Achieving life-cycle compliance of Service-Oriented Architectures: Open issues and challenges

Theodoor Scholte[1], Engin Kirda[2]

[1] SAP Research
805 Avenue du Docteur Maurice Donat
06254 Mougins Cedex France
theodoor.scholte@sap.com

[2] Institut Eurécom
2229, Route des Crêtes
06560 Valbonne, France
kirda@eurecom.fr

Abstract. The introduction of regulations such as the Sarbanes-Oxley act requires companies to ensure that appropriate controls are implemented in their business applications. Implementing and validating compliance measures in 'agile' companies is time consuming, costly, error-prone and a maintenance-intensive task. This paper presents an approach towards dynamically adapting a Service Oriented Architecture (SOA) such that business applications remain compliant. In order to ensure compliance, a compliance checking mechanism for the SOA is needed. Upon detection of a threat/violation, the components of a business application are adapted using aspect-oriented programming (AOP). In this paper, we discuss the fundamental problems and we give an architectural description of our approach.

Keywords: Business Process Management, Compliance Management, Compliance Checking, Service-Oriented Architectures, Aspect-Oriented Programming, Risk Assessment, Risk Mitigation

1 Introduction

In order to survive in today's business world which is characterized by fact-paced market development, emerging technologies, increased time-to-market pressure and shortened product life cycles, enterprises need to be able to quickly adapt in terms of business processes, partners and relations.

The introduction of regulations such as the Sarbanes-Oxley act [36], Basel II Accord [34], Code Tabaksblad [21], HIPAA [8], IFRS [5], MiFID [7] and LSF [17] requires organisations to implement an effective internal controls system in the enterprise. Non-compliance to rules and regulations can be the cause of juridical pursuits as financial scandals have shown. Examples include Enron and WorldCom in the US and Parmalat in Europe [10, 12]. More recently, it became clear that the absence of proper policies, regulations and controls are one of the factors that caused the subprime mortgage crisis which resulted in government bailouts of financial firms, bankruptcies or selling of banks at fire sale prices [38]. The term Compliance Management refers to identifying, modeling and implementing rules and regulations such that illegal and illicit behaviour will be avoided when performing business activities. Thus, proper Compliance Management helps in mitigating the risks to illegal, illicit and fraudulent behavior and financial losses. Regulations and legislations constrain the business and are organisation-centric, business-centric, information-centric, legal-aspects centric and human-centric descriptions [26]. They are often imposed by external entities such as the government. Implementing these rules and regulations is difficult as they are documented and communicated in natural language. Furthermore, they are expressed at a high-level of abstraction which means that they have to be translated into executable models and policies such that they can be enforced by the underlying infrastructure. This mapping is always done manually as there are no tools available to automate this process. For these reasons, designing a business process that satisfies laws, rules and legislations and implementing it on top of an IT-infrastructure is a time consuming, costly and error-prone process.

Business applications are used by companies to help them in achieving their business goals. Business goals are reached by performing business activities which can be described by business processes. A popular way to develop a business application is by modeling and implementing a business process through the use of the Service Oriented Architecture (SOA) paradigm. In this paradigm, the functionality of an IT system is structured in small units called services. Then, business processes are modeled to orchestrate these services in order to implement a business activity. The services providing the implementation may change independently from the process specification; enabling and accelerating business & IT alignment and agility. The implementation of a business process requires different stakeholders to be involved due to the increased size and complexity of today's organizations. This issue has been addressed in existing work on Business Process Management (BPM) [18, 23, 37] and Enterprise Architecture [44].

Compliance Management requires the modeling and the implementation of constraints in the implemented business process. Regulations and business objectives change independently and irregularly from each other. Business applications that are compliant to rules and regulations, are designed and managed through separate activities and by several different experts which have different domain knowledge [22] (e.g. risk and juridical experts). As mentioned above, the mapping of abstract and high-level compliance requirements to implementable rules and policies is a manual process. Therefore, managing compliance is not only time consuming, costly and error-prone but also maintenance-intensive [26]. A scalable, robust and powerful approach is desired to solve the above issues.

In this paper, we make the following contributions:

– We identify the problems related to the semi-automatic adaptation of business applications given a set of constraints that mitigate the risk that illegal/illicit behavior will occur.
– We propose an architecture of an application that can potentially solve the identified problems.

