Monitoring business process compliance : a view based approach

Samir Sebahi

To cite this version:

Samir Sebahi. Monitoring business process compliance : a view based approach. Other [cs.OH]. Université Claude Bernard - Lyon I, 2012. English. NNT : 2012LYO10032 . tel-00866483

HAL Id: tel-00866483
https://theses.hal.science/tel-00866483

Submitted on 26 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 32-2012 Année 2012

THESE DE L'UNIVERSITE DE LYON

Délivrée par

L'UNIVERSITE CLAUDE BERNARD LYON 1

Ecole doctorale informatique et mathématiques

DIPLOME DE DOCTORAT

(arrêté du 7 août 2006)

Soutenue publiquement le 22 Mars 2012

par **Samir SEBAHI**

BUSINESS PROCESS COMPLIANCE MONITORING A VIEW BASED APPROACH

Composition du jury :

Rapporteurs	
M. Kamel Barkaoui	Professeur - Conservatoire National des Arts et Métiers – Paris
M. Olivier Perin	Professeur, Université Nancy 2
Examinateurs	
M. Farouk Toumani	Professeur, Université Blaise Pascal – Clermont-Ferrand
M. Hamamache Kheddouci	Professeur à l'Université Claude Bernard Lyon1
Co-directeurs de thèse	
M. Mohand-Said Hacid	Professeur à l'Université Claude Bernard Lyon1
Mme. Salima Benbernou	Professeur à l'Université Paris Descartes

Remerciements

Je remercie tous ceux qui m’ont accompagné et soutenu pendant ces années, en particulier, je voudrais exprimer toute ma reconnaissance à mes directeurs de thèse, Salima Benbernou et Mohand-Said Hacid, pour leur soutien et leurs conseils précieux qui ont permis à ces travaux de devenir ce qu’ils sont aujourd’hui.

Enfin, je tiens à remercier ma famille pour tout ce qu’elle m’apporte au quotidien d’attention, de soutien et d’amour.

Abstract

Nowadays, business processes allow more automation of tasks and complex interconnections within the same system and across different systems, which is particularly facilitated by the emergence of Web services. In this context, the need to ensure compliance to different sources (regulations, internal commitments) is more necessary than ever. However, the task of specifying and checking compliance at runtime becomes particularly challenging.

In this thesis, our goal is twofold: monitoring and security in the context of Service Oriented Architecture (SOA). First, we propose a view-based monitoring approach and a framework that target monitoring of business process compliance at runtime, including internal business process monitoring, business protocol monitoring, services choreography monitoring, and monitoring over a specific view of a business process or services choreography.

Our monitoring framework aims to offer an easy way to specify properties to be monitored and to facilitate its integration with SOA based environments. Thus, we have developed a new monitoring language called BPath, which is an XPath-based language that offers among others, the ability to express and to check temporal and hybrid logic properties at runtime, making the execution of business processes visible by expressing and evaluating statistical indicators, in order to detect any compliance violation at runtime.

A specific compliance monitoring concern in SOA based environment is security, which is also an important aspect for companies willing to give access to some of their resources over the Web. Thus, we propose a domain specific language (DSL) based architecture for ensuring security in SOA environments. We particularly focus on access control by proposing a graphical language to facilitate the specification and generation of access control policies. It is composed of a low level DSL that allows a security expert to create a set of access control patterns and a set of custom transformation modules in order to generate access control policies in some desired language. In addition, it includes a high level DSL that allows a business expert to specify access control policies in abstract way by reusing access control security patterns, without worrying about technical aspects, such as targeted platforms and languages.

Our approaches are implemented and integrated within a complete end to end compliance framework developed within the COMPAS project[1].

Keywords: Service oriented architecture, Compliance, monitoring, XPath, temporal logic, business process, business protocol, services choreography, access control, domain specific language, XACML.

[1] COMPAS (http://www.compas-ict.eu/), which is a Specific Targeted Research Project (STREP), funded by the European commission under the 7th Framework Programme.

Résumé

De nos jours, les processus métiers permettent une automatisation croissante des tâches et des interconnexions complexes au sein du même système et entre différents systèmes, ce qui est particulièrement facilité par l'émergence des services Web. Dans ce contexte, la nécessité d'assurer la conformité à différentes sources (réglementations, politiques internes) est plus que jamais nécessaire. Toutefois, les tâches de spécification et de vérification de la conformité pendant l’exécution deviennent particulièrement intéressantes.

Dans cette thèse, on s’intéresse à deux aspects, le monitoring et la sécurité dans le contexte de l’Architecture Orienté Service (SOA). Ainsi, nous proposons une approche fondée sur le concept de vue et une plateforme qui vise le monitoring de la conformité des processus métiers pendant leur exécution, incluant : le monitoring de l’activité interne des processus métiers, le monitoring d’un protocole métier, le monitoring d’une chorégraphie de services, et également le monitoring en se basant sur une vue particulière d’un processus métier ou d’une chorégraphie de services.

Notre plateforme de monitoring est destinée à offrir un moyen pour spécifier les propriétés à vérifier dans un environnement SOA. Ainsi, nous avons développé un langage de monitoring appelé BPath, qui est un langage basé sur XPath, qui offre entre autres, la possibilité de spécifier et de vérifier des propriétés de la logique temporelle linéaire et hybride, des requêtes visant à évaluer des indicateurs statistiques sur l’exécution d’un processus métier, ceci dans le but de détecter toute violation des règles de conformité pendant l’exécution.

Une des préoccupations spécifiques du monitoring de la conformité pour les environnements basés sur SOA est la sécurité, qui constitue un aspect important pour les entreprises, en raison de l’ouverture que la technologie Web leur impose. Ainsi, nous proposons une architecture fondée sur des langages dédiés (DSL) pour assurer la sécurité dans un environnement SOA. Nous nous sommes particulièrement focalisé sur les contrôles d'accès, où nous avons développé une DSL graphique pour faciliter la spécification et la génération des contrôles d’accès. Celle-ci englobe une DSL de bas niveau permettant à un expert en sécurité de créer des patrons de contrôle d'accès ainsi que des modules de transformation afin de générer des politiques de contrôle d'accès dans des langages cibles, et une DSL de haut niveau qui permet à un expert métier de spécifier des politiques de contrôles d’accès de manière abstraite en réutilisant des modèles de contrôles d’accès prédéfinis, sans se soucier des aspects techniques tels que la plateforme et les langages cibles.

Nos approches sont mises en œuvre et intégrés dans une plateforme développée dans le cadre du projet Européen COMPAS[2] qui vise à assurer la conformité de bout en bout dans les environnements basés sur SOA.

Mots-clés: Architecture Orientée Services, monitoring, conformité, XPath, logique temporelle, processus métier, protocole métier, chorégraphie de services, contrôle d'accès, langages dédiés, XACML.

[2] COMPAS (http://www.compas-ict.eu/), which is a Specific Targeted Research Project (STREP) funded by the European commission under the 7th Framework Programme.

This work was conducted at the LIRIS Laboratory (Laboratoire d'InfoRmatique en Image et Systèmes d'information).

LIRIS, UFR d'Informatique, Université Claude Bernard Lyon 1. 43, boulevard du 11 Novembre 1918. 69622 Villeurbanne cedex. France.

This work was conducted under the European project COMPAS: Compliance-driven Models, Languages and Architectures for Services", which is funded by the European commission, funding reference FP7-215175

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF LISTINGS

LIST OF DEFINITIONS

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF FORMULAS

Part 1. INTRODUCTION

Outline

1.1 Context

Companies increasingly place compliance in the heart of their concerns. Compliance targets the goal of ensuring that a system complies with regulatory or legislative provisions, or similar business requirements given through outer influences. A typical example is compliance to the regulations set forth in Basel II[3], and the Sarbanes-Oxley[4], Act, which cover issues such as auditor independence, corporate governance, enhanced financial disclosure, and security. In this context, compliance is any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process. Different types of compliance sources include:

- Policies internal to an organization (business rules)
- Functional or non-functional, e.g., QoS-security policies.
- Mutually acceptable agreements, e.g., SLAs that drive a business transaction.
- Policies external to an organization.
- Public policy e.g., privacy/data protection, consumer protection.
- Laws & regulations (universally applicable)
- Sectorial regulations, e.g., transportation & delivery.

This work is conducted under the European project COMPAS (http://www.compas-ict.eu/), which is a Specific Targeted Research Project (STREP) funded by the European commission under the 7th Framework Programme. The project has a budget of 3.920.000 € and started in February 2008 with a duration of 36 months. Furthermore, COMPAS is a NESSI Project and plans standardization of some parts of its contributions. Project partners are: Technische Universität Wien (Austria), Stichting Centrum voor Wiskunde en Informatica (the Netherlands), Université Claude Bernard Lyon 1 (France), Universität Stuttgart (Germany), Stichting Katholieke Universiteit Brabant (the Netherlands), Universita degli Studi di Trento (Italy), Telcordia Poland (Poland), Thales Theresis (France), PricewaterhouseCoopers Accountants N.V. (the Netherlands).

[3] http://www.basel-ii-risk.com/

[4] http://www.soxlaw.com/

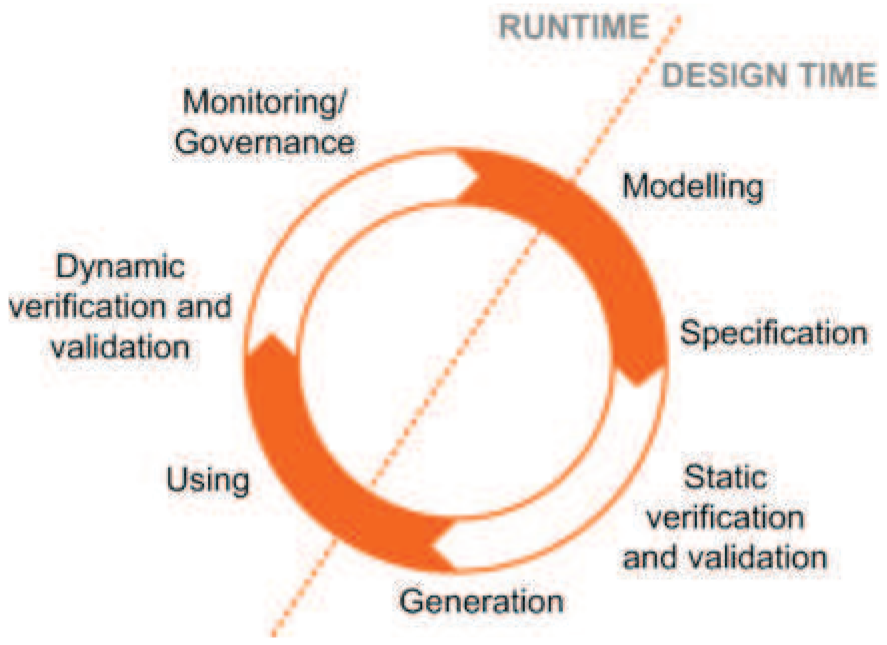


Figure 1: The COMPAS lifecycle

The COMPAS project targets the design and implementation of novel models, languages, and an architectural framework to ensure dynamic and on-going compliance of software services to business regulations and stated user service-requirements. Our research efforts in this dissertation are mainly concerned with monitoring of business process compliance, which is a part of the end-to-end business compliance software framework proposed by COMPAS that aims to capture the entire compliance lifecycle (Figure 1), starting from the modeling to monitoring and governance of business processes, this framework aims to provide the following advantages:

- Reducing the development complexity of composing services and processes with compliance concerns.
- Supporting reuse of services and processes with compliance concerns.
- Supporting changeability and extensibility of services and processes with compliance concerns.
- Developing expressive languages and models to specify compliance concerns.
- Verification and validation of services and processes with compliance concerns at design time and runtime.
- Governance of services and processes with compliance concerns.

1.2 Research issues

Since the verification system of Hoare [Hoa69] to model checking, which earned Edmund Clarke, Allen Emerson, and Joseph Sifakis the prestigious Turing Award in 2007 [Tur07], software verification continues to be an active research area. Despite the success of methods developed for design time verification, which help to discover errors, previously very difficult to discover by human analysis, it is never sure that a software system successfully checked at design time will display the expected behavior during its execution. Therefore, it is necessary to bring verification to the execution phase by continuously observing and checking the correct behavior of a system during runtime.

The advent of Web services has made a considerable progress in the way applications are developed and used, leading to the opening of new borders for information systems, with more automation of tasks, complex and multiple interconnection scenarios between applications within the same system and across different systems. In this context, the task of monitoring SOA based systems at runtime becomes particularly challenging. The need to monitor SOA at runtime has inspired a large number of research projects, both academic and industrial. The differences between these research proposals are manifold, and none of the approaches can cover all concerns of monitoring SOA based systems. But at the end, these approaches address either functional requirements or nonfunctional requirements, or both of them. While monitoring functional requirements focus on observing and checking whether a system produces the expected function in the expected behavior, monitoring nonfunctional requirements consists in observing a system and evaluate its behavior according to a predefined criteria, such as availability, performance, and security.

This dissertation focuses on the development of a business process compliance monitoring system, including the capability of checking compliance of business process execution at runtime and deriving statistical indicators regarding their execution. We also address a specific security goal, namely access control. Among the various issues that can be addressed by research approaches, two of them have guided our efforts:

Usability of the specification language

There is a need to have an expressive language to be able to express different kinds of properties, but the usability of the monitoring language should be a central point to any verification system, in order to guarantee its use. For example, it should provide some facilities for writing specifications.

Integration with SOA environment

In the framework of COMPAS project, we have tested several verification systems, both at design time and runtime. We found that having a powerful verification system is not enough, if it is unable to integrate with SOA environments, and to be in synergy with different technologies available with SOA. There is a need to leverage existing technologies and provide a solution that minimizes the changes required to SOA environments, which is certainly an important criterion towards its adoption by users and companies.

1.3 Contributions

Our work includes two main contributions, each contribution displays several features. Our first contributions is concerned with the design of a view-based monitoring approach, which allows monitoring of business processes and services choreography from different perspectives, each perspective may correspond to a particular set of concerns or related to some business process stakeholders. Thus, we have developed three monitoring systems supporting internal business process monitoring, by observing internal activities of the business process, capturing internal events, and evaluating monitoring properties over a single process instance execution or a set of process instances, by checking their compliance at runtime and reporting information about their execution. We have developed a business protocol monitoring system that is able to capture exchanged messages between a business process and its partners, and to perform monitoring over a business protocol view. Also, we have developed a choreography monitoring system, which observes exchanged messages between a set of Web services, performs verification and statistical measurement.

These monitoring systems are based on our monitoring language BPath, which offers a single expressive language based on XPath for both checking behavioral and statistical properties. BPath aims to be an expressive monitoring language by offering the possibility to use hybrid and linear temporal logic for expressing monitoring properties, and to be an easy to learn language, by the fact of being based on a widely known language in SOA, which is XPath. In addition, BPath offers the ability to express properties over various abstractions (views of a business process), which makes the expression of monitoring properties more easy and enhances their evaluation performance. Regarding the integration, BPath can be easily integrated with any SOA environment that gives support for XML processing. Also, we have proposed an algorithm and developed a tool for extracting business protocols from business processes implemented in Business Process Execution Language (BPEL). The extracted business protocol is used by our monitoring system as a particular view of a business process. The resulting tool can be reused by any approach for other purposes provided that the underlying model is a business protocol.

Security is an extremely important aspect for SOA based systems due to the open environment in which SOA systems interactions occur. Our second contribution is the design and development of an access control Domain Specific Language (DSL) providing a graphical language to express access control policies in an abstract way and offering a clear view of the designed access control policies. Our access control DSL comes in the same spirit of BPath language, by trying to provide an easy to use language that fit different abstraction levels existing in a company's organization. Our DSL is composed of two levels of DSLs, allowing to separate between abstract and technical aspects of access control: (1) A low level DSL allowing a security expert to extend a set of predefined access control patterns and custom transformation modules in order to generate access control policies in some desired language, and (2) high level DSL allowing a business expert to specify security requirements in an abstract way by reusing a set of predefined security patterns, without worrying about technical aspects, such as the targeted platform and language. We have implemented a DSL tool for specifying and generating access control policies and made a choice on an existing SOA language for encoding the generated access control policies, which is XACML, in order to facilitate the integration of our tool with SOA environments. Also, we have developed a framework to enforce XACML access control policies at runtime.

The outcomes of our scientific tasks include a number of publications at international conferences and workshops, which are:

- S. Sebahi and M.-S. Hacid, "Business Process Monitoring with BPath," in On the Move to Meaningful Internet Systems: OTM 2010, vol. 6426, R. Meersman, T. Dillon, and P. Herrero, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 446-453.

- S. Sebahi and M.-S. Hacid, “Business Protocol Monitoring,” the SERVICE COMPUTATION 2010, The Second International Conferences on Advanced Service Computing, 2010, pp. 62-67.
- S. Sebahi et M.-S. Hacid, “Business Process Compliance Monitoring,” 6th French-Japanese Workshop, ISIP, 2010, Lyon, France.
- S. Sebahi, S. Benbernou, M.-S Hacid, “An Access Control Domain Specific Language Tool,” The first international Workshop on business process compliance, organized by the COMPAS project, 2011, Warsaw, Poland.
- F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, and M. Hacid, “Business Compliance Governance in Service-Oriented Architectures,” in Advanced Information Networking and Applications, International Conference on Advanced Information Networking and Applications, Los Alamitos, CA, USA, 2009, vol. 0, pp. 113-120.
- M. A. Baazizi, S. Sebahi, M.-S. Hacid, S. Benbernou, and M. Papazoglou, “Monitoring Web Services: A Database Approach,” in Towards a Service-Based Internet, vol. 5377, P. Mähönen, K. Pohl, and T. Priol, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 98-109.

Also, we have contributed, as co-authors, to a set of deliverables in the COMPAS project:

- D. Schumm, S. Strauch, S. Sebahi, Supporting infrastructure, process engine, process artefact repository, process generation tool, 2010.
- S. Sebahi, D. Schumm, S. Strauch, Classification and specification of reusable process artefacts, 2010.
- A. Elgammal, O.turetken, S. Sebahi, Reasoning mechanisms to support the identification and the analysis of problems associated with user requests, 2009.
- S. Sebahi, D. Schumm, S. Strauch: State-of-the-art report on the existing approaches to improving reusability of processes and service compositions, 2009.
- F. Casati, F. Daniel, S. Sebahi, M. Hacid: State of art in the field of Adaptive Service Composition Monitoring and Management, 2008.

Finally, during this project, I was involved in the mobility programme. I made several research stays with our partners. The objectives were to develop a common research view regarding compliance for SOA, to prepare the deliverables, and to design the

COMPAS framework. We particularly mention the following collaborations that required mobility from our side:

- The Access control DSL tool was the result of our collaboration with security experts from Thales Theresis (France), This tool has been presented to Banque de France, and has received their interests toward its adoption.
- Used Business processes to demonstrate our approaches were developed together with our COMPAS partners, especially Telcordia (Poland) and Thales Theresis (France).
- Integration of our Securitiy DSL with a Compliance Rule Manager. This work was a collaboration work with our partner from ERISS laboratory (Netherlands).
- Integrating of our monitoring system within the COMPAS compliance runtime framework was done during an intership chez DISI Laboratory (Trento).

1.4 Outline

This manuscript is organized as follows. In Chapter 2, we provide a brief introduction to software verification by highlighting its need and its role, then we mention three verification axes: software testing, model checking and monitoring. Then, we present the emerging Service Oriented Computing (SOC) paradigm and provide an overview on business process monitoring in the context of SOC. In Chapter 3, we present our framework, a novel approach and tools for business process monitoring. In particular, we present the ingredients of our monitoring language BPath. In Chapter 4, we present a domain specific language for access control. Then, we conclude and anticipate future work in Chapter 5.

Part 2. Background On Soc And Monitoring

Outline

In this chapter, we introduce software verification by discussing three approaches, namely software testing, model checking, and software monitoring. Then we will focus on monitoring in the context of service oriented computing.

2.1 Introduction to software verification

Due to internal and external factors, software systems are continually changing and evolving, either by the development of new components, upgrading of existing ones, or by internal reorganization of all or some parts of these systems. The evolution can be induced according to multiple interconnection scenarios, with possible integration of external services or exposure of some services for use by external systems, increasing the risk of arrival of unexpected errors.

Human verification of software is no longer sufficient and almost impossible due to the inherent complexity of software systems to be checked. Thus, automatic verification becomes necessary. A need that grows as far as the consequences resulting from an expected behavior of a system has an important impact on the system itself or on its environment. For instance, unexpected errors in a subsystem of a company may have an impact on the correct execution of the whole system of the company, with possible effects on external systems that exchange services with the company. Also, in case of critical application fields e.g., chemical, nuclear fields, software verification must be strengthened, because the presence of errors can have very serious impact on the human and the environment. In addition, damages could affect the credibility of the company that sells software services. Thus, integrating software verification from the first steps of the life cycle of software development is more than necessary.

Before going further in our presentation, it is important to remember that, there is a distinction between software verification, and software validation. According to the IEEE Standard Glossary of Software Engineering Terminology [IEEE83]:

Definition 1 (Software validation)

The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Definition 2 (Software verification)

The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software verification is not a new technique. Its roots go back to 60's, where one can find the system proposed by Hoare (1969), Known as Hoare logic [Hoa69], which is one of the first formal verification systems [HKT00]. Hoare Logic allows statements in terms of pre and post conditions, and can be used to check the correctness of a transformational program (program that can be seen as a transformation from an initial state to a final state), as formulated by the following expression.

$$\mathbf{P\ \{Q\}\ R}$$

This is interpreted as "if the assertion P (*precondition*) is true before initiation of a program Q, the assertion R (*postcondition*) will be true on its completion".

However, Hoare's logic is of little use for nonterminating programs [Eme90], where there is no final state. Such programs are known as reactive systems (they are also called real-time systems [Pnu86]). In this class of programs, behavioral correctness depends not only on the produced output, but also on some temporal constraints, such as the time at which the output is produced and the time taken to produce it. Verification of such systems has motivated research works for many years and continues to be an active research area. In the following, we discuss some of them that fall in the following categories: software testing, model checking and monitoring.

2.1.1 Software testing

Software testing is probably the most used technique for detecting errors in software systems and hardware devices, which can be explained by the simplicity and the natural aspect of the idea behind this technique. In general case, tests are manually written or automatically generated and can be used to check whether a system produces the expected outputs and behavior or not. For instance, as shown in Figure 2, a finite set of inputs are injected to the system to be tested, then checking to see if the produced outputs are correct with respect to the expected outputs or not, but also in the expected behavior i.e. the order of produced outputs and the time of producing them are as expected.

