MC-Cube: Mastering Customizable Compliance in the Cloud

Tobias Anstett, Dimka Karastoyanova, Frank Leymann, Ralph Mietzner, Ganna Monakova, Daniel Schleicher, and Steve Strauch

Institute of Architecture of Application Systems, University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Abstract. Outsourcing parts of a company's processes becomes more and more important in a globalized, distributed economy. While architectural styles and technologies such as service-oriented architecture and Web services facilitate the distribution of business process over several departments, enterprises and countries, these business processes still need to comply with various regulations. These regulations can be company regulations, national, or international regulations. When outsourcing IT-functions, enterprises must ensure that the overall regulations are met. Therefore they need evidence from their outsourcing partners that supports the proof of compliance to regulations. Furthermore it must be possible to enforce the adherence to compliance rules at partners. In this paper we introduce so-called compliance interfaces that can be used by customers to subscribe to evidence at a provider and to enforce regulations at a provider. We introduce a general compliance architecture that allows compliance to be monitored and enforced at services deployed in any emerging cloud delivery model.

1 Introduction and Motivation

Service-oriented architecture has emerged as the architectural style that allows to recursively compose services that are run in a distributed fashion on heterogeneous infrastructures. Service-oriented systems are often used in conjunction with business process execution engines to build cross-organizational IT-support for the business processes in and across enterprises.

With upcoming service delivery models such as infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) enterprises can outsource computing and middleware resources to the cloud and use them "on demand". This allows enterprises to focus on their core competencies that may not lie in the acquisition and management of an IT infrastructure. As a consequence from financial and other scandals in the last years, companies are faced with more and more regulations that they need to obey to. These regulations range from internal regulations, such as business ethics or sustainability rules to external regulations, such as privacy laws that need to be obeyed or frameworks such as BASEL II [3] or SOX [20] that regulate financial transactions.

Many of these regulations mandate enterprises to provide enough evidence to auditors so that those auditors can judge whether regulations have been obeyed

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 592–606, 2009.

or violated. In the case of outsourced IT services, the gathering of evidence can be partially delegated to the outsourcing provider. However, some regulations still hold an enterprise (or even the CEO of an enterprise) liable even if an outsourcing provider violated a regulation. Furthermore as complex business processes can be partially supported by IT systems running at different providers and in the own data-center, the evidence must later be aggregated to provide a comprehensive view on the whole business process.

In this paper we deal with cross-organizational business processes that use services provided at multiple outsourcing providers in multiple delivery models. In such a setting, compliance to regulations is of utmost importance and has implications on the IT-infrastructure of both providers and consumers of services. Thus there is a need to monitor and correct the execution of business processes after reaching a service-level agreement. Therefore we introduce the notion of a *compliance interface* that allows enterprises to gather evidence from providers as well as enforce rules on these providers. We show that providers must allow their clients to customize the evidence they provide depending on the regulations the client has to be compliant with. The approach presented in this paper is very flexible and does not focus on a certain legal framework. However, specific focus is paid on outsourcing parts of applications, which must maintain the overall compliance rules imposed on the whole application.

We introduce a running example in Section 2 that will be used while presenting the main contributions of the paper. We then describe the requirements and architecture for a general compliance framework that we gathered from the case studies in various projects (Section 3). We apply this framework to different delivery models in Section 4. In Section 5 we then discuss how the compliance interfaces could be realized. We show a prototypical implementation in Section 6, compare our approach to other approaches in Section 7 and finish with a conclusion and an outlook to future work that we plan in the field.

2 Running Example

In [2] we investigated how security and trust issues affect the execution of WS-BPEL [16] processes in the cloud and discussed requirements on the middleware supporting the execution of WS-BPEL processes in the different cloud delivery models. In contrast, in this paper we investigate how compliance of business processes can be ensured during outsourcing.

Therefore we introduce the example of a fictional EU-based company manufacturing and shipping drugs named Pharmazon. Figure 1 illustrates a simplified view on the business process that Pharmazon follows when selling drugs.