This paper is structured as follows. In section 2 we discuss the problems related to compliance management, section 3 presents a solution architecture that allows adapting business applications automatically based on compliance rules and runtime information. In section 4 we discuss related work. Finally, a summary and an outlook on future research is given in section 5.

2 Problem Discussion

2.1 Case study

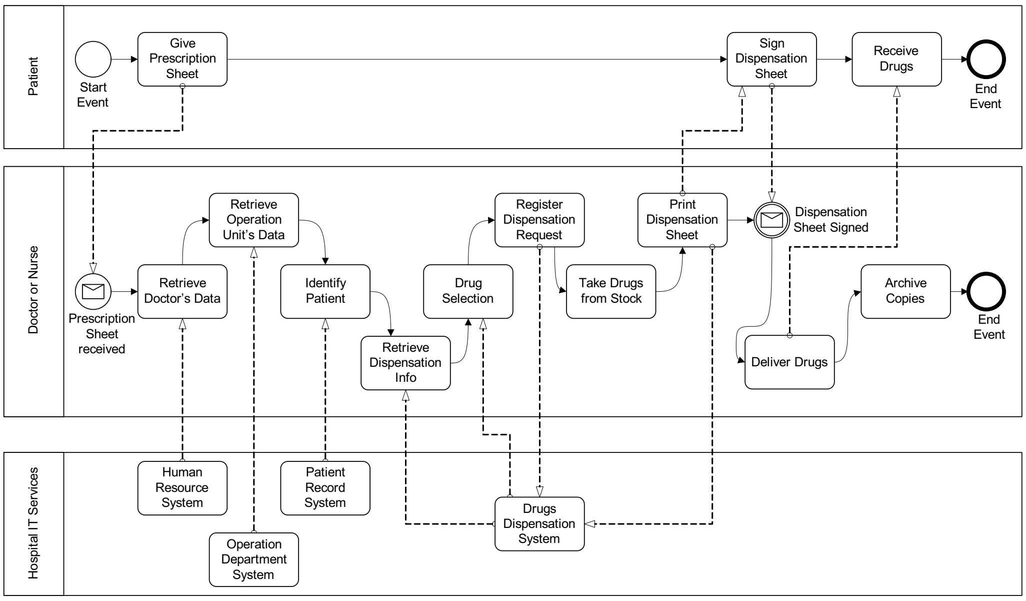

Fig. 1. *Drugs dispensation Business Process in use by one of the largest hospitals in Milan.*

The following use case is used in the EU FP7 project MASTER [9] and it contains the standard business processes that are in use by one of the largest hospitals in Milan. Here, the use case will be used to explain the basic concepts of business processes and internal controls. We use this example to motivate our research problem.

In this real hospital in Milan, drugs are dispensed to patients according to the business process depicted in Figure 1. Modeling a business process for dispensing drugs is normally a complex activity. Therefore, we use a simplified example. The business process starts with a patient who hands a prescription sheet to a doctor or nurse. The doctor/nurse logs into the dispensation software application, the operational unit of the doctor is identifier. Then, the doctor is able to select the patient who should receive the drugs. The system receives all the necessary information to select the drugs to be dispensed. The doctor chooses the drugs, registers this, takes the drugs from stock and registers this. Then, the doctor hands the drugs to the patient. The business process contains manual activities as well as activities that are implemented as IT Services. The presence of automated activities are illustrated by the arrows between the 'Doctor or Nurse'-lane and the 'IT Services'-lane. For the rest of this paper we assume that the dispensation software application adopts the SOA paradigm. Thus, the specification of the business process model as shown in Figure 1 is deployed on a business process engine which orchestrates Web Services.

The managers of the hospital wish to make sure that all the medical and non-medical operations that are performed in the hospital conform to a set of relevant internal controls. Examples of relevant compliance rules for the business process depicted in Figure 1 include:

1. Only doctors or nurses are allowed to access the dispensation software application.
2. A doctor/nurse cannot dispense drugs to him or herself (i.e. being a patient and doctor at the same time is not allowed).

The business process depicted in Figure 1 describes *how* a business activity should take place in order to meet a business objective. This is in contrast with compliance rules such as the ones listed above as they are declarative meaning that they indicate *what* the hospital can do in order to satisfy a control. The compliance rules that are listed above typically *imply* certain behaviour and they constrain certain behaviour. The business applications in use by the hospital might or might not be compliant with the compliance rules stated above. Our aim is to adapt the business application automatically when a violation of a compliance rule occurs resulting in compliant business applications and thus, mitigating the risk of additional illegal and illicit behavior occurring.