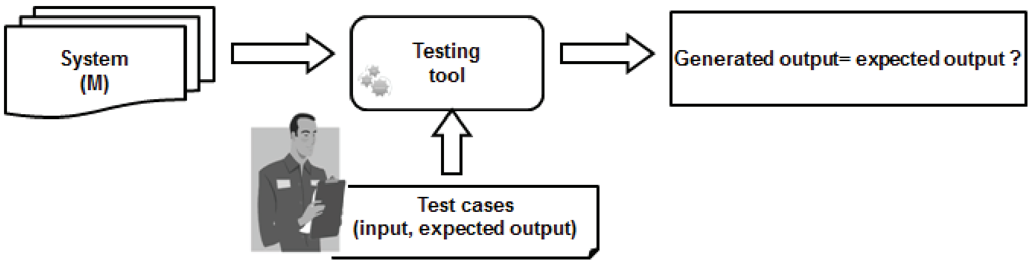


Figure 2: Testing process

Tests can be derived from requirements and specifications, design artifacts, or the source code. Different levels of testing accompany each distinct software development activity [AO08]:

- Acceptance testing: Assess software with respect to requirements.
- System testing: Assess software with respect to architectural design.
- Integration testing: Assess software with respect to subsystem design.
- Module testing: Assess software with respect to detailed design.
- Unit testing: Assess software with respect to implementation.

In the current state-of-the-practice, the various techniques for software testing can be grouped in three main categories [Che02]:

- **Functional testing**: It uses a "black box" approach in which the programmer creates test data from the program specification. Then the program is executed using these test data as input, and the corresponding program behavior and output are compared with those described in the program specification. If the program behavior or output deviates from the specification, the programmer attempts to identify the erroneous part of the program and correct it.
- **Structural testing:** It is a "white box" approach in which the programmer examines the source code of the program and then creates test data for program execution based on the percentage of the executed program's statements.
- **Code reading:** The programmer identifies major modules in the program, determines their functions, and composes these functions to determine a function for the whole program, then compares this derived function with the intended function as described by the program specification.

Several research approaches and industrial frameworks have been proposed. Table 1 lists some of existing tools for software testing.

Table 1: Software testing tools

Testing tools	**Description**
soapUI *http://www.soapui.org/*	soapUI is a free and open source cross-platform Functional Testing solution, with an easy-to-use graphical interface, and enterprise-class features, soapUI allows the easy and rapid creation and execution of automated functional, regression, compliance, and load tests. soapUI supports several standard protocols and technologies.
xUnit	Architecture for unit testing frameworks. Several tools have been developed according to this architecture, each of them targets a specific language: JUnit for java program, PHPUnit for PHP Web programming language, NUnit for dotNet, etc.
SOAPSonar *http://www.crosschecknet.com*	SOAPSonar is a software testing tool specifically designed for SOA, and supporting several features: performance testing, functional testing, compliance testing, and security testing.

Finally, it is important to mention that, the main intent of software testing is to allow finding software bugs, in order to increase the confidence level we can have on the correctness of the system. But at the end, as explained by Dijkstra in [DDH72]: software testing cannot guarantee that the system is error-free.

2.1.2 Model checking

Model checking has received a considerable attention from the formal methods community within the last decade. Model checking is a technique used to verify concurrent systems such as sequential circuit designs and communication protocols, by exhaustively exploring the state space of a finite-space description of the system involved. It describes the problem of determining whether a given system satisfies or not some properties expressed in some formal language. As shown in Figure 3, given a model M of the system to be checked, and a correctness property ϕ, model checking technique can be used to check if all computations of M satisfy ϕ. Automata based model checking techniques are able to compute a counter example that leads to the non-satisfaction of the correctness property ϕ.

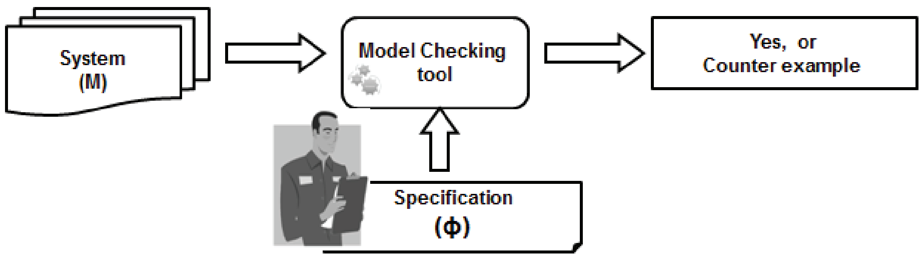


Figure 3: Model checking process

One of the widely used formalism for specifying properties that a system should satisfy is temporal logic. Temporal logic describes the ordering of events in time without explicitly introducing time. They were developed by philosophers and linguists for investigating how time is used in natural language arguments. Temporal Logic is a special type of modal logic. It provides a formal system for qualitatively describing and reasoning about how the truth values of assertions change over time [Eme90].

Lamport [Lam77] classifies temporal properties into safety and liveness properties.

- Safety property: Asserts that nothing wrong happens in the system (e.g., a car does not stay on the bridge more than 20 minutes").
- Liveness property: States that something which is considered as "good" will eventually happen (e.g., a car will cross the bridge).

Model checking has the advantage to be an automatic technique. However, model checking is mainly applicable to finite-state systems, for which all computations can exhaustively be enumerated [LS09]. Additionally, model checking has a practical limitation in case of industrial-size systems where a huge amount of time and memory are needed to explore and store the state-space of the system [Pet99].

Since the first proposals on model checking of temporal logic formulas by E. M. Clarke and E. A. Emerson [EC80] and by J. P. Queille and J. Sifakis [QS82], model checking continues to be an active research area. A plenty of tools have been developed. Table 2 shows a non-exhaustive list of existing model checking tools.

Table 2: Model checking tools

Model checking tools	**Description**
Vereofy *http://www.vereofy.de*	Component-based systems for operational correctness. It makes use of constrained automata as formal semantics for the components behavior and Reo as Coordination Language. It allows for linear and branching time model checking.
Prism *http://prismmodelchecker.org*	PRISM is a probabilistic model checker, a tool for formal modeling and analysis of systems which display random or probabilistic behavior.
mCRL2 *http://www.mcrl2.org*	Formal specification language with an associated toolset that can be used for modeling, validation and verification of concurrent systems and protocols.
Uppaal *http://www.uppaal.org*	Integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

2.1.3 Software monitoring

While the design time verification (model checking, tests…) is supposed to ensure the execution correctness, in practice a software system, successfully tested and checked, might not lead to the expected compliant outcomes and behavior during its execution, this is for several reasons.

It is possible that the used verification technique has some theoretical limits as it is the case with automata based model checking, which suffers from the well-known state explosion problem, leading to the use of abstract model with simplified assumptions about the considered system: in order to reduce the number of states in the model, some events and actions are not taken into account. This means that the considered model will not reflect the system as it is in the real world. In case of testing techniques, they only help to discover errors, by proving that a system is not working as expected for a given set of inputs, but cannot guarantee that a system is error-free. In addition, there are some practical limits: the use of different verification tools in the same time (to cover different properties that cannot be covered only by using one verification technique) could increase the probability of errors, stemming from the poor control by the users of various and complex verification tools. Also, in some cases, assumptions about behavior of a system cannot be verified before it is put in operation. Human factors, but also hardware failures may cause deviations from the expected behaviors, without forgetting that deviations do not always correspond to undesirable behavior: sometimes users deviate for good reasons-for instance in order to add more flexibility to a system, users may explicitly break some rules during the execution.

Consequently, the real behavior of a system at runtime may be different from what is supposed to be at design time. Thus, it is necessary to spread verification to the execution phase, by continuously observing and checking the correct behavior of a system during runtime. This activity is known as runtime verification, which is a cornerstone activity in a software monitoring system that may ensure, among others: analysis and recovery from detected faults by providing additional defense against catastrophic failure. Even if the system meets the specified monitoring properties at runtime, monitoring system may provide information that can improve the performance and reliability of the monitored system, and helping manager to have a visibility on the system by offering a clear view on how components of the system work within their operational environments, and enabling management decision with the ability to perform control actions on the system.

However, it is important to make distinction between design time verification and runtime verification. For instance, while model checking deals with the verification of all computation paths of a system i.e. all possible executions of a system will be checked, runtime verification deals only with observed executions as they are generated by a system [DGR04]. Although it may seem a weakness, it is an advantage. The reason is that by considering a single execution, the monitor which usually runs in parallel with the system to be checked, will consume much less resources and perform checking relatively faster. Remember that runtime verification is applicable to black box systems for which no system model is available. In model checking, however, a suitable model of the system to be checked must be available prior to the start of the verification [LS09].

2.2 Service Oriented Computing

2.2.1 Introduction

Information systems always had the purpose to meet the functional requirements of the business. But adapting technology to business requirements has long been a difficult task, following the previous technologies such as CORBA, DCOM, or RMI, that were more programmatic approaches, difficult to link in a sustainable manner with the company's business reality.

To meet this challenge, Service Oriented Computing (SOC), an emerging computing paradigm proposes a new conceptual framework to reorganize information systems around its processes and through its business view, on the basis of the notion of service, as the basic constructs to support the development of rapid and easy composition of distributed applications, even in heterogeneous environments. SOC promotes the idea of assembling application components into a network of services that can be loosely coupled to create flexible, dynamic business processes and agile applications that span organizations and computing platforms [PTDL07].

Built on the basis of loosely coupled business services, applications can be easily integrated to different processes and can be collected, made according to changes in business strategy in a flexible way, but also will allow to develop and to expose the company expertise to any customer, or to interoperate with different partners.

Currently, the common practice for developing service-based systems is to employ the Service-Oriented Architecture (SOA) paradigm [MP09]. Thus, we propose an overview of this architecture in the next section. We mainly focus on the fundamental issues of this architecture.

2.2.2 Service Oriented Architecture

Service-Oriented Architecture (SOA) is the main architectural concept in the field of SOC. In this architecture, all functions and services are defined using a description language, and have invokable platform-independent interfaces. Each service is an endpoint of a connection, which can be used to access the service, and each interaction is relatively independent of each other interaction.

As proposed in [PTDL07], SOA can be viewed as a trihedral (Figure 4):

- **Service foundations at the bottom:** The service foundations plane consists in a service oriented middleware backbone that realizes the runtime SOA infrastructure. This infrastructure connects heterogeneous components and systems and provides multiple-channel access to services over various networks including the Internet. It lets application developers define basic service functionality in terms of the description, publishing, finding, and binding of services.
- **Service composition in the middle:** The service composition plane encompasses roles and functionality for aggregating multiple services into a single composite service. Resulting composite services can be used as basic services in further service compositions or offered as complete applications and solutions to service clients
- **Service management and monitoring on top**: Service management spans a range of activities, from installation and configuration to collecting metrics and tuning, to ensure responsive service execution. Service monitoring involves monitoring events or information produced by the services and processes; monitoring instances of business processes, etc.

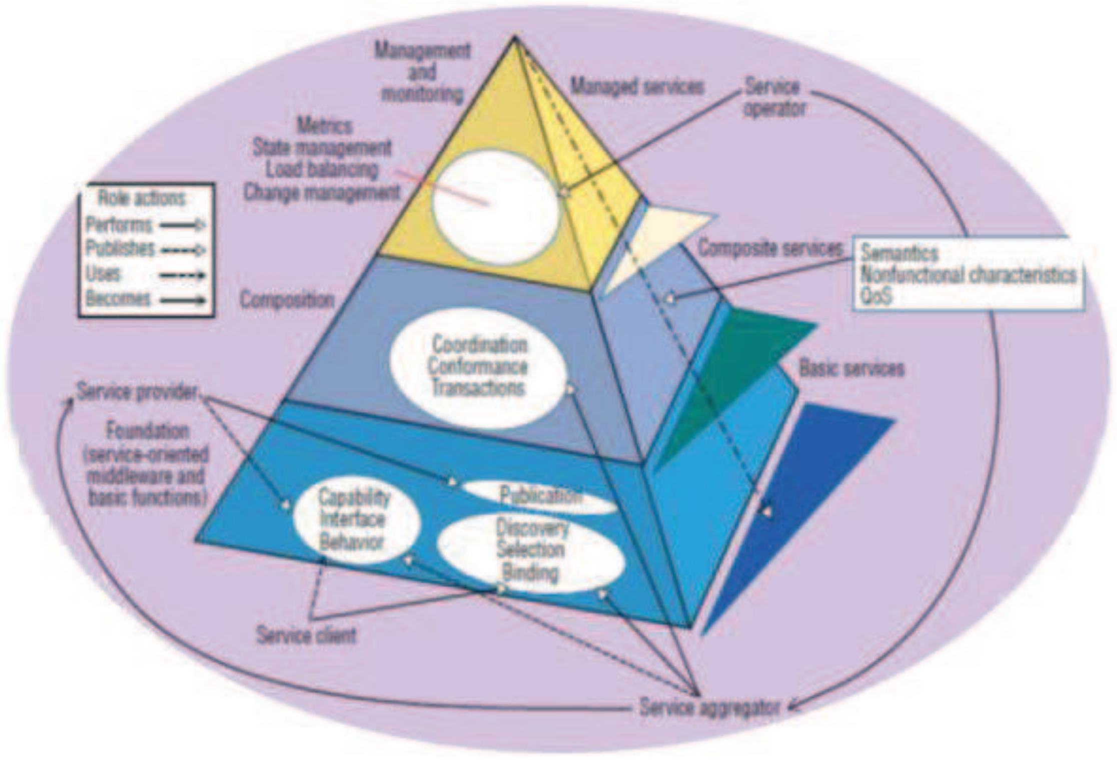


Figure 4: Extended SOA (taken from [PTDL07])

The need of exhibition of services across the Web have derived the standardization of service interfaces, and given rise to the concept of Web services.

Definition 1 (Web service)

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [W3C].

Three actors are identified in the usual Web services working scenario (Figure 5): service provider, service registry, and service consumer.

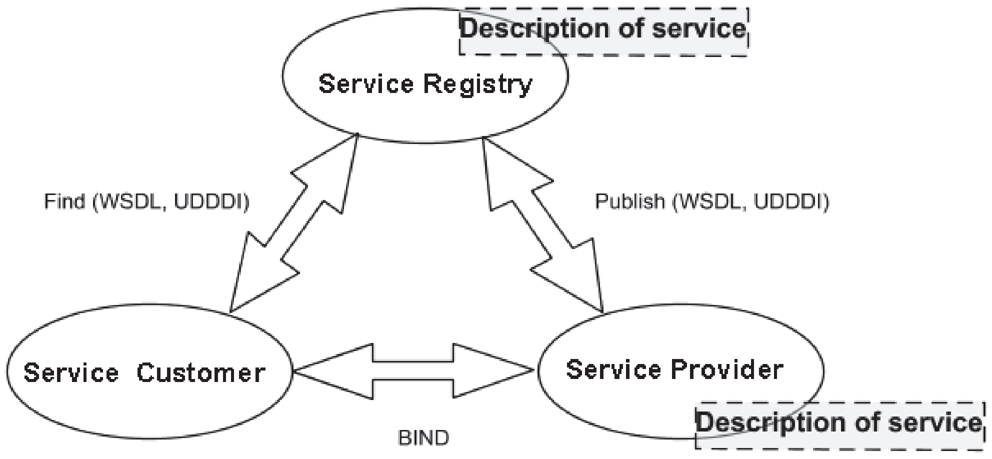


Figure 5: SOA components

A service provider describes its service by publishing a description document and storing it in a service registry. A service customer submits a search query to the service registry and therefore receives a service description. Then eventually, the service customer can start the invocation of a service based on its description.

Three essential properties characterize Web services:

- The self-description: Web services architecture defines a standard way to describe a Web service.
- The modularity: Web services are based on a multi-tiered architecture.
- Loosely coupling: the only mission that is imposed to each Web service is to recognize the self-describing text messages from other services.

XML Technologies

XML technology play fundamental role in several SOA specifications that have been developed around Web services. XML Technologies include a series of languages, and specifications:

- Extensible Markup Language (XML) [XML]: simple, very flexible text format, which play an important role in the exchange of a wide variety of data on the Web and elsewhere.
- XML Schema [STM+09]: provides a means for defining the structure, content and semantics of XML documents.

- eXtensible Stylesheet Language Transformations (XSLT) [Cla01]: Used to transform an XML document into another XML document or another type of document that may be recognized by a browser, or by any other application.
- XML Path Language (XPath) [CD99]: Defines expressions for addressing parts of an XML document.
- XML Query Language (XQuery) [CSR+09]: A query language for XML to extract data, similar to the role of SQL for databases, or SPARQL for the Semantic Web.

Several SOA specifications have been developed for Web services, the following basic specifications originally defined the Web Services space.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) [SOAP] is a protocol for communication between applications based on XML, intended for exchanging structured information in a decentralized, distributed environment.

Web Services Description Language

Web Services Description Language (WSDL) [WSDL] is an XML format for describing network services as a set of endpoints operating on messages, and defines the messages communicated between the endpoints. To describe the content of messages, WSDL uses XML schema.

Universal Description, Discovery, and Integration protocol

The universal description, discovery, and Integration (UDDI) [UDDI] defines a protocol to query and update a repository of information on Web services. UDDI aims to build a repository that is for Web services and business what the domain names and IP addresses are for web sites.

Together, these SOA specifications allow applications to find each other and interact following a loosely coupled, platform independent model, but systems integration requires more than the ability to conduct simple interactions by using standard protocols. The full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model [AAB+05].

Business Process Execution Language for Web services

A business process is a set of activities that are performed to achieve the business objectives of an enterprise. Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short) [AAB+05] defines a model and an XML grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a partner link.

Business Process Model and Notation

BPEL is an executable language, including technical activities that are hard to understand by the business analysts and managers. That is why it is necessary to have a high level notation to design the business process of a company. Business Process Model and Notation (BPMN) [Whi04] is an agreement between multiple modeling tool vendors, who had their own notations, to use a single notation for the benefit of end-user understanding and training. BPMN provides a mechanism to generate an executable Business Process (BPEL) from the business level notation.

Web Services Choreography Description Language

The Web Services Choreography Description Language (WS-CDL) [BKF+05] is an XML based language that describes peer-to-peer collaborations of Web service participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal.

2.2.3 Business process compliance monitoring

A Business processes is a reactive system that interacts with its environment in continuous manner. Its correct behavior does not depend only on its correct outputs, but also on different constraints such as: temporal constraints, correct order of the produced outputs, constraints imposed by internal policies, business contract reflecting the commitment with external partners, and any rule imposed by existing regulation.

Typically, a set of formal compliance rules are automatically extracted from compliance sources or manually specified by a compliance expert (internalization step in Figure 6) and sent as input to the monitoring system in order to be able to monitor compliance of a business process.

When a business process is executed, the monitoring system tracks service activities, captures events during business execution (according to the events specified in the design phase), and accommodates visibility into various metrics and statistics. If parts of the business execution are automated, e.g., with the help from customer relationship management (CRM) or workflow management (WFM) systems, events will be generated by the process engine, in charge of running the process specification. The parts which are not automated may, however, generate user-driven events such as notifications or emails, which can be captured to derive the status of the business execution. Captured events can be used to feed a monitoring dashboard with up-to-date information for human consumption or to log process execution data for data warehousing for later analysis.

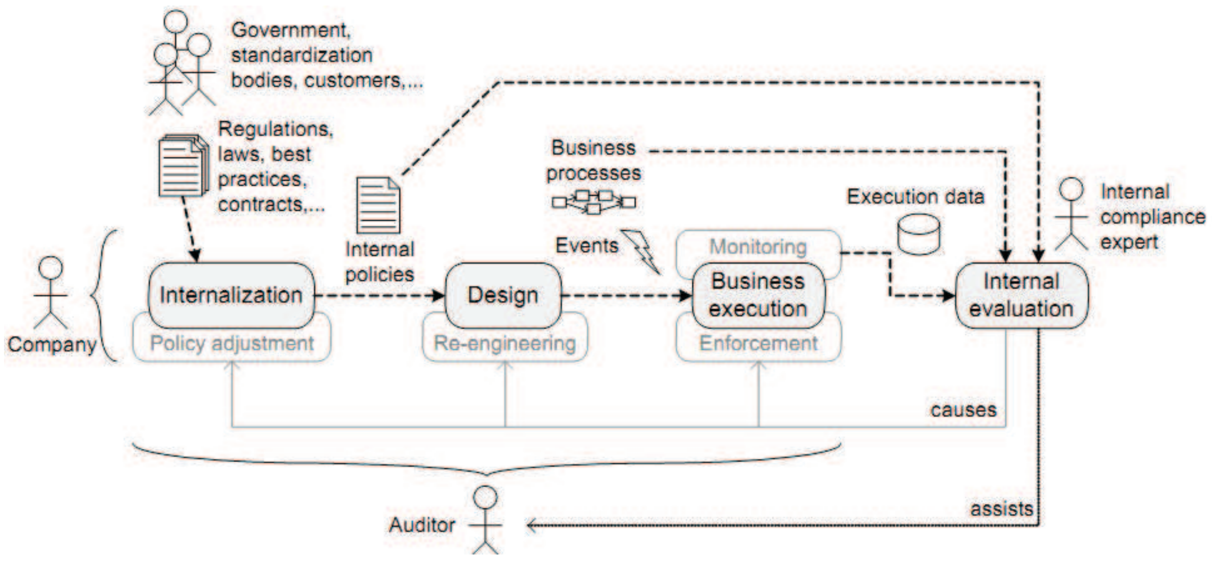


Figure 6: Compliance monitoring.

In case non-compliance is detected, the typical countermeasures that can be enacted in order to ensure compliance or mitigate possible negative effects, the following actions may be performed [DCA+09]:

- Enforcement of automatically controllable actions: In some cases, in response to raised events it might be necessary to invoke some actions on the business process like enforce a compensation or corrective action.
- Re-engineering of the design for compliance: When a business process has been designed in a non-compliant fashion in the first place; and the enforcement actions fails to bring the process to its expected behavior. The redesign of the initial design of the business process should be considered.
- Adjustment of the internal policies: non-compliance problems might also be due to the wrong interpretation and, hence, the wrong internalization of compliance regulations, or simply because regulations change over time, or contains inconsistency due to some contradiction in the specification. In such cases, the policies need to be adjusted, and the business execution might require a re-engineering.

2.3 Summary

In this chapter, we briefly introduced software verification by presenting three main approaches, namely software testing, model checking and software monitoring. Then, we have expanded our presentation to compliance monitoring in the context of SOC, by introducing some notions on Web services and theirs underlying technologies. We finally mentioned business process compliance monitoring.

We can summarize this chapter by saying that software verification is a relatively established discipline, but monitoring is gaining more and more importance, especially with the advent of Web services.

Part 3. A View-Based Business Process Compliance Monitoring Approach

Outline

In this chapter, we will present our business process monitoring approach that allows monitoring over different views of a business process and services choreography. We will present the underlying new monitoring language that aims to be expressive and easy to use in order to specify monitoring properties.

