An Evidence Model to Enable Constraint-Based Runtime Monitoring in SOA[*]

Ganna Monakova[†], Philip Miseldine[‡], and Frank Leymann[*]

Abstract— **One of the major challenges for businesses today is to ensure that their processes are regulatory compliant. This implies that business processes must be appropriately constrained for their correct and legal operation. To evaluate such constraints, evidence of the implementation of the business process execution is needed. In a SOA, a business process is commonly realised as an orchestration of services. It is therefore necessary to observe the runtime behaviour of these services. In this paper we propose a common evidence model, which allows constraints to be modelled upon service behaviour and mapped to the available evidence. We propose an architecture to provide the runtime monitoring needed to evaluate the constraints based on this model.**

Keywords: SOA monitoring, constraints in SOA, evidence model SOA, signalling model SOA

1 Introduction

Business goals imply actions to be taken or tasks to be accomplished by a company. Compliance regulations exist to ensure a business operates in a legal, standardised manner. Regulations thereby constrain businesses, providing confidence in a business's governance. Business goals are commonly realised as business processes, whose implementation can be automated by IT systems. It follows therefore that regulatory constraints must be related to and enacted by the implemented processes. To assess whether a business runs in a compliant way, monitoring of the performed operations is required. To this end, the business operations must produce evidence suitable for constraint-based monitoring. Thus not only must a business satisfy the constraints necessary for compliance, but it must also produce evidence required for the constraint evaluation [1].

A popular way for a company to model and implement business processes is through the adoption of the Service Oriented Architecture (SOA) paradigm. SOA requires an IT application to be designed in such a way that it can be exposed as a service. These services externalise internal behaviours of the system through a common, standardised interface. Business processes are then modelled to reach the business goal by orchestrating these services. This allows the orchestrated services to change independently from the process specification. As such, a service may be dynamically chosen at runtime as long as it provides the same functionality. This allows external entities to implement part of a business process, thereby allowing the process owner to outsource some process activities. Accordingly, it must be assumed that parties other than the process owner must be included into the modelling process.

Just as business processes orchestrate services together to achieve business goal, a business process choreography aims to define a pre-determined plan that each party in the business collaboration must follow. Choreographies therefore determine the protocol that must be followed between each partner in a business process. This is achieved by way of modelling the messages and their order that must be followed by participating parties. To support regulations governing the interactions between businesses, constraints must be placed on these interactions. Each party may have different regulations and thus may require different constraints to govern the interactions. Therefore evidence that can be used by each party to prove the compliance to their regulations must be provided.

1.1 Paper Outline

Much work exists in providing the verification of a business process specification against a set of constraints. The existing approaches can be divided into two main categories: verification of the business process specification (based on the process model and performed during the design time of the process) and verification of the process implementation (based on evidence collected during the business process execution, and thus only achievable during, or after runtime). We place our work into the second category and discuss the problems and advantages of constraint-based monitoring at the runtime. We motivate the requirements for the runtime monitoring of service behaviour by showing that verification of the business process specification to the design time is not sufficient to evaluate a range of constraints that could be applied to a business process. We provide an example business pro-

[*]The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement no FP7-216917 (project MASTER).

[†]Institute of Architecture of Application Systems, University of Stuttgart, Germany.

[‡]SAP Research, Karlsruhe. Germany.

cess choreography in Section 2 as motivation. Further, we argue that there is currently no technical solution to represent evidence of service behaviour in a common model that could be interpreted by each party in a business process collaboration. Similarly, there is no way to represent a constraint in a way that the evidence required for its evaluation can be derived from the constraint specification. These arguments are detailed in Section 3. We then propose a descriptive model of service behaviour that provides a standardised way of representing what evidence may be provided by a service. This model is described in Section 4. We then use this model as a basis to define constraints, which can be applied to inform each party in the choreography of what evidence must be provided to the other parties. This is shown in Section 5. We then describe how these constraints can be modelled and evaluated in practice, through an architecture described in Section 6. Finally, the paper positions the work in the current state-of-the-art in Section 7, with concluding remarks in Section 8.

2 Motivating Example

Figure 1 shows an example of a business process collaboration that automates the delivery of clinical treatment to a patient by a hospital. There are three parties in the choreography: the *Patient*, the *Hospital*, and the *Insurer*. The activities of each party are encapsulated within a *pool*, a modelling notation provided by BPMN [2]. The collaboration is modelled from the *Hospital*'s point of view, therefore the partner processes on the *Patient* and *Insurer* sides show an external (also called abstract) view of the internal behaviour.