2.2 Solving non-compliance

In the previous sections, we have explained what compliance management means and why it is difficult to manage compliance of business applications. We now explain the problems of the existing approaches towards compliance management in more detail.

Companies perform audits to ascertain the validity and reliability of information and to assess to which extent the systems implement compliance measures.

Audits are performed in order to be certified as being compliant to certain regulations. The outcome of an audit is a set of risks that are relevant to the organisations assets and an evaluation of the effectiveness of the controls that mitigate risks. The auditing-approach to compliance management generates high-costs as it requires auditors with the necessary expertise and knowledge and audits have to be performed on a regular basis. Audits require experts on regulations as well as experts of the organisation's business and IT-infrastructure to be involved. Due to the complexity, audits check only a part of the business and IT landscape by adopting statistical sampling. The outcome of an audit can be that risks have increased and/or existing compliance measures are not effective enough due to, for example, new versions of regulations being introduced, changes being made in business processes and/or IT-infrastructure. Then, the business processes and the underlying IT-infrastructure need to be adapted such that risks are mitigated. As mentioned in the previous sections, managing compliance is time consuming, costly, error-prone and maintenance-intensive. A software solution that detects violations and potential violations, also called threats, of compliance rules can support the management of compliance. In addition, the software should 'solve' threats and violations by adapting the business process, the business logic and its underlying IT-infrastructure such that non-compliant behaviour will be prevented or compensated, and risks for the organisation are thus mitigated.

In the following sections, we explain the problems that are specific to this software solution.

Modeling behavior and constraints. Business processes and its underlying IT-infrastructure can be described by behavioural models including orchestration models and choreography models. While a choreography model like WS-CDL [40] specifies a collaborative behavior of two or more participants, a business process orchestration model like WS-BPEL [33] specifies a composition of activities designed to achieve a certain business goal. The orchestrations are defined by specifying which services and operations should be invoked. Compliance rules *imply* and *constrain* certain behaviour. We can identify four types of constraints:

- Security constraints
  This type of constraints include all the constraints that have to be put on the system in order to meet security requirements such as confidentiality, integrity, authentication, authorization, availability and non-repudiation. A well-known example of a security constraint is the Segregation of Duties or four-eye principle constraint. This constraint requires multiple persons to complete a task. The second compliance rule in the case study is an example of a Separation of Duty constraint.
- Domain-specific constraints
  These constraints refer to the business rules of the enterprise and are specific to the context/domain of the enterprise. An example in the context of drugs dispensation could be: when a patient gets Paracetamol and a Blood Thinner dispensed, a warning should be raised.

- Orchestration constraints
  Orchestration constraints include dependencies between the activities in a business process and are specified in a business process model. An example based on the scenario in section 2.1 is that a ‘Take Drugs from Stock’ activity must be followed by a ‘Print Dispensation Sheet’ activity.
- Choreography constraints
  This type of constraints are the ones that are enforced over the interaction between business partners and are specified in a choreography model. Consider a business process where doctors prescribe drugs and a pharmacy dispense drugs. A doctor can only send a prescription to the pharmacy if he received an acknowledgement of the previous prescription from the pharmacy.

Please note that this classification does not enforce that a particular constraint falls within one specific class of constraints. For example, a constraint such as ‘a patient should not get Paracetamol and Blood Thinner dispensed at the same time’ is a domain-specific constraint. But it is also an orchestration constraint when the orchestration model specifies an activity ‘check dangerous drugs combinations’ followed by a ‘warn doctor’ activity. The verification of the implementation of a business process requires a language to describe the model of the target system and a language to describe the constraints that have been put on the system. This language should be expressive enough to model the semantics of the rules and regulations that exist in the real-world, and yet abstract enough for the purpose of validation and analysis.