Of course the company Pharmazon and its business processes have to comply to several European laws and national laws for example concerning production, distribution, advertising and labeling of medical products for human and veterinary use [6]. For example, Pharmazon must be compliant to directive 2001/83/EC [21] of the European Parliament and Council, which deals among other things with the advertising of medicinal products for humans. Their

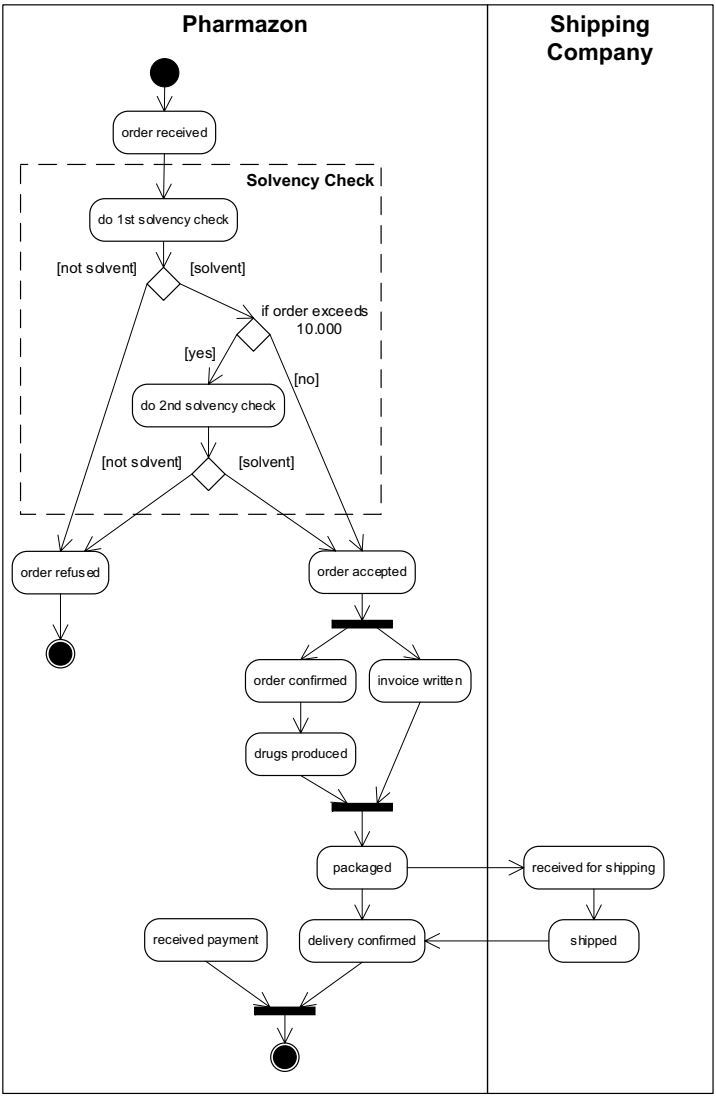

Fig. 1. Simplified view on Pharmazon's business process

business processes also have to be compliant to their own internal regulations. Examples for such internal rules are: *'Every time an order is received, a solvency check has to be done. If the amount exceeds €10.000 a second solvency check has to be done by a person different from the first one'* or *'Every order has to be delivered within 24 hours'.*

Pharmazon is forced to maintain lower costs to stay competitive. The first step in reducing cost is to reduce the enormous shipping costs to non European countries by subcontracting pharmaceutical companies in the US and Asia

Pacific countries. Furthermore parts of the business process such as the above mentioned solvency check have to be outsourced.

While outsourcing parts of the business (process) it must be ensured that the outsourcee still complies with the European, national or internal regulations that formerly were ensured by the company itself. Thus Pharmazon has to have the possibility to check, whether the outsourcee complies to the given regulations imposed on Pharmazon's processes.

In this paper we will use the example with focus on the internal regulations mentioned above, ensuring that the outsourced solvency check is made as defined and the cross-cutting business concern of orders delivered within 24 hours is achieved.

3 Requirements and Resulting Architecture

As mentioned in Section 1 it is required that a specified set of compliance constraints is ensured during a business process execution.