The basic flow of the process is the following: the *Patient* initialises the process by instructing the *Hospital* that they require treatment. The *Hospital* receives this request, locates the Electronic Patient Record (EPR) of the patient, and determines the insurer of the patient. The *Hospital* then requests a cost for the treatment of the *Patient* to the *Insurer*. Note that multiple insurance companies can implement the abstract behaviour modelled for the party *Insurer* in this example. At runtime, the insurance company of the patient who initiated the treatment request is bound to the abstract description. The *Insurer* receives the cost request and calculates a cost for the treatment based on the insurance policy of the *Patient*. The *Hospital* uses this cost as a basis to provide a treatment plan for the patient. The *Patient* receives this treatment plan, including the cost, and confirms that the treatment should take place. For brevity, we do not model a negative confirmation. Upon receiving the confirmation for treatment, the *Hospital* needs confirmation from the *Insurer* indicating that the insurer is willing to pay for the treatment. Again, we do not model a negative confirmation. The *Hospital*, on receipt of this confirmation, treats the *Patient*. After treatment, the *Hospital* requests a payment from the *Insurer*. The *Insurer* initiates a transfer of funds to the *Hospital* to pay for the treatment. The *Hospital* processes this payment, and completes the process.

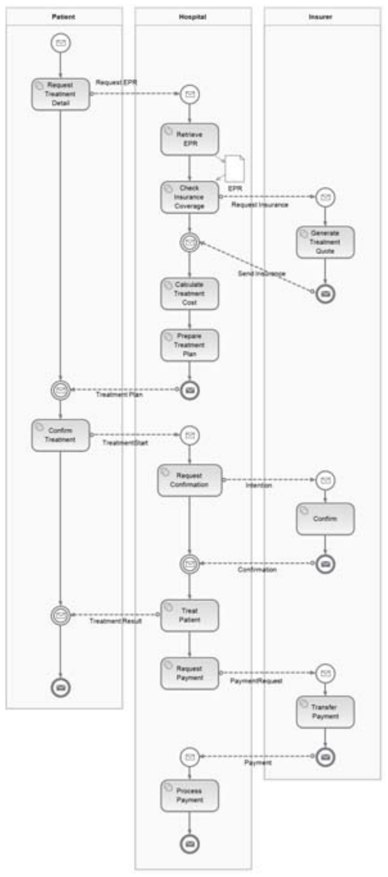

Figure 1: Insurance for Healthcare Business Process Choreography

In classic SOA, the services used to implement these activities will be described operationally, via a WSDL description. This describes the data types consumed and produced by the operations exposed in the service. WSDL however does not provide any form of standardised description for evidence that might attest to the execution of an operation. In the example, such a standardised evidence model is required, as the *Hospital* requires evidence from the *Insurer* that it correctly managed the patient data it sent to it. As multiple *Insurer* parties can be chosen dynamically at runtime, each must follow the standard for the *Hospital* to be able to assess their evidence.

Similarly, the BPMN standard does not allow the *Hospital* or any other party to describe what evidence they require from other parties in the choreography. One could use the modelling notation of a data object to represent an SLA view of such a set of requirements, however fine-grain relations between these requirements and the possible evidence available from another party cannot be modelled in BPMN. Likewise, each party, if dynamically bound at runtime, may have different evidence models based on their own service implementations. Again, neither WSDL nor BPMN would enable this to be modelled, described, or enacted.

3 Problem Analysis

In this section we analyse the implementation layers of a business process collaboration in a SOA and outline how the collaboration depicted in the example may be implemented using SOA principles. Constraints that are required for compliance regulations are described and analysed in relation to the described implementation layers. Finally, requirements for runtime monitoring of the specified constraints using evidence obtained from the collaboration implementation is discussed.

3.1 Business Process Implementation Analysis

Figure 2 shows the four main layers of a SOA implementation of a business collaboration. The top layer contains the specification and implementation of the process choreographies. A choreography describes interactions between two or more parties, whose behaviour is defined in the business services layer. These interactions can be specified using choreography languages like BPEL4Chor [3], Let's Dance [4] and WS-CDL [5]. The actual message exchange can be considered as the choreography implementation, and can be conducted for example over a service bus. While the choreography specifies a multi-party collaboration protocol, a business process orchestration specifies a collaboration of activities designed to accomplish a given business goal. The specification and implementation of the service orchestrations is contained in the business service layer. The orchestrated services themselves are depicted in the application service layer. An application service performs actions on resources which are depicted in the bottom layer.

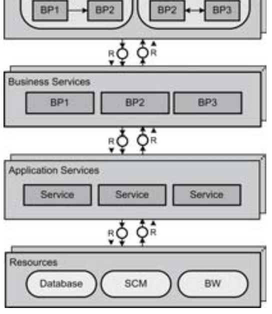

Figure 2: Implementation levels of a SOA

Note that the level of abstraction increases from the bottom to the top: an application service can use multiple resources and perform multiple actions on the resources but it is atomic from the point of view of a business service. Thus, a business service abstracts from teh implementation details of the application service and is only interested in the business value (or effect) it can produce in combination with other application services. Similarly, a choreography abstracts from the business semantic of the business process and is only interested in the external communication of the business processes.

On the other hand, an application service and can be a used in different business processes and a business process can participate in different choreographies. While these principle of loose coupling makes SOA systems very flexible, it also makes them very complex. In the dynamic environment, a static analysis of such a system is not possible, therefore runtime monitoring is required to ensure fulfilment of the constraints put on the system.