Detecting threats and violations. Threats and violations have to be detected during the whole lifecycle of business processes. This can be achieved by compliance checking which refers to the verification of the status of compliance measures in the enterprise [15]. We can identify two complementary approaches for compliance checking: design-time and run-time. Compliance checking at design-time means the verification of a behavioural model (formal model) of the implemented business process against a set of formally specified constraints (compliance rules). The behavioural model is compliant if its definition complies with the predefined set of compliance rules. Runtime compliance checking is based on the evidence collected at runtime and is required here to detect whether threats or violations of constraints occurred in practice. With respect to orchestration and choreography constraints, verifying the behavioral models (at design-time) is possible but it is not sufficient enough as unexpected behavior might occur at runtime. In addition, the validation of security and (data) integrity constraints requires information that is not available at design-time. The challenge for compliance checking, is to come up with an approach that does not only detect non-compliant behaviour but is also able to trace back to the causes of non-compliant behaviour.

The problem of adaptation. When threats and/or violations of compliance rules have been detected, business applications should be repaired automatically such that violations can be prevented or violations can be compensated and risks are mitigated. Software adaptation can either be done at *design-time* or at *runtime* [6]. Design-time adaptation refers to all types of changes made to software before the software system is running. This can include modifications of requirements/specifications, modifications of source code or changes of configuration files. An important property of this type of software adaptation is that all the steps in the adaptation process are known and have been planned in advance. This is in contrast to dynamic or runtime software adaptation that refers to techniques that allow to change running pieces of software. In the context of business process driven applications, adaptations can be made by modifying the business process model. A business process can be modified by inserting, deleting or shifting activities in the service orchestration [11]. Just as with generic software adaptation, modifications of the orchestration and/or choreography model can be applied at design-time or at runtime. Design-time adaptation is needed when the business application has to meet new (compliance) requirements and there is an intention to reuse the modifications. Dynamic or runtime adaptation allows us to bring a running business application to a compliant state without the need to restart the application which might result in loss of data. Since we would like to prevent and compensate violations before the execution of a business application is finished, dynamic software adaptation techniques should be applied. An important issue here is maintaining the consistency of the control flow and of runtime data. This requires the development of adaptive middleware. A second issue with respect to dynamic adaption is that the compliance rules do not specify *how* to repair non-compliant behaviour. Depending on the business process, its implementation and a compliance rule, there might be more than one way or *strategy* to repair a business application. Thus, a compliance rule is always associated with one or more repair strategies. The challenge here is the modeling and coding of these repair strategies.

3 System Overview

Figure 2 depicts the global overview of our system. Business process models define the way how Web Services are orchestrated and these models are deployed on a BPEL engine [33]. In addition, the system includes an XACML policy engine [32] which is responsible for evaluating access control requests originating from Web Services against a set of access control policies. The adaptors that are responsible for collecting evidence of software behavior and the adaptors for repairing the software are implemented using aspect-oriented programming (AOP). The main reason for choosing an AOP approach is that it supports *compile-time, load-time and runtime weaving* [35, 39].

Aspect-Oriented Programming and Annotations The following paragraphs gives a very short introduction on AOP and annotating source code. For a more comprehensive introduction, consider reading [24, 25] and [4] for source code annotations.

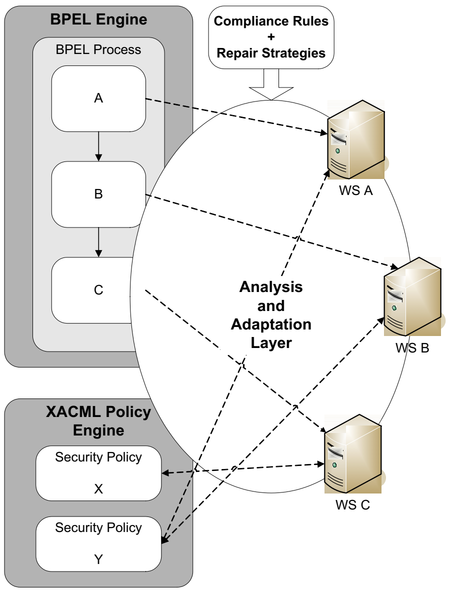

Fig. 2. *High-level overview of an adaptive Service-Oriented Architecture.*

Aspect-oriented programming is a programming paradigm which aims to provide modularizing techniques supporting the separation of cross-cutting concerns in complex software systems. Examples of cross-cutting concerns include security constraints, logging functionality and communication protocols. The main idea is to separate the cross-cutting concerns in stand-alone modules called *aspects*. An aspect is related to one or more places in the code which are called *join points*. In order to identify join points, the notion of a *pointcut* is introduced. The additional behavior at a join point is specified in an *advice* and this code can run before, around or after a join point. The AOP framework is responsible for combining the base functionality with the additional code, this step is called weaving. Weaving can be done at compile-time (by the compiler), load-time (by the classloader) or at runtime. With runtime weaving, targets can be declared at runtime. The class bytecode can be redefined at runtime without reloading the class.

