1$ *and* $e_2 \in E_2$*:*

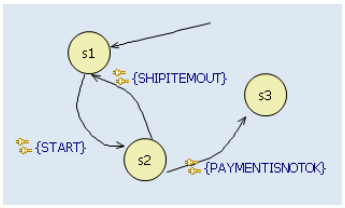
- *If* $e_1 = (s_1, N_1, g_1, s_1')$*,* $e_2 = (s_2, N_2, g_2, s_2')$*,* $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1 = \emptyset$ *and* $g_1 \wedge g_2$ *is satisfable, then* $e = (\langle s_1, s_2 \rangle, N_1 \cup N_2, g_1 \wedge g_2, \langle s_1', s_2' \rangle)$*.*
- *If* $e_1 = (s_1, N, g, s_1')$ *where* $N \cap N_2 = \emptyset$*, then* $e = (\langle s_1, s_2 \rangle, N, g, \langle s_1', s_2 \rangle)$*.*
- *If* $e_2 = (s_2, N, g, s_2')$ *where* $N \cap N_1 = \emptyset$*, then* $e = (\langle s_1, s_2 \rangle, N, g, \langle s_1, s_2' \rangle)$*.*

Figure 5(a) shows the CA for one instance of the purchase order scenario, that is, only states that are reachable from the initial state after a single reading operation. Such CA are automatically obtained from Reo process models. Intuitively, each state of a CA without hiding corresponds to a unique combination of empty/full buffers of the corresponding Reo circuit. We reflect this dependency in state names by writing 1 for a full FIFO1 channel and 0 for an empty FIFO1 channel assuming their top-down left-to-right order in Fig. 3. CA transition labels correspond to the names of Reo nodes where data flow is simultaneously observed during the transition. After hiding internal ports, CA control states represent process states observable by an external user. In this example, three logical states have been identified: the initial state s_1, state s_2 corresponding to the started process execution, and state s_3 that implies the presence of the deadlock in the process model due to the payment failure.

Reo timer channels can be exploited for time-aware analysis of business process models with ECT. In our case study, one may be interested to know how much time is required to process a single purchase order. The operational model for time-aware Reo connector circuits is given with the help of *Timed Constraint Automata (TCA)*, which can be defined as follows. Additionally to the notation introduced for CA, let $\mathcal{C}$ be a finite set of clocks. A clock assignment means a



(a) Without hiding



(b) With hiding

Fig. 5. Constraint automata for the purchase order scenario

function $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$. If $t \in \mathbb{R}_{\geq 0}$ then $v + t$ denotes the clock assignment that assigns the value $v(x) + t$ to every clock $x \in C$. If $C \in \mathcal{C}$ then $v[C := 0]$ stands for the clock assignment that returns the value 0 for every clock $x \in C$ and the value $v(x)$ for every clock $x \in \mathcal{C} \setminus C$. A clock constraint (denoted cc) for C is a conjunction of atoms of the form $x \bowtie n$ where $x \in \mathcal{C}, N \in \{<, \leq, >, \geq, =\}$ and $n \in \mathbb{N}$. $CS(\mathcal{C})$ (or CS) denotes the set of all clock assignments and $CC(\mathcal{C})$ (or CC) the set of all clock constraints.

Definition 3 (Timed Constraint Automaton (TCA) [3]). *A TCA is an extended CA* $\mathcal{A} = (S, S_0, \mathcal{N}, E, C, ic)$ *where the transition relation* E *is a finite subset of* $S \times 2^N \times DC \times CC \times 2^C \times S$ *such that* $dc \in DC(N)$ *for any transition* $e = (s, N, dc, cc, C, s') \in E$*,* C *is a finite set of clocks, and* $ic : S \to CC$ *is a function that assigns to any state* s *an invariance condition* $ic(s)$*.*

The time required to perform certain actions in the process may depend on the availability of resources. For example, the time to deliver products in our case study may depend on the capacity of a purchase delivery service. Moreover, most of the systems have to change their states if an interaction has not occurred or an operation has not been completed within a certain timeout. For modeling such requirements in business processes, we extend Reo with time and resource-awareness information. The formal model for this extension relies on the notion of *Resource-aware Timed Constraint Automata (RSTCA)* [4].