3.1 Introduction

A business process is a set of related, structured activities that are performed to achieve the business objectives of a company, each activity may belong to some part addressing a particular concern. In fact, a business process model may encompass various tangled concerns, such as control flow, data processing, service and process invocations, fault handling, event handling, human interactions, transactions, etc. The entanglement of these concerns increases the complexity of process model, as the number of involved services and processes grow. Consequently, there is a difficulty for users to understand the process model in order to perform needed tasks of process verification and monitoring. To address this issue, we propose a view-based monitoring approach that is able to perform compliance monitoring of business processes, services choreography, or on their related views.

Definition 3 (Business process (Choreography) view)

We call a business process (Choreography) view, a representation of a part or the whole business process (Choreography) from the perspective of a related set of concerns that the system stakeholders are interested in.

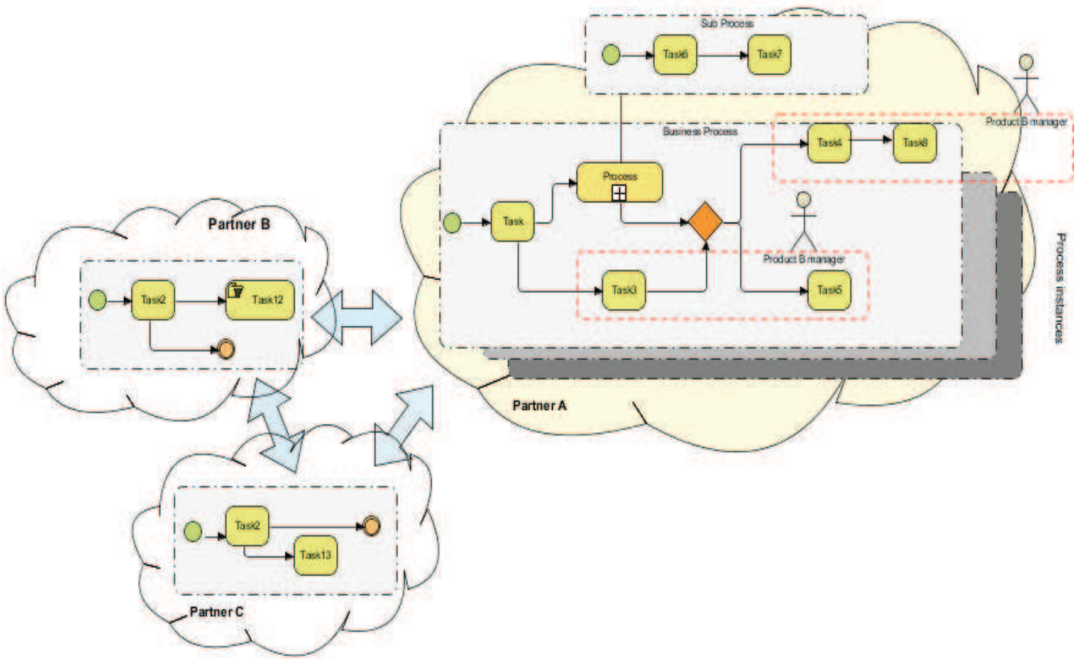


Figure 7: Views of a Business Process and a choreography

As shown in Figure 7, examples of such views may include: views focusing on activities of a business process involved in the creation of some product, activities exchanging messages with some partners, a subset of exchanged messages in services choreography, or those covering some nonfunctional aspects such as activities devoted to ensure security.

Existing industrial tools for creating business processes do not or partially separate between business process concerns. We used the View-based Modeling Framework (VbMF) [Tra09], which was developed within the COMPAS project. VbMF provides three major advantages: first, it captures different perspectives of a business process model in separate, (semi-)formalized views; second, it separates different abstraction levels in business process architecture; third, it is an extensible model-driven approach to integrate the different view models and abstraction levels. Moreover, VbMF automatically generates platform-specific descriptions and an executable code of business processes in WSDL and BPEL.

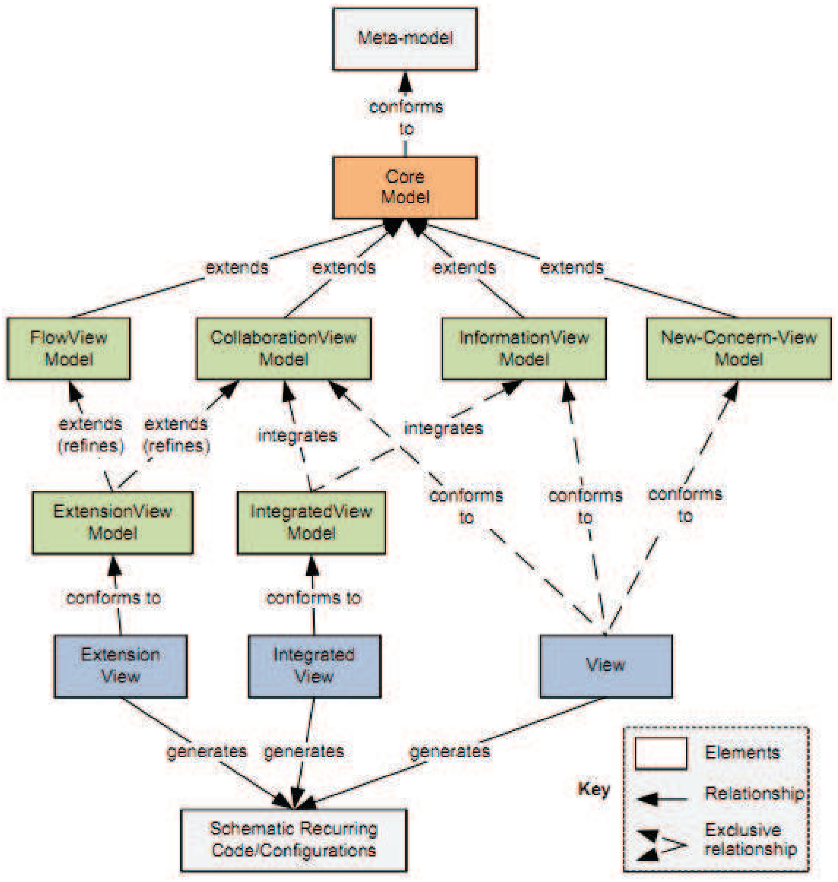


Figure 8: Fundamental concepts of the VbMF framework [Tra09]

Figure 8 presents the formalizations of basic process concerns such as the control flow, service invocations, and data processing, in terms of the *FlowView*, *CollaborationView*, and *InformationView* model, respectively. These view models are built up around a core model. The core model is intentionally developed for conceptually representing the essence of a business process. That is, the core model covers three distinct concepts: The process, the relationships between the process and the environment, i.e., the services, and the internal representation of the process, i.e., the process views. Process concerns described by the view models merely relate to these concepts in the sense that each concern involves either the process's interior or exterior, or both. A new view model can be added to VbMF through the extension of the core model.

From the monitoring perspective, in order to monitor a process view, we proceed by simulating the view execution in correspondence with the execution of the represented business process. This simulation will generate a set of metadata about the view execution that are useful for monitoring. The detailed description of these view execution metadata will be presented further. Our approach supports monitoring of business processes at different levels, which includes:

- Internal business process monitoring, by observing internal activities of a business process, capturing internal events, by focusing either on process class or on process instance:
    - Instances of a business process: deals with the execution of a single instance of BPEL process.
    - Classes of business process: checks the execution over a set of process instances, and report aggregated information about all the instances of a business process.
- Business protocol monitoring: At this level, the focus is on the external behavior of the business process, as it can be seen from an external observer, by tracking exchanged messages from, and to a business process.
- Services choreography monitoring: By tracking and checking messages exchanged between involved partners in the choreography.

Figure 9 describes the working of our approach, which can be summarized in the following steps:

- Monitoring properties are specified at design time or runtime, possibly by making reference to some process view (1)
- Business process or a choreography of a process to be monitored are put on run (2),
- At runtime, execution events are collected from the messaging framework. They may be filtered, according to a predefined set of rules (3),
- Execution events are stored into an execution data (5)
- A special event (state event) computing the state change of a business process or choreography are generated and stored in a special event log called state trace (step 4).
- State events (generated in step 4) may depend on a special abstraction (process view) (1),
- Process view may also generate some high level events (relevant event from the business point of view), that can be used to control the monitoring task.
- Monitoring (6) is performed according to the specified monitoring properties in (1), information stored in the state trace (4) and the execution data (5).
- Monitoring results are shown in a dashboard component (7). Some actions can be done on the state of the business process, such as changing the value of the variables defined within a business process.

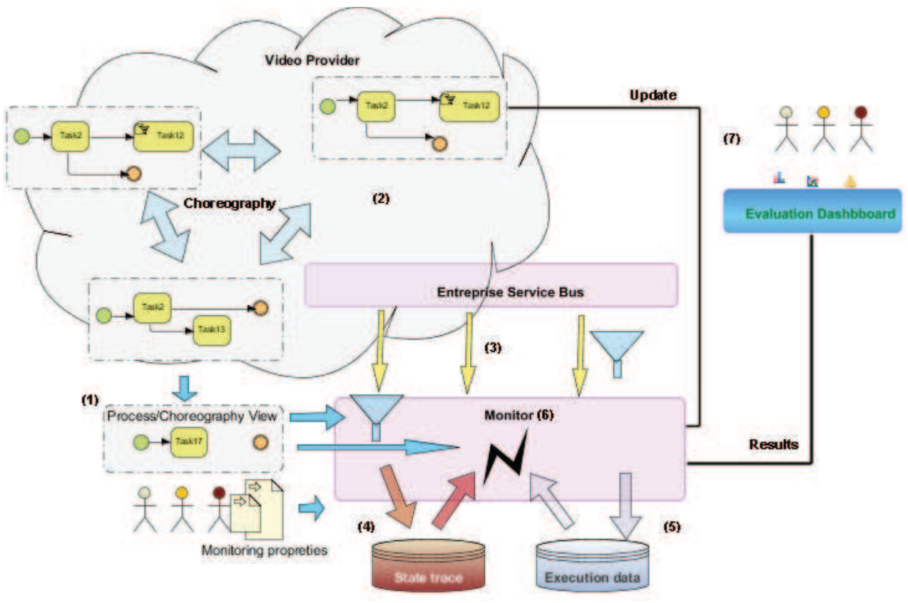


Figure 9: View based monitoring approach

3.2 BPath monitoring language

3.2.1 Business process execution

An execution of a business process can be captured as a sequence of states, independently of the fact that it may contain different process instances, or parallel activities inside the same process instance. Given an initial state, a business process instance will go through a series of intermediate states, until it reaches a final state. The transition from a given state to another state may be caused by receiving or sending a message, the assignment of instance variables, and so on. Each state represents an instantaneous description of the business process instance execution in terms of variables and their corresponding values at a given time. The value of a given variable belongs to its associated domain, such as integer, string, and so on. In practice, the types of variables are specified within WSDL documents through an XML schema. The basic instruction leading to the change of a state is variable assignment.

Definition 2 (Variable assignation)

A function that assigns a value to a given variable of the business process instance at a given time. We assume that several variable assignments can occur at the same time. Let us consider (id, t)$\in N\times N$, N being the set of natural numbers, and x $\in X$, where X is the set of variables of a process, and let D be the definition domain of those variables.

$$A_{id,x}(t) : N\times X\times N \rightarrow D$$
$$A_{id,x}(t) \in D \qquad (1)$$

Definition 3 (Instance State)

A function that assigns a value to each variable within the business process instance at a given time.

$$S: N \times N \times X \rightarrow D$$

$$S_{id,t}(x) = A_{id,x}(t) \qquad (2)$$

It is important to make a distinction between what we can consider as a state of a business process instance, which is an abstraction that captures some interesting issues, and ignores other less important ones from the point of view of what we want to do with this abstraction, and what is the state of the business process instance in reality. The transition from a state to another state in a business process instance leads to a sequence of states, which is known as business process instance execution trace.

Definition 4 (Business process instance execution trace)

A time ordered sequence of instance states that occurs from the execution of an instance of a business process starting from a particular input state.

When a state of process instance changes, the process state will change.

Definition 5 (Process state)

A process state at a given timestamp t is a state composed from all instance states at the given time t.

The sequence of process states corresponding to an execution of a business process is known as business process execution trace.

Definition 6 (Business process execution trace)

A time ordered sequence of process states that occurs from the execution of a business process starting from a particular input state.

For instance, let us consider the business process shown Figure 10 (A), consisting of three assignment activities, and two variables *(x,y).* The first activity, initializes *x* to some received value, and depending on some condition, *y* will be assigned the square or the square root of ***x*.** We assume two process instances (*x1, x2* are respectively the *x* variable of the first and the second instance). Figure 10 (B, C) show possible instance execution traces, and Figure 10 (D) shows the business process execution trace, which is a possible combination of the two process instance traces of Figure 10 (B, C).

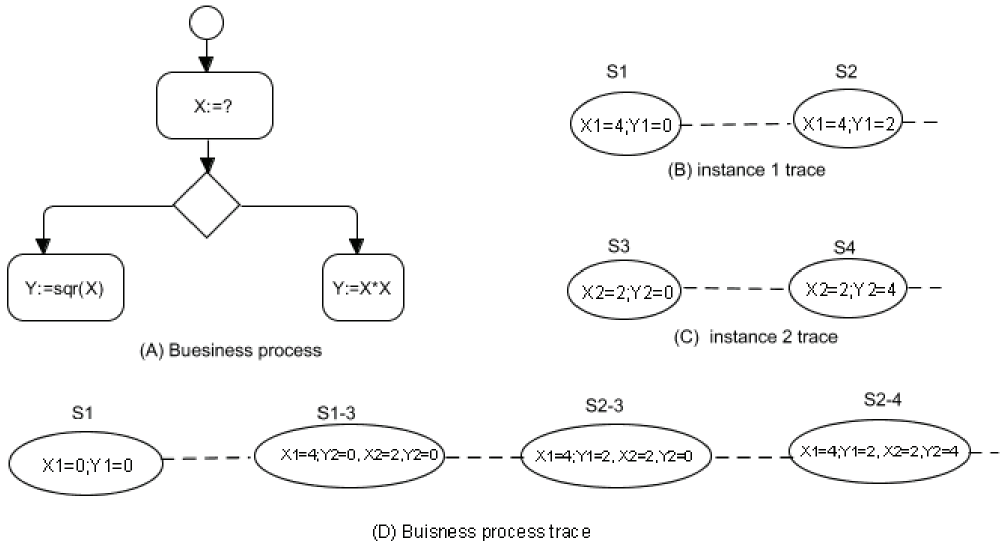


Figure 10: Instance and process execution trace

One can use a similar reasoning when a business process contains parallel activities, as it is the case in the process shown in Figure 11 (A). In this case, the two activities defined within the flow activity will both be executed, but in random order, leading to two possible execution traces.

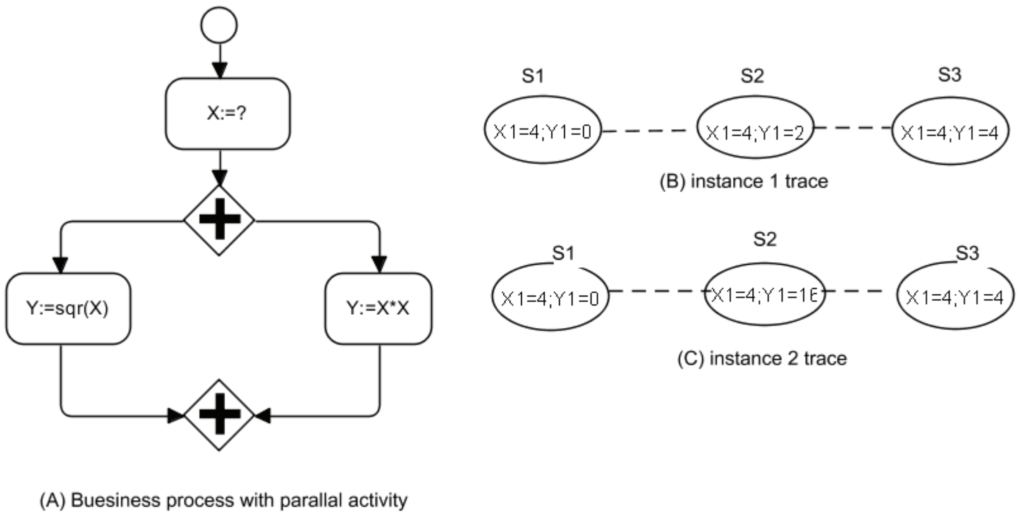


Figure 11: Process with flow activity execution trace

Execution trace will allow us to store states of the execution and to access to the history of variable values as the execution progresses. A monitoring language will allow us to query the execution state in order to evaluate some statistical indicators on the execution and to check whether the execution conforms to what is expected or not. In this perspective, we have developed a new monitoring language called BPath, which will be presented in details in the next section.

3.2.2 Introduction to BPath

BPath aims to be both a monitoring language for business process runtime verification and a query language. BPath offers, among others, the ability to use both linear temporal logic (LTL), and hybrid logic (HL) to express monitoring proprieties.

LTL is a commonly-accepted formalism for specifying properties of finite-state discrete systems [MP95] and widely used for design time verification. Temporal logic does not only allow checking the correctness of a system in terms of a relation between inputs and outputs, but also to check whether a given property is verified with respect to temporal constraints. As a simple and intuitive example, let's consider the execution trace of Figure 10 (D). By using temporal logic, we can specify that every time the value of *x* is greater than 4 ($x > 4$), the value of *y* should be greater than 2 at least once during the execution (3), or that *x* has always a positive value during the execution (4).

$$\mathbf{G}(x>4 \rightarrow \mathbf{F}(y>2)) \qquad (3)$$
$$\mathbf{G}(x>0) \qquad (4)$$

These properties use the linear temporal logic (LTL) modal operators F (future), and G (globally) which express respectively that a given formula should be true one or more times in the future, and always in the future. A temporal logic formula is evaluated by fixing some initial state. As we can see from the execution trace showed in the Figure 10 (D), both properties will be checked successfully.

LTL proposes an interesting language for software verification, both at design time as well as at runtime, as it allows checking properties over different execution states. However, in temporal logic, execution states cannot be explicitly accessed. To do this, we need to add further expressive power. This can be done by using hybrid logic. Hybrid logic makes it possible to store a reference to a given state and to come back later to this state when we are in another state in order to check some properties, or to access to a value of some variables at this referenced state.

For instance, we can check whether a variable *x* preserves its value or not during the execution, compared to its value in the first state.

$$\mathtt{S!}\ \mathbf{G}(x{=}\mathtt{S}{:}x) \tag{5}$$

This formula introduces a new expression *S!,* which stores the current state of the execution on the state variable *S*, navigates to the future states by the temporal logic operator *F*, then compares the value of *x* on the current state to the value of *x* on the state referenced by the state variable *S*.

BPath supports checking temporal and hybrid logic properties at runtime through the extension of XPath 1.0 language. There are several reasons in the use of an XPath based language for runtime monitoring instead of automata based approach similar to model checking for design-time verification:

- An XPath expression can be deployed on any XPath engine, which facilitates the integration of our monitoring system to any business process framework supporting XML technologies.
- The computational complexity of checking a document against an XPath 1.0 expression is shown to be in P-TIME [GKP03]. Therefore, XML query processors are efficient for runtime monitoring.
- XPath is not restricted on checking qualitative properties, in contrast to LTL, but also on evaluating quantitative properties, which gives us the ability to evaluate statistical indicators on the business process execution. For instance: Counting the average time of execution of a business process (instance).
- BPath is expected to be an easy to learn language because XPath is supposed to be a well-used in the area of SOA.

Using XPath 1.0 rather than using XPath 2.0 or XQuery 1.0 languages is motivated by the fact that XPath 1.0 is the basic language for XPath 2.0 and XQuery 1.0. So, any extension to XPath 1.0 will be automatically supported by XPath 2.0 and XQuery 1.0. We have:

$$\text{XPath 1.0} \subset \text{XPath 2.0} \subset \text{XQuery 1.0} \tag{6}$$

The main idea behind BPath is first to consider a part of the execution trace as an XML tree. In this tree, each node represents a possible state, and its direct following node represents the direct next state, then we use XPath to query this part of execution trace. In fact, we call this part of execution trace that will be queried by BPath a state trace. It is a collection of metadata about the execution states that do not contain execution data (variables and theirs corresponding values).

Definition 7 (State trace)

A state trace is an XML tree of nodes (states-nodes): w_1, w_2, w_3...where w_2 is the direct next following sibling of the node w_1, w_3 is the unique following sibling node of w_2, etc. Each state-node has a name $n \in LAB / n=label(w_i)$, and three or more attributes (called static attributes): processid (process identifier) $\in ATTS$, instid (instance identifier) $\in ATTS$, and a timestamp (an attribute recording a time) $\in ATTS$. LAB is a set of node names, and ATTS is a set of attribute names, which are called the set of static attributes.

```
<?xml version="1.0" encoding="utf-8" ?>
<Trace>
  <S1 processid="001" instid="001" timestamp="00001"/>
  <S1 processid="001" instid="002" timestamp="00002"/>
  <S2 processid="001" instid="001" timestamp="00003"/>
  <S1 processid="001" instid="003" timestamp="00004"/>
  <S1 processid="001" instid="004" timestamp="00005"/>
  <S2 processid="001" instid="002" timestamp="00006"/>
</Trace>
```

Listing 1: Execution trace

The fact that BPath is built on top of XPath provides a mean for navigating between state nodes. In other terms, this means that with BPath it is possible to explore process instances and theirs corresponding states.

Example 1: From the execution trace shown in Listing 1, if we want to navigate to the last "instance" state node, or to access to the second state *S2* of the process instance with id is "001", we can respectively write in BPath the following expressions, which will be evaluated from the first state node of the execution trace:

```
following-sibling::*[position()=last()]          (7)

following-sibling::S1[@instid=001]/ following-   (8)
```

```
sibling::S2[@instid=001]
```

State trace stores only a sequence of execution states; the real execution data is stored separately in what we call an execution data source.

Definition 8 (Execution data source)

An execution data source is a collection of entries: el = {processid, instid, timestamp, ls}, where ls is a list of pairs (key:value), key is a variable name and value is the value of the variable, processid and instid are respectively the process, and the instance identifiers related to the variable, and a timestamp recording the time.

A BPath expression is evaluated over a state trace, which contains only metadata about an execution, in terms of a sequence of states nodes and their static attributes. However, if we want to check the execution, we necessary need to access to execution data, i.e. the data stored in the execution data source. This is possible in BPath by resorting to dynamic attributes. In fact, BPath makes distinction between variables of a state. This done by considering four possible types of variables: static attributes, dynamic attributes, state variables and global variables.

Definition 9 (Static attribute)

A static attribute is a variable associated with a state, and concretely defined in static way within the state node in the state trace XML document.

Comparing to XML, this is what is called a node attribute. For instance: *processid*, *instanceid*, and *timestamp* in the Listing 1 are static attributes.