Annotations, and in particular Java annotations, are a special form of metadata that can be added to Java source code. All types of declarations can be annotated: packages, classes, variables and methods. Unlike javadoc, Java annotations may be available at runtime. The Java VM may retain annotations and make them retrievable at runtime. Annotations do not directly affect the application semantics, but they can be processed by tools at design-time or at runtime. Then, these tools can affect the application behavior. A similar concept exists for .NET.

3.1 Architecture

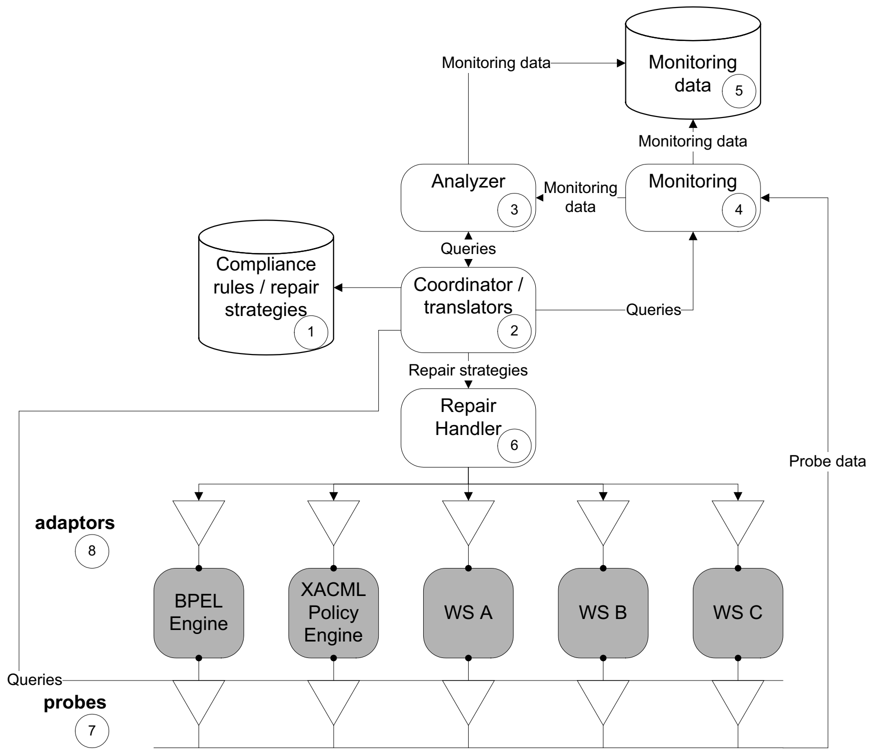

Fig. 3. *Detailed run-time architecture of an adaptation system for a Service-Oriented Architecture.*

Figure 3 depicts the runtime architecture of the approach taken. Due to space constraints, we left out the architecture of the design-time infrastructure. The compliance monitoring and adaptation process starts with modeling compliance rules and repair strategies using a modeling tool. These models are stored in a repository (1). Whenever a new compliance rule or repair strategy is added to the repository, the coordinator (2) is notified and translates the compliance rule to queries for the probing infrastructure (7), the monitoring infrastructure (4) and the compliance analyzer (3). The probing infrastructure (7) is responsible for collecting evidence of the system's behavior. Each probe is a 'hook' into a component of the Service-Oriented Architecture (BPEL engine, policy engine, Web Service) and emits events that represent the behavior of a component. In order to reduce the number of events that are emitted by the probes and to 'enrich' the evidence, there is a monitoring component (4) which is implemented by a *complex event processor*. This component filters, aggregates and correlates events. The monitoring component fills a repository (5) which contains historical data of the actions that were performed by the Service-Oriented Architecture. The compliance analyzer (3) analyzes continuously the historical evidence from the repository (5) and the accurate evidence originating from the monitoring infrastructure (4). If the compliance analyzer detects a violation of a compliance rule while analyzing the evidence, the coordinator (2) is notified. The coordinator retrieves the corresponding repair strategies (advice) from the repository (1) and sends them to the repair handler (6) together with the location of the source of the evidence. Based on this location, the repair handler chooses the locations of the adaptation. This location is composed of a component identifier and a pointcut which identifies the join point to insert the additional code. The component identifier identifies the adaptor (8) to which the repair handler should send the advice.