It is possible to enable QoS analysis of Reo process models by assigning certain properties to Reo basic channels such as the *execution time* required to transmit a data item, the *cost* of a single data transmission, the *bandwidth* that limits simultaneous data transmission, or *reliability* which represents the probability of a successful data transmission. Operations over these parameters can be formally specified using a notion of Q-algebra [15]. A Q-algebra is an algebraic structure $R = (C, \oplus, \otimes, ||, \mathbf{0}, \mathbf{1})$ where C is the domain of R and represents a set of QoS values. The operation $\oplus$ induces a partial order on the domain of R and is used to define a preferred value of QoS dimension, $\otimes$ is an operator for sequential channel composition, while $||$ is an operator for parallel channel composition. For example, the Q-algebras corresponding to the above QoS dimensions are as follows:

- *Execution time*: $(\mathbb{R}_{\geq} \cup \{\infty\}, min, +, max, \infty, 0)$,
- *Cost*: $(\mathbb{R}_{\geq} \cup \{\infty\}, min, +, +, \infty, 0)$,
- *Bandwidth*: $(\mathbb{N} \cup \{\infty\}, max, min, +, 0, \infty)$,
- *Reliability*: $([0, 1], max, \times, \times, 0, 1)$.

Taking into account this definition, *Quantitative Constraint Automata (QCA)* [2] is as an extended CA $\mathcal{A} = (S, S_0, \mathcal{N}, E, R)$ where the transition relation E is a finite subset of $\cup_{N \in \mathcal{N}} S \times \{N\} \times DC(N) \times C \times S$ and $R = (C, \oplus, \otimes, ||, 0, 1)$ is a labeled Q-algebra with domain C. However, this model is not sufficient for practical applications as certain QoS (e.g., execution time) may change the intended behavior of Reo circuits. For example, consider a circuit consisting of two channels, SYNC(A, B) and ASYNC_DRAIN(A, B), whose execution times are t_1 and t_2, respectively. Assuming that the asynchronous drain accepts data at port A at time t_0 only if there is no data flow on port B within the time interval $[t_0, t_0 + t_1]$, while the synchronous channel accepts data at port A only if it can dispose it at time $t = t_0 + t_2$, the overall connector will accept data if $t_2 > t_1$, and get blocked, otherwise. Therefore, QoS-aware Reo models require more expressive formalisms to represent their behavior. Indeed, depending on whether delays are attributed to input/output operations on source/sink ends or to data transmission across the channel, the computation of a delay of data transmission across a composite connector may differ. Another type of automata, namely, *Quantitative Intentional Automata (QIA)*, have been introduced to specify the semantics of stochastic Reo: a version of Reo where one or more delays can be assigned to input/output operations on channel ends and transmission delays. QIA can be converted to Continuous-Time Markov Chains (CTMC) and used for process performance analysis using PRISM model checker [2]. More details about this work can be found in [16].

5 Verifying Business Process Specification

The main purpose of the formal models presented above is to enable automated verification of compliance-aware business processes and web service compositions. This can be accomplished with the help of the *Vereofy* model checking

[2] http://www.prismmodelchecker.org/

tool [17] developed at the University of Dresden. Vereofy is integrated into ECT, but also can be executed from a command shell. It uses two input languages, namely, Reo Scripting Language (RSL), and a guarded command language called Constraint Automata Reactive Module Language (CARML) which are textual versions of Reo and CA, respectively. Scripts in these languages are automatically generated from graphical Reo/CA models, however, they can be written manually as well, e.g., to specify connectors composed of a huge number of channels with repeating patterns.

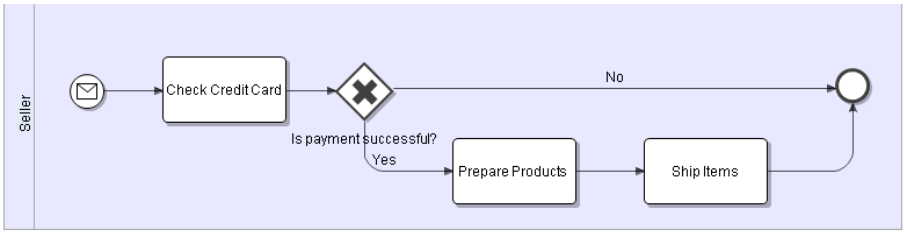
Vereofy supports linear and branching-time model checking. Properties of the Reo circuits are specified either in Linear Temporal Logic (LTL) or Alternating-time Stream Logic (ASL). LTL allows designers to encode formulae about the future of execution paths such as that some condition will eventually be true or will be true until another condition remains true. Computation Tree Logic (CTL) is a branching-time logic which models time as a tree-like structure and allows designers to encode formulae about the future of possible execution paths. Branching Time Stream Logic (BTSL) is a logic specifically designed for Reo [18]. It extends CTL with the ability to express conditions on data flow in channel nodes using regular expressions. A standard Alternating-time temporal Logic (ATL) aims at reasoning about existence or absence of a coalition's strategy to achieve or avoid a specific temporal goal given the behavioral specification of each component. ASL is a CTL-like branching-time logic which combines features of BTSL and ATL.