To give an example of a business collaboration implementation on these layers, consider the business process outlined in Section 2. The process implementations of the *Patient*, *Hospital* and *Insurer* business processes are placed in the business services layer. The specified interactions between these processes, such as *RequestEPR* and *RequestInsurance* are placed in the business interactions layer. The implementations of the activities modelled by *Hospital*, such as *Calculate Treatment Cost* activity, are placed in the application services layer. Finally, all resources the business processes operate on, such as *PatientData* and *TreatmentPlan*, are placed in the resources layer.

The next section analyses the constraints that can be placed on the different layers, and relates them to the motivating example from Section 2.

3.2 Constraint Analysis

According to the abstraction layers and relationships between the layers depicted in Figure 2, constraints can be classified in the following three types:

Constraint Type 1 *Choreography constraints or constraints over business process interactions.*

Many constraints on the interactions between parties are commonly defined within Service Level Agreements (SLAs) [6]. SLAs provide contracts of expected performance, behaviours, and other criteria that are agreed upon by both parties, the service provider, and the service consumer. For example, the *Hospital* may be viewed as a consumer of the services of the *Insurer*. SLAs are typically defined without a specific relation to the activities defined in any business processes of the parties involved, but rather relate to individual service interactions. A high-level example of such a constraint could be *"Invocation of service ConfirmPayment must give a response within 2 hours"*. A business process choreography describes when the service interactions between the parties involved occur, and thus determines when and at which point of execution the SLA constraints must be applied and assessed.

Another example of a constraint of type 1 related to the motivating example described in Section 2 is to ensure that the activity *Confirm Payment* on the *Insurer* side is always executed before activity *Treat Patient* on the *Hospital* side.

A more elaborate example would be to ensure that if the *Insurer* receives a payment request, then it must be guaranteed that this request was sent by a *Hospital* party and the payment cost was not changed by any third party. In addition, the *Insurer* wants to ensure that the treatment the *Insurer* is supposed to pay for was actually performed by *Hospital*. These constraints enforce data integrity between interacting parties and require analysis of the resource layer for their evaluation. The constraint's logic however is specified on the choreography level and involves analysis of the interactions, therefore these constraints are considered to be of type 1.

Constraint Type 2 *Business process constraints or constraints over an orchestration of application services.*

Constraints of type 2 include temporal dependencies between activities in a business process, for example that activity *Request Payment* must follow activity *Treat Patient*. Such constraints are normally enforced by the process model and can be verified to the design time. This does not negate the need for the monitoring of the actual implementation of the process specification, as unexpected behaviour can occur due to the software and hardware bugs and unforeeseen operational situations.

An example of type 2 constraint is to ensure that the treatment cost calculated in activity *Calculate Treatment Cost* is consistent with the treatment cost sent to the *Insurer* in activity *Request Confirmation* and *Request Payment*. Note that this constraint implies analysis of the resources used in the business process. This constraint is placed into the category 2, because the constraint logic follows from the business integrity required on the business process level.

Constraint Type 3 *Application service constraints or constraints over resource usage.*

Constraints may also be placed upon the use of resources, independent from the application service logic. In the healthcare domain for example, the regulatory act HIPAA demands protection of patient data being disclosed to external parties [7]. This constraint can be applied to the EPR data object in the process, as described in [8]. Its fulfilment however must be ensured by the services that are processing and requesting the EPR, namely the application services.

The next section motivates evidence based monitoring of the specified constraint types and discusses evidence available on each abstraction layer and required for the monitoring of the specified constraints.

3.3 Evidence Analysis

Where a complete process specification is present, many design time techniques can be applied to prove properties of the process, a review of which is provided in Section 7. A process can be considered as an ordered graph with nodes as activities, and flow dependencies as edges [9]. Temporal relations, for example, can be verified between activities where each activity is modelled (i.e. activity *Confirm Payment* must always precede an activity *Treat Patient*) by analysing the business process graph: every path leading to the *Treat Patient* node must pass the *Confirm Payment* node. To verify this happened in practice however, monitoring of the underlying implementation is still required. In the case of the constraint type 1, only an incomplete process specification, and thus an incomplete graph, is available. This is due to the choreography specification over the abstract description of the interacting business processes. This means that any constraint the *Hospital* wants to ensure at design time on the choreography can only be applied to the abstract process specifications exposed by the other parties. A runtime constraint verification on the other hand is based on the evidence provided by the interacting parties and can contain more information than the abstract process specification. For example the *Insurer* may be asked to provide evidence to the *Hospital* if it outsources some activities (e.g. to ensure that some critical activities will always be performed by *Insurer*), but *Insurer* does not have to specify the logic behind this step. As such, evidence generated at runtime that indicates a request sent by *Insurer* to any third party is required for evaluation of this constraint.