Probes and Adaptors. The probes and adaptors are both implemented using the aspect-oriented programming (AOP) paradigm. These components are added to the base functionality of the Service-Oriented Architecture as advices. The main reason is that this approach allows us to enable or disable a probe or adaptor (and the adaptor's associated repair strategy) at compile-time, load-time or runtime. The coordinator determines *where* in the target system to put probes and the repair handler determines this for the adaptors. The coordinator and repair handler specifies this in a pointcut. Pointcuts can be specified using a language which syntax is based on the base language, like Java signatures when Java is used. In order to determine and define the crucial parts of a target's system component where evidence collection and adaptation should take place, a more fine-grained way of specifying point cuts is needed. In our approach, we annotate the source code of each component in the target system and we expose the annotations to the coordinator and the repair handler. By referring to the annotations in the pointcuts, the coordinator and the repair handler are able to determine and define the locations for evidence collection and adaptation.

3.2 Case study revisited

In this section, we describe the behavior of our Compliance Management Solution when a violation of the compliance rule 'Only doctors or nurses are allowed to access the dispensation software application' (section 2.1) occurs. This rule implies that every Web Service implements an access control mechanism. As mentioned above, we assume that there is a centralized access control policy engine (Policy Decision Point in XACML terminology) and every Web Service (Policy Enforcement Point) implementing an activity of the business process depicted in figure 1 sends requests to this policy engine. Now, the compliance rule can be refined to the following compliance rule 'always when a Web Service needs access control the PDP has to receive an access control request message'. This rule can be expressed in LTL as follows:

```
[]((Q & !R & <>R) -> (P U R))
```

where Q *refers to the event representing the need for access control,* P *refers to the event representing the access control request message at the PDP and* R *is a time-bound equal to* $\mathsf{Q} + 30$ *seconds*

This LTL property is stored in the 'compliance rules / repair strategies' repository together with a repair strategy (advice) that includes all the source code for making an XACML request, sending it to the policy decision point and processing the response. An XACML request requires parameters such as subject's and resource' identity and the action. We assume that the variable-names of these parameters can be found in the annotations. Then, the repair handler can retrieve them and generate the correct repair strategy. Due to space constraints, we do not give an example here of the source code of a XACML request. When the combination of a compliance rule / repair strategy is added to the repository, the coordinator gets notified and translates the compliance rule to queries for the probing infrastructure, the monitoring components and the compliance analyzer. Consider now that a user performs the activity 'Register Dispensation Request' of the business process depicted in figure 1. The Web Service implementing this activity starts executing the method RegisterDispensation. The coordinator identified this as a critical point (based on the annotation and compliance rule) and put in advice in place that emits an event when this method is called. The event represents the need for access control. The XACML policy engine does not emit an event representing an access control request. The compliance analyzer detects this as there is a violation of the compliance rule. The coordinator passes the source of the violation of the compliance rule to the repair handler together with the repair handler. The repair handler concludes that the web service does not implement a valid access control request mechanism and decides to apply the repair strategy that performs a XACML request to the centralized policy engine.

4 Related Work

Related work in the context of compliance focuses mainly on modeling controls and compliance validation but not on software adaptation techniques.

Compliance modeling. The ability to model compliance rules are essential for our solution. Models of compliance rules are stored in the repository and the coordinator translates them to queries that can be deployed on the evidence collection and processing infrastructure. The control pattern introduced in [30, 31] acts as a pattern-based abstraction layer that separates business process and compliance management by annotating the process model with compliance rules. The approach is promising because the patterns have been used for runtime compliance validation. However, it lacks support for modeling constraints between process instances and modeling context. Moreover, the applicability of control patterns for dynamic adaptation of SOAs has not been shown. In [41, 42], Wolter et al. propose to use annotated business process models to model security requirements. The approach allows to extract security policies such as

AXIS2, XACML and WS-Policy security configurations from an annotated business process model. However, compliance rules include more than only security requirements such as authorization, access control and encryption.