For model checking, a constraint automaton needs to be associated with an arbitrary finite data domain (Data), which collects all possible data items transmitting through the corresponding Reo circuit or stored within the local variables of components. Data is a global data type, which can be Bool, int, or enum, depending on the user settings. The default data domain is int(0,1) and in our case it is used for control flow analysis.

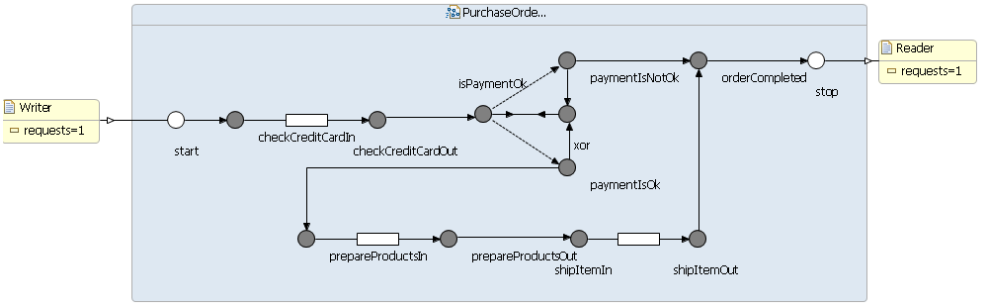
There exist a number of studies on how system properties can be expressed using logical formalisms. COMPAS deliverable D2.2 [19] examines the suitability of Deontic logic, LTL, and XML-based approaches for formal specification of regulatory compliance requirements. It demonstrates that basic compliance requirements can be successfully expressed in all these languages, but advocates the use of LTL as the most comprehensible notation by end users. In our case study, the following LTL formula

$$\mathbf{G}(PrepareProductsOut \rightarrow \mathbf{F}\, ShipItemsIn)$$

states that whenever the data flow is observed in *PrepareProductsOut* port meaning that the activity *PrepareProducts* is finished, a data flow must be eventually observed in *ShipItemsIn* port corresponding to the invocation of the *ShipItems* activity. An ASL formula $\mathbf{AG}[\mathbf{EX}[true]]$ which literally means *"for all paths, it is globally true that there exists a next state"* can be used for deadlock detection. Both these formulae fail for the Reo process model presented in Figure 3. Indeed, in this scenario, if customer payment fails, the products remain prepared (e.g., packed), but will never be shipped. A proper model for this scenario is shown in Figure 6.



(a) BPMN model



(b) Reo process model

Fig. 6. Refined purchase order scenario

Once the process model is refined to satisfy all necessary conditions, it can be turned into an executable process. Figure 7 shows our scenario with Reo primitives called *components* attached to in/out ports of the FIFO1 channels which simulate activity invocations and replies. The observable behavior of real world services expressed by means of extended CA can be associated with these components. An executable code that realizes such a service composition is automatically generated by code generation tools of ECT.

Certain compliance requirements can be seen as informal descriptions of ideal business process fragments. Such process fragments in their turn can be modeled using Reo and/or CA. In this case, the compliance of software systems actually used in organizations with the ideal processes can be established by checking bisimulation equivalence of their corresponding CA models. Beforehand, one can abstract from unimportant details of an existing process by hiding data flow of the automata ports that are not relevant to a particular compliance policy. Algorithms for finding bisimilar states, and, thus, checking behavioral equivalence of CA or Reo circuits are presented in [20].

6 Compliance-aware Process Modeling by Examples

One of the popular resource-aware constraints is a dual control or so-called four eyes principle. It is applied, for example, in investment banking, to segregate the

