Combining Enforcement Strategies in Service Oriented Architectures

Gabriela Gheorghe[1], Bruno Crispo[1],
Daniel Schleicher[2], Tobias Anstett[2], Frank Leymann[2],
Ralph Mietzner[2], and Ganna Monakova[2]

[1] University of Trento, Italy
First.Last@disi.unitn.it
[2] University of Stuttgart, Germany
Last@iaas.uni-stuttgart.de

Abstract. Business regulations on enterprise applications cover both infrastructure and orchestration levels of the Service-Oriented Architecture(SOA) environment. Thus, for a correct and efficient enforcement of such requirements, full integration among different enforcement middleware is necessary. Based on previous work [1], we make a comparison between enforcement capabilities at business and infrastructure levels. Our contribution is to make a first step towards a policy enforcement model that combines the strengths of the orchestration level enforcement mechanisms with those of the message bus. The advantage of such a model is (1) that infrastructure and orchestration requirements are enforced by the most appropriate mechanisms, and (2) the ability to enforce regulations that would be otherwise impossible to enforce by a single mechanism. We present the architecture and a first prototype of such a model to show its feasibility.

Keywords: policy enforcement, SOA, BPEL, ESB.

1 Introduction

There is an increasing number of regulations that all enterprise applications have to comply with. These regulations usually concretise in policies on data protection, system behaviour and resource or organisational management (e.g., EU Directives[1], Basel3 and Sarbanes-Oxley[14]). Such constraints crosscut enterprise applications.

Achieving compliance with such regulations can be done through policy enforcement. From our point of view, this enforcement comes in two flavours: message or infrastructure level enforcement mechanisms and orchestration level enforcement mechanisms. Mechanisms at the message level focus on technical details like communication protocols (e.g., SOAP, REST), message transformation (e.g., XSLT) or choice of message routes. They are typically embedded within an Enterprise Service Bus (ESB). Regulatory enforcement at the ESB level cannot normally go higher than controlling message flows between physical endpoints or binding virtual endpoints to actual endpoints. Concepts

[1] European Commision EC-95/46 http://ec.europa.eu/justice_home/fsj/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 288–302, 2010.

at higher levels address issues related to orchestration or choreography of activities. At this level business processes and business process engines are used. Examples on this level are based on the Business Process Execution Language (BPEL).

The gap between these two enforcement capabilities at the two levels is illustrated by a real use case: the Hong Kong Red Cross[2] implements its blood donation process in a SOA. In order to stay compliant with existing laws and regulations, the Red Cross needs to enforce the following policy: mutual exclusiveness of doctors approving blood donation and distribution. The first policy is a policy most easily enforced at the message level because location information is only managed at this level. The second is a Separation of Duty (SoD) requirement saying that the same actor in an application is not allowed to perform two conflicting operations. This can be enacted on a BPEL engine by inhibiting one operation if the other operation has been previously performed by the same actor. While the BPEL engine can disallow a process to handle the request to perform an action, the problem is that the process has *already* received the request. If this request contains sensitive information, the BPEL engine cannot prevent it from arriving on the machine where it will be inhibited, and this may cause a data privacy breach over which the BPEL process has no control.

It follows that enforcing policies of message level at the orchestration level or vice versa is awkward for a number of reasons. First, it requires pulling data and associations that are not available to one layer or the other. Second, the policy specification is more complicated and hence prone to errors. Third, there is a tight dependency between the policy languages at each level, and the limits of their enforcement. Also, for management reasons it is advisable to have all policies in one language, and in one place, since this makes deployment and review much easier. We follow this idea and suggest a centralised enforcement model that discovers enforcement capabilities advertised at both process engine and message level. The model decides where to enforce which policy constraints, and how to map these constraints to the existing capabilities at the right level. In our Red Cross example, presented in the next section, we envision the same SoD constraint enforced correctly by combining the BPEL-level check with ESB's capability to resolve the second operation to an endpoint that is surely operated by a different actor. Hence, our approach extends typical ESB and BPEL engine features, and combines rather than separates the resulting enforcement strengths.

The remainder of the paper is structured as follows. Giving a motivating example, the gaps between BPEL engine and ESB enforcement as non-integrated components are described in Section 2. Section 3 presents extensions of the ESB and BPEL engine to address the enforcement limitations and also a general enforcement model. Section 4 shows how to join the two different sets of capabilities. Implementation considerations are presented in Section 5. Problems with current approaches are given in Section 6, and Section 7 concludes.