Example 2: To access to the static attribute *processid* of the first state node *S1*, we write:

```
Self::S1/@processid                    (9)
```

Definition 10 (Dynamic attribute)

A Dynamic attribute is a function associated with nodes. It allows to access to external data associated with a context node (the current node under evaluation). Concretely, for *monitoring purposes, we use dynamic attributes to access to values of variables of a business process.*

Example 3*:* The following BPath expression gets the value of the *variable y* of the last process state:

```
following-sibling::* [position()=last()]/@y          (10)
```

Definition 11 (Global variable)

A global variable in BPath is like an XPath variable that can change its value during the evaluation of a path expression by means of an assignment expression.

Example 4: In the following expression the global variable *$i* receives the value of the dynamic attribute *x* at the first state node.

```
self::*[$i:=@x]                          (11)
```

Definition 12 (State variable)

A state variable is a variable that stores a reference to a state node.

Example 5: The following expression shows how to use a state variable to store a reference to the current state node, in order to use it to access to a dynamic attribute *x,* and to compare it with the value of a dynamic attribute *y* in each state of the next sibling node.

```
$S! following-sibling::*[$S/@x >@y]              (12)
```

3.2.3 Core BPath syntax

Formally, a basic BPath expression is built according to the following syntax:

<u>*A formula:*</u> $\boldsymbol{\varphi}$ *::= true() | false() | (φ) | φ and φ | φ or φ| T=T| P(T…T) | not φ |* ***LTL*** *|* ***HL***

LTL*::=****X****(φ)|* ***F****(φ) |* ***G****(φ) |* ***U****(φ, φ) |* ***B****(φ) |* ***P****(φ) |* ***H****(φ)*

HL*::=$S* ***!*** *φ |$S [φ] | $x:=T,φ*

<u>*A term:*</u> ***T****::= c |π|* ***f*** *(T, T...) | $x | $S | $S/@q | π/@q*

<u>*A path:*</u> $\boldsymbol{\pi}$ *::= Axis::N | (π) | π [φ] | π / π | π '|' π*

N *::= n | **

Axis ::= self | following-sibling | preceding-sibling

where $x \in FVAR$ (a set of first-order variables), $n \in LAB$ (a set of state names), such that for each element *w* from *W* (a set of state nodes) we associate an element *n* from LAB such that: *n=label (w)*, $q \in ATTS$ (a set of unary function symbols that we call static attributes) $\cup$ *ATTD* (a set of unary function symbols that we call dynamic attributes) $\cup$ *FUN* (a set of one or more arity functions). $S \in SVAR$ (a set of state variables). $c \in CON$ (a set of first order constants).

In order to simplify the writing of some expressions, we consider that the expression *self::*/@q* can be simply written as *@q* , *not (φ)* $\vee \alpha$ can be abbreviated as $\varphi \rightarrow \alpha$, $*x:=T, true()* is simply written as $*x:=T*. By convention, state variables are written in uppercase and first order variables in lowercase.

3.2.4 Core BPath semantics

BPath implements a fragment of first order hybrid logic, which does not support the use of nominal and quantifiers. Here, we present the semantics of this fragment by using BPath syntax. For a detailed presentation of first order hybrid logic, the reader may refer to [BM02].

The semantics of this fragment is interpreted in first-order modal model *M (W, R, D, I_w),* *w* $\in$ *W*, such that *W* is a set of states (state nodes) *{w_1, w_2...}. D* is the interpretation domain. (W, R) is the modal frame.

For every w $\in$ W, (D, I_w) is an ordinary first-order model such that:

- $I_w(c) = I_{w'}(c)$, where: w, w' $\in$ W, c $\in$ CON.
- $I_w(q) \in D$, for q $\in$ ATTS $\cup$ ATTD $\cup$ FUN.
- $I_w(P) \subset D^k$, for P a k-ary predicate symbol.
- $I_w(\pi) \subset W$, for π a path expression returning a set of nodes.

To interpret formulas with free variables, we define an assignment function *g* such that:

g: SVAR $\cup$ FVAR $\rightarrow$ D $\cup$ W

- g(x)$\in$D if x$\in$FVAR
- g(x)$\in$W if x $\in$ SVAR.

We denote by g_d^x an assignment that is the same as *g* except that *g(x) = d*.

Given a model and an assignment *g*, the interpretation of the term *t* at a given state *w*$\in$W, denoted by $\bar{t}$ is defined as:

- $\bar{x}$= g(x) for x $\in$ FVAR
- $\bar{s}$ = g(s) for s$\in$ SVAR
- $\bar{c}$ = $I_w(c)$ for c $\in$ CON
- $\overline{s/@q} = I_{g(s)}(q)$ for s a state variable, and q $\in$ {ATTS $\cup$ ATTD $\cup$ FUN}.
- $\overline{\pi/@q} = \{I_{w1}(q), I_{w2}(q), \ldots I_{wn}(q)\}$ such that $w_1, w_2 \ldots w_n \in I_w(\pi)$.

The satisfaction relation of a BPath expression is defined as follows:

- M, g, w |= t $\neq$ Ø. $\Leftrightarrow$ $I_{M,w,g}(t) \neq$ Ø.
- M, g, w |= $P(t_1, \ldots, t_n) \Leftrightarrow (\overline{t1}, \ldots, \overline{tn}) \subset I_w(P)$
- M, g, w |= t = u $\Leftrightarrow$ $\bar{t} = \bar{u}$, where: *t* and *u* are terms.

- $M, g, w \models \text{not } \varphi \Leftrightarrow M, g, w \not\models \varphi$.
- $M, g, w \models \varphi \text{ and } \psi \Leftrightarrow M, g, w \models \varphi \wedge M, g, w \models \psi$.
- $M, g, w \models \varphi \text{ or } \psi \Leftrightarrow M, g, w \models \varphi \vee M, g, w \models \psi$.
- $M, g, w \models s[\varphi] \Leftrightarrow M, g, g(s) \models \varphi$ for $s \in SVAR$.
- $M, g, w \models s!\varphi \Leftrightarrow M, g_w^s, w \models \varphi$.
- $M, g, w \models x{:=}T,\varphi \Leftrightarrow M, g_T^x, w \models \varphi$.

The interpretation of a path expression on the model M, starting from the state-node w, and given an assignment g is defined as follows:

- $I_{M,w,g}(\pi_1 \mid \pi_2) = I_{M,w,g}(\pi_1) \cup I_{M,w,g}(\pi_2)$.
- $I_{M,w,g}(\pi_1/\pi_2) = \{w'' \mid \exists\, w' \in I_{M,w,g}(\pi_1), w'' \in I_{M,w',g}(\pi_2)\}$.
- $I_{M,w,g}(\pi[\varphi]) = \{w' \mid w' \in I_{M,w,g}(\pi) \wedge M,g,w' \models \varphi\}$.
- $I_{M,w,g}(\text{self::n}) = \{w \mid \mathit{label}(w)=n \vee n=*\}$.
- $I_{M,w,g}(\text{following-sibling::n}) = \{(w' \mid w\ R\ w') \wedge (\text{label}(w')=n \vee n=*)\}$.
- $I_{M,w,g}(\text{preceding-sibling::n}) = \{(w' \mid w'\ R\ w) \wedge (\text{label}(w')=n \vee n=*)\}$.

BPath does not provide by itself a means for expressing quantified expression as it is based on XPath 1.0. However, this becomes possible by using XPath 2.0 or XQuery. In fact, XPath 2.0 supports existential and universal quantification as defined by the following expressions (Where π an XPath 2.0 expression and φ a condition that can be a BPath or XPath 2.0 expression):

- Existential quantification: *Some $x in π satisfies φ.*
- Universal quantification: *Every $x in π satisfies φ.*

3.2.5 Temporal logic with BPath

LTL is a commonly-accepted formalism for specifying properties of finite-state discrete systems; it provides a formal system for qualitatively describing and reasoning about how the truth values of a formula change over time [Eme90]. LTL has future and past operators. The LTL future operators with their respective meaning are:

- X (φ): φ should be true on the next state.
- F (φ): means that φ should be true on the current state or at least in one state in the future.
- G (φ): φ should be true on the current state and on every state in the future.
- φ U ψ: φ has to be true at least until ψ, which is true now or in the future.

The use of the past tense operators might be useful simply in order to make the formulation of specifications more natural and convenient [Eme90], these LTL past operators are:

- B(φ): φ should be true on the previous state.
- P(ϕ): ϕ true somewhere in the past.
- H(ϕ): ϕ true always in the past.

Classical semantic models for LTL are typically sequences, which are infinite toward the future and finite toward the past [MNP05]. In the context of business processes, this semantics should be adapted, because instances are supposed to be finite, and at the time of evaluation of an LTL formula, we have only a finite execution trace. Several semantics for LTL over finite trace were proposed [Kam68, MP95, HR01, EFH+03, BLS08]. Mainly, they differ from the way they interpret the next state operator *X()* at the end of the trace. For the sake of simplicity, we adopted the traditional LTL semantics for finite trace [Kam68], which considers the LTL X() operator as a strong next operator i.e. X(φ) is considered false if no further state exists at the end of the trace. Then, LTL semantic can be defined by the following satisfaction relation, where $i,k,j \in N$ and w_i is the i state of W:

- $M, g, w_i \models X(\varphi) \Leftrightarrow i < |W|$ and $M, g, w_{i+1} \models \varphi$
- $M, g, w_i \models G(\varphi) \Leftrightarrow \forall j, i <= j <= |W|: M, g, w_j \models \varphi$
- $M, g, w_i \models F(\varphi) \Leftrightarrow \exists j, i <= j <= |W| : M, g, w_j \models \varphi$
- $M, g, w_i \models \varphi \, U \, \psi \Leftrightarrow \exists k, i <= k <= |W| : M, g, w_j \models \psi$

  and $\forall j, i <= j <= k: M, g, w_j \models \varphi$
- $M, g, w_i \models B(\varphi) \Leftrightarrow 1 < i <= |W|$ and $M, g, w_{i-1} \models \varphi$

- $M, g, w_i \models H(\varphi) \Leftrightarrow \forall j, 1 <= j <= i: M, g, w_j \models \varphi$
- $M, g, w_i \models P(\varphi) \Leftrightarrow \exists j, j <= i: M, g, w_j \models \varphi$

We support LTL in our framework by defining a mapping from LTL operators to XPath 1.0 that has been extended with some functions. XPath has the advantage to be easy to integrate with SOA environment that natively provides XML processing capabilities. But, this mapping can be easily modified to implement a different LTL semantics if needed. Table 3 describes this mapping from LTL to XPath.

Table 3: LTL to XPath

BPath Syntax	**Mapping with XPath 1.0**
X(φ)	following-sibling::*[1][φ].
F(φ)	(self::* | following-sibling::*) [φ].
G(φ)	not (self::* | following-sibling::*) [not(φ)]).
U(φ, ψ)	$S! (self::* | following-sibling::*) (ψ and not((self::*| preceding-sibling::*) [not (P($S= self::*) → φ))]
B(φ)	preceding-sibling::*[position()=last()-1][φ]
P(φ)	(Self::*| preceding-sibling::*)[φ]
H(φ)	not(Self::*| preceding-sibling::*) [not(φ)])

The mapping of hybrid logic operators as well as BPath dynamic attributes to XPath expressions needs the use of extended XPath functions. They will be presented in the implementation section.

3.2.6 Trace view

A business process view is a representation of a business process model from a given perspective. At runtime, a process view captures relevant information from an execution of a business process, and then generates a state trace containing a set of relevant states from the view perspective. From the monitoring side, we expect from using process views several benefits, for instance: they provide simplified and clear abstractions, which help user to write monitoring properties, but also to make the evaluation of monitoring properties faster, because monitoring properties will be evaluated just over a smaller set of states, by assuming that each process view will produce less states than the business process at runtime.

In addition to process views, it is interesting to be able to filter state traces in order to select only those satisfying some condition at runtime as illustrated in Figure 12. For instance: select only the set of process instances (instance state of a process trace) related to some user, and then evaluate monitoring properties just over this subset of instances. This is what we call a trace view, which can be used as abstractions in BPath expressions in order to facilitate writing of monitoring properties like by using SQL view in relational databases.

Definition 13 (Trace view)

A trace view is a subset of a state trace. It is the result of the evaluation of a BPath path expression over a state trace, which returns a subset of state nodes.

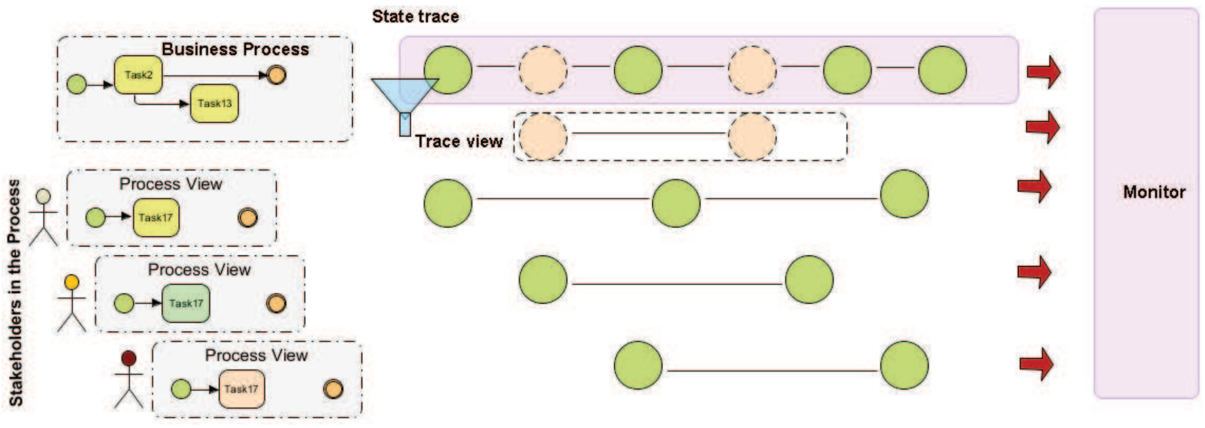


Figure 12: Process view and Trace view

Several trace views are predefined in our framework (state trace global view, process trace view, instance trace view, protocol trace view).

Definition 14 (State trace global view)

A state trace global view is the execution state trace itself, we make reference to it as following:

```
$Trace:= (self::* | following-sibling::*)                    (13)
```

Definition 15 (Process trace view)

A state trace containing only all last process instance state nodes. It can be obtained by the following BPath expression, which returns all state nodes, which have no following-sibling state node with a same instanceid.

```
$Process:=  (self::*  |  following-sibling::*)[$S!  not
                                                             (14)
(following-sibling::* [@instanceid=$S/instanceid])]
```

Definition 16 (Instance trace view)

A state trace containing only all state nodes of a given process instance, which is defined by the following BPath expression:

```
$Instance:= $S! $Trace/following-sibling::*
                                                             (15)
[@instanceid=$S/@instanceid]
```

Definition 17 (Protocol trace view)

A protocol trace view is a state trace containing only all last process instance state nodes, relating to a protocol view, which is obtained by the following BPath expression that returns all state nodes having no following-sibling state node with a same instanceid.

```
$Protocol:= (self::* | following-sibling::*)
                                                             (16)
[source='protocol']
```

3.3 Business process monitoring

In this section, we will present through a concrete scenario, how our monitoring framework is able to monitor business process with BPath. This concrete scenario (WatchMe) is concerned with a Mobile Virtual Network Operators (MVNO) and focus on the monitoring of compliance to a set of predefined licensing rules.

WatchMe services combine value-added application capabilities with Internet and the next generation of mobile telecommunication network features. All these capabilities are integrated by MVNO services to provide combinations of call/session control, messaging features, presence and location features, multimedia content streaming, parental monitoring, etc. The MVNO environment is particularly challenging due to the heterogeneous characteristics of the network infrastructure. Many of the applications that provide the MVNO service components are owned and managed by different enterprises (i.e. the MVNO, the network providers, and third-party application providers). Moreover, the current trends of such systems of services include organizing them in a SOA-oriented fashion, thus fostering loosely coupled networks of interoperating Web services.

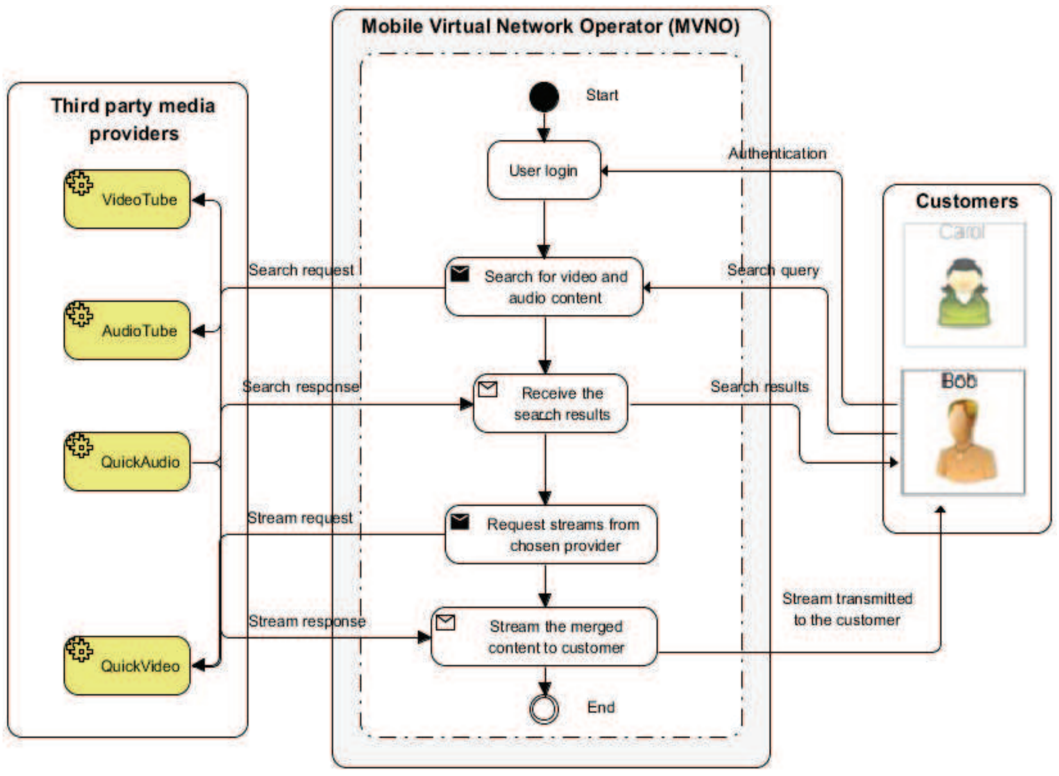

Mobile Virtual Network Operator (MVNO)
Third party media providers
VideoTube
AudioTube
QuickAudio
QuickVideo
Start
User login
Authentication
Search request
Search for video and audio content
Search query
Search response
Receive the search results
Search results
Stream request
Request streams from chosen provider
Stream response
Stream the merged content to customer
Stream transmitted to the customer
End
Customers
Carol
Bob

Figure 13: WatchMe business process

Figure 13 shows part of the WatchMe scenario: The left-hand side of the figure shows the video providers (*Videotube* and *QuickVideo*) and audio providers (*AudioTube* and *QuickAudio*) available in the WatchMe system, which can be dynamically composed and consumed by customers (*e.g., Bob*) via multimedia streaming. The video providers impose their licenses, which they wrote in conjunction with the audio providers they have agreements with. These licenses, therefore, state which audio providers can be used to merge an audio stream with the video provider's video stream (e.g., a video from the video provided *QuickVideo* can be only downloaded with audio offered by *QuickAudio*).

Table 4 summarizes the list of compliance requirements we want to be compliant with in WatchMe scenario regarding licensing.

Table 4: Compliance requirements of the WatchMe scenario

Compliance requirements	**Description of compliance requirements**
Composition permission	Only pre-defined combinations of video and audio providers are allowed due to the licenses specified by the video provider.
Pay-per-view plan	When the WatchMe company subscribes for the Pay-per-view plan it acquires a limited number of streams based on the amount paid to the media supplier.
Time-based plan	When the WatchMe company subscribes for the Time-based plan it acquires any number of times any possible streams in a certain period, based on the amount paid to the media supplier.

These compliance requirements can be expressed and monitored using BPath as following:

Composition permission

The Composition permission allows only some combination of video and audio providers together. In the BPath expression (see **Formula 1**), "*$Process [position()=last()]*" allows to access to the last updated instance of the process where the checking of allowed composition should be performed.

Formula 1: Composition permission

```
$Process [position()=last()] [
( @VideoProvider = 'VideoTube' → ( @AudioProvider
='AudioTube' or @AudioProvider ='QuickAudio' ) )
and
( @VideoProvider = 'QuickVideo' → @AudioProvider
='QuickAudio' )
]
```

Pay-per-view plan

In this requirement, we should check that the imposed limit number of streams from a given provider is respected. In the following expression, we check that the number of streams from a video provider is less than 20. To count the number of streams from a given video provider, we need to explore the history of all process instances that have a same video provider as the current one (the last instance).

Formula 2: Pay-per-view plan license

```
$Process [position()=last()][
@VideoProvider → (
$S!
count (
   (self::*|preceding-sibling::*)
    [@VideoProvider= $S/@VideoProvider]
 ) <20
)
]
```

Time-based plan

This requirement allows acquiring unlimited number of streams in a certain period (time window), based on the amount paid to the media supplier. Thanks to BPath, which defines time explicitly through the *@timestamp* static attributes and by using the native XPath date function *dateTime-less-than*, we can easily define a time window, as shown in the following BPath expression.

Formula 3: Time-based plan license

```
$Process [position()=last()]
[
@VideoProvider='VideoTube' →
(
  dateTime-less-than('12/29/2010 23:11:43',  @timestamp))
  and
  dateTime-less-than ( @timestamp,'3/09/2011 15:15:43')
)
And
@VideoProvider='QuickVideo' →
(
  dateTime-less-than('11/01/2010 23:11:43' , @timestamp)
  and
  dateTime-less-than (@timestamp, '4/10/2011 15:15:43' )
)
]
```

3.4 Business protocol monitoring

As an example of monitoring business process view, we propose in this section to focus on a special view of a business process, which is called a business protocol. The purpose of business protocol is essentially to specify the set of conversations (sequence of messages) that are supported by a business process. Business protocols can be useful in several analysis contexts, both at development and runtime [BCT+03, BCT04, BM06].

Their immediate benefit is that they provide developers with information on how to write clients that can correctly interact with a given service or with a set of services. In addition, once protocols become an accepted practice and service descriptions become endowed with protocol information, the middleware can be significantly extended to better support service development, binding, and execution in a number of ways, considerably simplifying the whole service life-cycle [BCT06].