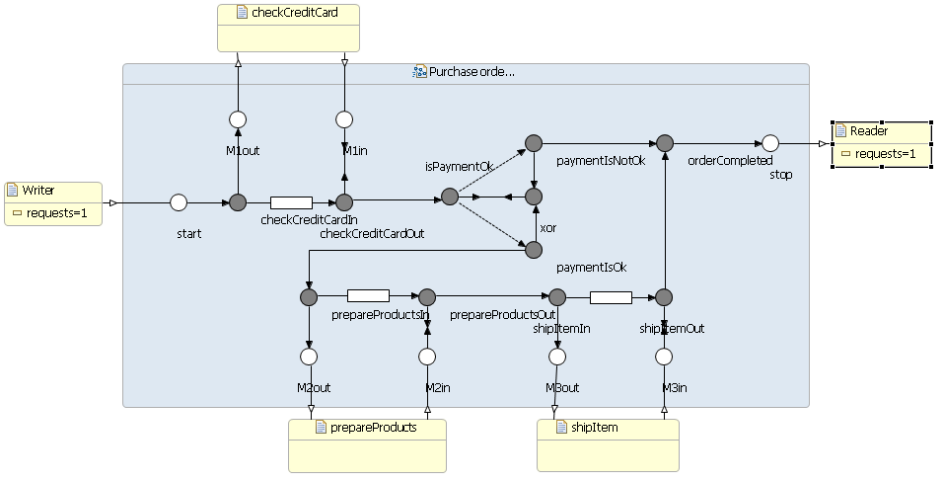


Fig. 7. Reo model of a service composition for the purchase order scenario

duties of a trader from the duties of an internal auditor. In the corresponding process model, it is important to ensure that generally each bank clerk can play both roles, but he/she cannot play both roles in a single instance of the process. Later, the term SoD was introduced for referring to a principle of information protection and fraud prevention by limiting user access to vulnerable data and/or operations. This category of compliance requirements is extensively discussed in [21, 22].

Figure 8 shows a Reo process model which consists of sequential invocation of two activities, $T1$ (investment) and $T2$ (authorization), simulated using FIFO1 channels in separate Reo connectors. In this model, the activity $T1$ can be executed by three authorized clerks, *Alice*, *Bob* and *Clara*, while the activity $T2$ can be executed by *Alice*, *Bob* and *Frank*. These access control rights are modeled by means of filter channels FILTER(T1in, T1start) and FILTER(T2in, T2start) with conditions

$$\#T1in.clerkName \in D1 = \{Alice, Bob, Clara\}$$

and

$$\#T2in.clerkName \in D2 = \{Alice, Bob, Frank\},$$

respectively. Here we use "#X" to refer to data observed at port X. Parts of the *Coordinator* circuit emphasized with dashed rectangles impose the dual control constraint in this scenario. The two *Writer* components connected to ports $U1$ and $U2$ correspond to two users, *trader* and *internal auditor*, who login to the system and perform the investment and authorization operations, respectively. The synchronous drain channel with filter FILTER_SYNC_DRAIN(U2, A5) uses a condition $\#U2 \neq \#A5.trador$ to ensure that the internal auditor differs from the trader who performed the investment operation in this process instance. This

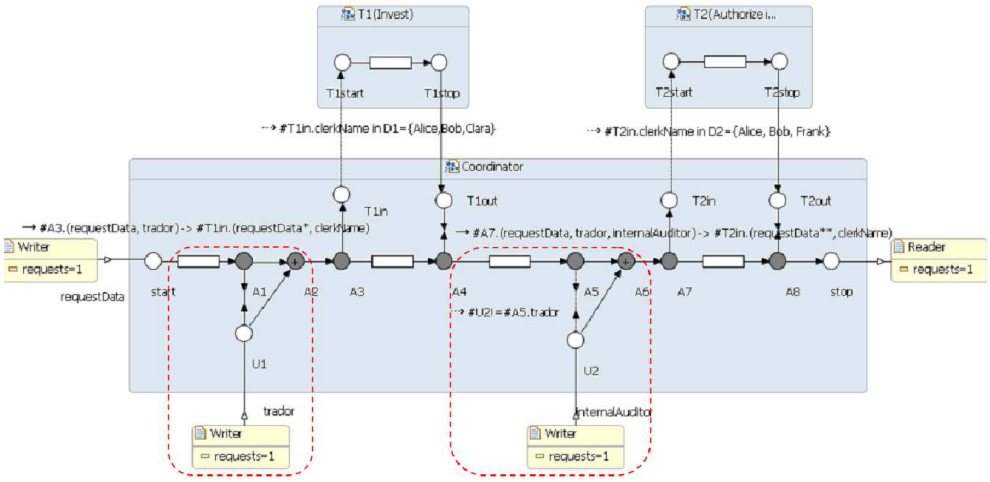


Fig. 8. Modeling a process with segregation of duties in Reo

circuit uses two special join nodes $A2$ and $A6$ which merge the data items from the incoming channels. Transformer channels TRANSFORMER(A2, T1in) and TRANSFORMER(A6, T2in) are employed to transform data objects from the *Coordinator* circuit to the format used in circuits representing operations $T1$ and $T2$.