2 Motivating Example

In this section we use a real example to show the drawbacks of performing enforcement by focusing on either BPEL engines or ESBs. In many countries the name *Red*

[2] `http://www.redcross.org.hk/en/home.html`

Cross encompasses those organisations dedicated to emergency situations and health care. The Hong Kong Red Cross (HKRC) organisation manages, among other things, blood donations for a number of public hospitals [12]. The organisation acts as an intermediary between blood donors, hospitals, and patients (blood receivers). Both the HKRC and the hospitals have to comply with regulations defined by government health agencies. These regulations relate to the blood treatment and donor data:

- The same doctor cannot approve both a blood donation and the distribution of that blood sample to a public hospital. (**Regulation 1**)
- The temperature of the blood samples must always be below zero degrees Celsius. (**Regulation 2**)
- The Red Cross must distribute the donated blood within 2 days after the blood is collected in an HKRC branch. (**Regulation 3**)

Complying with these regulations means understanding how to detect and react to violations. For example *Regulation 1* requires a differentiation between users of the Hong Kong Red Cross so that a doctor approving a blood donation cannot approve the blood distribution afterwards. *Regulation 2* requires sensors detecting temperature changes to a component responsible for monitoring temperatures and triggering reactions. *Regulation 3* requires that blood can only be collected from an HKRC branch and forbids blood distribution after more than 2 days after the collection.

Regulations 1-3 above also concern internal business processes of the HKRC, such as the *blood donation management process*. Figure 1 illustrates the interface between the donors and the blood donation management process. We use this process in a slightly changed form in the remainder of this work. The process is modeled as a WS-BPEL process that orchestrates the Web services connected by an ESB.

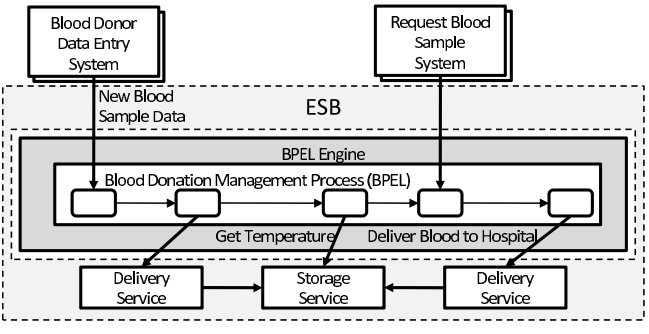

Fig. 1. The Red Cross blood sample management process

The blood donation management process is modeled with respect to regulation 2. For every blood donation a process instance is created; this instance continuously polls the temperature of the corresponding blood storage system until the blood sample is requested or regulation 2 was violated. The violation is detected by the evaluation of a

condition (similar to if-else structures) defined on the returned temperature. Despite being the only implemented solution of enforcing SoD constraints, this BPEL-only implementation falls short in several respects: (1) tagging and sending the blood for cooling are logical invocations whose execution the BPEL engine cannot control down to the message level; (2) the BPEL process does not distinguish between service invocations emitted or served by the same host; (3) the BPEL process does not manipulate service location information. These three aspects are covered by ESB capabilities, and to our knowledge there is no approach that tries to link ESB with BPEL engine features for the purpose of fine-grained policy enforcement.

In the following section we discuss in more detail the limitations of performing policy enforcement only at the level of business processes, or only at the ESB level. We show how Regulations 1-3 can be realised with our combined approach.

3 SOA Enforcement: Current Approaches versus Our Approach

In this section we show architectural approaches to do enforcement and present the enforcement capabilities of current BPEL engines and ESBs. We also show a policy model describing how to use these enforcement capabilities in a distributed and automated environment. In what follows, we use the term *SOA enforcement* to refer to the operation of the service-oriented mechanisms whose job is to ensure that requirements of an enterprise policy (on security or performance) are satisfied. We also use the concept of an *enforcement life cycle* as a three-step mechanism. Given a policy to be enacted, the first step is *to detect* the events relevant to the policy; the second step is *to decide* whether the detected events constitute a violation of the policy. The last step is *to react* as specified by the policy. Drawing this line between the enforcement steps helps to localise the shortcomings of current approaches.

3.1 Different Architectural Approaches

Let us consider the separation of duty regulation (Regulation 1) from above: *The same doctor cannot approve both a blood donation and the distribution of that blood sample to a public hospital*. Enforcing this constraint requires to detect (1) on which doctor's behalf the donation approval service operates, (2) on which doctor's behalf the blood distribution service operates, and (3) the identifier of the blood sample. Figure 2 shows different approaches to enforce such a policy. An initial approach is to weave enforcement features into the donation approval service and the blood distribution service (Figure 2.A). The enforcement mechanism here couples all three stages of the enforcement life cycle (detect, decide, react). Changing the reaction or detection criteria implies that the donation approval service or blood distribution service code must be changed. An improved approach is wrapping: bundling the donation approval service and the blood distribution service with enforcement stubs to make any policy change independent from the service logic (Figure 2.B). This approach does not scale with the policy growth and burdens the two service nodes with heavy processing because the wrapper is on the same node as the business logic. Figure 2.C shows a further improvement: *enforcement as a service*. External enforcement services make enforcement reusable for