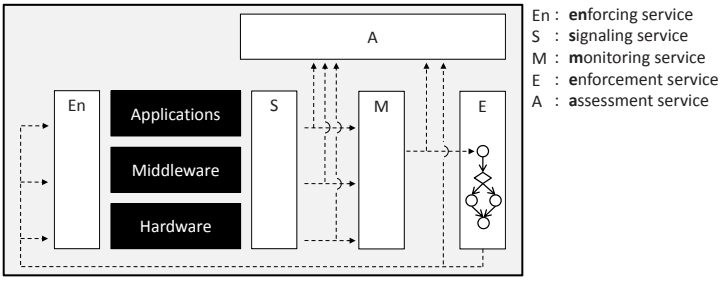

Fig. 2. Abstract Compliance Supporting Architecture / Infrastructure

Figure 2 presents an abstract architecture using compliance services to enable the control and assessment of compliance. Here the term *En* is a shortcut for **en**forcing service, *S* for **s**ignaling service, *M* for **m**onitoring service, *E* for **e**nforcement service and *A* for **a**ssessment service. The architecture as well as terms used in this paper are based on the research work [13] in the European Community's FP7 project MASTER [1].

In the following each compliance service is described in detail:

Signaling Service: Provides evidence in form of events emitted on action state changes. An example of an action state can be state *ready*, denoting that an action is ready to be executed, state *running*, denoting that an action is currently executing or a state *completed*, denoting that an action was executed. Signaling services can be implemented at any level of abstraction where events must be

[1] http://www.master-fp7.eu

emitted, e.g. at application, platform middleware or hardware level. Logically they are however related to a certain business process behavior as evidence to prove the satisfaction of constraints.

Runtime Monitoring Service: Aggregates events based on its situation information, payload or any other data, which can be resolved using an external service (such as a database entry lookup or a Web service call). Thus, runtime monitoring services can also be viewed as complex event processing, for short CEP [14], services. Depending on the aggregation rules, events may either provide evidence for a compliant execution of the process or the detection of a constraint violation.

Enforcement Service: In order to prevent a violation the enforcement service reacts to an event denoting a possible threat of violation and guides (controls) the system in such a way that the invalid state (the state, which violates the constraint) cannot be reached. In case of a detected violation the goal of the reactive process is to transfer the system from the current invalid state into a valid one. This can be done through the compensation of the actions, which caused the invalid state transfer. If such compensation is not possible, other actions can be taken to minimize the impact of the violation on the business value. In both cases, an enforcement process is an orchestration of services, which are able to influence the state of the system. Note that the actual execution of the services can only be influenced if corresponding enforcement capabilities are provided by the corresponding middleware. For example a process can be terminated from the outside only if the engine provides an interface supporting a *terminate* operation. A component providing such enforcement capabilities is in the following called *enforcing service*.

Assessment Service: Assess the satisfaction of constraints as well as the effectiveness of the implemented enforcement process.

Because compliance concerns may vary between different business processes an infrastructure realizing the presented architecture has to allow dynamic configuration of its services. Compliance policies allow to describe the configuration of the compliance services required to monitor and enforce a certain compliance constraint. The configuration can be divided into the following blocks: i) description of the signals (events) required to monitor the behavior of the system, ii) description of the monitoring rule (predicate), which allows detection of the violation or a violation threat, iii) description of the reactive and preventive actions and iv) description of the assessment function.

When translating compliance regulations to compliance policies, for short doing compliance regulations refinement, the responsible person has to take decisions about the granularity of events, where they occur and where and how they can be aggregated to express the required semantics. Depending on the capabilities of the underlying execution environment these events can be emitted and aggregated at different levels. For example an event denoting the read access to a specific database resource could be emitted by the database using its trigger

mechanism, by a service that provides an interface to the database functionality or by the business process that initiated the access. Furthermore the required *signaling* and *runtime monitoring* capabilities can be part of the business process itself, implemented as internal controls, or provided as *compliance services*. Reactions to certain events, e.g. to enforce the satisfaction of constraints, may be realized by single operation calls or require more complex *reactive* processes. In Section 5 we present a concrete example using the running example of Section 2.

4 Refining the General Architecture to Cloud Delivery Models

One of the biggest challenges of mastering compliance in the cloud is that the customer is not able to transfer its obligations regarding compliance regulations to the outsourcing provider. This section discusses the requirements as well as responsibilities of customer and provider in outsourcing scenarios. We describe the requirements for the following three categories of delivery models, namely infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

4.1 IaaS

In this delivery model a customer basically rents the required hardware from an IaaS provider. Like in the traditional on-premise model, he has to take care for configuring the platform and application on his own. Amazon Elastic Compute Cloud (Amazon EC2)[2] is a prominent example of an infrastructure as a service. The configuration of the platform includes the installation of *operating system*, *platform middleware* such as database management system (DBMS), enterprise service bus (ESB) or a BPEL engine, and *application*. In the following compliance services are considered as a specialized platform middleware. Although only responsible for providing the hardware the provider may also provide signaling, monitoring and enforcement capabilities for its hardware. A provider may offer these compliance services as agreed in the service agreements with the customer. He may also require compliance services for ensuring his own compliance. E.g. he has to check that his hardware works as expected and furthermore is not abused to run illegal software like file sharing servers.