The compliance of this process model to the dual control principle can be checked using the ASL formula

$$\mathbf{A}[\#T1start.clerkName \neq \#T2.start.clerkName]true,$$

which requires clerk names executing the involved operations to be different.

Suppose now an organization providing a composite service needs to ensure compliance to a privacy policy stating that user personal data can be transferred to a third party only if the user explicitly authorized such a transfer. For example, in the above investment process, the trader may invest on behalf of a bank client who entrusts his/her personal data (name, passport number, organization, address, etc.) to the bank, but does not want them to be shared with other partners/services (e.g., trading organization). On the other hand, some of these data can be vital for involved services, and to complete the process the bank needs to get the client's permission to transfer particular data to particular services. Such permissions can be formalized by means of privacy rules and stored in the following format:

$$r_i = (ruleID,\ clientID,\ dataItem,\ recipientID,\ action,\ permission),$$

where *ruleID* is a rule identifier, *clientID* is a client identifier, *dataItem* is a data item that requires authorization, *recipientID* is a partner (service) to whom the data item will be transferred, *action* is an action on data item performed by the

recipient (e.g., *use*, *retain*, *share*, etc.) *permission* is a boolean value that permits or prohibits the transfer of the specified data item to the specified partner for the particular purpose defined by the action. For example, *Alice* can authorize the transfer of her *passport number* to the *T1 (Investment)* service if it is needed for processing her request (*use*), but prohibit to *retain* this data after her request has been processed.

Although the majority of the publicly available service policies are published in plain natural languages, they usually provide sufficient information about the intended use of personal data. XML-based specifications such as WP-Policy and XARML allow designers to express privacy policies in a more structured manner. Recent approaches to privacy management suggest the transfer from static policies to customizable solutions which allow parties to negotiate the use of personalized data. This assumes a formalization of possible actions performed by each partner on these data. In our case, providers of composite services may store privacy policies of individual services in the form

$$p_j = (\textit{ruleID, serviceID, dataItem, action, purpose, necessity, disclosure}),$$

where *ruleID* is a rule identifier, *serviceID* is a service used in the composition, *dataItem* is a private information concern, *action* is an action on data item performed by the service, *purpose* explains the intended use of this data item, *necessity* indicates whether this data item is vital for a service or optional, while *disclosure* specifies whether it can be shared with other partners.

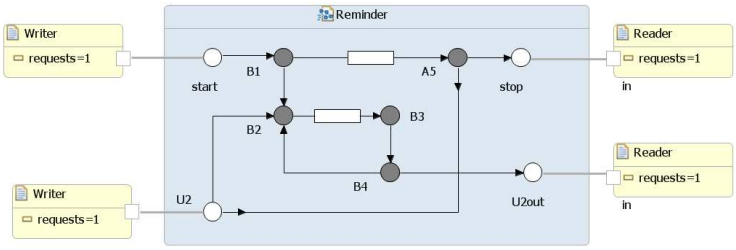
The selection of permitted actions on protected data items regarding the invocation of a particular service by a particular client can be modeled using Reo transformer channels. Assuming that $R : r_i, i = \overline{1,n}$ is a table of client permissions, an SQL request

```
SELECT action FROM R
  WHERE clientID = %currentClientName
    AND dataItem = 'passport number'
    AND recipientID = 'T1'
    AND permission = 'true'
```

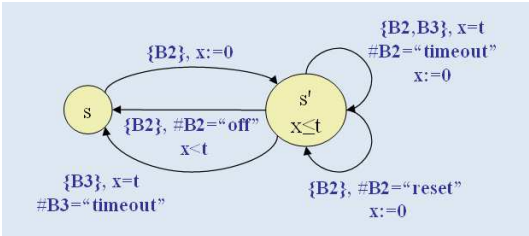
can be realized by a channel TRANSFORMER(A2, T1in) to select a set $\mathcal{P}$ of actions permitted for the service $T1$ over a client's passport number. Here the variable %currentClientName refers to a client name from the current investment request (Alice). Similarly, assuming that $P : p_j, j = \overline{1,m}$ is a table of rules formalizing service privacy policies, a set $\mathcal{R}$ of requested actions can be obtained using the following SQL request

```
SELECT action FROM P
  WHERE serviceID = 'T1'
    AND dataItem = 'passport number'
    AND necessity = 'vital'.
```

A constraint $\mathcal{R} \subseteq \mathcal{P}$ for the FILTER(T1in, T1start) channel will ensure that the data transition through this channel is possible only if the set of requested



(a) Auditor notification



(b) TCA for a t-timer with off and reset options

Fig. 9. Modeling time-aware business processes

actions is included in the set of permitted actions. Checking an appropriate formula over the domain $Act = \mathsf{enum}\ \{$'use', 'retain', 'share',...$\}$ we can automatically establish whether privacy policies match (state $T1start$ is reachable) or mismatch (state $T1start$ is unreachable). Potentially, more complex matching functions, e.g., ones that take into account action implication (e.g., 'retain' implies 'use', etc.) can be implemented.