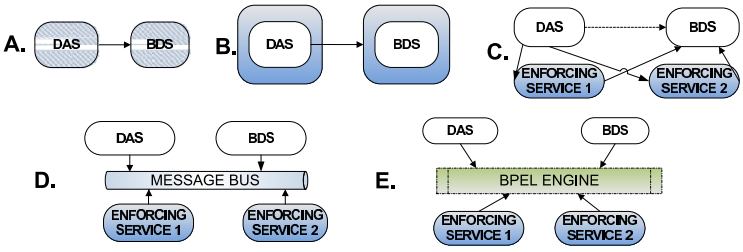

Fig. 2. Different architectures for SOA enforcement. BDS stands for Blood Distribution Service. DAS stands for Donation Approval Service.

other services, but are bypassable and not very scalable. If the blood distribution service or the donation approval service is contacted on a channel that cannot be reached by the enforcement service it cannot be detected that the same doctor performs the same operation for both services. Worse, for every new constraint on the communication between the two services there will have to be a new enforcement service to enforce it.

A further logical step is to employ the message bus (ESB) as a communication channel to and from all services in an enterprise system (Figure 2.D). The ESB solves the bypassability and overloading issues mentioned before. Since it mediates all communication between the donation approval service and the blood distribution service, all relevant messages will be unnoticed before they are dispatched to their destination. All enforcement mechanisms for any policy will have to be deployed to the bus, and the ESB will ensure that messages are dispatched to the right destinations. The ESB falls short in handling correlated enforcement actions that need access to the semantics of the messages: the ESB by itself cannot easily detect that the same doctor performs the same operation on the same blood sample. This correlation problem is solved by the BPEL engine (Figure 2.E). In case there are several constraints on the communication related to the donation approval service and the blood distribution service, an enhanced BPEL engine can be used to orchestrate the way in which the different policies are enforced by their respective services. Conversely, message routing and endpoint resolution is where BPEL falls short. The BPEL engine can enforce the separation of logical users running the approval and distribution of the blood sample. But it cannot impose that this happens in every case. The mapping between the users and their machines is done by the ESB and not BPEL. Hence for a given blood sample, the ESB can ultimately resolve the blood distribution service endpoint to a machine that is different from the one on which the donation approval service has happened. Since it is realistic to assume each doctor has its own machine, then the separation of duty constraint is satisfied because different machines mean different doctors.

Having shown some important limitations of current ESB and BPEL approaches, in what follows we present a more detailed study of the enforcement capabilities of these two SOA components. Based on this we show how the enforcement capabilities of the ESB and the ones of the BPEL engine can be combined to do policy enforcement.

3.2 BPEL Engine Enforcement Capabilities

There are three main categories of *standard capabilities* that a common BPEL engine can employ for the detection of policy violations: observation, event triggering, and event aggregation.

- Observation of events: BPEL engines can observe events occurring during the execution of a process. The execution history of all process instances is saved in the audit trail, and the data can be later used for an analysis if a certain condition has become true.
- Triggering of events: BPEL engines are capable of triggering events at state changes during the execution of a BPEL process. Events that occur during the execution of a BPEL process are saved in the audit trail. The audit trail can be used to monitor the execution of all running instances of all deployed process models on a BPEL engine. A process instance can also be started when a message or event has arrived at the BPEL engine.
- Aggregation of events. In a BPEL engine, event aggregation is implicitly done by an internal component called the process navigator. The navigator is responsible for the execution of the activities defined in a BPEL process. One feature of the navigator is the initialisation of a new activity if the required previous activities have completed or faulted. The navigator of a BPEL engine is notified by an event that the preceding activity has ended. Such events can also be made visible to the outside for complex event aggregation, e.g. by emitting an event containing the aggregated information of several events.

Functionalities covering the reaction step of the enforcement life cycle include the suspension and termination of process instances. These enforcement abilities are part of the standard abilities of the BPEL engine we use in our prototype. To further support enforcement we enriched the BPEL engine with *additional* reaction abilities: functionality to block and unblock processes, to insert, delete, and modify activities, and to modify variables [3].