In order to go in progress with our presentation of monitoring business protocols, let us consider the following scenario shown in Figure 14, which is inspired from [MS04], of an online Car Rental System (CRS). CRS offers a car location service: whenever a rent car request is received (*RentCar*), the availability of the requested car will be checked by invoking a sensor service. If it is not available, then a list of cars will be sent to the client, otherwise, the requested car is reserved, and a confirmation message is sent to the client

(*CarReservation*). Then, the customer will send her/his bank information (*BankInfo*), which will be validated by invoking a bank service, then the rent request will be validated by a CRS agent. After that, the keys of the car will be sent to the customer. After returning the keys, the customer receives a payment confirmation (*BankConfirmation*). The CRS agent may refuse the rent for various reasons, including the case when the received bank information is not valid. Thus, a *RentRejected* message will be sent to the customer, and the process instance is terminated.

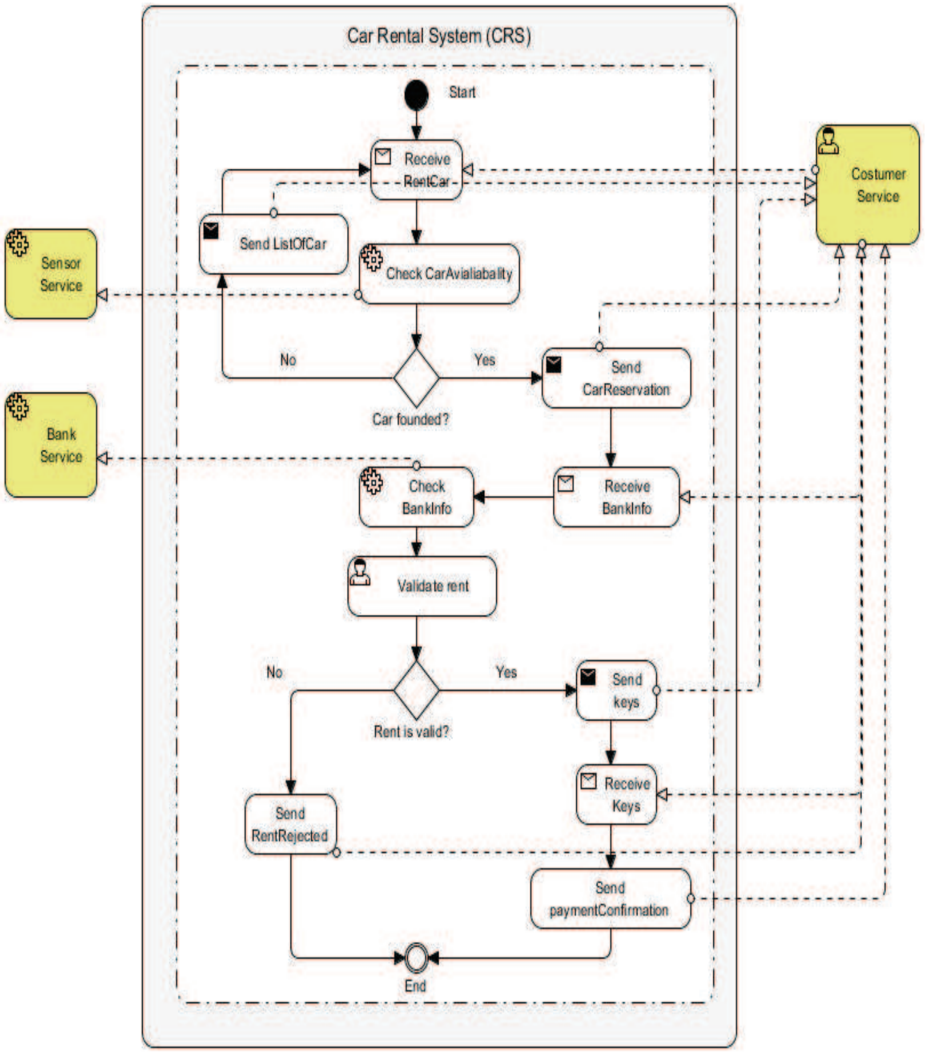


Figure 14: CRS business process

Let us consider a list of compliance requirements as described in, which should be continuously monitored at runtime.

Table 5: CRS compliance requirements

Compliance requirements	**Description of compliance requirements**
Filtering customers	Process should not accept rejected customer until a given period. Example: if a customer bank information is rejected, customer must wait one hour before requesting a new car
Coherence of the system	The behavior of the process should be in coherence with the sensor. Customer should not get a car reservation when the car keys are taken by another customer
Quality of service	When a customer sends a rent request, the average time to make a car reservation should not exceed 180 seconds.
Internal commitment	Rejected rents should not exceed 20 in a given time windows

In order to give an example on how process view can be used to monitor a business process, we propose to focus on a special process view, capturing the exchanged messages between the CRS business process and the customer. These exchanged messages can be captured by a business protocol, as shown in Figure 17. We have developed a tool to automatically extract business protocols by a transformation of business process expressed in BPEL language. The details will be provided later.

3.4.1 Overview of the monitoring process

Figure 15 depicts the main steps of our approach in order to perform monitoring over a business protocol view. This encompasses the monitoring of the business protocol itself in terms of checking that messages sent and received are in the expected order. First, a BPEL business process external behavior is represented by means of a business protocol. Then, monitoring properties can be formulated using BPath over the business protocol i.e.

monitoring properties will explicitly use the states defined within the business protocol model.

At runtime, the business protocol model is executed in correspondence to the execution of the business process. So, all incoming or outgoing messages will be captured by the monitoring framework. The process engine as well as the business protocol will respectively publish events, which will be stored in the execution trace. The execution trace is of two types: state trace, generated by the business protocol, and execution data source (concretely it may be an event log) generated by the process engine. On the basis of these generated execution logs, the monitor will check the correctness of the current execution and evaluate the specified statistical query to return statistical indicators about the execution, then monitoring results will be published on a dashboard.

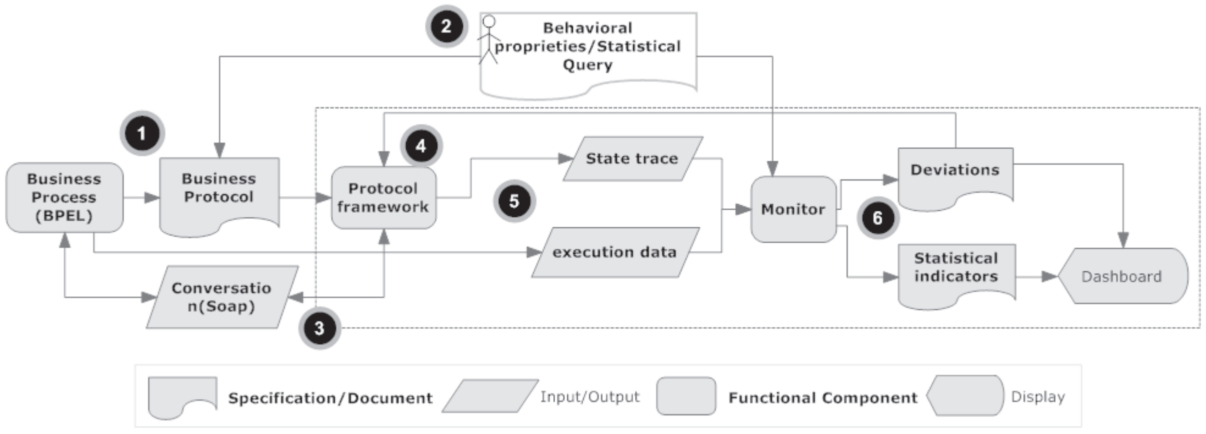


Figure 15: Protocol monitoring

Protocol events

Additionally, the business protocol execution framework provides a set of business protocol execution events that are useful to capture information about exchanged messages and control the message exchanging, but also to specify when verification tasks should be performed. Table 6 presents a set of business protocol events. For instance to perform verification, whenever a message is received or sent, we write:

```
OnMessageReceived (EventArgs  e){
---Check a property here}
OnMessageSent(EventArgs e){
---Check a property here}
```

Table 6: Business protocol events

Protocol event	Description of the event
OnEvent	Triggers every time an event listed in this table occur.
OnNewInstance	Occurs when a new instance is started
OnNewState	Occurs when a state is entered
OnMessage	Occurs when a message is sent or received
OnMessageReceived	Occurs when a message is received
OnUnknwonMessage	Occurs when a received message is not defined in the protocol
OnUnexpectedMessage	Occurs when a received message is defined in the protocol, but not expected from the current state
OnMessageSent	Occurs when a message is sent
OnTransition	Occurs when a transition from a state to another state happens
OnEndInstance	Occurs when an instance is completed

3.4.2 Business protocol extraction

Formally, we define a business protocol as a tuple P = (S, s_0, F, M, T):

- S is a finite set of states the process goes through during its execution.
- s_0 is the initial state.
- F represents the finite set of final states.
- M is a set of messages.
- $T \subseteq S \times S \times M \times \{+,-\}$ is the set of transitions, where every transition is labeled with a message name and its polarity, when a message is consumed by the protocol, the transition is assigned the polarity sign (+), and when it is produced by the protocol, the transition is assigned the sign (**-**).

In order to create a business protocol from a given business process specified in BPEL, we propose to proceed as described in the following steps.

Step 1: Cleaning BPEL process

Remove all elements from the XML BPEL process, except communication (sending and receiving messages activities), and structured activities (control activities: if, flow…). Then, let *xBpel* the resulting XML tree. For instance, Listing 2 shows an example of the CRS BPEL process at the end of this step.

```
<sequence>
  <receive name="RentInfo" partnerLink="Customer" operation="RentCar"
portType="ns1:portType1" variable="RentCarIn">
  </receive>..
  <if name="validRent">
    <reply name="KeysOut" partnerLink="Customer"
operation="SendBankInfo" portType="ns1:portType1" variable="KeysOut"/>
    <else>
      <reply name="RentRejected" partnerLink="Customer"
operation="SendBankInfo" portType="ns1:portType1" variable="CardRejected"/>
    </else>
  </if>
</sequence>
```

Listing 2: xBpel Fragement of CRS BPEL process

Step 2: Build an object representation of the cleaned process

Build an object representation "*oBpel*" corresponding to the cleaned business process *xBpel*, where each activity of a type T in *xBpel* is represented as an instance of a class of the same type, and reflecting the same structure as in *xBpel*. Each instance has two properties: *input, ouput*, that can contains a set of protocol states and a method '*Transform'*. Figure 16 shows such a tree of the previous *xBPel*.

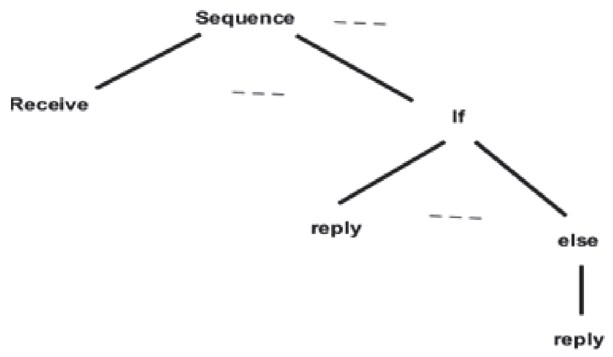


Figure 16: oBPEL activities tree

Step 3: Generate the protocol

The protocol automaton can be generated as described by the Algorithm 1, which takes as input the *oBPEL* activities tree obtained in the Step 2.

Algorithm 1: BPEL process to business protocol

```
Input: oBpel activities tree created in the step 2
Output: A business protocol P = (S, s0, F, M, T)
    1. Initialize the protocol P to ({s0, f0}, s0, {f0}, Ø, Ø )
    2. Let a the root activity of oBpel
    3. Call the transform method of the activity a
    4. Replace states in the input property of the activity a by s0
    5. Replace states in the output property of activity a by f0
```

We start by initializing the protocol to be generated, and then call the transform method of all sub activities of the root activity. The transform method of each type of activity is described in the following by distinguishing between BPEL communication and structured activities.

BPEL communication activities

Communication activities are those used in order to send or receive messages from and to a business process. We focus on three of them: receive, reply, and invoke activities.

Receive activity: Specifies the partner link from which to receive information and the port type and operation for the partner link to invoke (a service defined within the business process). This activity waits for an asynchronous callback response message from a service. The content of this response is stored in a response variable in the process.

Algorithm 2: Transform receive activity

```
Input: Receive activity in oBpel
Output: Create a set of protocol elements/ input, output initialization
    1. Create two states: $s_1$, $s_2$
    2. Create a new message: $m_1$=(pl, op, inMessage)
    3. Create a new transition: $t_1=(s_1, s_2, m_1,+)$
    4. Add to P: $s_1$, $s_2$, $m_1$, $t_1$
    5. Set input property to $s_1$
    6. Set output property to $s_2$
```

Pl is the name of the partner which sends the message (partnerlink in BPEL), *inMessage* is the input message of the operation *op*, which can be extracted by parsing the WSDL file defining messages involved in the BPEL process definition.

Reply activity: Allows the process to send a message in response to a previously received one through a receive activity. The combination of a receive activity and a reply activity forms a request-response operation on the WSDL port type for the process.

Algorithm 3: Transform reply activity

```
Input: Reply activity in oBpel
Output: Create a set of protocol elements/ input, output initialization
    1. Create two states: $s_1$,$s_2$
    2. Create a new message: $m_1$= (pl,op,outMessage), where
```

```
     outMessage is the output message of the operation op
  3.  Crate a new transition: $t_2 = (s_1,s_2,m_1,-)$
  4. Add to P: $s_1,s_2, m_1, t_1$
  5. Set input property to $s_1$
  6. Set output property to $s_2$
```

Invoke activity: Invoke activity allows a business process to invoke a one-way or request-response operation on a portType offered by a partner. In the request-response case, the invoke activity completes when the response is received.

Algorithm 4: Transform invoke activity

```
Input: Invoke activity in oBpel
Output: Create a set of protocol elements/ input, output initialization
If the activity is one-way operation then
 Consider this activity as a reply activity
Else
 If the activity is a requiest-response operation then
  1. Create three states: $s_1,s_2,s_3$
  2. Create a message $m_1=(p_l,op,outMessage)$, where outMessage is
     the output message of the operation op
  3. Create a message $m_2=(p_l,op,inMessage)$, where outMessage is the
     input message of the operation op
  4.  Create a transition: $t_2=(s_1,s_2,m_1,-)$
  5. Create a transition: $t_2=(s_2,s_3,m_2,+)$
  6. Add to P: $s_1,s_2,m_1,t_1,t_2$
  7. Set input property to $s_1$
  8. Set output property to $s_3$
```

BPEL structured activities

Structured activities offer a way to structure BPEL processes. They describe the flow of a business process by structuring basic activities. In this way control patterns, data flow, fault handling and coordination of messages can be achieved.

If activity: This activity is used to select exactly one activity for execution from a set of choices.

Algorithm 5: Transform if activity

```
Input: If activity in oBpel
Output: Transforming sub-activities/input, output initialization
    1. Transform all sub-activities
    2. Set input property to the set of input of all sub activities
    3. Set output property to the set of output of all sub  activities
```

Sequence activity: States that contained activities should be performed sequentially in lexical order.

Algorithm 6: Transform sequence activity

```
Input: Sequence activity in oBpel
Output: Transforming sub-activities/ input, output initialization
    1. Transform sub activities
    2. For each a, b sub-activities, and b the following sibling of a
        a. Create a new state: s
        b. Add s to P.
        c. Replace all states in the a output property, and b input
           property by s in P
```

```
3. Set input property to first sub-activity input property

4. Set output property to first sub-activity output property
```

RepeatUntil activity: This activity is used to define that the sub-activity is to be repeated until a specified condition becomes true. The condition is tested after the sub-activity completes. This activity is used to execute the sub-activity at least once.

Algorithm 7: Transform repeatUntil activity

```
Input: Sequence activity in oBpel

Output: Transforming sub-activities, output initialization

1. Transform the sub-activity;
2. Create a new state: s
3. Add s to P
4. Replace the state in the input property of the sub-activity, and the
   state in the output of the sub-activity by s in P
5. Set input, and output property to s
```

Flow activity: This activity enables to specify one or more activities to be performed concurrently. A flow activity completes when all activities in the flow are completed. Completion of a flow activity includes the possibility that it can be skipped if its enabling condition is false.

Algorithm 8: Transform flow activity

```
Input: Flow activity in oBpel

Output: Transforming sub-activities / input, output initialization

1. Let a, b...the sub-activities
2. Let $P_x$= Protocol (a) × Protocol (b)...( ×: Product of automata)
3. Set Input to the initial state of $P_x$
```

4. Set Output to the final state of P_x

3.4.3 Execution scenario

In this section, we will show through a concrete execution scenario, how BPath can be used to monitor a business process over a business protocol view. The business protocol under consideration is shown in Figure 17, which is extracted from the business process defined in Figure 14, by applying the steps presented in the previous section, with a small change at the step 1, consisting of keeping only communication activities, which have customer service as a partner Link.

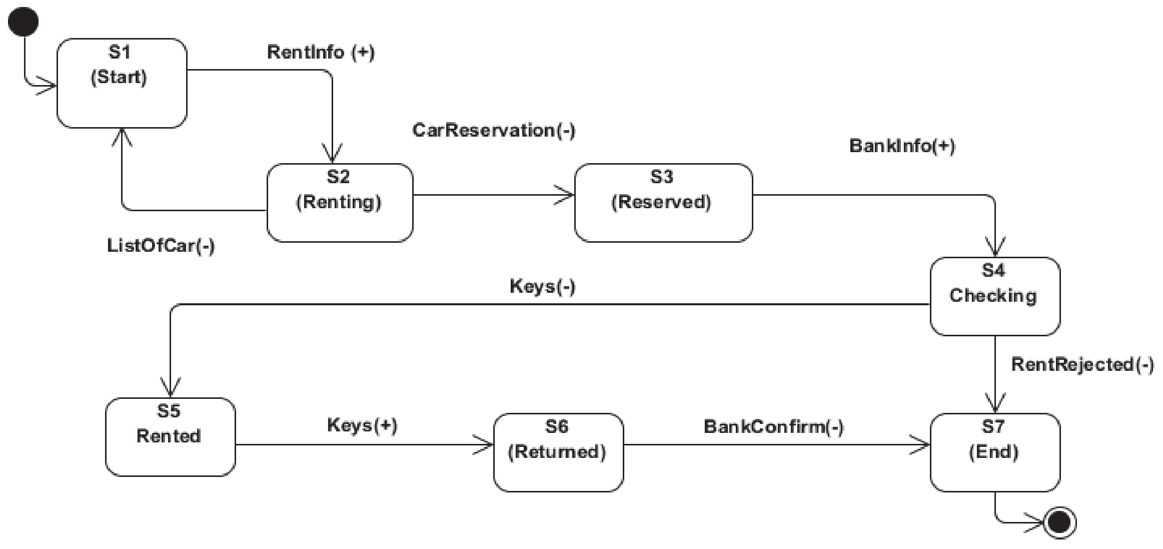


Figure 17: CRS business Protocol

Let us assume that the car rental system manages three cars (*RedCar, GreenCar, BlueCar*), and receives requests from three clients (*John, Mark and Bob*), that we consider as web services interacting with the CRS business process: First, John sends a request for red car. His credit card will be rejected, but he tries again and gets the car reservation. Mark requests a green car, gets a reservation and keys, and then receives a payment confirmation after returning the *keys*. *Bob* requests the same car as Mark and obtains a reservation.

At runtime, messages exchanged between different instances of the process and external partners will be captured and stored in the event log. Listing 3 shows an example of an event log, generated from the supposed execution scenario of the CRS business process.

```
L1: RentInfo, ClientInfo=John, CarInfo=RedCar, processid=crs, instid=1,
timestamp=1
L2: CarReservation, carReserved=yes, processid=crs, instid=1, timestamp=3
L3: RentRejected, cardInfo=798799979879, processid=crs, instid=1,
timestamp=5
L4: RentInfo, ClientInfo=Mark; CarInfo=GreenCar, processid=crs, instid=2,
timestamp=8
L5: CarReservation, carReserved=yes, processid=crs, instid=2, timestamp=10
L6: BankInfo, cardInfo=798799979879, processid=crs, instid=2, timestamp=12
L7 : RentInfo, ClientInfo=John, CarInfo=BlueCar, processid=crs, instid=3,
timestamp=15
L8: CarReservation, carReserved=yes, processid=crs, instid=3, timestamp=17
L9: Keys, keysOutimestamp=KY123, processid=crs, instid=2, timestamp=19
L10: RentInfo, ClientInfo=Bob; CarInfo= GreenCar, processid=crs, instid=4,
timestamp=22
L11: CarReservation, carReserved=yes, processid=crs, instid=4, timestamp=24
L12: Keys, keysIn=KY123, processid=crs, instid=2, processid=crs, instid=2,
timestamp=26
L13: BankConfirm, payeConfirmed =yes, BankTransation=Trans0001,
processid=crs, instid=2, timestamp=28
```

Listing 3: Execution data source

Additionally, the business protocol will generate events related to a transition from a state to another state, when a message is received or sent by an instance of the process. These events are stored in the state log (Listing 4).

```
<Trace>
  <Start processid="crs" instid="1" source="protocol" timestamp="0"/>
  <Renting processid="crs" instid ="1" source="protocol" timestamp ="2"/>
  <Reserved processid="crs" instid ="1" source="protocol" timestamp ="4"/>
  <Checking processid="crs" instid ="1" source="protocol" timestamp ="6"/>
  <Start processid="crs" instid ="2" source="protocol" timestamp ="7"/>
  <Renting processid="crs" instid ="2" source="protocol" timestamp ="9"/>
  <Reserved processid="crs" instid ="2" source="protocol" timestamp ="11"/>
  <Checking processid="crs" instid ="2" source="protocol" timestamp ="13"/>
  <Start processid="crs" instid ="3" source="protocol" timestamp ="14"/>
  <Renting processid="crs" instid ="3" source="protocol" timestamp ="16"/>
  <Reserved processid="crs" instid ="3" source="protocol" timestamp ="18"/>
  <Rented processid="crs" instid ="2" source="protocol" timestamp ="20"/>
  <Start processid="crs" instid ="4" source="protocol" timestamp ="21"/>
  <Renting processid="crs" instid ="4" source="protocol" timestamp ="23"/>
  <Reserved processid="crs" instid ="4" source="protocol" timestamp ="25"/>
  <Returned processid="crs" instid ="2" source="protocol" timestamp ="27"/>
  <End processid="crs" instid ="2" source="protocol" timestamp ="29"/>
</Trace>
```

Listing 4: Business protocol state trace

Now, by using BPath we can write the following properties in order to monitor compliance of the CRS system:

Filtering customers

Check that in case where the credit card of a customer was rejected, the customer should wait one hour to be able to get a car reservation. We formulate this property in BPath as follows:

Formula 4: Filtering customers

```
$Protocol [position()=last()] [
$s! (
self::CarReserved →
not(
      P(
       self::End/@RentRejected and @ClientInfo=$s/@ClientInfo
        and
         ($s/@timestamp - @timestamp < 60 )
       )
    )
 )
]
```

In this property, whenever a car is reserved, we explore the history of protocol messages to check that the rent was not rejected for the same customer before one hour, which requires a correlation between processes instances of the same costumer on the basis of customer information. This property makes explicitly reference to two states of the business protocol, which are: The *CarReserved* and *End* states.