Using IaaS a customer will always trust his own installation and therefore does not have to worry about the validity of the events generated by this software.

4.2 PaaS

The PaaS model offers both, the *infrastructure* (hardware) as well as the *platform middleware* to deploy *applications*. The customer neither has to take care for

[2] http://aws.amazon.com/ec2/

reserving hardware resources nor for configuring the platform. Google's App Engine[3] is an example of platform as a service.

In PaaS the customer has to specify his functional and non-functional requirements to the provider and the provided infrastructure. While the functional requirements might for example only specify the need for a certain type of middleware, the non-functional part includes the specification of the compliance requirements on the provided middleware or even hardware. For example the customer might specify that the BPEL engine, which is part of the platform, must be able to send events about the actual state of process instances. Furthermore the engine must allow to enforce certain actions on process instances and must therefore provide a specific enforcement interface. If the customer is for example interested in using monitoring or enforcement services, he has to provide its configuration using the compliance policies. The provider is in charge of installing and executing these policies on his middleware in a similar way he would have to do it when deploying a BPEL process on a BPEL engine. Thus policies can be considered at the level as BPEL processes, which realize applications. This allows PaaS providers to offer monitoring, enforcement and assessment services in a SaaS delivery model. Because signaling and enforcing services are bound to specific platform middleware they can not be outsourced independently of that platform middleware.

In contrast to the IaaS model, the provider may not be able to offer all the required information or services to the customer or even may decide to offer only a limited subset of information he could generate for use by its customers. Thus the customer is constrained to the offered granularity and semantics of the provided events as well as monitoring and enforcement capabilities of the provider.

One of the main deficiencies of using PaaS in this context is the perceived lack of trust, that events provided to the customer are authentic. Because everything except the business process model and its explicitly generated events are hosted, the customer must trust his provider. There are several ways to increase the trust level. On the one hand the trust level could be increased by applying more complex monitoring rules, which execute additional checks on the middleware or even hardware event level. On the other hand there is need for a compliance certification agency to increase trust and allow chains of trust similar to Verisign's[4] role and functionality in the Internet. Certification agencies might base their certificates on audits of the assured compliance, refined compliance policies and the implementation of compliance services. This also leads to the need for trusted middleware.

4.3 SaaS

Software as a service provides different customers the functionality of an application that is completely hosted in the cloud. The user does not have to worry about the required infrastructure or setting up and configuring the platform.

[3] http://appengine.google.com
[4] http://www.verisign.com

When recalling the presented compliance architecture, which is mainly based on the presence of signaling services to provide evidence, the concept of variability becomes very important not only for SaaS but also for the other delivery models. The set of supported events must be made available to the user at each level of abstraction. These abstraction levels should support but not be limited to the categorization used throughout this paper, namely *hardware*, *platform middleware* and *application*, but also introduce more convenient perspectives such as *resources*, *actions* and *states*. Based on a the provided compliance capabilities the customer should be able to define the compliance policies.

5 Technical Architecture

In this section we discuss how the signaling and enforcement capabilities can be described. For this purpose we extend the model described in [5], which is based on relations between *actions*, *resources*, *services* and *events*. As motivated in Section 1 and discussed in Section 3 the customer requires evidence of the behavior of the actual executed business process. This behavior is defined by the set of actions being executed and their ordering relations. Thus the behavior can be represented by and reconstructed from event traces [1] [23] [22]. An event basically represents a specific execution state of the process or an action within the process and contains additional payload information such as resources.