- Termination and suspension of process instances: A BPEL engine is capable of terminating and suspending running process instances. This means it is possible to stop a process instance that behaves in a way that violates policies. Terminated process instances cannot be reactivated; suspended process instances can be resumed.
- Block or un-block process instances: The execution of running BPEL processes can be blocked when a certain event occurs. This is the difference to the suspension of a process instance described before. To unblock a process instance another event needs to occur. This can be, for example, an unblock message coming from a policy decision component.
- Insertion, deletion, and modification of activities of a running process instance: It is possible to insert new activities into running process instances. For long running process instances this is a means to implement new policies that were not present when the process instance was initiated. The changes performed onto process instances can be mapped to the underlying process model. This procedure has been previously called *instance migration* [10].

– Modification of variables and activities: Variables and activities of running process instances can be modified to lead the process execution to a direction that avoids the violation of policies.

The BPEL engine inherently keeps track of the state of the process instances running on it. In case of an emergency shut down most BPEL engines persist the states of all running processes on disk so that they can be resumed after a restart. BPEL internal fault and compensation handling is not always visible to the outside, thus it must be signalled and enforced within the BPEL engine. State keeping helps to make BPEL-level enforcement decisions, but an enforcement decision maker on the BPEL engine cannot come off-the-shelf.

3.3 ESB Enforcement Capabilities

In this section we show enforcement aspects that are typical for an ESB and cannot happen on a BPEL engine. Following the enforcement life cycle model, detection in the ESB happens specifically on message flows. It encompasses endpoint resolution, observation, triggering of events, blocking of a message flow and transformation of messages. This functionality is provided by default in standard ESB platforms.

– Endpoint resolution: Endpoint resolution is one of the two main functions of the ESB. Resolving endpoints means matching a virtual endpoint to a physical endpoint in a service registry. The ESB uses its own protocols for this matching.
– Observation: The ESB mediates all messages flowing through. These message flows can be logged for later analysis.
– Triggering of messages: ESBs are capable of emitting events, usually when a certain message arrives. Hence when a certain message is received, the system is injected with a generated event. At message-level, both the message and the event can be either a service invocation, a service response, or a fault raised in the application.
– Blocking message flows: ESBs offer the possibility to react synchronously to incoming or outgoing messages. This means that when a certain message has been received or is about to be dispatched, the ESB can perform an action before routing the message to its destination.
– Transformation of messages: ESBs are capable of aggregating, splitting, filtering, and processing the messages that they route from one endpoint to another. Processing of messages on certain criteria is covered generically by enterprise integration patterns. Standard patterns like the message splitter, message aggregator, and message filter are already implemented in a number of ESB platforms.

The added enforcement functionalities that were suggested and implemented in our previous work [1] cover the reaction step of the ESB enforcement life cycle. They include:

– Modification of a message: The ESB has control over the messages between the source endpoint and destination endpoint. This implies that it can modify parts of the message, be it the message metadata (source, destination, security headers, etc.) or the message payload. Usually the ESB can discern between payload and routing information. Still, the payload is not always accessible to the ESB, for instance when it is encrypted.

- Rerouting messages: Rerouting messages can be done on the fly by sending a message not to its intended destination, but to another endpoint. Rerouting can serve multiple enforcement functions, e.g., block the initial destination endpoint from receiving a message or to retain the message for a limited amount of time.
- Inserting messages: ESB traffic can be duplicated among endpoints, as the ESB can insert messages into existing flows.

While the BPEL engine handles logical flows between logical endpoints, the ESB handles correspondence with the real endpoints. The ESB features native endpoint name resolution, message processing, message reliability and delivery mechanisms. For instance, the ESB has access to service metadata such as: position information, interface information, intra-service protocols, user tags, and ranking or price of a service. Endpoint resolution means the ability to choose another endpoint for a message depending on various parameters. Non-functional runtime aspects like traffic load and trust can influence the flow of messages between endpoints. The message flow can be controlled by the ESB at the infrastructure level. Similar to the case of the BPEL engine, the ESB does not come with decision making support with respect to any kind of constraints.

Table 1 shows a comparison of the new enforcement capabilities we added to the ESB [1] and the BPEL engine. It summarises our efforts to provide two middleware components with extended support for enforcement in an SOA. The fact that the BPEL engine and the ESB have two distinct feature sets in terms of enforcement emphasises that there is a need to combine these enforcement capabilities.

Table 1. Comparison of new Enforcement Capabilities of ESB and BPEL engine

New Enforcement Capability	BPEL engine	ESB
Modification of Messages	-	×
Rerouting Messages	-	×
Insertion of Messages	-	×
Block / Unblock Process	×	-
Termination / Suspension of Process Instances	×	-
Insertion, Deletion, and Modification of Activities	×	-
Modification of Process Variables	×	-

3.4 Enforcement Policy Model

To automatically react to critical states of a system, there have to be machine readable policies describing what should happen when the system reaches a certain state or tries a transition from one state to another. In this section we present a model of the information needed in an enterprise policy for regulatory compliance. With this model it is possible to describe enforcement actions on single SOA components like a BPEL engine or an ESB or on a combination of these components. We use this policy model in the Sections 4 and 5 to show the interaction between an ESB and a BPEL engine during the execution of an enforcement action.