In the above model of the investment banking process the system waits until a trader and an auditor perform their activities. Using timer channels we can model a system that notifies the trader about pending requests and the auditor about the need to authorize the performed investment operations if they do not execute their activities in a required time period. Figure 9(a) shows an example of a Reo circuit that uses a *t-timer with off- and reset-option* TIMER(B2,B3) to achieve this goal. A TCA for this channel is shown in Figure 9(b). A *t-timer* channel accepts any input data and returns on its sink end a timeout signal after a delay of t time units. In our case, we use a t-timer to measure how long the investment request waits for authorization. The *off-option* allows the timer to be stopped before the expiration of its delay when a special "off" value is consumed through its source end. This option is used to switch off the timer when the authorization message is received from the auditor. The *reset-option* allows the timer to be reset to 0 after it has been activated, when a special "reset" value is consumed through its source end. We reset the timer after sending a notification message to the auditor.

Additionally, timer channels can be exploited to initiate the rollback of an investment activity that was not authorized in a certain time period after auditor notification. Modeling of long-running business transactions with Reo is discussed in [23].

7 Related Work

The problem of formal business process modeling and high-level property specification has received plenty of attention in the research community. Formal structures such as Petri-nets, transition systems, process algebras and logic-based approaches have been widely employed to formalize the semantics of BPMN [5, 24], UML ADs [25] and WS-BPEL [26–28]. A comparative analysis of Petri-nets, transactional logics and temporal logics can be found in [29]. At a first glance, Reo is somewhat reminiscent of Petri-nets. However, Petri-nets normally offer synchronization at each transition of a net, whereas in Reo synchronization is defined by the types of channels connected together. This enables more concise representation of complex workflow patterns. Synchronous drain channels in Reo are convenient for modeling processes where token cleaning is required, while Petri-nets are usually extended with inhibitor and reset arcs for this purpose, which significantly reduces the number of software tools able to analyze such models [30]. Due to the compositional nature of Reo, designers can easily model various sub-processes separately, assemble them for verification and testing purposes and later on deploy and execute coordination code on separate machines without any changes in the system behavior. Process algebras have been used for formal modeling and analysis of business processes. Like other traditional models of concurrency, process algebras offer an action based model of concurrency, where more complex actions (i.e., processes) are composed out of simpler ones. In these models, action is the first class concept making the interaction protocols implicit in the static structures of the compound processes that manifest themselves as the sequences of the matching actions of process pairs only as they unravel their behavior in the temporal domain. In contrast, Reo offers a model of concurrency where interaction constitutes the only first class concept. The distinction between Reo and traditional models of concurrency is analogous to the distinction between constraint programming and imperative programming. Every channel in Reo explicitly represents a primitive interaction, as a binary relation that imposes a constraint on the actions at its ends. More complex interaction protocols are constructed by composing such binary constraints into Reo circuits. Having interaction protocols as explicit constraints makes it easier to associate other properties and constraints, such as QoS or compliance, to them. Moreover, Reo/CA-based models are easier than process algebras, which make them a promising technique for practical applications for designers without a strong formal background.

In the area of SOC, the aforementioned formalisms have been applied for web service compatibility checking [31, 32] and composition verification [33, 34]. Extended CA are suitable for time-, resource- and QoS-aware behavioral com-

patibility analysis as well. An interesting property of CA as a formal model for web service compositions is their ability to deal with interaction transactions. For example, if a user has to provide his *name*, *birth date*, *passport number* and *home address* to a system, it is often not important in what order he/she introduces these data. CA allow us to abstract from such details by modeling the whole interaction as a single transition.

Various types of logic-based languages (First-Order Logic [35, 36], LTL [8], CTL [37, 38], deontic logic [39, 40], temporal deontic assignments [41], concurrent transaction logic [37], etc.) have been applied for high-level process property specification. Existing approaches provide means for formal specification, verification and enforcement of compliance requirements, but their integration appears to be very problematic. Each model is specifically designed to deal with a certain set of requirements representing a single compliance category, e.g., temporal constraints on control flow [42, 43], security requirements [44], privacy policies [45, 32], task based entailment constraints [46], segregation of duties [47, 48], or performance evaluation [49]. The advantage of our approach is that it allows designers to check various types of compliance requirements represented as constraints on transitions in a single CA model. LTL/ASL property specification formats provide powerful means for formalizing and model checking these requirements.