For constraints that check the data integrity outlined in the previous sections, the values of the data to check can only be known at run time as a value is assigned during the business process execution by the activities that generate the data values. Thus it is not possible to verify this constraint the modelling level. Other examples of constraints based on runtime data include those relating to separation of duty (ensuring that the same person did not execute two or more tasks). Such constraints are demanded by Sarbanes-Oxley and the ISO 27002 standard [10]. To evaluate such a constraint, evidence of who executed an activity must be observed from the underlying implementation of the process. Such evidence could be obtained from the Business Process Management System, which can act as a service itself, providing runtime evidence of the process execution state. For instance, in the SAP NetWeaver®Business Process Management Solution [11], activities involving human interaction are assigned to users via a service-based worklist component. This can be monitored to capture when an activity was assigned to a user, and to whom the activity was assigned.

Where a constraint implies evaluation of a resource or data, as in type 3, we assume its evaluation can only occur at runtime as this requires constraining the application service using the resource or data itself. This means that there must be a description of which services interact with the resource or data so that this evidence can be obtained. For example, to verify the behaviours of the *Insurer*, such as their use of a patient record, monitoring of the actual resource usage of the resource *PatientData* is required at runtime. The application services muat provide appropriate evidence to indicate their usage.

We now consider how the source of the evidence can affect the scope and evaluation of a constraint. The different layers shown in Figure 2 can provide evidence on different abstraction levels. For instance if a business process is a BPEL implementation, then the events can be emitted by the BPEL engine when an activity changes its state, to show that the activity was for example completed or terminated. If a single activity represents a complex process on an application service level, in some cases there is a need to go down on the abstraction levels to obtain more fine grained evidence. To demonstrate the relation and difference of the evidence on different abstraction levels, consider the following example. Interactions between business processes, as well as invocations of the application services, normally happen via a service bus. This means that evidence about a service invocation can be obtained from three sources: from a business service that invokes a service identifying the intended invocation, from a service bus which routes the invocation request and eventually performs service discovery and dynamic binding, and from the invoked service identifying the execution of teh invoked operation. Although all of these events relate to the occurrence of the same actions performed by the invoked service, they have different semantics: events emitted by the business process engine indicates the intention to perform an action, the event emitted by service bus can indicate the selection of the action implementation (or lack of a suitable implementation), the delivery of the request or the request timeout, and the application service can emit an event when the actual execution of the service operation has started, completed or faulted. By monitoring these three levels of events relating to the service invocation, certain constraints such as a service response time specified in a SLA can be verified. This allows different values to be obtained for the response time, depending on the chosen abstraction level. If the events on the business process layer indicating the start and end times of an invocation are correlated to calculate the service response time, then it will differ from the response time calculated from the events on the application service layer. This is due to the additional time taken for service discovery, message routing and queue waiting time. As a consequence, the expected SLA might be violated from the business process view while it is fulfiled from the application service point of view.

To be able to capture evidence available on all abstraction layers, a common evidence model is required. This model must be suitable for modelling constraints on all layers of abstraction. The next section shows why the action state change related evidence is sufficient to monitor all types of discussed constraints and presents the evidence model.

4 Evidence Model

We now formalise our understanding of a SOA system so that a model attesting to its operation can be formalated. In general, a system in a SOA can be described as follows. Let Σ denote a SOA system, then $\Sigma = \mathscr{S} \times \mathscr{R} \times \mathscr{I}$, where $\mathscr{R} = \{R_1, ..., R_l\}$ denotes the set of all resources, $\mathscr{S} = \{S_1, ..., S_n\}$ denotes the set of all services, and $\mathscr{I} = \{I_1, ...I_m\}$ denotes the set of all interactions between services (or service choreographies).

Every service S_i performs a set of actions $\{A_{i_1}, ..., A_{i_k}\}$. At any point of time every action is located in a specific state denoting the action progress. Let Γ denote the set of all action states. The state function $\sigma_{\mathscr{A}} : \mathscr{A} \times \mathscr{T} \to \Gamma$ returns the state of an action at specific time point, where $\mathscr{T}$ denotes time. The state of the service S_i at time t is defined by the states of its actions: $\sigma_{\mathscr{S}} : \mathscr{S} \times \mathscr{T} \to 2^{\Gamma}$, or $\sigma_{\mathscr{S}}(S_i, t) = (\sigma_{\mathscr{A}}(A_{i_1}, t), ..., \sigma_{\mathscr{A}}(A_{i_k}, t))$

The state of the system is defined by the current state of the system services, resources and interactions. As the previous section discusses, every resource can be abstracted through an application service, which captures all accesses to this resource. This means that any state change of this resource implies invocation of an action of the application service. Therefore any resource state change can be related to the state change of the corresponding action of the application service. Thus, the resource states of the system can be derived from monitoring application service action state changes. For example, if an action *update* on resource *data* changes its state to *completed*, then it can be derived that resource *data* is in state *updated*.

All service interactions happen through the service bus. Thus, monitoring of the state changes of the service bus actions provides enough information to derive the current state of service interactions. A similar approach for monitoring predefined choreographies in the service bus was described in [12].

Thus, the complete state of a SOA system can be derived from the state changes of the actions on the application service level, business service level, and the interaction service level. Therefore the evidence model presented in this section considers the evidence that can show state changes of the service actions and provide information about the current state of the resources the action operates on, which allows monitoring for the resource state changes.