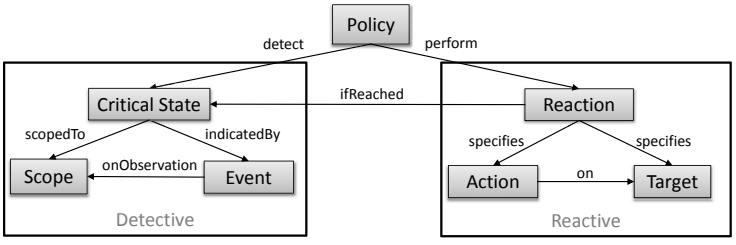

Fig. 3. Policy Content

Figure 3 shows the model specifying the content of the enforcement policy. It consists of two parts. The detective part specifies the critical state of the system to observe and how this state can be detected. The components that need to be observed in order to detect the critical state constitute the scope of the detective part. For example, in order to ensure encrypted communication between services S_1 and S_2, all messages sent between these services must be observed. A critical state in this case would be indicated by the detection of an unencrypted message. The reactive part specifies what actions should be taken if the critical state is reached and what are the targets of the actions. In the previous example, a reaction can be the encryption of the message. In this case *message* is the target of the action *encrypt*.

To enforce a policy, the decisions have to be made as to where the detective measures for the policy have to take place and where the reactions have to be executed. The decision is made based on the policy content, particularly on the observational scope and on the targets the reactive actions need to be performed on, and on the capabilities of the enforcement components, in particular ESB and BPEL engine, which are described in a service catalogue. Thereby detective and reactive parts can be performed by different enforcement components. Currently, the decision regarding whether the policy is enforced on a BPEL engine or on an ESB is made at the design time and must satisfy the following two criteria:

1. Detective part: The enforcement component must be able to observe the components specified in the scope
2. Reactive part: The enforcement component must be able to perform specified actions on the specified target

These criteria restrict the number of options to the components which are capable to enforce the given policy. For example, a BPEL engine would not be able to observe communications between services if this communication is not part of a business process running on this engine. Similarly, an ESB would not be able to control internal control and data flow of a business process. The ESB actions primarily target a *message* and are scoped to the observation of the messages that flow between the services that use this ESB. The BPEL engine actions, on the other hand, target *process*, *process instance*, *process activity*, *process activity instance*. Thus, if a policy specifies *Ensure that every instance of a business process performs activity Approve on Blood Sample*, then the target is clearly a business process and at least the detection of the critical state

should go to the BPEL engine. If the policy on the other hand says *Ensure that all messages between* $S_1, ..., S_n$ *are encrypted*, then the observation scope and the action target is a *message* and clearly belongs to the ESB. However, service invocations can be controlled on both ESB and BPEL level: on the BPEL level by controlling *invoke* activities that invoke a service, and on the ESB level by controlling the invocation message.

To demonstrate the differences and similarities between ESB and BPEL level enforcements in our case, consider the first example regulation specified in Section 2: *The same doctor cannot approve both a blood donation and the distribution of that blood sample to a public hospital*. The critical state of a policy guarding this regulation can be specified as *The doctor to have approved a blood donation wants to distribute the blood sample* with the corresponding reactive part of *Block blood sample distribution by the doctor*. To detect the critical state, two services S_1 performing *blood donation* and S_2 performing *blood distribution* need to be observed. If these services are invoked from the same business process BP_1, then the observation can be mapped to the observation of the corresponding *invoke* activities of BP_1 and thus can be done on the BPEL engine level. The reactive part in this example is specified by action *block* on target S_2, which can also be mapped to the blocking of the corresponding *invoke* activity on BPEL level. However, if usage of S_1 and S_2 is not coordinated by a single business process, for example when blood donation and blood distribution are parts of different business processes, then additional correlation of the service invocations is required. In this case the ESB can carry out the necessary message correlation to detect a critical state and block messages to the service S_2. In general, when the policy controls the usage of services S_1, ..., S_n, both ESB and BPEL can be used for its enforcement. In this case, the decision of which to use must be made by the designer based on the current system implementation and with the goal to minimize distribution of the policy enforcement. For example, when these services are invoked from multiple points but all invocations flow through one ESB, this ESB can intercept requests from different processes and services at one centralized point. A BPEL engine in this case would need an extra policy for each BPEL engine that runs processes that invoke these services, and thus would have multiple enforcement points. Similarly, if a set of services that need to be observed are invoked from a single business process, then their usage can be controlled at one centralized point on the BPEL engine level.