8 Conclusions and Future Work

In this paper, we discussed how Reo/CA can be used for compliance-aware business process modeling and web service composition verification. We aligned various extensions of CA and illustrated their applications using simple but realistic examples. We showed how structural errors in workflow models can be detected, and formalized the problem of process compliance verification to segregation of duties and privacy policies as reachability problems in CA. All steps of business process development, including control/data flow modeling, property specification, process verification and code generation are accomplished with the ECT.

We plan to extend the presented work in several ways. First, additional tools for property specification and verification will be introduced. For example, by integrating appropriate real-time model checking tools with TCA, we can support more powerful time-aware business process analysis. Second, by integrating syntactic/semantic interface matching algorithms with CA bisimulation checking we can enable (semi-)automated service discovery and composition given Reo process models and CA-based specifications of required service operations. Another line of work is related to the generation of graphical Reo circuits from RSL which will provide the basis for efficient process model reconfiguration using RSL-like scripts. Moreover, there is ongoing work on enabling Reo/CA to express priority of certain alternatives and transitions. Such models can be useful for implementing exception and compensation handling in Reo process models.

9 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. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science **14**(3) (2004) 329–366
2. Arbab, F., Chothia, T., Sun, M., 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
3. Arbab, F., Baier, C., Boer, F., Rutten, J.: Models and temporal logical specifications for timed component connectors. Software and Systems Modeling **6**(1) (2007) 59–82
4. Sun, M., 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
5. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. In: Information and Software Technology (IST). Volume 50 of 12., ACM Press (2008) 1281–1294
6. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: Proc. of the Int. Conf. on Business Process Modeling (BPM'08). (2008, to appear)
7. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. In: Proc. of the Int. Conf. on Business Process Modeling (BPM). Volume 4714 of LNCS., Springer (2007) 6479
8. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal **46**(2) (2007) 335–361
9. 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
10. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming **61** (2006) 75–113
11. Arbab, F., Koehler, C., Maraikar, Z., Moon, Y.J., Proenca, J.: Modeling, testing and executing Reo connectors with the Eclipse coordination tools. In: Proc. of the Int. Workshop on Formal Aspects in Component Software, Elsevier (2008)
12. Arbab, F., , Kokash, N., Sun, M.: Towards using Reo for compliance-aware business process modelling. In: Proc. of the Int. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Volume 17 of LNCS., Springer (2008)
13. Arbab, F., Sun, M.: Synthesis of connectors from scenario-based interaction specifications. In: Proc. of the Int. Symposium on Component Based Software Engineering (CBSE'08). Volume 5282 of LNCS. (2008)
14. Tasharofi, S., Vakilian, M., Moghaddam, R.Z., Sirjani, M.: Modeling web service interactions using the coordination language Reo. In: Proc. of the Int. Workshop on Web Services and Formal Methods. Volume 4937 of LNCS., Springer (2008) 108–123