Design-time compliance checking. Approaches towards a priori or design-time compliance checking are based on the concept of validating a specification of a process model against a certain set of compliance properties including the ordering of activities, liveness and correctness properties. Although our approach does not include design-time compliance checking, we describe here some work in that area. The approaches proposed in [3, 13, 16, 22, 27, 43] are all based on a priori compliance checking. The differences between the approaches are: 1) the languages used to specify the process models and the compliance properties 2) the model checker or reasoning techniques used. In [13], Concurrent Transaction Logic (CTR) is used as the language to specify, analyze and to schedule workflows. The compliance properties or the constraints are specified as CTR formulas and also workflow graphs are transformed to CTR formulas. Liu et al. proposed in [27] a compliance-checking framework that allows to model process models in BPEL and compliance properties in the graphical Business Property Specification Language (BPSL). Model transformation techniques are used to map the BPEL process models to FSMs and the compliance properties to LTL properties. [3] does not use BPEL but BPMN diagrams which are translated in REO models. The approach presented in [16] focuses on verifying the compliance of service interactions against obligation policies. These policies describe what actions a subject must or must not do to a set of target objects. The service interactions are specified in BPEL, the obligations in Message Sequence Charts (MSC). A similar approach is [43] in which a BPEL process is validated against properties expressed using property patterns [14].

Runtime compliance checking. In our solution architecture, run-time compliance checking is used to detect the cause of non-compliant behaviour in a SOA. In [19, 20] a method and meta-model is introduced that captures compliance requirements in a language. Using this framework, abstract policies can be translated to implementation artifacts such as business process definitions, data retention policies, access control lists and monitoring policies. This model-transformation process can, at least partially, be carried out automatically. Agrawal et al. addressed in [1] the importance of using database technology for run-time and a-posteriori compliance checking. The work in [2] focuses on an event-based language that can be used for run-time monitoring of Web Service interactions. The work presented in [15, 29, 28] propose different frameworks for compliance management. All of them adopt both design-time and run-time compliance checking techniques. The work looks only at the level of business process execution while we are planning to look at a lower-level. In [30, 31], a semantic mirror is used to collect run-time information of process instances. Violations to pre-defined control patterns are detected by the semantic mirror.

5 Conclusion and Outlook

The introduction of laws and regulations leads to the need to identify, model and implement proper controls in the IT-landscapes of organisations such that illegal and illicit behaviour can be avoided when performing business activities. Managing compliance in 'agile' companies requires the use of a software solution that is able to detect non-compliant behaviour and adapts the components of business applications accordingly. In this paper, we identified the problems this software solution should cope with and we presented an architecture which uses the aspect-oriented programming paradigm for evidence collection and software adaptation. Future work will focus on developing a proof-of-concept of the proposed architecture. Moreover, one of the open issues is determining the relevant join points for evidence collection and adaptation. We gave some hints and directions, but additional research is necessary. Another open issue is the translation from compliance rules to queries that can be evaluated on probes, the monitoring infrastructure and the compliance analyzer.

Acknowledgements
This work was supported by funds from the European Commission (contract N 216917 for the FP7-ICT-2007-1 project MASTER).

References

1. Rakesh Agrawal, Christopher Johnson, Jerry Kiernan, and Frank Leymann. Taming compliance with sarbanes-oxley internal controls using database technology. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 92, Washington, DC, USA, 2006. IEEE Computer Society.
2. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, Sergio Storari, and Paolo Torroni. Computational logic for run-time verification of web services choreographies: Exploiting the *ocs-si* tool. In *WS-FM*, pages 58–72, 2006.
3. Farhad Arbab, Natallia Kokash, and Sun Meng. Towards using reo for compliance-aware business process modeling. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA*, volume 17 of *Communications in Computer and Information Science*, pages 108–123. Springer, 2008.
4. Calvin Austin. J2se 5.0 in a nutshell.
5. International Accounting Standards Board. International accounting standard 1: Presentation of financial statements.
6. C. Canal, J.M. Murillo, and P. Poizat. Software adaptation. 14(13):2107–2109, 2008.
7. European Commission. Markets in financial instruments directive.
8. United States Congress. Health insurance portability and accountability act of 1996.
9. EU FP7 MASTER Consortium. Managing assurance, security and trust for services. http://www.master-fp7.eu.
10. Julie Creswell. Citigroup agrees to pay 2 billion in enron scandal. *The New York Times*, June 2005.