4 Combination of Enforcement Capabilities

The ESB complements the process engine enforcement because it performs complete mediation, endpoint resolution and leverages communication disparities. These disparities are often unavoidable because providers do not reveal the implementation of their services. To the usually stateless controls offered by the ESB, the BPEL engine provides a central point of coordination and global state management. ESB capabilities fit better with a cross-domain stateless enforcement, while the process engine better serves enforcing complex actions to achieve adherence to a common policy.

As described in sections 3.2 and 3.3, we have extended an ESB and a BPEL engine with added functionality for enforcement. Both components have been extended independently to meet enforcement requirements in their respective domain. In order to

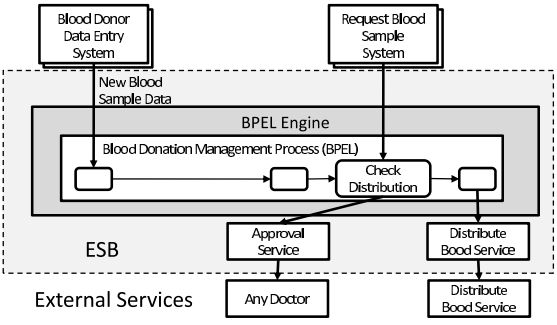

Fig. 4. Original implementation of the blood donation management process

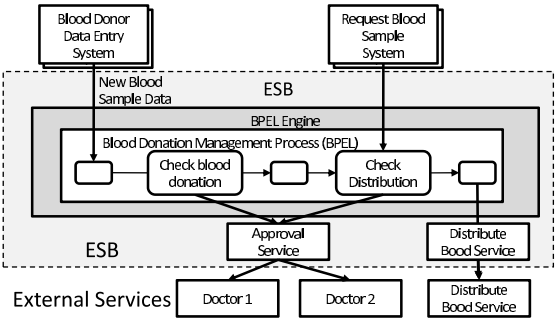

Fig. 5. Modified implementation of the blood donation management process. Also shows ESB responsible for choosing the doctor to approve the requests.

make best use of enforcement functionalities of both ESBs and BPEL engines we need to combine them. In the following we show how the combination of both enforcement capabilities works in a use case scenario.

Let us assume the Hong Kong Red Cross has a BPEL process for managing blood donations shown in Figure 4. This process is out-dated: it does not implement Regulation 1 as stated in Section 2 because it only has one check activity instead of the required two. There also is no policy deployed in the ESB ensuring the adherence to Regulation 1. The activity labelled *Check Distribution* is responsible to check if a blood sample can be distributed. The ESB forwards any approval request to any doctor being currently in charge for approvals in the hospital.

Figure 5 shows the new blood donation management process with Regulation 1 implemented. To implement Regulation 1, a new activity labelled *Check blood donation* has been inserted into the process model. Already running instances have been changed on the fly by using the activity insertion enforcement mechanism of the BPEL engine. The Approval service in the ESB has not been changed. No new endpoint has been introduced for the new check activity. Instead a new enforcement policy has been added to the ESB stating that two approval requests from the same process instance are routed

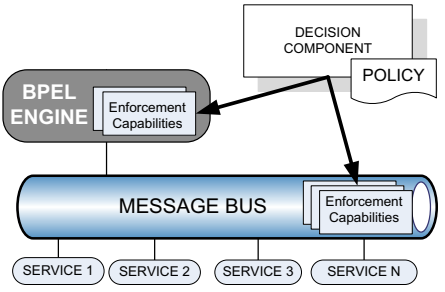

Fig. 6. The architecture implementing the enforcement life cycle. The arrows to the right indicate that the decision component uses the interfaces provided by the ESB and BPEL engine for detection and reaction to violations.

to distinct doctors. Of course, an underlying assumption is that every doctor is using the system from a physically different endpoint, so that the ESB can differentiate between doctors. When the same endpoint is operated by two different doctors for two different actions, the ESB cannot go beyond ESB endpoint information, which is normally not associated with user data. With extra information from the outside, the ESB can be helped to route to distinct doctor endpoints on the same machine.

Enforcement actions affecting the control flow of a business process are meant to be executed on a BPEL engine. It is possible to influence the control flow of a workflow from the outside by modifying input messages. To do this, deep knowledge of the workflow is needed in order to be capable of doing the right modifications to the input data. Therefore we propose to execute enforcement actions that affect the control flow of a workflow directly on the BPEL engine. On the other hand enforcement actions that have to do with message routing or endpoint resolution of Web services are better to be executed on the ESB.