The description of the signaling capabilities contains the following information:

- actions the service performs
- states an action supports, e.g. [7] describes the states BPEL activities support
- resources the service uses
- events emitted on action state change
- resource/information an event may contain
- event properties such as event timestamps

The description of the enforcement capabilities contains the following information:

- enforcement actions the service performs. The enforcement actions are specific actions, which can be used to influence the service execution.
- resources on which the enforcement action is performed
- end point reference (EPR) of the enforcement action to enable action invocation

Figure 3 illustrates the model for describing signaling and enforcement capabilities. The relation *onState* is an abstract relation, which can be subtyped with the relations *onStarted, onRunning, onFaulted, onTerminated, onCompleted*, depending on the states the corresponding action supports. Note that because an enforcement action is a subtype of the general action concept, events also can be emitted on the state changes of the enforcement action. Note also that because

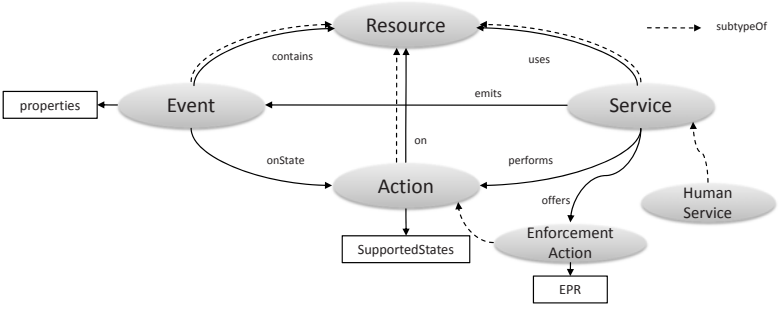

Fig. 3. Signaling and Enforcement Model

action is a subtype of a resource, an enforcement action can be executed on another action, e.g. an enforcement action *block* can be executed *on* action *check*. A signaling policy is a serialization of the events of a concrete instance of this model. An enforcement policy is a serialization of all actions of type *EnforcementAction*. If both customer and provider specify their requirements/capabilities using the same model with domain specific actions and resources, two signaling as well as enforcement policies can be matched using policy matching algorithms (for example that of WS-Policy [25]) to determine if the service provider provides sufficient evidence and actions for external control.

As described in Section 2 Pharmazon decided to outsource the solvency check part of its business process. To be compliant with internal regulations, Pharmazon wants to ensure that the *check* action is performed twice in case the order exceeds €10.000 and that these check operations are performed by different persons. Thus, Pharmazon requires events every time a *check* action was executed, denoting the completion of this action, on action state *completed*, containing information about the person who executed this action. Figure 4 shows an example of a solvency check process signaling and enforcement description offered by an SaaS provider.

Note, that this model does not describe the structure of the solvency check process. The provider can in addition describe the structure of the process using existing standards, such as abstract BPEL [16].

In general there are two options to bind to a service: i) dynamic binding and ii) static binding. Dynamic binding as defined in [25] is based on the operations publish, find and bind. A service requestor finds a service by providing its requirements to the discovery facility, which is responsible for matching service descriptions as well as policies. If a service matching the required capabilities is found, the service requestor binds itself to the service to use it. Because the customer also has to subscribe to the events he described within his signaling policy the bind step has to include the subscription to the events using the compliance interface offered by the provider. A compliance interface has to provide the following operations, which could be implicitly contained in the service description (WSDL [25]) or offered as a standalone service:

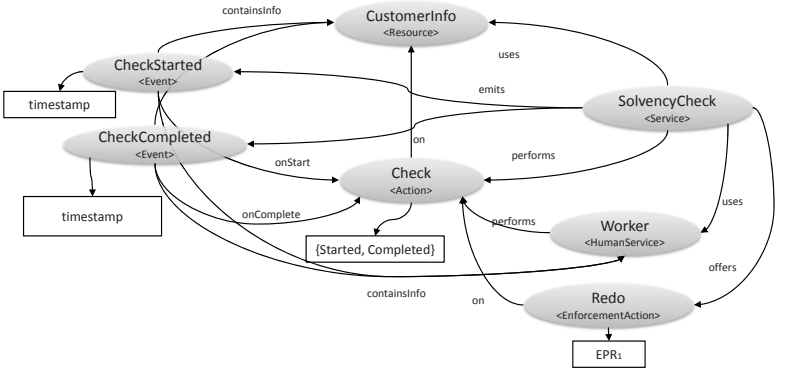

Fig. 4. Signaling capabilities of the service provider

- getSignalingCapabilities(service) returns the signaling policy for the specified service
- getEnforcementCapabilities(service) returns the enforcement policy for the specified service
- subscribeTo(service, event) subscribes to a specific event of the service

Static binding assumes that a suitable service was already found and only the subscription to signaling events has to be done manually. For example a SaaS provider could offer a graphical user interface to describe an abstract view on the business process including signaling and enforcement capabilities available for this process. The customer can use the offered tooling to select the events he wants to subscribe to and the provider automatically performs the subscription in the background. Especially in SaaS delivery models this approach might be interesting because providers already allow customers to customize the provided business processes using variability descriptors [15].