Figure 3 shows a graphical representation of the evidence model based on the action state changes.

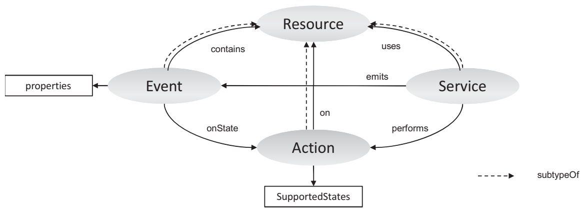

Figure 3: Evidence Model

The evidence model contains the following concepts:

- *Service* - represents the described service.
- *Action* - captures the actions service performs. A set of all actions builds action taxonomy. The sub- and super-class relationships between actions can be used to enhance modelling of the observational requirements. For example, if an evidence on execution of an action of type *AccessData* is required, and it is known that the actions *ReadData, UpdateData, DeleteData* are sub-classes of the action *AccessData*, then the evidence requirement can be propagated to all sub-class actions. Every action can specify a set of supported states as attributes. Every action can have it own set of supported states, we assume however that a superset of all possible states exists. This means that an action specific state set must always be a subset of the superset. An example of such a superset can be based on the BPEL activity state diagram [13] and can include states *Started, Running, Faulted, Repaired, Suspended, Terminated, Completed* and *Compensated*
- The *Event* - describes events a service can emit which are related to a certain action state change. The relation *onState* between *Event* and *Action* concepts is an abstract relation. It can be refined with the *onStarted, onRunning, onFaulted, onRepaired, onSuspended, onTerminated, onCompleted* and *onCompensated* relations, depending on the states the corresponding action supports. Events can have properties, for example event timestamp. As a payload, events can contain information about resources the action operates on. This provides contextual information at the current execution state.
- *Resource* - Describes the resources which can be used by a service and on which the actions are performed. In an abstract way, everything can be considered as a resource: an action can be executed on a service, action or an event. Therefore a resource can be viewed as a super concept. Resources can have relations to other resources, which are captured in an ontology. A specific type of the resource ontology is an action taxonomy described above.

5 Application of Model

The previous section detailed the design of the evidence model. This section shows how this model can be used to describe the service behaviour in terms of performed actions and the available events. Following this, we outline how the presented model can be used for specifying constraints, and the mapping of these specified constraints to the available evidence for their evaluation. Finally, an architectural example is presented which supports the implementation of the approach proposed.

5.1 Service Description

Figure 4 shows a description of the activity *Transfer Payment* on the *Insurer* side. This activity is represented with action *Pay* and is performed on the resource *Payment*, which includes another resource *SumToPay*. Action *Pay* supports states *Started* and *Completed* and can emit events E_1 and E_2 on these states. Events can include information about the *Payment* as the payload and contain *Timestamp* of their generation as a property.

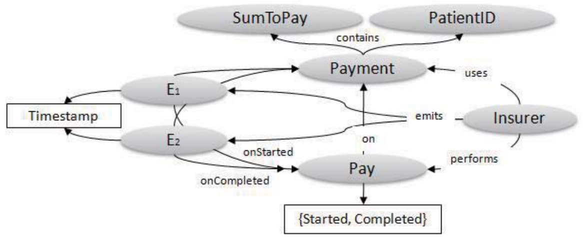

Figure 4: *Insurer* description of the *Transfer Payment* activity

Figure 5 shows an example of the service description on the *Hospital* side which is responsible for patient treatment. Note that even if a treatment itself is a complex process which consists of human activities, the *Hospital* service only provides an abstraction of this process captured in the action *Treat*. In practice every part of patient treatment is normally logged into an IT system, which means that an IT abstraction of the patient treatment can be represented by this system. The action *Treat* is performed on a *Patient* according to the *TreatmentPlan*. The *Hospital* can inform other parties when the treatment has been Started (event E_3), Terminated (event E_4) and Completed (event E_5). All events can contain information about the *TreatmentPlan* the action *Treat* follows and a *Timestamp* of the event generation.

Figure 6 shows the description of the *Request Payment* activity on the *Hospital* side. The *Request Payment* activity is represented by *Invoke* action that is performed on the action *Pay*. Note that in this case action *Pay* is a resource for action *Invoke*. Action *Invoke* sends *TreatmentPlan* to the partner, which contains the *PatientID* and

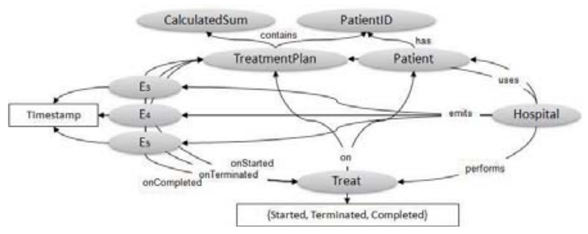

Figure 5: *Hospital* description of the patient treatment

CalculatedSum. Events E_6, E_7 and E_8 containing *TretamentPlan* information and the *Timestamp*of their generation can be emitted on action states *Started, Faulted* and *Completed*.