We envision an architecture that joins the capabilities of the message bus with those of the BPEL engine, as shown in Figure 6. There can be one or more BPEL engines deployed to the bus to orchestrate the execution of a business process. The enforcement capabilities of these engines, as well as that of the ESB, are made available to a higher-level enforcement component that we call the *decision component*. As mentioned earlier, neither the message bus nor the BPEL engine offer any explicit support for enforcement decision making. Our suggested decision component will discover the ESB and BPEL detection capabilities (as described in Section 3.4), and will make a decision of how to react to a possible violation of a policy. This reaction can be either at the BPEL, at the ESB level, or at both levels. For the moment, the way to decide between these two cases is by checking an explicit flag in the policy, stating that its target it located at one level or the other. Thus, we rely on the policy writer to specify what part of the policy is linked to the ESB and what part to the BPEL instance. Keeping the decision component away from either the ESB or the BPEL engine makes the decision component independent of the limitations of either layer. In this way, the enforcement assurance of the composite will incorporate both types of message-level and BPEL engine capabilities.

5 Implementation Considerations

In this section we show how the prototypes of an ESB and a BPEL engine implement the enforcement life cycle. We use Regulation 1 from Section 2 to show an example of how the ESB and BPEL engine work together to perform an enforcement action.

First, we instrumented Apache ServiceMix 3.0[3] to offer hooks to message interception, default endpoint resolution and message modification. ServiceMix offers an interface for intercepting all messages at the level of the Normalised Message Router. We employ this interface for *detection*: we direct all intercepted messages to a Decision Component shown in Figure 6. Before being directed to this component, messages can be filtered based on message metadata like message destination, message source, and user data. As *reaction*, we added a set of Web services to the message bus that can perform the following enforcement actions: block a message, modify a message, delay a message for a defined period of time and insert a message into the message flow. These Web services act within the message router, before or after a message is dispatched.

Second, we extended the Apache ODE BPEL[4] engine with enforcement capabilities to block and unblock a running BPEL process. The concepts of the prototype are based on two previous approaches [3,4]. The first approach proposes an architecture to extend ODE to emit all events occurring when a process is executed. The BPEL engine is enhanced with a component to extract the internal events (*detection*). The other approach shows a management framework for BPEL based on Web service Resource Framework (WS-RF). Here, we use WS-RF to expose every BPEL process deployed on the Apache ODE BPEL engine as a resource (*detection*). This resource can be queried and changed. The changes are mapped back to the referred BPEL process (*reaction*).

Web service interfaces link enforcement on the Apache ODE BPEL engine with enforcement on the modified message bus. All SOAP messages produced by the ODE BPEL engine and possible application services are routed through the ESB. The ESB can access the BPEL engine capabilities, and hence ask for orchestration of enforcement actions or events occurring at the BPEL level. Conversely, the BPEL engine will access ESB messages and ask to perform message modification actions at the message level.

How is Regulation 1 enforced with this implementation? The ESB detects all incoming requests to the check blood donation and the check distribution services. All incoming requests to the former are let to pass unaltered. All incoming requests to the check distribution service are blocked waiting for an enforcement decision. A copy of the request to the check distribution services is sent to the decision component. The decision component will record in a forbidden address parameter the physical location of the doctor to which this request was directed, as well as the blood sample id. For each request, the decision component will check if the blood sample id is the same as in a previous check blood donation request.

If so, the decision component will invoke an enforcement operation at the BPEL engine in order to send the check distribution request again with a new addressee. The new request is again blocked by the ESB. After a check by the decision component, the block is released and the message is forwarded to the right doctor performing the check.

[3] `http://servicemix.apache.org/home.html`
[4] `http://ode.apache.org`

The Decision Component shown in Figure 6 is for now manual. It is performed by a human administrator declaring a list of doctors who cannot invoke the same Web service operations.

6 Related Work

In SOA enforcement, there has been no approach, yet that covers the enforcement life cycle in its entirety: (1) observation of events, (2) decision of right countermeasures, and (3) execution of a reaction. Moser et al. introduce the VieDAME4BPEL framework for monitoring BPEL processes and enforcing the replacement of partner services [8]. An interception and adaption layer allows to intercept outbound messages, monitor messages, apply message transformations, and replace services. Monitoring inbound messages and state changes is not discussed. This is a basic requirement to react to possible faulty behaviour of a BPEL process. Another approach supporting the dynamic change of workflows is $ADEPT_{flex}$ [9]. $ADEPT_{flex}$ defines a set of change operations which can be used to adapt the structure of a running workflow. Conducting automatic changes to the running workflow is not intended by the solution.

In [13] Tsai et al. propose an event driven framework to enforce policies in an SOA environment. In this work a BPEL engine is instrumented to emit events for certain state changes of a running BPEL process. These events are then aggregated by a so called global policy engine and forwarded to a local policy engine. This policy engine is responsible for executing the necessary enforcement action. There is no explanation on what enforcement actions are possible and how these actions are executed.