As we can see from the previous execution log (Listing 3) that this property is violated, when John obtains a car reservation (L8), knowing that his credit card was rejected less than one hour before (at L3).

Coherence of the system

A costumer should not get a car reservation when the keys are taken by another customer. This property can be expressed using BPath as follows:

Formula 5: Coherence of the system

```
$Protocol[position()=last()][
$S!
(
 self::Reserved →
 B(
   P[$s/@CarInfo=@CarInfo → (@Rented → @KeysReturned) ]
  )
)
]
```

In this property we express that whenever a car is reserved, if the same car was rented in the past, it should be returned to the parking. Otherwise, we conclude that there is an inconsistency between the sensor service and the system.

As we can see from the previous execution log, this property is violated at line L11: the green car was reserved for Bob (L11), but this car is still assigned to Mark (L9), and the keys of the car are returned by Mark only after (L11), exactly at (L12).

Quality of service

BPath is also a query language that can be used to return statistical indicators on the execution of a business process. For example, in the quality of service compliance requirement, we should check the average time to make a car reservation should not exceed three minutes.

Formula 6: Quality of service

```
Sum(
(self::Reserved |Following-sibling::Reserved)/@(@timestamp-
$Instance[1]/@timestamp) )
Div
count (Following-sibling::Reserved)
< 180
```

In order to compute the time spent to perform a reservation, we compute the difference between the *timestamp* of each *Reserved* state and the *timestamp* of the first state of the current instance (The variable *$Instance* is a trace view containing all states of a given process instance). Then, we divide the sum obtained on the number of all made reservations. We used two functions *sum* and *count,* which respectively compute the sum and the number of elements of a sequence.

Internal commitment

The internal commitment compliance requirement reflects an internal policy to the RSC system, which should be respected by the system agents, and services. It imposes a maximum number on the rejected rents in a given period of time.

Formula 7: Internal commitment

```
Count(
  $Protocol[
  dateTime-less-than('11/01/2010 23:11:43' , @timestamp)
  and
  dateTime-less-than (@timestamp, '4/10/2011 15:15:43' )
  ]
  /Following-sibling::End[@RentRejected]
)
< 20
```

This property counts the number of *RentRejected* messages sent from the business process, during a given period of time. To this end, the protocol trace is filtered using a time window.

3.5 Web services choreography monitoring

A choreography monitoring system observes exchanged messages between a set of Web services and performs verification and statistical measurement. Often, a choreography monitoring system stores the exchanged messages in a global execution log, in order to access to the history of exchanged messages. This global execution log is called conversation [BFH+03].

Although our monitoring system is able to perform monitoring in such scenario, this way of monitoring choreography becomes an issue, particularly when:

- Some data within exchanged messages are encrypted for security reasons.
- It is not allowed to access or use some data within the messages, for confidentiality or privacy protection reasons.

- Some data needed for monitoring are simply not exchanged within messages.
- Some legislation may explicitly prohibit storing conversation messages.

We can imagine a monitoring solution where the monitoring system obtains needed security and privacy agreements for reading encrypted data or using private data, but this solution will need to deploy a complex infrastructure for negotiating such agreements with each Web service. Also, monitoring properties can be dynamically specified at runtime by involving new data, which probably requires renegotiating the established agreements, making the monitoring system even more complex. However, the two last issues remain challenging for a choreography monitoring system, where some needed data for monitoring are not exchanged between Web service partners, or when legislation prohibits storing of conversations. Thanks to BPath, our monitoring approach is able to address these two last issues, by being able to perform monitoring without storing conversation messages.

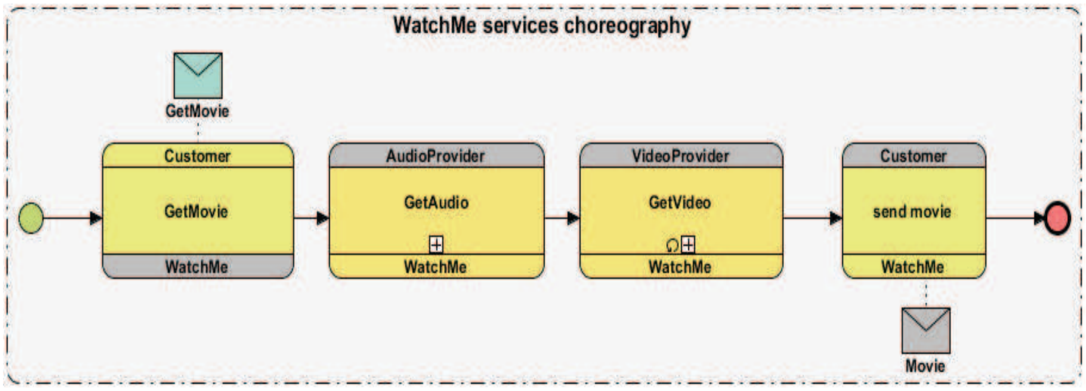


Figure 18: WatchMe choreography

In order to demonstrate this, let us consider an adapted version of the WatchMe scenario, as observed from the outside of the business process. Thus, we can see choreography of four Web services, as illustrated by the BPMN choreography diagram in Figure 18. Table 7 provides a description of each Web service with its corresponding messages.

Table 7: WatchMe choreography services

Choreography service	**Messages description**
Customer service	Sends a *GetMovie (encrypted:title, language)* message to

	WatchMe service Receives a *Movie* message from WatchMe service.
WatchMe service	Receives a *GetMovie (encrypted:title, language)* message from customer service Sends a *Movie* message to customer service. Sends a *GetAudio (language)* message to audio provider service Receives an *Audio* message from audio provider service. Sends a *GetVideo (title)* message to video provider service. Receives a *Video* message from provider service.
Video provider service (QuickVideo, VideoTube)	Receives a *GetVideo(title)* message from video WatchMe service Sends a *Video* message to WatchMe service.
Audio service (AudioTube, QuickAudio)	Receives a *GetAudio(language)* message from WatchMe service Sends an *Audio* message to WatchMe service.

The Figure 19 illustrates the working of our choreography monitoring approach. At runtime, our monitoring system observes exchanged messages between Web services, and whenever a message is observed; our monitoring system generates a new conversation state entry, and stores it into a conversation state trace (Listing 5), without storing the message somewhere. The choreography execution state is distributed over local execution traces on each Web service.

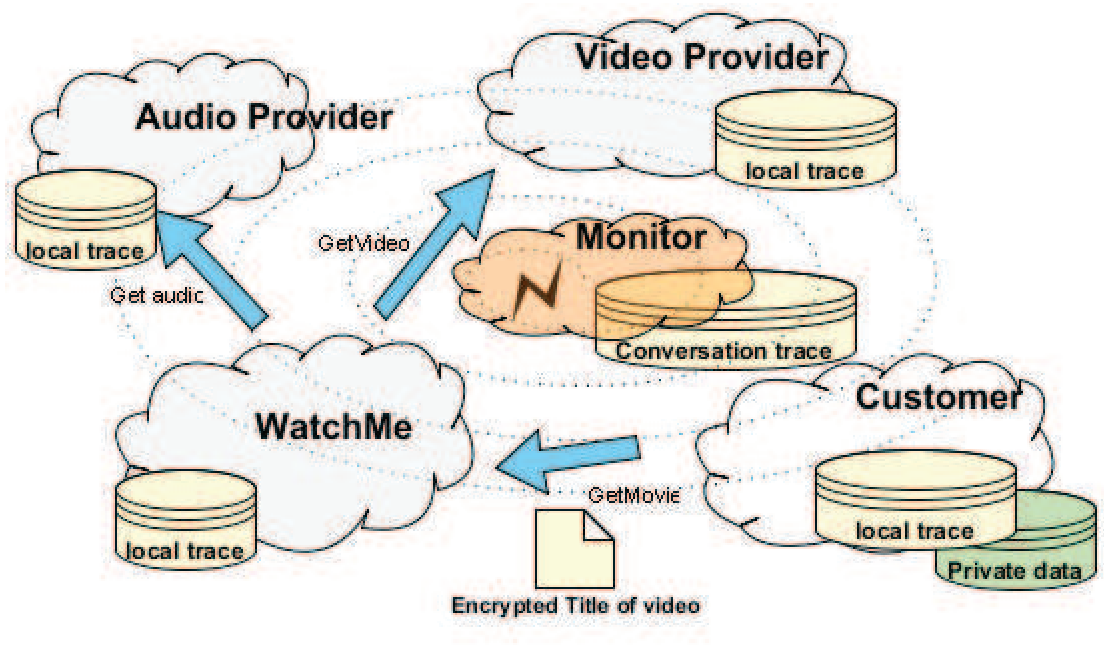


Figure 19: Service choreography monitoring

```
<Trace >
  <C1 processid="Customer" instid="1" timestamp="0"/>
  <C2 processid="WatchMe" instid ="1" timestamp ="2"/>
  <C3 processid="WatchMe" instid ="1" timestamp ="4"/>
  <C4 processid="VideoProvider" instid ="1" timestamp ="6"/>
  <C5 processid="AudioProvider" instid ="2" timestamp ="7"/>
---
</Trace>
```

Listing 5: Conversation trace

In case where a choreography Web service is executed by an external process engine, we may be unable to know information such as process id or instance id. In this case, this information can be replaced respectively by the name of the service sending a message, and the correlation information defined within the message.

In fact, we assume that each service in the choreography keeps a local execution trace that is accessible by our monitoring system, but each service can decide which information can be sent to the monitoring system. We assume that each service implements a special service giving access to its local execution trace, by implementing a common operation interface:

Object GetAttribut(Context, attributeName)

C*ontext* specifies a set of information about the monitoirng context such as *timestamp*, *processid*, *instanceid, etc.* The *attributeName* is the name of the dynamic attribute to retrieve. This operation returns the value of the dynamic attribute to retrieve.

When a BPath property is checked by our monitoring system, whenever the value of a dynamic attribute cannot be obtained from a given Web service, the checking of the property is ended with an exception message: *external state is not accessible from the Web service.*

As an example of compliance requirements, we can check the composition permission requirement of the WatchMe process, but as a requirement on the choreography that involves a set of services, which are viewed externally. This requirement can be expressed using BPath as follows:

Formula 8: Choreography composition permission

```
$Conversation [position () =last ()]
[
$S!
(
 @processid= ‘VideoTube’ and @SendVideo→
P(
  @SendAudio and @instdest=$S/@instdest and
 (@processid=’AudioTube’ or @processid=’QuickAudio’)
 )
)
And
(
@processid= 'QuickVideo'  and @SendVideo→
P(@SendAudio and @instdest=$S/@instdest and
@Processid=’AudioTube’)
)
]
```

This property checks messages sent from two services (*VideoProvider*, AudioProvider). In the first part of this property, we check that if a sender of a message is the *VIdeoTube* service and the sent message is *SendVideo* message, then there is a previous *SendAudio* message from *AudioTube* or *QuickAudio* services that has a same instance destination as the current one (*@instdes*t). *$Conversation* is the conversation global trace view.

Also, as a statistical query, we can also evaluate the average age of customers requesting a given movie, by the following BPath expression:

Formula 9: Average age of customers

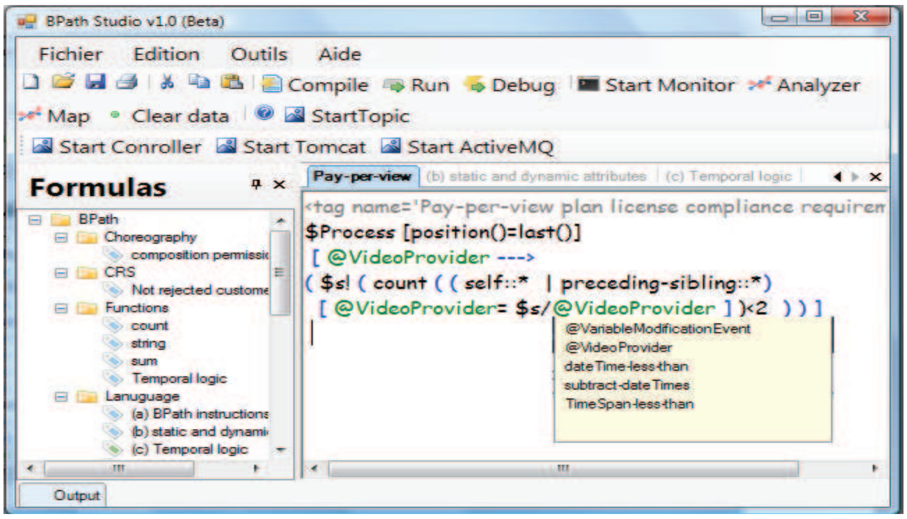
```
Sum(following-sibling::*[@processid='customer' and
@title='dexter']/@age)
Div
count(following-sibling::*[@processid='customer' and @title='
dexter'])
```

3.6 Implementation and evaluation

In this section, we will present our monitoring framework from the implementation perspective, together with the used technologies.

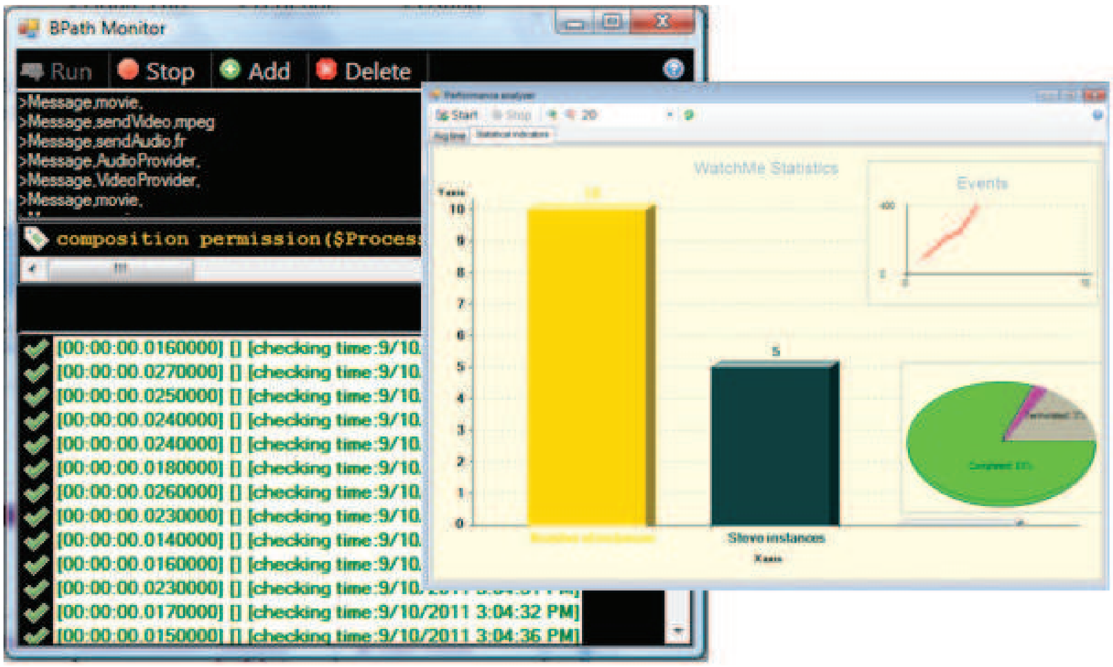
3.6.1 BPath tools

A BPath expression can be written using BPath studio editor (Figure 20), which offers a set of editing facilities, such as text coloration and auto-completion list, which shows a list of BPEL variables, and a set of predefined BPath functions that help user to write BPath expressions. This tool allows users to compile BPath expressions and to discover lexical and syntactical errors. It also allows users to start multiple servers, such as apache ODE (process engine), and ActiveMQ (messaging framework), and to run our monitoring system (Figure 21). This tool is implemented as a Microsoft C# window form.



```
<tag name='Pay-per-view plan license compliance requiren
$Process [position()=last()]
 [ @VideoProvider --->
( $s! ( count ( ( self::*  | preceding-sibling::*)
 [ @VideoProvider= $s/@VideoProvider ] )<2  ) ) ]
```

Figure 20: BPath studio



```
>Message.movie.
>Message.sendVideo.mpeg
>Message.sendAudio.fr
>Message.AudioProvider.
>Message.VideoProvider.
>Message.movie.
composition permission($Proces
[00:00:00.0160000] [] [checking time:9/10.
[00:00:00.0270000] [] [checking time:9/10.
[00:00:00.0250000] [] [checking time:9/10.
[00:00:00.0240000] [] [checking time:9/10.
[00:00:00.0240000] [] [checking time:9/10.
[00:00:00.0180000] [] [checking time:9/10.
[00:00:00.0260000] [] [checking time:9/10.
[00:00:00.0230000] [] [checking time:9/10.
[00:00:00.0140000] [] [checking time:9/10.
[00:00:00.0160000] [] [checking time:9/10.
[00:00:00.0230000] [] [checking time:9/10.
[00:00:00.0170000] [] [checking time:9/10/2011 3:04:32 PM]
[00:00:00.0150000] [] [checking time:9/10/2011 3:04:36 PM]
```

Figure 21: BPath monitor

Also, we have developed a Business protocol tool which allows extracting a business protocol from a BPEL business process or to design it manually (Figure 22). This tool is implemented using Visual Studio Visualization & Modeling SDK [DSL].

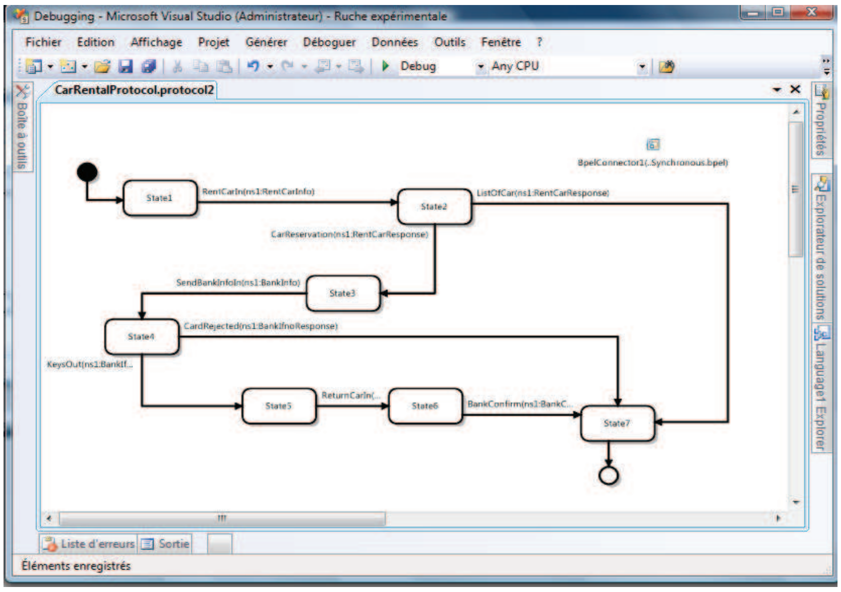

Debugging - Microsoft Visual Studio (Administrateur) - Ruche expérimentale
Fichier
Edition
Affichage
Projet
Générer
Déboguer
Données
Outils
Fenêtre
?
Debug
Any CPU
CarRentalProtocol.protocol2
Boîte à outils
Propriétés
Explorateur de solutions
Language1 Explorer
BpelConnector1(..Synchronous.bpel)
State1
RentCarIn(ns1:RentCarInfo)
State2
ListOfCar(ns1:RentCarResponse)
CarReservation(ns1:RentCarResponse)
SendBankInfoIn(ns1:BankInfo)
State3
State4
CardRejected(ns1:BankIfnoResponse)
KeysOut(ns1:BankIf...
State5
ReturnCarIn(...
State6
BankConfirm(ns1:BankC...
State7
Liste d'erreurs
Sortie
Éléments enregistrés

Figure 22: Business protocol tool

3.6.2 BPath API

We have developed two APIs for BPath. The first one is a Microsoft C# library that can be used within a programming language (like C# or Java languages). The second API is designed for use within an XSLT language.

The C# API offers a set of methods that can be used to run a BPath expression. Table 8 gives a short description of these methods.

Table 8: BPath C# API

Members	**Description**
Boolean Check (expr)	Check a BPath expression
Object Eval (expr)	Evaluate a BPath expression, the result may be of any type
Boolean Debug (expr)	Check a BPath expression, and build the execution tree.
Context Context	Get the context associated with the evaluation of a BPath expression
Boolean TraceEnabled	Enable recording all variables and events of an evaluation of a BPath expression
Boolean InMemoryEnabled	Enable keeping relevant data in memory

Our API allows to access to a set of information about the evaluation of a BPath expression itself, which we call the BPath context. BPath context can be accessed through the *Context* class, which has the following properties.

Table 9: BPath context API

Context properties	**Description**
List SVAR	Get the set of state variables
List FVAR	Get the set of global variables
List DATT	Get a set of dynamic attributes
XMLDocument ExecutionTree	In case of debugging, return the execution tree of a the debugged BPath expression

Additionally, the BPath API exposes a set of events that capture events published into the ESB (Enterprise Service Bus) by the event producers of our monitoring framework (see section 3.6.3.2), which can be used to control when a monitoring property must be checked.

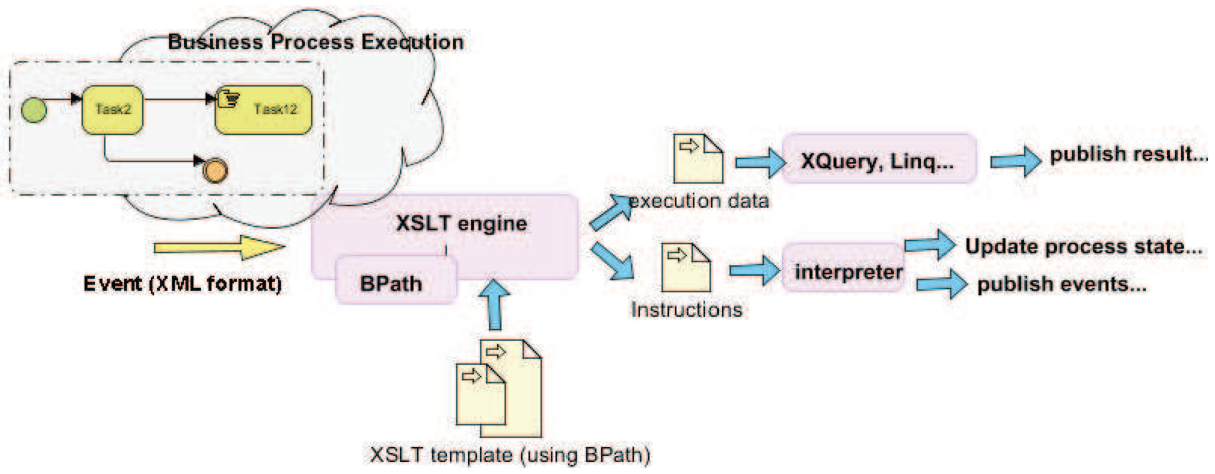


Figure 23: BPath and XSLT

The second BPath API offers the possibility to use BPath within XSLT templates. XSLT is a language for transforming XML document, by an application of a series of XSLT transformation templates. In the same spirit, as illustrated in Figure 23, we use XSLT to filter events (serialized in XML format), select which rules to check, and produce different execution data that can be queried by using a query language, such as: SQL, XQuery, or produce a set of instructions (Table 10) that can be executed by the command interpreter component in order to make some actions on a business process.