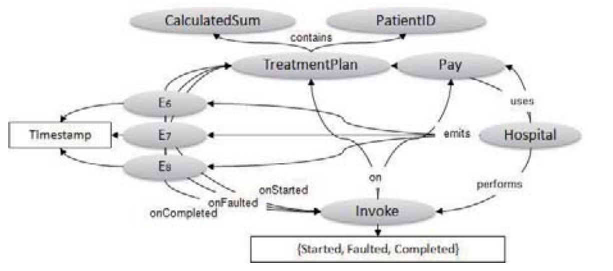

Figure 6: *Hospital* description of the *Request Payment* activity

5.2 Constraint Specification

This section shows how constraints can be specified using the concepts of the evidence model and relations between them. We consider the following constraints as an example:

1. Before *Insurer* pays for the treatment, the treatment must be completed by *Hospital*.
2. If *Insurer* receives a payment request, then it was sent by a *Hospital* party and the payment cost was not changed by any third party.

The first constraint describes a temporal dependency between actions *Treat* and *Pay*. This kind of dependency is often expressed using Linear Temporal Logic (LTL) [14]. The relations between concepts in the evidence model can be used as the predicates in the constraint specification. Using concepts from the evidence model as the domain vocabulary, relations between these concepts as predicates over these domains, and LTL operators to connect these predicates, the first constraint can be specified as follows:

$$\begin{aligned} \Box(Performs(Insurer, Pay) \rightarrow \\ Performs(Hospital, Treat) \wedge \\ \neg \mathit{Started}(Pay)\mathbf{U}\mathit{Completed}(Treat))) \end{aligned}$$

Each party in a choreography can have different constraints on the overall behaviour. For example, the above constraint is of interest for the *Insurer* side. The *Hospital* might provide their own constraint, as the previous constraint is also satisfied when hospital treats the patient and does not receive the payment:

$$\begin{aligned} &\Box(Performs(Hospital, Treat) \wedge Started(Treat) \rightarrow \\ &\qquad \Diamond Completed(Insurer, Pay) \end{aligned}$$

The second constraint is slightly more complicated as it involves comparison of the resource states. Thus, to be able to monitor this constraint, the events must contain information about the resource states. The second constraint can be modelled as follows:

$$\begin{aligned} \forall insurer \in Insurer, \forall pay \in Pay, \forall payment \in Payment : \\ Performs(insurer, pay) \wedge On(pay, payment) \rightarrow \\ (\neg(\mathit{Started}(pay)\mathbf{U}(\exists hospital \in Hospital, \\ \exists invoke \in Invoke : Performs(hospital, invoke) \wedge \\ On(invoke, pay) \wedge Started(Invoke) \\ \wedge \exists plan \in TreatmentPlan : On(invoke, plan) \\ \wedge \exists S_1 \in TreatmentSum : Contains(payment, S_1) \\ \wedge \exists ID_1 \in PatientID : Contains(payment, ID_1) \\ \wedge \exists ID_2 \in PatientID : Contains(plan, ID_2) \\ \wedge \exists S_2 \in SumToPay : Contains(plan, S_2) \\ \wedge Equal(S_1, S_2) \wedge Equal(ID_1, ID_2))) \end{aligned}$$

The above formula states that if an insurer performs an action pay, then there must be a hospital, which invoked this action. Furthermore, the values which were sent by the hospital must be the same as the ones the action pay on the insurer side is performed on.

Whilst constraints and their representation can differ depending on whatever formalism is used, the objects it refers to within the predicate must be understood by all parties. This is so that evidence across the choreography regarding the operation of each party can be collected and provided for all parties wishing to evaluate their constraints. As an example, consider the *Hospital* as the party wishing to enforce the constraint modelled above. It needs to be aware of what states the *Insurer* can provide for observation that a payment was performed, i.e. an event on completion of action *Pay* containing information about *Payment*, so that it may evaluate its constraint. In turn, the *Insurer* needs to be aware that it must provide the evidence required for the *Hospital*.

In this context, the service model described in Section 4 forms a common vocabulary for constraint modelling across the choreography. By defining the objects that may be targeted by constraint predicates, each party may construct predicates that require the events described by each service for its evaluation. For example, given the descriptions of the *Insurer* and *Hospital* actions from the previous section, the above constraints can be mapped to the events as follows:

$$\neg E_1 \mathbf{U} E_5$$

Taking into account the delivery time of the events this constraint can be refined into the following:

$$E_5 \rightarrow (E_1 \wedge E_5.timestamp > E_1.timestamp)$$

Note that this constraint must be applied to every process instance, the correlation information for the events is not shown in this example. For the second constraint the following mapping can be derived:

$$\begin{aligned} \neg E_1 \mathbf{U} E_8 \\ \wedge E_1.Payment.SumToPay = \\ E_8.TreatmentPlan.CalculatedSum \\ \wedge E_1.Payment.PatientID = \\ E_8.TreatmentPlan.PatientID \end{aligned}$$

6 Architectural Support

We now detail how the described observation and evaluation could take place in practice. In Figure 7, a simplified architecture of a system is given that evaluates rules using evidence gathered from events described in the proposed model. The basic design of the system is described, with a description of the information flow.