15. Chothia, T., Kleijn, J.: Q-automata: Modelling the resource usage of concurrent components. Electronic Notes in Theoretical Computer Science: Proc. of the Int. Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006) **175**(2) (2007) 79–94
16. Arbab, F., Chothia, T., van der Mei, R., Sun, M., Moon, Y., Verhoef, C.: From coordination to stochastic models of QoS. In: Proc. of the 11th International Conference (COORDINATION). Volume 5521 of LNCS., Springer (2009) 268–287
17. Baier, C., Blechmann, T., Klein, J., Klüppelholz, S.: A uniform framework for modeling and verifying components and connectors. In: Proc. of the 11th International Conference (COORDINATION). Volume 5521 of LNCS., Springer (2009) 268–287
18. Klüppelholz, S., Baier, C.: Symbolic model checking for channel-based component connectors. Electronic Notes in Theoretical Computer Science **175**(2) (2007) 19–37
19. Concortium, C.: Initial specification of compliance language constructs and operators. COMPAS Deliverable (2008)
20. Blechmann, T., Baier, C.: Checking equivalence for Reo networks. In: Proc. of the Int. Workshop on Formal Aspects of Component Software (FACS). (2007)
21. Gligor, V.D., Gavrila, S.I., Ferraiolo, D.: On the formal definition of separation-of-duty policies and their composition. In: Proc. of IEEE Symposium on Research in Security and Privacy. (1998)
22. Schaad, A., Lotz, V., Sohr, K.: A model-checking approach to analysing organisational controls in a loan origination process. In: Proc. of the eleventh ACM symposium on Access Control Models and Technologies (SACMAT). (2006)
23. Kokash, N., Arbab, F.: Applying Reo to service coordination in long-running business transactions. In: Proceedings of the ACM Symposium on Applied Computing (SAC'09), ACM Press (2009) 318–319
24. Wong, P., Gibbons, J.: A process semantics for BPMN. In: Proc. of the Int. Conf. on Formal Engineering Methods. Volume 5256 of LNCS., Springer (2008)
25. Störrle, H., Hausmann, J.H.: Towards a formal semantics of UML 2.0 activities. Software Engineering (2005) 117–128
26. Lucchia, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming **70**(1) (2007) 96–118
27. 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
28. 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
29. Oren, E., Haller, A.: Formal frameworks for workflow modelling. Technical Report 2005-04-07, DERI - Digital Enterprise Research Institute (2005)
30. Raedts, I., Petković, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L.: Transformation of BPMN models for behaviour analysis. In: Proceedings of the International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS). (2007) 126–137
31. Guermouche, N., Perrin, O., Ringeissen, C.: Timed specification for web services compatibility analysis. Electronic Notes in Theoretical Computer Science (ENTCS) **200**(3) (2008) 155–170
32. Mokhtari, K., Benbernou, S., Said, M., Coquery, E., Hacid, M., Leymann, F.: Verification of privacy timed properties in web service protocols. In: Proc. of the Int. Conf. on Services Computing, IEEE Computer Society (2008) 593–594

33. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: Proc. of the Australasian Database Conf. (ADC'03), ACM Press (2003)
34. Yang, Y., Tan, Q., Xiao, Y.: Verifying web services composition based on hierarchical colored Petri nets. In: Proc. of the Int. Workshop on Interoperability of Heterogeneous Information Systems, ACM Press (2005) 47–54
35. Dingwall-Smith, A., Finkelstein, A.: Checking complex compositions of web services against policy constraints. In: Proc. of the Int. Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS). (2007)
36. Halpern, J.Y., Weissman, V.: Using first-order logic to reason about policies. In: Proc. of the Computer Security Foundations Workshop (CSFW). (2003)
37. Mukherjee, S., Davulcu, H., Kifer, M., Senkul, P., Yang, G.: Logic based approaches to workflow modeling and verification. In: Logics for Emerging Applications of Databases. (2003)
38. Koehler, J., Tirenni, G., Kumaran, S.: From business process model to consistent implementation: A case for formal verification methods. In: Proc. of the Int. Enterprise Distributed Object Computing Conf., IEEE Computer Society (2002) 96–107
39. Sadiq, S., Governatori, G., Naimiri, K.: Modeling control objectives for business process compliance. In: Proc. of the Int. Conf. on Business Process Management (BPM'07). Volume 4714 of LNCS., Springer (2007) 149–164
40. Cederquist, J., Corin, R., Dekker, M., Etalle, S., den Hartog, J., Lenzini, G.: Audit-based compliance control. Int. Journal of Information Security **6**(2) (2007) 133–151
41. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: Proc. of the Int. Workshop on Business Process Design (BPD'06). Volume 4103 of LNCS. (2006) 5–14
42. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: Proc. of the Int. Enterprize Distributed Object Computing Conf., IEEE Computer Society (2006) 221232
43. Ghose, A., 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
44. 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
45. Hamadi, R., Benatallah, B., Paik, H.: Conceptual modeling of privacy-aware web service protocols. In: Proc. of the Int. Conf. on Advanced Information Systems Engineering. Volume 4495 of LNCS., Springer (2007) 233–248
46. Wolter, C., Schaad, A., Meinel, C.: Task-based entailment constraints for basic workflow patterns. In: Proc. of the ACM Symposium on Access Control Models and Technologies. LNCS, ACM Press (2008) 51–60
47. Li, N., Wang, Q.: Beyond separation of duty: An algebra for specifying high-level security policies. In: Proc. of the ACM Conf. on Computer and Communications Security, ACM Press (2006) 356–369
48. Knorr, K., Stormer, H.: Modeling and analyzing separation of duties in workflow environments. In: Proc. of the Int. Conf. on Information Security: Trusted Information: the New Decade Challenge. (2001) 199–212
49. Koizumi, S., Koyama, K.: Workload-aware business process simulation with statistical service analysis and timed Petri net. In: Proc. of the Int. Conf. on Web Services (ICWS), IEEE Computer Society (2007) 70–77