Table 10: BPath instructions

Instruction	**Description**
<update processid='' instanceid='' variable='' />	Update a value of a variable defined in a business process instance.
<publish > event data </publish>	Publish an event to the ESB
<stop processid='' instanceid=''/>	Stop an instance execution
<resume processid='' instanceid=''/>	Resume an instance execution
< suspend processid='' instanceid=''/>	Suspend an instance execution

3.6.3 Monitoring events

3.6.3.1 Event model

Events share some common features that can be expressed as an event model. Each event has a set of related attributes to express its execution performance and details. Events may be expressed at different levels of granularity, but can define a common ground. By abstracting from the details, we can say that an event is a tuple *e = <type, processid, instanceid, source, destination, timestamp, properties, body>, w*ith the following characteristics:

- Type$\in${NewProcessInstanceEvent, ProcessInstanceStartedEvent, ProcessTerminationEvent, ,…}, a set of event types.
- Processid: The related process id.
- Instanceid: The related instance id.

- Source $\in$ {ProcessEngine, MonitoringSystem, ViewName, …}, the set of sources that generate the events.
- Destination: The destination of the event.
- Timestamp: The time at which the event occurred.
- Properties = {(key, value), …} is a set of properties.
- Body: is the optional body of the event message.

This can be seen as a generic event model that applies to whatever syntax we choose for the serialization of the events.

3.6.3.2 Event producers

We distinguish three main producers of events in our framework (process engine, process and choreography views, monitoring system).

Process engine

The process engine describes the current status of the execution of processes by generating and emitting execution events that indicate certain steps within a running process instance. The generated events may differ according to the used process engines.

Our framework uses Apache ODE [ODE], which is an open source process engine, produces the following types of events as the execution of a process instance progress:

- ActivityEnabledEvent: An activity has been enabled.
- ActivityDisabledEvent: An activity has been disabled (due to dead path elimination).
- ActivityExecStartEvent: An activity has started its execution.
- ActivityExecEndEvent: An activity has terminated its execution.
- ActivityFailureEvent: An activity has failed.
- NewProcessInstanceEvent: A new process instance has been created.
- ProcessCompletionEvent: A process instance has been completed.
- ProcessInstanceStartedEvent: A process instance has been started.
- ProcessInstanceStateChangeEvent: The state of a process instance has changed.

- ProcessMessageExchangeEvent: A process instance has received a message.
- ProcessTerminationEvent: A process instance has been terminated.
- ScopeCompletionEvent: A scope has completed.
- ScopeFaultEvent: A fault has been produced in a scope.
- ScopeStartEvent: A scope has been started.
- VariableModificationEvent: The value of a variable has been modified.
- VariableReadEvent: The value of a variable has been read.

Process and choreography views

Since the views are abstractions over a business process model or service choreography, they ignore or capture some execution events, reinterpret them according to the view perspective, and publish new events to the messaging framework.

For instance, a business protocol which is a view on a business process (represented as an automaton) captures exchanged messages from and to a business process, then publishes some events corresponding to a transition from a state to another state, when receiving or sending of message events occurs.

Monitoring system

In order to perform monitoring tasks, our monitoring system first consumes process events and views events, and then publishes the result of the monitoring as events to the messaging framework. The published events may concern: alert of detected deviations, performance indicators, etc. Our monitoring system is also able to produce some high level events, as it will be explained in the next section.

3.6.3.3 Generating high level events

High level events include events produced as the result of the occurrence of several process execution events, and business events occurring during the execution of a business process or process view, and may have relevance from a business point of view. High level events provide an additional abstraction that can be useful to control monitoring tasks. For example, the verification may be done only when the receipt of an invoice event is received, which allows to avoid doing verification during intermediate steps of process execution. Hence, minimizing the number of verifications during runtime is important in order to enhance the performance of the monitoring system. Also, high level events may encompass information related to several events.

In order to be able to produce high level events form low level events, we need an event correlation mechanism to capture a relation between events occurring at different time during the execution of a process. An emerging and powerful technology to process and correlate events is Complex Event Processing (CEP). But, alternatively to deploy such CEP tools, our monitoring system offers the possibility to produce high level events, by listening to events exchanged through the messaging framework, and by the application of BPath based rules (with XSLT) to create new high level events, as shown in Figure 24.

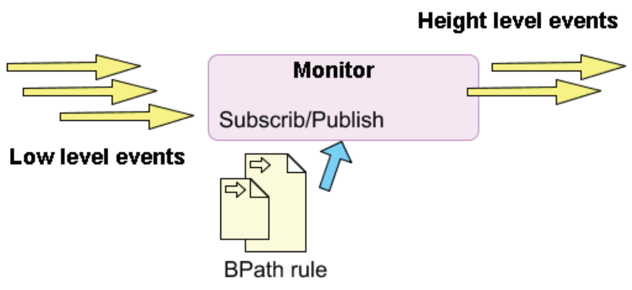


Figure 24: High level event with BPath

Example 6: Consider the process shown Figure 25 of a bank process, where a user sends his information to login in the system, in order to be able to activate other activities defined within the process (for instance, check credit worthiness activity). But to perform this, he should first be allowed by an authorization system.

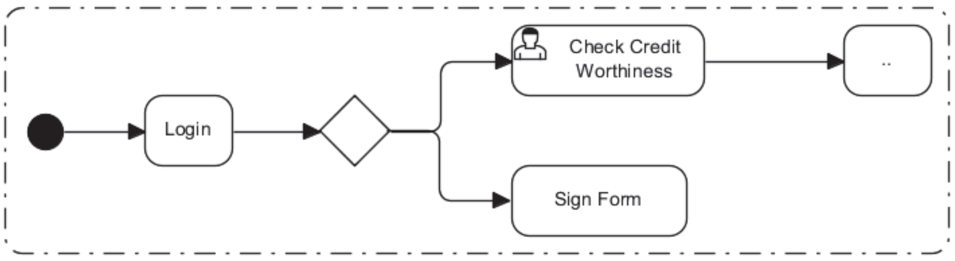


Figure 25: Bank process

For instance, during the execution, the process engine will publish two low level events: one corresponding to the receiving a login message (Listing 6), and the second one to start executing a check credit worthiness activity (Listing 7). Then, thanks to BPath, these two events can be correlated (Listing 8) in order to generate a high level event (Listing 9). The generated high level event will be sent to an authorization system to decide whether the activation of the activity will be allowed or not.

```
<Event>
          <id>1</id>
          <processid>123456</processid>
          <instanceid></instanceid>
          <timestamp>...</timestamp>
          <type>ProcessMessageExchangeEvent</type>
          <source>ProcessEngine</source>
          <properties>
            <Property>
              <key>user</key>
              <value>steve</value>
            </Property>
            <Property>
              <key>role</key>
              <value>CreditBroker</value>
            </Property>
            <Property>
              <key>action</key>
              <value>login</value>
            </Property>
            </properties>
 </Event>
```

Listing 6: Login message low level event

```
<Event>
          <id>...</id>
          <processid>      </processid>
          <instanceid>  </instanceid>
          <timestamp>...</timestamp>
          <type>ActivityExecStartEvent</type>
          <source>ProcessEngine</source>
          <properties>
            <Property>
              <key>name</key>
              <value>check credit worthiness</value>
            </Property>
          </properties>
 </Event>
```

Listing 7: Check credit worthiness low level event

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="utf-8"/>
  <xsl:template name="high-level-access-control-event" match="Event/
type/ActivityExecStartEvent">
    <publish>
      <xsl:variable name="inst" select="$Process[Event/processid=@processid
and Event/instanceid=@instanceid]" />
        <Event>
          <id>...</id>
          <processid>
            <xsl:value-of select="$inst/@processid"/>
          </processid>
          <instanceid>
            <xsl:value-of select="$inst/@instanceid"/>
          </instanceid>
          <timestamp>...</timestamp>
          <type>AccessControlEvent</type>
          <source>MonitoringSystem</source>
          <properties>
            <Property>
              <key>user</key>
              <value>
                <xsl:value-of select="$inst/@user"/>
              </value>
            </Property>
            <Property>
              <key>role</key>
              <value>
                <xsl:value-of select="$inst/@role"/>
              </value>
            </Property>
            <Property>
              <key>action</key>
              <value>
              <xsl:value-of select="Event/properties/Property/value"/>
             </value>
            </Property>
            </properties>
          </Event>
        </publish>
          </xsl:template>
</xsl:stylesheet>
```

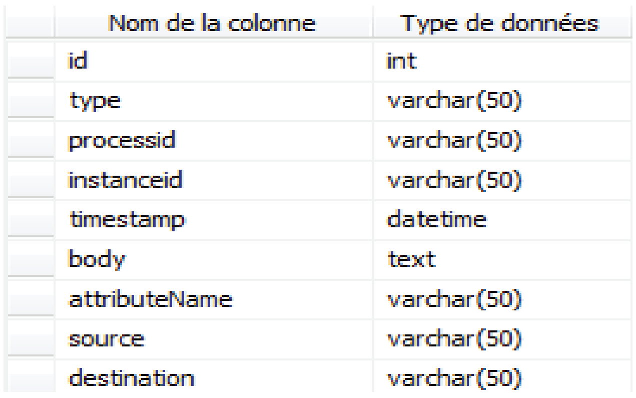
Listing 8: High level event rule

```
<publish>
        <Event>
          <id>...</id>
          <processid>      </processid>
          <instanceid>  </instanceid>
          <timestamp>...</timestamp>
          <type>AccessControlEvent</type>
          <source>MonitoringSystem</source>
          <properties>
            <Property>
              <key>user</key>
              <value>steve</value>
            </Property>
            <Property>
              <key>role</key>
              <value>CreditBroker</value>
            </Property>
            <Property>
              <key>action</key>
              <value>Check credit worthiness</value>
            </Property>
            </properties>
          </Event>
</publish>
```

Listing 9: High level access control event

3.6.3.4 Event log

Events are carriers of information that are essential to any monitoring system. When an event is received by our monitoring framework, it is parsed in order to extract relevant information for monitoring, for instance those related to variable assignments of business processes. This extracted information can be stored in an event log database. Figure 26 shows the event log database schema used in our framework in order to store dynamic attributes and their corresponding values.



Nom de la colonne	Type de données
id	int
type	varchar(50)
processid	varchar(50)
instanceid	varchar(50)
timestamp	datetime
body	text
attributeName	varchar(50)
source	varchar(50)
destination	varchar(50)

Figure 26: Event log database schema

3.6.4 BPath engine

The core component in our monitoring framework is the BPath engine, which includes various components as shown in Figure 27. Before running a BPath expression, it should be compiled, which involves three components:

- Lexical analyzer: which extracts BPath tokens (state variables, global variables, and dynamic attributes), which will be used by the extended XPath engine
- BPath2XPath component translates a BPath expression into an XPath expression according to the mapping defined in Table 11.
- XPath compiler: Compile a produced XPath expression from the BPath2XPath component, which allows detecting the presence of syntactic errors.

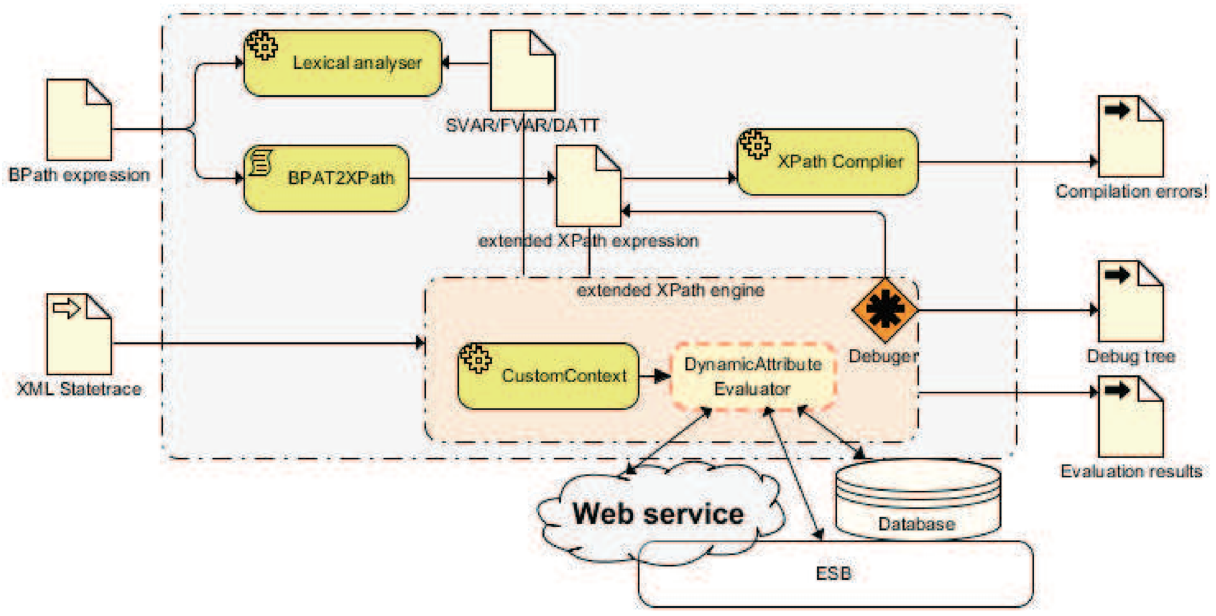


Figure 27: BPath engine

To be evaluated, a BPath expression will be mapped into a standard XPath expression, extended with some custom functions. Our implementation is based on *System.Xml* APIs of the Microsoft .NET Framework SDK. The Table 11 shows how BPath is mapped to XPath 1.0 expression.

Table 11: Mapping BPath to XPath

BPath Syntax	**Mapping to XPath 1.0**
X(φ)	following-sibling::*[1][φ].
F(φ)	(self::* | following-sibling::*) [φ].
G(φ)	not (self::* | following-sibling::*) [not(φ)]).
U(φ, ψ)	$S! (self::* | following-sibling::*) (ψ and not((self::*| preceding-sibling::*) [not (P($S=self::*) → φ))]

B(φ)	preceding-sibling::*[position()=last()-1][φ]
P(φ)	(Self::*| preceding-sibling::*)[φ]
H(φ)	not((Self::*| preceding-sibling::*) [not(φ)])
$S! φ *($S a state variable)*	boundSVAR('S') and φ
π/@q *($q \in FUN \cup ATTD$)*	π/self::*[setValue(@q)]/@crurrentAttributeValue
$S/@q *($q \in FUN \cup ATTD$)*	$S/self::*[setValue(@q)]/@crurrentAttributeValue
@q	getATTD('q')
$x:=T,φ	*setFVAR($x, T) and* φ

So, dynamic attributes, variable assignment and state variable binding inside a BPath expression are implemented through custom XPath functions. The custom functions *setFVAR*, *boundSVAR* respectively assign a value to a global variable, and the current node to a state variable. *setValue* changes the value of the *@currentAttributeValue* attribute, which is a special static attribute accessible only internaly for BPath engine. These custom functions always return the boolean value True. The custom function *getATTD* returns the value of a dynamic attribute. Whenever the XPath engine tries to get the value of a dynamic attribute, it invokes the attribute custom function *getATTD* before, which gives access to the current context of the XPath expression under evaluation that specifies the context state node, which can be used to retrieves the value of the dynamic attribute at the given context state node from different possible data sources.

The class diagram in Figure 28, describes the working of our XPath extension, which can be summarized in the following steps:

- *CustomContext* provides the XPath processor with the custom context for resolution of BPath functions and variables. *CustomContext* implements two key methods, *ResolveFunction()* and *ResolveVariable().*
- When the XPath processor detects BPath custom function, it invokes the *ResolveFunction()* method of the custom context. *ResolveFunction()* returns the appropriate custom function, which derives from BPathCutomFunction. Then the XPath processor calls the *Invoke()* method on this custom function at runtime with the provided arguments.
- When the XPath processor detects a BPath variable, it invokes the*ResolveVariable()* method of *BPathCustomContext*. *ResolveVariable()* returns the appropriate Bpath custom variable that derives from BPathCustomVariable. Then, the XPath processor calls the *Evaluate()* method on this custom variable at runtime.

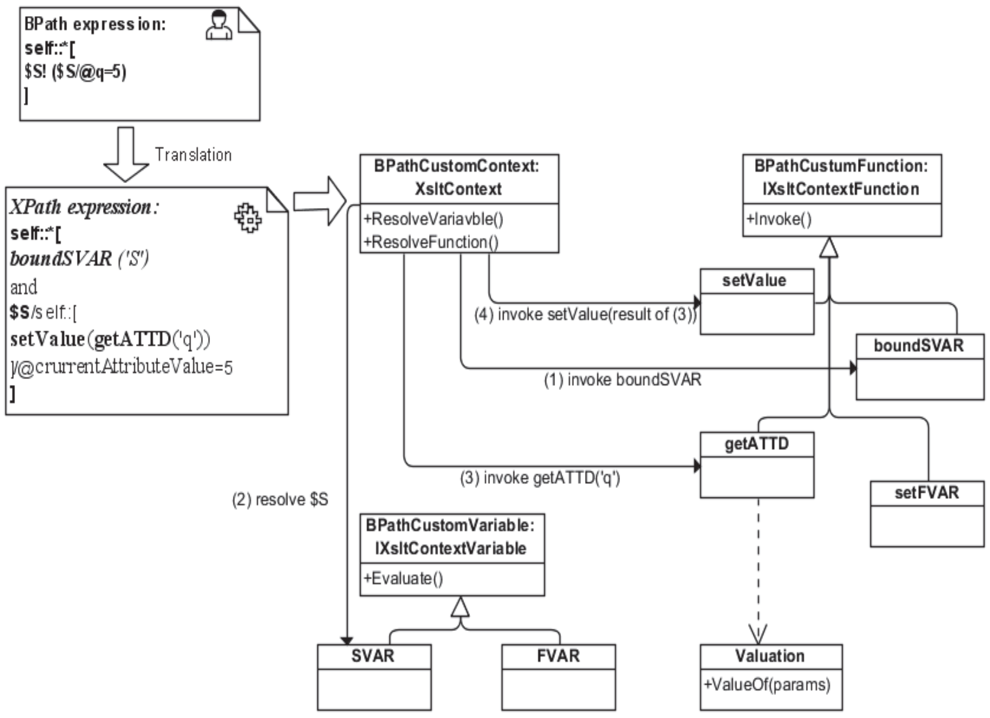


Figure 28: BPath custom context

So, when the *getATTD* custom function is invoked during the evaluation of a BPath expression, it gives access to the current state information: *processid*, *instanceid*, source, and the timestamp, in addition to the attribute name. Then, we use this information to query an execution data source in order to extract the value of this dynamic attribute.

Concretely, one possible implementation is to use an SQL procedure to query an event log database, in case where the event log is a local execution data source, or by invoking a web service method for a remote execution data source. Listing 10 shows an SQL procedure returning the value of a dynamic attribute from an event log database.