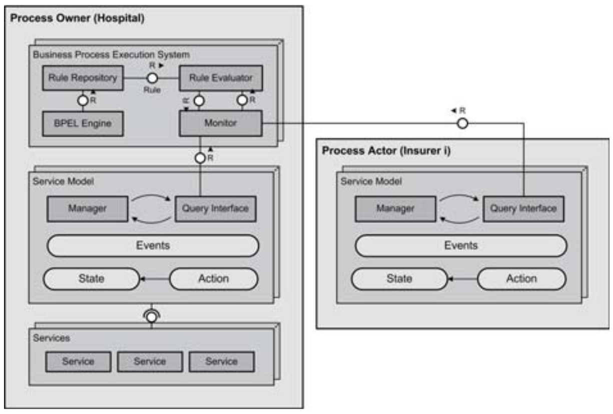

Figure 7: Monitoring Architecture

The architectural schematic describes two entities: the Process Owner (in our examples, the *Hospital*), and a third-party in a Business Process Choreography (for example, an *Insurer*). As the process owner is unaware of the actual implementation of the services of the third-party, the architecture reflects that only a service evidence model provided by the third-party is exposed. We assume additional components are required for the service evidence model to be managed, and exposed for query. Such components would provide interfaces to relate a service endpoint or WSDL operation to an evidence model, as well as provide a routing of the events produced by the underlying service to a designated endpoint.

A business process execution component (here the *BPEL Engine*) instantiates a business process, and queries the *Rule Repository* to return predicates that govern the current execution state of the process. These predicates refer to the events described by the evidence models of each party in the choreography. The *BPEL Engine* can perform this task as it is aware of which services are being invoked through the process, and which operations are to be invoked. This information is forwarded to the *Repository* to build a list of rules that reference the events that can be produced by the corresponding actions performed by these services. The actions are determined based on a query to each party's service evidence model manager. The matching rules are relayed to a *Rule Evaluator* who instructs a *Monitoring* component to inform it when events needed to evaluate the rules are observed. The *Rule Evaluator* instructs each service evidence model manager to inform it when the events needed occur.

During the business process execution, the service operations are invoked by the *BPEL engine* to provide the activities described in the process. Upon an invocation of an operation it provides, a service emits an event as described in its service evidence model. This includes the properties that are represented in the event payload. The local service evidence model manager relays this event to the *Monitoring* component that subscribed to this event. This could be achieved in multiple ways: the use of a service bus for example, allows invocations and messages sent by and to services to be monitored. The *Monitor* forwards this to the *Rule Evaluator*. The event payload (detailed information regarding the event) is also sent so that operational state leading to the event can be captured and represented in the rules too. The *Rule Evaluator*, upon receiving this event and the others it must observe to evaluate the rule, then evaluates the predicate. This can also lead to another event being emitted by the Rule Evaluator, that others can subscribe to, for example, to assess how often positive evaluation of a predicate was achieved.

7 Related Work

We now situate the work presented within the current state-of-the-art. We consider exisiting approaches that provide design-time and run-time verification of constraints over business processes, and general approaches that aim to provide constraint modelling and management.

The area of formally verifying properties of a business process against a set of constraints is rich and mature. Extensive work has been conducted in proving structural properties of a business process control flow using model-checking techniques. These rely on formalising the workflow in different representations, for example in a Petrinet [15]. A review of the different formalisations is given in [16]. Extensions of existing formalisms can be used to verify additional constraints, for example logical relationships between variables in a process, as shown in [17]. Constraints can be modelled in relation to these formalisations. For example Promela can be used to model business processes, and the process requirements are specified in temporal logic (LTL). SPIN [18], another model checking tool, can be used to verify LTL specifications in Promela models [19]. These techniques can be used to verify, for example, structural dependencies [20], and entailment constraints [21] (constraints where the execution of one activity is constrained by the execution of another activity) as shown in [8].

Similarly to the control flow, data flow has been recognised as a fundamental aspect of workflow specification [22]. Techniques such as described in [23] can be used to model data dependencies in a workflow specification, and verify these against a formalisation. All these techniques however lack the ability to verify the implementation of the business process.

In [24] the author propose to model monitoring rules based on the concept model of the domain and using an extension of a temporal logic. However they do not propose a common evidence model, which does not allow the use of this approach across different organisations.

Another monitoring approach proposes monitoring of the choreography in the message bus [12, 25, 26]. This allows comparison of the actual message exchange with the choreography specification. In contrast to our approach, the approach described in these works only allows to monitor the external messages and cannot handle the events related to the internal behaviour of the partner processes.

8 Conclusions and Outlook

This work presented an approach for constraint-based monitoring of the services behaviour in a SOA. We analysed implementation layers of a business process and gave a constraint classification based on the layer this constraint is applied to. We then proposed an evidence model that allows the description of the service behaviour to be defined in terms of the actions it performs, the interactions these actions have with other resources in the system, and the evidence the service can produce upon the different execution states of the action. We showed how constraints could be specified using the concepts defined in this model to enable the mapping of the specified constraints to the evidence provided by the service.