11. Peter Dadam and Manfred Reichert. The adept project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development*, (23):81–97, 2009.
12. Eric Dash. Parmalat sues citigroup over transactions. *The New York Times*, July 2004.
13. Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 25–33, New York, NY, USA, 1998. ACM.
14. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. Technical report, Amherst, MA, USA, 1998.
15. Marwane El Kharbili, Sebastian Stein, Ivan Markovic, and Elke Pulvermller. Towards a framework for semantic business process compliance management. In *Proceedings of the First International Workshop on Governance, Risk and Compliance (GRCIS)*, Montpellier, France, June 17, 2008.
16. Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based analysis of obligations in web service choreography. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, page 149, Washington, DC, USA, 2006. IEEE Computer Society.
17. Gouvernement Francais. La loi de scurit financire.
18. George M. Giaglis. A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228, April 2001.
19. Christopher Giblin, Alice Y Liu, Samuel Müller, Birgit Pfitzmann, and Xin Zhou. Regulations expressed as logical models (realm). Technical Report RZ 3616, IBM Research, Zurich, 07 2005.
20. Christopher Giblin, Samuel Müller, and Birgit Pfitzmann. From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Technical Report RZ 3662, IBM Research, 2006.
21. Commissie Corporate Governance. De nederlandse corporate governance code: Beginselen van deugdelijk ondernemingsbestuur en best practice bepalingen.
22. Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business processes and business contracts. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pages 221–232, Washington, DC, USA, 2006. IEEE Computer Society.
23. A. H. M. Ter Hofstede and M. Weske. Business process management: A survey. In *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*, pages 1–12. Springer-Verlag, 2003.
24. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.
25. Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean M. Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.
26. Ulrich Lang and Rudolf Schreiner. Managing business compliance using model-driven security management. In *Proceeedings of ISSE 2008 Securing Electronic Business Processes*, 2008.

27. Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.*, 46(2):335–361, 2007.
28. Linh Thao Ly, Kevin Gser, Stefanie Rinderle-Ma, and Peter Dadam. Compliance of semantic constraints - a requirements analysis for process management systems. In *Proc. 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08)*, Montpellier, France, 2008.
29. Linh Thao Ly, Stefanie Rinderle, and Peter Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1):3–23, 2008.
30. Kioumars Namiri and Nenad Stojanovic. A formal approach for internal controls compliance in business processes. In *Proceedings of the 8th Workshop on Business Process Modeling, Development, and Support*, Trondheim, Norway, 2007.
31. Kioumars Namiri and Nenad Stojanovic. Pattern-based design and validation of business process compliance. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 59–76. Springer Berlin / Heidelberg, 2007.
32. OASIS. extensible access control markup language (xacml) version 2.0, February 2005.
33. OASIS. Web services business process execution language, 2007.
34. Basel Committee on Banking Supervision. International convergence of capital measurement and capital standards: A revised framework.
35. Andrei Popovici, Thomas Gross, and Gustavo Alonso. Dynamic weaving for aspect-oriented programming. In *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*, pages 141–147, New York, NY, USA, 2002. ACM.
36. Paul Sarbanes and Michael Oxley. Sarbanes-oxley act of 2002 (pub.l. 107-204, 116 stat. 745).
37. Wasana Sedera, Guy G. Gable, Michael Rosemann, and Robert W. Smyth. A success model for business process modeling: findings from a multiple case study. 2004.
38. David Streitfeld and Gretchen Morgenson. Building flawed american dreams. *The New York Times*, October 2008.
39. A. Vasseur. Dynamic aop and runtimeweaving for java - how does aspectwerkz address it? In *In Workshop on Dynamic AOP*, 2004.
40. W3C. Web services choreography description language version 1.0.
41. Christian Wolter, Michael Menzel, Andreas Schaad, Philip Miseldine, and Christoph Meinel. Model-driven business process security requirement specification. *Journal of Systems Architecture*, page 13, 2008.
42. Christian Wolter, Andreas Schaad, and Christoph Meinel. A transformation approach for security enhanced business processes. In *Proc. SE2008 of 26th IASTED International Multi-Conference*, February 2008.
43. Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In *WISE*, pages 156–168, 2006.
44. John A. Zachman. A framework for information systems architecture. *IBM Syst. J.*, 26(3):276–292, 1987.