Research in SOA enforcement that considers message-level enforcement has a lot of generic models [11,5,2]. Neither of these solutions discusses how the enforcement at the message bus layer actually happens, so that it can be linked with the orchestration level. More relevant to our viewpoint are the message-level enforcement frameworks that are scoped to SOA governance [6,7]. These approaches hint to the centralization of policy management, when policies pertain to different abstraction layers. None of the solutions above separate between message-level and BPEL engine-level enforcement capabilities. To our knowledge, our work is the first to suggest to combine important SOA components on the stages of the enforcement life cycle.

7 Conclusion and Future Work

This paper introduces a new approach and model for ensuring policy enactment in enterprise applications. Our approach extends and combines BPEL and ESB capabilities on the framework of a SOA policy model. It is an architecture by which BPEL enforcement can use ESB abilities for policy enforcement, and vice versa. This is done by exposing these capabilities via enforcement interfaces and already implemented in the prototype. For future work, we will add mechanisms to automatically react to policy violations. For now, the decision making component decides whether to delegate the enforcement actions to the ESB or the BPEL levels based on flags contained in the policies. We aim to investigate the extent to wich this process can be automated, along with its granularity. Also, the ongoing work on our prototype will be extended with several case studies to show what is the impact of our model on the business process execution.

Acknowledgement

The work published in this article has partially received funding from the European Community's 7th Framework Programme Information Society Technologies Objective under the MASTER project[5] contract no. FP7-216917.

References

1. Gheorghe, G., Neuhaus, S., Crispo, B.: xESB: An enterprise service bus for access and usage control policy enforcement. In: Uehara, T. (ed.) IFIPTM 2010. LNCS, vol. 321, pp. 63–78. Springer, Heidelberg (2010)
2. Goovaerts, T., De Win, B., Joosen, W.: Infrastructural support for enforcing and managing distributed application-level policies. Electron. Notes Theor. Comput. Sci. 197(1), 31–43 (2008)
3. Khalaf, R., Karastoyanova, D., Leymann, F.: Pluggable framework for enabling the execution of extended bpel behavior. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 376–387. Springer, Heidelberg (2009)
4. van Lessen, T., Leymann, F., Mietzner, R., Nitzsche, J., Schleicher, D.: A Management Framework for WS-BPEL. In: Proceedings of the 6th IEEE European Conference on Web Services 2008, pp. 187–196. IEEE Computer Society, Los Alamitos (November 2008)
5. Leune, K., van den Heuvel, W.J., Papazoglou, M.: Exploring a multi-faceted framework for soc: how to develop secure web-service interactions? In: Proc. 14th Intl. Workshop on Research Issues on Data Engineering, pp. 56–61 (March 2004)
6. Maierhofer, A., Dimitrakos, T., Titkov, L., Brossard, D.: Extendable and adaptive message-level security enforcement framework. In: International conference on Networking and Services, ICNS 2006, pp. 72–72 (2006)
7. Hafner, M., Mukhtiar Memon, R.B.: SeAAS - a reference architecture for security services in SOA. Journal of Universal Computer Science 15(15), 2916–2936 (2009)
8. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for ws-bpel. In: WWW, pp. 815–824 (2008)
9. Reichert, M., Dadam, P.: Adeptflex: Supporting dynamic changes of workflow without loosing control. Journal of Intelligent Information Systems 10, 93–129 (1998)
10. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. T. Petri Nets and Other Models of Concurrency 2, 115–135 (2009)
11. Svirskas, A., Isachenkova, J., Molva, R.: Towards secure and trusted collaboration environment for european public sector. In: Collaborative Computing: Networking, Applications and Worksharing. CollaborateCom 2007. International Conference on, pp. 49–56 (2007)
12. Trojer, T., Kwong Lee, C., Fung, B.C.M., Narupiyakul, L., Hung, P.C.K.: Privacy-aware health information sharing. In: Privacy Aware Knowledge Discovery: Novel Applications and New Techniques, Chapman and Hall/CRC Press, Boca Raton (2010)
13. Tsai, W.T., Zhou, X., Chen, Y.: Soa simulation and verification by event-driven policy enforcement. In: ANSS-41 2008: Proceedings of the 41st Annual Simulation Symposium (anss-41 2008), pp. 165–172. IEEE Computer Society, Washington (2008)
14. United States Code: Sarbanes-Oxley Act of 2002, pl 107-204, 116 stat 745. Codified in Sections 11, 15, 18, 28, and 29 USC (July 2002)

[5] `http://www.master-fp7.eu`