However, which of the discussed approaches is used, the events, a customer has subscribed to, have to be monitored to provide evidence for a compliant execution of the process or the detection of a constraint violation. In Pharmazon's case it has to be ensured that every time an order exceeds €10.000 two check operations are performed by different persons.

Using the provided description of the signaling capabilities, the monitoring rule for the seperation of duties (SoD) objective can be specified as follows:

$$\forall r \in SolvencyCheck\ \exists e_1, e_2 \in CheckCompleted:$$

$$(e_1.pid = r.pid) \wedge (e_2.pid = r.pid) \wedge (e_1 \neq e_2) \wedge (e_1.Worker \neq e_2.Worker)$$

where *pid* denotes the id of the current process instance run. This rule states that at least two different events of type *CheckCompleted* should be available for every run of the *SolvencyCheck* service. This implies that the action *Check* must be performed at least twice for every process run. In addition, the above

condition requires the two actions being executed by different workers. In our case this rule must only be fulfilled if the order amount exceeds €10.000. Because the order sum is not directly available from the event payload of the provided events, this information has to be extracted on the customer's side before the solvency check service is invoked. Assuming that an additional event e_3 of type *SolvencyCheckInvoked* containing order sum and the process ID is sent to the monitoring component by the Pharmazon process before the solvency check is invoked, the above rule can be rewritten as follows:

$$\forall e_3 \in SolvencyCheckInvoked(e_3.sum > 10.000) \rightarrow \exists e_1, e_2 \in CheckCompleted:$$

$$e_1.pid = e_3.pid \wedge e_2.pid = e_3.pid \wedge (e_1 \neq e_2) \wedge (e_1.Worker \neq e_2.Worker)$$

If the separation of duties criteria was not met by the provider, Pharmazon can for example enforce the compliance of its process by invoking a reactive process, which enforces a *redo* on one of the *check* activities.

If the outsourcing provider always executes the check operation twice and the check operations are performed by different persons, then the monitoring rule is not violated. Because checking things twice may take significant longer than not doing so, Pharmazon's business process has to wait longer for the reply of solvency checks made for orders less than €10.000. Because Pharmazon has to be also compliant to its internal regulation *Every order has to be delivered within 24 hours* this might not be applicable. Thus another monitoring policy is needed to express the relation between the order volume, time available and checks to be performed. This policy could trigger an enforcement process, which either skips the activity waiting for the result of the solvency check if the order is less than €10.000, and the first check evaluated to true, or changes the shipping partner or method to a faster one depending on the time left.

6 Implementation Aspects

In this section we show a prototype, which fulfills the requirements of Section 3 and can be used to implement the described architecture.

We extended the Apache ODE BPEL engine[5] with the capability of sending events to the outside. Apache ODE supports BPEL 2.0 process models.

Figure 5 shows where the BPEL engine is placed in a compliance supporting architecture. It also shows the signaling service (S) and the enforcing service (En), which are part of the engine. The enforcing service provides operations to for example influence the running processes on the engine.

During the execution of a process the BPEL engine produces many events and stores them in an internal database called the *audit trail*. These events can be used to check compliance concerns. In order to fulfill the requirement of emitting events to the outside, the engine has been extended with two Web services. The first one is a publish-subscribe Web service, which provides operations to

[5] http://ode.apache.org

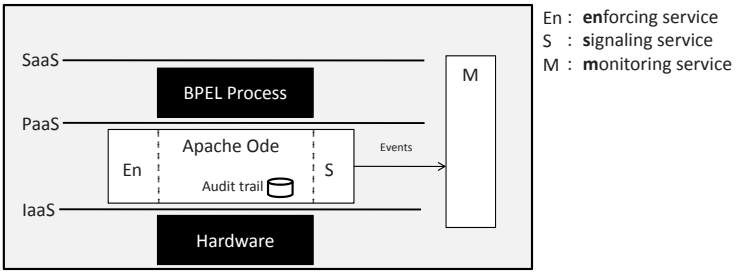

Fig. 5. Placement of BPEL engine in a compliance supporting architecture

subscribe to certain events occurring during process execution. For technical details of the subscription mechanism we refer to [24]. The second Web service is a signaling Web service. It sends the events as SOAP messages to the subscribers. These events are based on the common base event model proposed by IBM[6]. This model for example comprises information of the service, which sent a certain event and in what situation it was sent.