As part of the ongoing work we investigate how this evidence model can be used as part of the agreement between services, similar to an SLA. In this case, the service should also be able to describe the constraints it satisfies. The required constraints can then be matched with the provided constraints as part of the agreement, and then monitored on the provided evidence.

As future work, we plan to extend the model so that it can capture information about the trustworthiness of the provided description and evidence. For instance, if a service claims to perform actions on financial data, we need mechanisms to ensure that this description is correct. Such mechanisms could include providing security assertions or certification from trusted authorities that support the claims made in the description.

References

[1] A. A. Arens and J. K. Loebbecke, *Auditing, an integrated approach / Alvin A. Arens, James K. Loebbecke*, 2nd ed. Prentice-Hall, Englewood Cliffs, N.J. :, 1980.

[2] OMG, "Business process modelling notation (BPMN) specification version 1.2."

[3] G. Decker *et al.*, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *IEEE International Conference on Web Services*. IEEE Computer Society, 2007.

[4] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Let's Dance: A Language for Service Behavior Modeling," in *CoopIS 2006: Proceedings 14th International Conference on Cooperative Information Systems*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 4275. Springer, 2006, pp. 145–162.

[5] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon, *Web Services Choreography Description Language Version 1.0*, Nov. 2005. [Online]. Available: http://www.w3.org/TR/ws-cdl-10

[6] A. D. Heiko, H. Ludwig, and G. Pacifici, "Web services differentiation with service level agreements," 2003.

[7] J. B. E. David Baumer and F. C. Payton, "Privacy of medical records: It implications of hipaa," NCSU Dept. of Computer Science, Tech. Rep.

[8] C. Wolter, A. Schaad, P. Miseldine, and C. Meinel, "Model-driven business process security requirement specification," *Journal of System Architecture, Elsevier B.V.*, April 2008.

[9] O. Gerb, R. K. Keller, and G. W. Mineau, "Conceptual Structures: Theory, Tools and Applications," *Conceptual Graphs for Representing Business Processes in Corporate Memories*, 1998.

[10] "ISO/IEC 27002:2005 - Information technology – Security techniques – Code of practice for information security management," Tech. Rep.

[11] SAP, "The Road Ahead for Business Process Management," *SAP SDN*, 2008.

[12] O. Kopp, T. van Lessen, and J. Nitzsche, "The Need for a Choreography-aware Service Bus," in *YR-SOC 2008*. Online, June 2008, Workshop Paper, pp. 28–34.

[13] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann, "BPEL Event Model," University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, Technical Report Computer Science 2006/10, November 2006.

[14] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*. New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[15] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting BPEL processes," in *Business Process Management*, ser. Lecture Notes in Computer Science, vol. 4102. Springer Berlin / Heidelberg, 2006, pp. 17–32.

[16] F. v. Breugel and M. Koshkina, "Models and Verification of BPEL," 2006.

[17] G. Monakova, O. Kopp, F. Leymann, S. Moser, and K. Schäfers, "Verifying Business Rules Using an SMT Solver for BPEL Processes," in *Proceedings of the Business Process and Services Computing Conference: BPSC'09*, ser. Lecture Notes in Informatics. Gesellschaft für Informatik e.V. (GI), März 2009, Konferenz-Beitrag.

[18] G. J. Holzmann, "The model checker spin," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.

[19] M. d. M. Gallardo, P. Merino, and E. Pimentel, "A generalized semantics of promela for abstract model checking," *Form. Asp. Comput.*, vol. 16, no. 3, pp. 166–193, 2004.

[20] C. Pajault, J. F. Pradat-Peyre, and P. Rousseau, "Adapting petri nets reductions to promela specifications," in *FORTE '08: Proceedings of the 28th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 84–98.

[21] J. Crampton, "A reference monitor for workflow systems with constrained task execution," in *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2005, pp. 38–47.

[22] F. Leymann and D. Roller, *Production workflow: concepts and techniques*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[23] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger, "Data flow and validation in workflow modelling," in *ADC '04: Proceedings of the 15th Australasian database conference*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004, pp. 207–214.

[24] C. Giblin, A. Y. Liu, S. Müller, B. Pfitzmann, and X. Zhou, "Regulations expressed as logical models (realm)," in *JURIX*, ser. Frontiers in Artificial Intelligence and Applications, M.-F. Moens and P. Spyns, Eds., vol. 134. IOS Press, 2005, pp. 37–48.

[25] L. ke Fredlund, "Implementing WS-CDL," in *Proceedings of JSWEB 2006 (II Jornadas Cientfico-Tcnicas en Servicios Web)*, 2006.

[26] Z. Kang, H. Wang, and P. C. Hung, "WS-CDL+ for web service collaboration," *Information Systems Frontiers*, vol. 9, no. 4, pp. 375–389, 2007.