In [7] a static event model for BPEL is proposed. In this context static means, that every activity has a static set of events and one could subscribe to all events at any time. There is no way of constraining the events, which are visible outside of the BPEL engine. This document also contains definitions of all events, which can be emitted during execution of a BPEL activity. This is useful for subscriptions to certain events of a particular activity.

To process the events emitted by a BPEL engine a monitoring service (M) could for example subscribe to certain events to check the validity of the running processes in the BPEL engine. If a violation has occurred a new event can be issued to a so called *enforcement service*. The enforcement service then could carry out reactive actions in order to respond to compliance violations. Actions, which react to a compliance violation, can for example stop the business process running in the domain of the business partner. Such actions could also be modeled in a business process. This process is then called a *reactive process*.

The life cycle management operations, ODE provides, can be used as enforcement capabilities. ODE is capable of pausing, resuming, terminating, and deleting of process instances. So this prototype can be used as an enforcement component specified in Section 3. The engine is also capable of blocking a process instance when a certain event has occurred. This is in some cases useful when a decision has to be made before the process could continue.

For example, if someone has placed an order to buy drugs from Pharmazon exceeding the price of € 100.000 the process will be blocked and an event will be emitted notifying the enforcement service. The engine then can be unblocked by the enforcement by invoking a reactive process, which then takes the necessary actions to unblock the process.

[6] http://www.ibm.com/developerworks/library/specification/ws-cbe/

7 Related Work

Outsourcing is a technique used in process re-engineering in order to optimize or improve business processes in terms of optimization according to different criteria. Additionally it can be used as a mechanism for adapting business processes. The outsourcing of applications can be implemented using different techniques, which depend on the paradigms and technology used for implementing these applications. One of the approaches used to enable outsourcing of processes is process splitting. In the field of Web services compositions approaches have been created for outsourcing service compositions in order to optimize the processes they implement with respect to organizational resources or infrastructure performance. E.g. the approach in [11] enables splitting of BPEL processes into so-called partitions, which can be run as stand-alone processes on different BPEL engines at different locations/organizations in such a manner that the overall semantics of the original business process is maintained. The work in [4] reports on another approach for splitting service compositions in an optimal manner according to criteria like execution time, response time, cost etc. The approach allows for splitting a composition in the so-called strata and is based on the concept of stratified transactions, while the communication among strata is enabled via a queuing infrastructure (MOM in general). This approach views a service composition as a transaction and all the tasks of a composition - as nested transactions. The resulting stratified compositions maintain the original logic but improve its performance. The work introduces several algorithms for optimal stratification of service compositions. Multiple coordination protocols for partitioned/split processes exist and they are dependent on the approach used for the splitting. Worth mentioning are the WS-BA [17], which is a part of the WS protocol stack; for the above mentioned approaches there are corresponding coordination protocols based on either WS-Coordination Infrastructure [12] or on other coordination mechanisms. To enable the communication among the parts of a global process that run at different locations and hence be able to coordinate these partitions/fragments each service composition engine needs to provide information about events related to the life-cycle of process instances. Usually, an engine implements an event model, which is used to publish information for the purposes of monitoring; this has been used for enabling monitoring [9], adaptation [15] [8] [9] and a framework for coordination of service compositions [10]. These are all examples of the use of the events published by the engine, based on an event model. The existence of such an model is crucial also for enabling the outsourcing of parts of service-based applications and enabling their compliance. Compliance to a global process logic in the area of process splitting has been enabled by design in the approaches presented in [11] and [4]. Compliance of processes to a process model has been enabled by the work of [18] for the case of adaptation by means of model evolution. The difference with respect to the subject of compliance between this approach and the one we present here is that our approach focuses on ensuring compliance of process instances, whereas the approach of [19] enables the compliance or correctness of adaptation/modifications on the process model level only.

8 Conclusion and Future Work

In this paper we presented MC-Cube, an approach to deal with compliance requirements in cross-organizational applications build upon a service-oriented architecture. We introduced compliance interfaces as a means to allow subscribers of services to customize the evidence they need from a provider. On the other hand when enforcing compliance at an oursourcing provider the enforcement part of the compliance interface can be used. We introduced a general architecture for outsourcing and compliance and mapped this infrastructure to different delivery models such as IaaS, PaaS and SaaS. We described a prototype that shows how the presented concepts can be applied to a BPEL engine that can then be used at providers to offer customizable compliance to their customers. In future work we will extend this work to describe how suitable services that offer the required compliance can be automatically discovered. We will also investigate in detail how changes to compliance requirements will affect running applications and how this affects the underlying middleware (such as BPEL engines).

Acknowledgments

The work published in this article has partially received funding from the European Community's 7th Framework Programme Information Society Technologies Objective under the COMPAS project[7] contract no. FP7-215175, the MASTER project[8] contract no. FP7-216917 and under the Network of Excellence S-Cube[9] contract no. FP7-215483.

References

1. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, p. 469. Springer, Heidelberg (1998)
2. Anstett, T., Leymann, F., Mietzner, R., Strauch, S.: Towards BPEL in the Cloud: Exploiting Different Delivery Models for the Execution of Business Processes. In: IWCS 2009 (2009)
3. Basel Committee on Banking Supervision. International Convergence of Capital Measurement and Capital Standards (2006)
4. Danylevych, O., Karastoyanova, D., Leymann, F.: Optimal Stratification of Transactions. In: ICWS 2009 (2009)
5. Flegel, U., Kerschbaum, F., Miseldine, P., Monakova, G., Wacker, R., Leymann, F.: Insider Threats in Cybersecurity - And Beyond. Springer, Heidelberg (to appear, 2009)
6. Gordon, J.W., Appelbe, E.: Dale and Appelbe's pharmacy law and ethics. Pharmaceutical Press (2005)

[7] http://www.compas-ict.eu
[8] http://www.master-fp7.eu
[9] http://www.s-cube-network.eu

7. Karastoyanova, D., Khalaf, R., Schroth, R., Paluszek, M., Leymann, F.: BPEL Event Model. Technical Report Computer Science 2006/10
8. Karastoyanova, D., Leymann, F.: BPEL'n'Aspects: Adapting Service Orchestration Logic. In: ICWS 2009 (2009)
9. Karastoyanova, D., Leymann, F., Nitzsche, J., Wetzstein, B., Wutke, D.: Parameterized BPEL Processes: Concepts and Implementation. In: IWCS 2009 (2009)
10. Khalaf, R., Karastoyanova, D., Leymann, F.: Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. In: WESOA 2007 (2007)
11. Khalaf, R., Leymann, F.: A Role-based Decomposition of Business Processes using BPEL. In: ICWS 2006 (2006)
12. Khalaf, R., Leymann, F.: Coordination Protocols for Split BPEL Loops and Scopes. Technical Report Computer Science 2007/01
13. Lotz, V., Pigout, E., Fischer, P.M., Kossmann, D., Massacci, F., Pretschner, A.: Towards Systematic Achievement of Compliance in Service-Oriented Architectures: The MASTER Approach. Wirtschaftsinformatik (2008)
14. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman, Amsterdam (2002)
15. Mietzner, R., Leymann, F.: Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In: IEEE SCC (2008)
16. OASIS. Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)
17. OASIS. Web Services Business Activity (WS-BusinessActivity) Version 1.2 – OASIS Standard (2009)
18. Reichert, M., Dadam, P.: ADEPT flex - Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Systems (1998)
19. Reichert, M.U., Rinderle, S.B.: On Design Principles for Realizing Adaptive Service Flows with BPEL. In: EMISA 2006 (2006)
20. Sarbanes, P., Oxley, M.: Sarbanes-Oxley Act of 2002. The Public Company Accounting Reform and Investor Protection Act. Washington DC: US Congress (2002)
21. The European Parliament and the Council of the European Union. Directive 2001/83/EC of the European Parliament and the Council. Official Journal of the European Communities 311 (2001)
22. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. (2003)
23. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering (2004)
24. van Lessen, T., Leymann, F., Mietzner, R., Nitzsche, J., Schleicher, D.: A Management Framework for WS-BPEL. In: ECOWS 2008 (2008)
25. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR, Englewood Cliffs (2005)