```
PROCEDURE [dbo].[GetAttribute]
@processid varchar(50),
@instanceid varchar(50),
@source varchar(50),
@attributeName varchar(50),
@timestamp datetime
AS
BEGIN
 -- Return the value of a dynamic attribute
SELECT body from trace where processid=@processid and
instanceid=@instanceid and source=@source and attributeName=@attributeName
and timestamp<=@timestamp order by id desc;
END
```

Listing 10: *GetAttribute* procedure

3.6.5 Performance evaluation

In order to evaluate performance of our monitoring system, we conducted three simulations: internal business process monitoring of the WatchMe business process, business protocol monitoring of the CRS system, and monitoring of WatchMe services choreography.

All simulations are performed on an M2300 dell computer, with Intel(R) Core(TM)2 Duo CPU 2.00GHz, and 4.00 GB Memory(RAM), using Windows Vista Business edition. In our runtime framework, we made use of: Apache Tomcat [TOM] as an application server, Apache ODE as process engine, and activeMQ [AMQ] message broker, and we used NMS API [NMS] (*.Net Message Service API*) as a .NET interface for activeMQ. The messages sent to each business process have approximately the size of 1 KO. They are generated by soapUI

Pro 3.6.1 [SUI] testing tool. The event log is implemented using Microsoft SQL Server 2008 Express Edition [SSE]. A dynamic attribute takes, approximately, 3 to 5 millisecond to be loaded from an event log database.

For each simulation, we have chosen to evaluate the property requiring more evaluation time. Figures 28-29-30 show the results of these simulations, the x-axis corresponds to the number of execution events, and the y-axis in the left side plots the total execution time including the time needed for loading dynamic attributes. The y-axis in the right side plots the execution time of BPath engine without the time needed to load dynamic attributes.

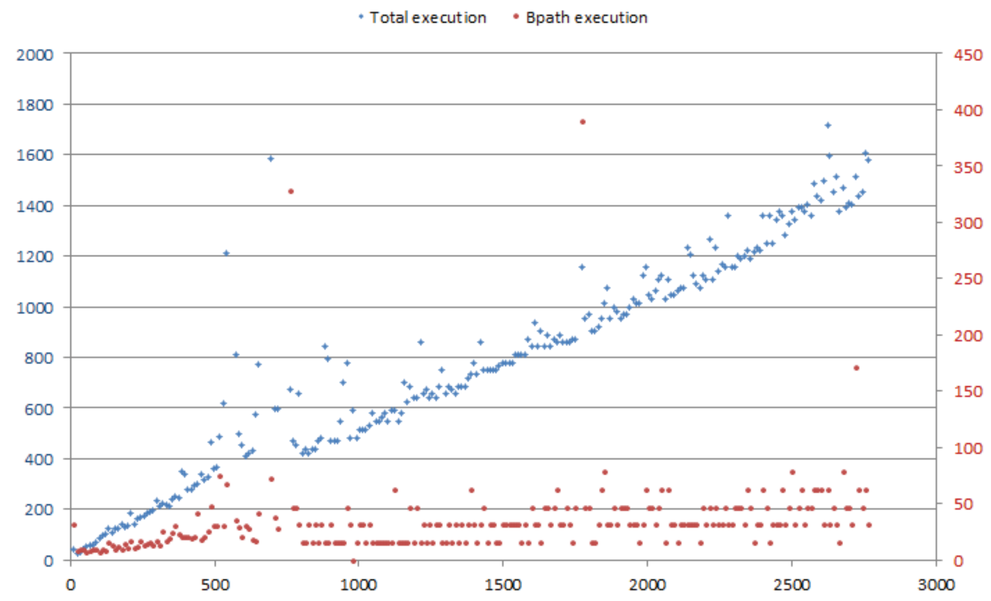


Figure 29: Pay per view compliance rule from WatchMe scenario

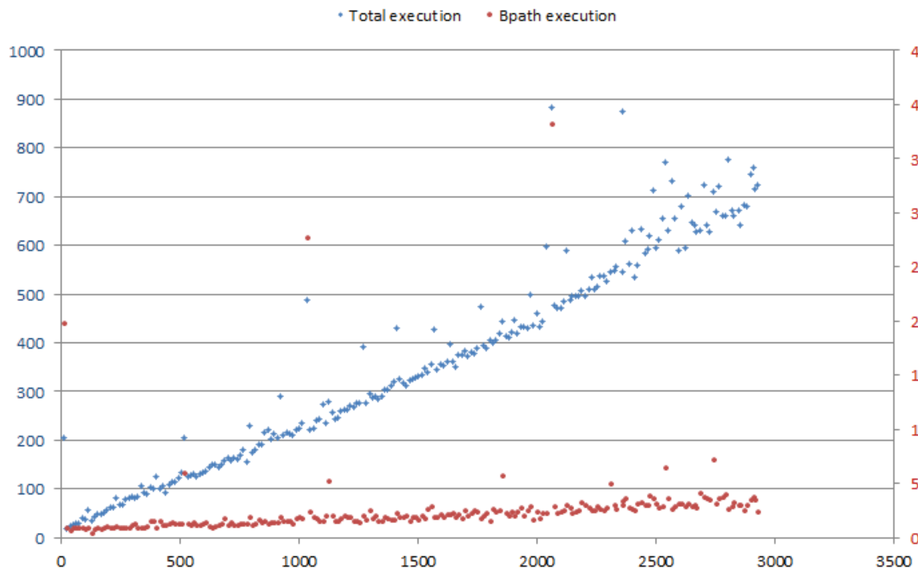


Figure 30: Filtering customers' compliance rule from the CRS scenario

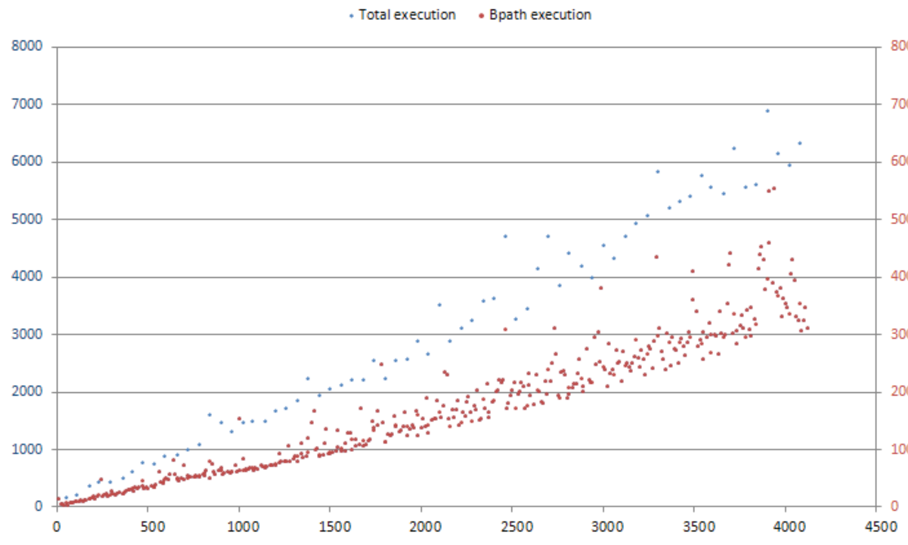


Figure 31: WatchMe Choreography composition permission

3.7 Related work

Several approaches have been proposed in the literature for web services monitoring, here we discuss some academic research approaches that are related to our work. The authors of [BG05] propose WSCoL Language for specifying constraints on execution by defining a set of monitoring rules. As our monitoring language BPath, WSCoL provides the necessary constructs to define both functional and non-functional properties, with the capability of setting the degree of monitoring at runtime such as: validity time frame, priority, and set of certified providers for which monitoring may be omitted. Also, it enables specifying expressions over the process variables and supports set of built-in functions, logical and mathematical operators, and quantification. This work was extended in [BBG+07], by adding support for specifying and checking temporal properties at runtime. This approach shares with ours the ability to specify both functional and non-functional properties in the same language. However, BPath has the advantage to be more expressive by providing both hybrid logic, past and future temporal logic operators, and the ability to perform monitoring over different views.

The authors in [MS04] consider monitoring web services as a problem of verification of requirements at runtime. This consists in checking if a process behaves in conformance with its specification, which states a set of behaviors that the process' services must display during their enactment, as well as assumptions on these behaviors. In this work, all the behaviors and assumptions were stated as Event Calculus predicates, a logic-based formalism used for representing actions and their effects on some variables called fluent. Event Calculus encompasses some constructs that express complex situations, using time variables, universally or existentially quantified, and implications between predicates. The monitoring consists of checking the messages sent between the different services against the stated behaviors and assumptions.

This approach needs a prior transformation of the business process specified in BPEL to event calculus predicates to be fully exploitable. An execution is said to be not conform, with regards to the specifications, when its entailed events are logically inconsistent to the behaviors or assumptions. The monitoring framework was implemented as a toolkit for monitoring service compositions specified in BPEL. The logs generated by the process engine were used to identify the events and update the corresponding formula templates in the monitors. In order to evaluate and validate the presented approach, the authors set up a comprehensive benchmark with many tests and generated events based on a simple case study parameterized by the frequency of events and the scale of the involved components.

This approach proposes interesting features, but we have two remarks to make on this approach: the first is on the real ability to transform a BPEL process into event calculus logic, which is not provided exhaustively in this work, and second, the representation of the behavior of a business process seems to be unreadable specification. Compared to automata based representation, which is the case for instance in our Business protocol automatically generated from business process, which is more readable specification, even though it represents only the external behavior of a business process.

In [BEM+07, BE07] Beeri et al. address the issue of monitoring business processes specified in BPEL. A visual language, called Business Process Query Language (BPQL), with query capabilities over BPEL processes was proposed, for easing the formulation of monitoring queries in the same way that graphical notations help BPEL designers generate specification code, using dedicated icons for each activity. Queries are written via a visual editor, and user can draw the query patterns that (s)he wishes to monitor from scratch, using a drag-and-drop items palette. Another interesting feature is that this graphically expressed query is automatically translated to a BPEL process, and deployed in the same framework as the monitored business process. This approach inspired us in the beginning of our work in order to build a monitoring language that easily integrate with SOA framework by reusing existing SOA language.

In [BMH07] Benbernou et al. address the problem of runtime monitoring of compliance of the privacy agreement defining the user's privacy rights and their possible handling by the service provider. This problem goes beyond the traditional access control management and defines the necessity to face the usage control management of the private user information.

The proposed solution presents the privacy agreement model, where the requirements on the management and handling of the privacy data are specified, together with the approach for runtime compliance monitoring. The privacy properties are given in the form of data-rights (authorized operations) and data-obligations (required actions) together with their validity frames and specified in the extended WS-Agreement specification. The set of privacy requirements, privacy units, and typical misuse scenarios are defined based on these properties. The formalism adopted for the representation of the privacy units and misuse relies on linear temporal logic. For the monitoring purpose, privacy units are transformed into state machine representation that correspond to the evolution of the privacy data management and define both correct and incorrect usage of these data.

Although this approach targets the monitoring of privacy, which is not addressed by our approach, some element are shared with this approach, as the running of business protocol (the privacy unit state machines in the authors' approach) in parallel with the service execution, in order to get a special execution view of the global execution of the service.

A closer work to our approach for supporting the monitoring of temporal logic properties at runtime is the work done in [HV08, HV09], in this work the authors propose an approach for monitoring web service choreography by means of the XQuery engine. Linear temporal logic properties are translated into an equivalent XQuery expression, and then evaluated over XML message traces representing the choreography. Messages trace are represented as following sibling xml elements, then the previously translated temporal logic operators to XQuery expression are evaluated in a streaming way as the messages are captured from the choreography.

If we share the common interest for using XPath based language in order to perform monitoring, the provided translation to XQuery can only be applied to monitor a sequence of states, where each state corresponds to exactly one XML message. Then, proposed temporal logic operators translated to XQuery cannot be reused to monitor business process where each process state may encompass messages (variables) that are captured at different time during the execution, which is achieved in BPath through dynamic attributes. Besides the fact that we address more aspects of monitoring (internal business process, choreography, business protocol), and providing more features, the performance of our monitoring engine is better due to the special structure we consider to evaluate BPath properties (state trace is a small size xml document). But as the authors use existing streaming XQuery engine, they have the advantage to evaluate their properties (future LTL operators) in streaming way, which is actually not supported by our implementation.

3.8 Summary

In this chapter, we presented our view-based monitoring approach. First, the general operation of our approach and capabilities of monitoring are presented. Then, a detailed presentation of BPath monitoring language was given. Finally, through case studies, we have shown how our monitoring framework can be used to monitor business and services choreographies from different views.

Part 4. An SOA Domain Specific Language Tool For Access Control

Outline

Security is an important compliance concern that needs specific tools in both design time and runtime. In this chapter, we will present a domain specific language tool and a framework for access control. Unlike the previous chapter, we will focus here on building a tool to enable automatic generation, rather than manual writing of compliance rules. We start by an introduction to security in the context of SOA based systems and an overview on the model driven architecture. Then, we present in detail our access control domain specific language tool, and we close the chapter by presenting some related work.

4.1 Introduction to SOA security

Security is an important concern for any software system and particularly for SOA based systems, where the opening to new boundaries accentuates the need of security. A lot of tools and technologies have been developed in order to take into account the changes and the new requirements induced by SOA. However, security requirements for SOA systems remain basically the same ones as for traditional system:

- Authentication: It is the evidence that an entity is the one claimed. Authentication functionality of system security is responsible for making sure that a user or a service is who they claim to be. Sometimes, the word identification is used instead of authentication to mean the same thing.
- Authorization: It is the granting of rights, which includes the granting of access based on access rights. Authorization functionality is responsible for making decisions whether or not to permit action on resource. Authorization cannot be enforced without reliable authenticating functionality of a system. Before access rights decisions can be made, it is critical to identify a user or a service.
- Confidentiality: Protecting secrecy of sensitive data.
- Data integrity: Detecting data tampering and making sure neither the sender nor the receiver can deny the message they sent or received.
- Privacy: Making sure the application does not violate the privacy of the users.

Assuring security in SOA based system is particularly challenging, because it is difficult for an application designer to foresee all possible interaction scenarios between services. Thus, there is a need for a new security approaches for SOA, which should be in alignment with the philosophy of SOA [KC08]. That is, any SOA security solution should keep services open as much as possible, easy to use as much as possible, technology boundaries should not be re-erected, interoperability should not suffer because of security, and SOA security infrastructure should be accessible independently from any technologies.

A fundamental key for the success of an SOA security solution is the use of existing security SOA standards. Among SOA standards for security, we give a description of some of them below.

XML Key Management Specification

XML Key Management Specification (XKMS) [HM05] is a W3C specification of validation and recording of public key in a joint way with XML signature. XKMS consists of two complementary standards: XML Key Registration Service and XML Key Information Service specifications. Together, they allow for the integration of a number of security technologies, including digital signatures, certificates, and revocation status checking.

Extensible Access Control Markup Language

eXtensible Access Control Markup Language (XACML) is an OASIS standard [XACML], which defines a language for access control, administration of rules and security policy of information systems. XACML is often used to ensure the function of authorization in SOA architectures. XACML defines a language capable of expressing policy statements for a wide variety of information systems and devices. The approach taken by XACML is to draw together long-established techniques for access-control and then to extend a platform-independent language (XML) with suitable syntax and semantics for expressing those techniques in the form of policy statements. The XACML specification consists of two related vocabularies: one for access control and one that defines a vocabulary for request and response exchanges.

Security Assertion Markup Language

Security Assertion Markup Language (SAML) [SAML] is an OASIS standard, which addresses the issue of allowing unique authentication on the Web that is known as Single Sign-On (SSO). The objective is to allow a user or a service to be authenticated only once on different sites, hence avoiding the exposition of too confidential information. SAML provides mechanisms for both authentication and authorization processes. Both request and response message formats are defined to facilitate the transmission of necessary credentials within a Web service activity. SAML provides components to express any assertion about a subject to be authenticated, a component for the authentication request-response protocols, a binding component (the SOAP-over-HTTP method of transporting SAML requests and responses), and profiles component for embedding and extracting SAML assertions in a framework or protocol.

XML-Encryption

XML Encryption [RE09] is a W3C standard for encrypting data. The data may be arbitrary data including both XML and non-XML data. The XML-Encryption proposes a standard model for encrypting data as well as a means of communicating information essential for recipients to decrypt the contents of received messages. The encrypted data is represented as XML document. XML Encryption is not intended to replace or supersede SSL (Secure Sockets Layer). Rather, it provides a mechanism for security requirements that are not covered by SSL.

XML Signature

XML Signature [YRH+09] specifies XML syntax and processing rules for creating and representing digital signatures. XML Signatures can be applied to any digital content (data object), including XML data. It allows establishing credibility within a message, as they assure the recipient that the message was in fact transmitted by the expected partner service, and that the message contents were not altered since it was sent. XML signature is used within various specifications such as SOAP, and SAML.

Web Service Security

Web Service Security (WS-Security) [WSS] is an extensible specification. It proposes a standard set of SOAP extensions that can be used when building secure Web services to implement message content integrity and confidentiality, which can be used in conjunction with other Web service extensions and higher-level application-specific protocols, in order to accommodate a wide variety of security models and security technologies, including Public Key Infrastructure (PKI), Kerberos, and SSL. Specifically, this specification provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies.

Despite the fact that the adoption of SOA security standards widely contributes to the successful of any security solution for SOA systems, it is still insufficient. In fact, the development of an SOA security solution can be cumbersome and complex, probably due to the number of used standards, and often the required configurations of the underlying execution platforms. Thus, there is a need for automatic generation of SOA security policies, and required configurations. An important ingredient is the ability to specify security policies at different levels of abstraction, which should be convenient for both technical experts as well as business experts.

4.2 Model driven architecture

Manually writing applications code may be both hard and boring, due to the complexity, or repetitive nature of the tasks to be achieved, which leads to the non-respect of application deliverance time contracts, and coding errors.

Thus, there is a need to reduce application development time, and the number of coding errors. To achieve this goal, the Object Management Group (OMG) proposes the Model Driven Architecture (MDA) [MDA]. MDA insists on the importance of models in the software development process. The aim is to make activity of modeling software systems as central concept on software development process [KWB03]. The mains principle of MDA is the elaboration of different models and the successive transformation of these models until generation of concrete realization of the system [Fra03]. The Figure 32 illustrates this MDA pattern.

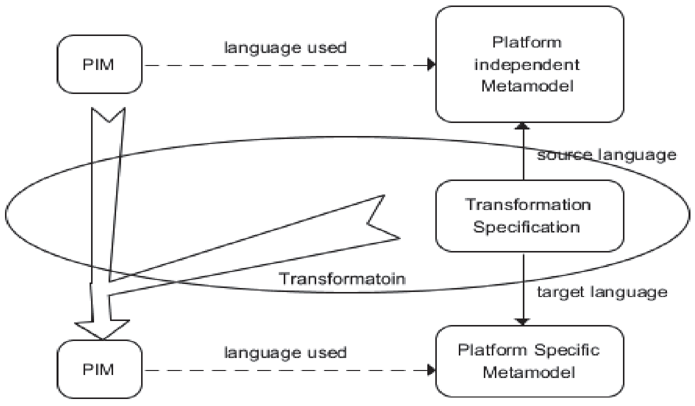

PIM
language used
Platform independent Metamodel
source language
Transformation Specification
Transformatoin
target language
PIM
language used
Platform Specific Metamodel

Figure 32: Meta-model Transformation

The first model that MDA defines is a model with a high level of abstraction that is independent from any technological platform used to implement it. This is called a Platform Independent Model (PIM). The PIM will be transformed into one or more Platform Specific Models (PSMs). A PSM is tailored to specify a system in terms of the implementation constructs that are available in one specific implementation technology.

In MDA, the transformations from model to model or from model to code are always executed by tools. Many industrial Model-Driven Development tools are available, which allow users to build a model of their problem, often using a graphical language such as the Unified Modeling Language (UML) [RJB99], and then generate concrete implementation or configuration of the desired system. Using MDA for software development process will offer several advantages, such as productivity, portability, and interoperability. It also makes easier software maintenance and documentation.

4.3 A DSL based approach for SOA security

A DSL is a custom language that targets a small problem domain, which it describes and validates in terms that are native to the domain [CJK+07]. DSL have the potential to reduce complexity of software development by raising the abstraction level towards an application domain. So, applications are directly specified using domain concepts. In contrast to a general-purpose programming language, such as C or Java, or general-purpose modeling language such as UML, domain-specific modeling languages are customized to a particular application domain, and allow programming by specifying the solution directly using domain con01cepts. Application domains are for instance: banking domain, military simulation, billing, and so on.

We propose an approach that is based on using of domain specific language and define a generic architecture (Figure 33) that can be reused to cover several security requirements. Our architecture follows the separation between business and technical levels, by proposing different usages of security DSL at each level. Through the low level DSL, a security expert creates a set of security patterns that can be reused as bricks to specify security policies, and a set of security transformation modules, which specify how a security pattern can be transformed into an executable security policy, and link it to necessary technical platform. High level DSL allows a business expert to specify security requirements in an abstract way by reusing predefined security patterns. Even if the syntax of a security DSL is created by a security expert, this should be done under commitment with business expert, in order to make its use closer to the application domain.

In fact, from a very general brids-eye perspective, a security pattern is a solution to a problem that arises within a specific context [SFH+06]. For instance, separation of duties pattern is a generic solution that allows avoiding the concentration of power in one hand, by preventing her to play different roles on a given resource. In our Security DSL, each security pattern has a graphical shape and a validation rule, determining to which security elements it can be connected. Security elements are any piece of information needed to define a security requirement (e.g., users, activities, and resources of a given system are considered as security elements).

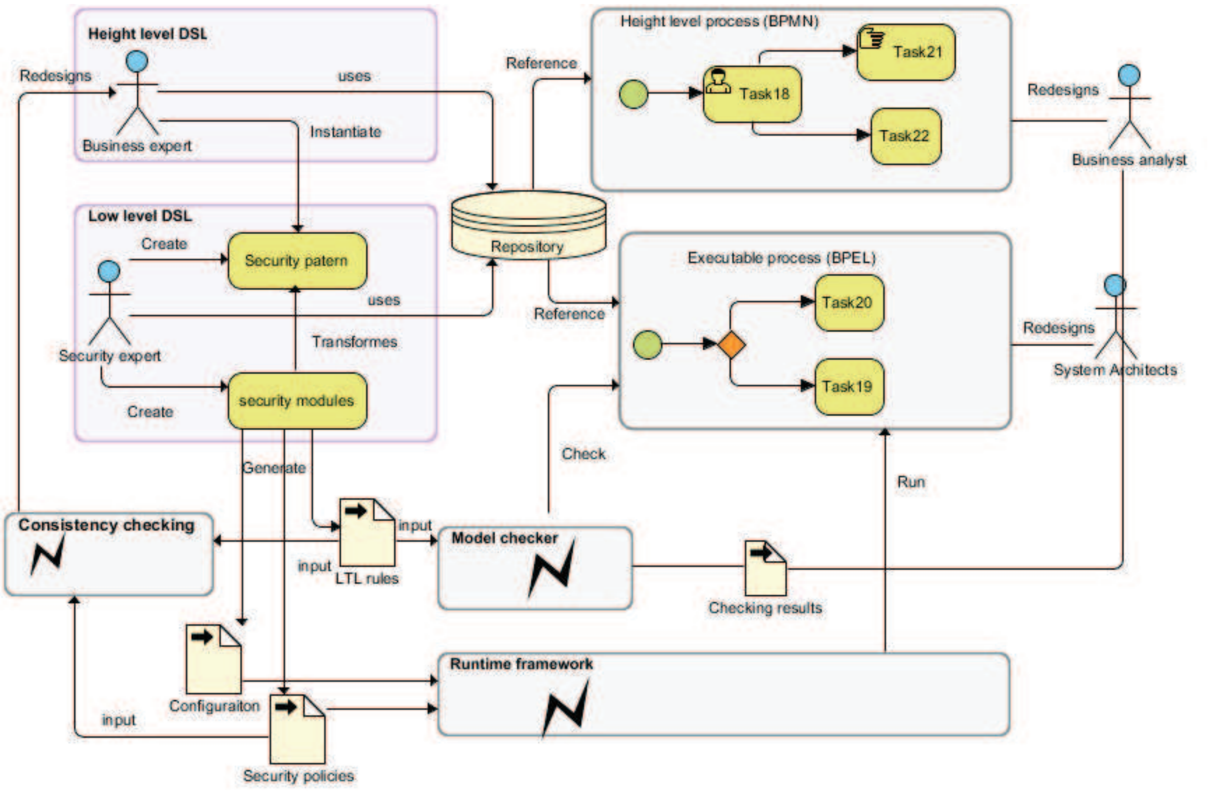


Figure 33: DSL based architecture for SOA security

Each security element is either predefined within the security DSL meta-model or is in a model repository. As an example of model repository, we used the Model-Aware Repository and Service Environment (Morse) [HZD09] developed within the COMPAS project, which is a service-based environment for the storage and retrieval of models and model-instances at both design- and runtime. It provides versioning capabilities so that models can be manipulated at runtime and new and old versions of the models can be maintained in parallel. Morse allows to explore model elements, and also to discover relationships between different model elements such as those that may exist between a BPMN model and its corresponding

BPEL process model. By using a model repository as an intermediary between security DSL and business process model, we support the idea of separating between user interfaces for defining security policies and those for business process modeling, in order to avoid what can be considered as business process pollution. Some advantages that are expected with our approach are:

- A clear view of the security requirement.
- Policies are directly specified using domain concepts.
- Only valid relations between the domain concepts exist.
- Separation between business and technical levels.
- The productivity and the software quality can be improved because they are easier to maintain and the generated code does not contain bugs.
- The generated code can be tailored for the particular technology.
- Technical aspects are not shown to business experts.

A security policy may define several rules that may lead to incoherence of the policy. Thus, there is a need to check consistency of a policy, in case an inconsistency is identified, it must be reported, in order to redesign the policy. Security policy may target a design time or runtime framework, for instance: temporal logic may be generated from a security policy, then checked by a model checker on the system model.

4.4 Access control DSL

Access control is a fundamental security requirement. It is used to control which user or subject (principal) has access to which resource in a system. Several access control models have been proposed. Probably, one of the most well-known access control model is the Role Based Access Control (RBAC) [FK92]. The Figure 34 shows the RBAC model.

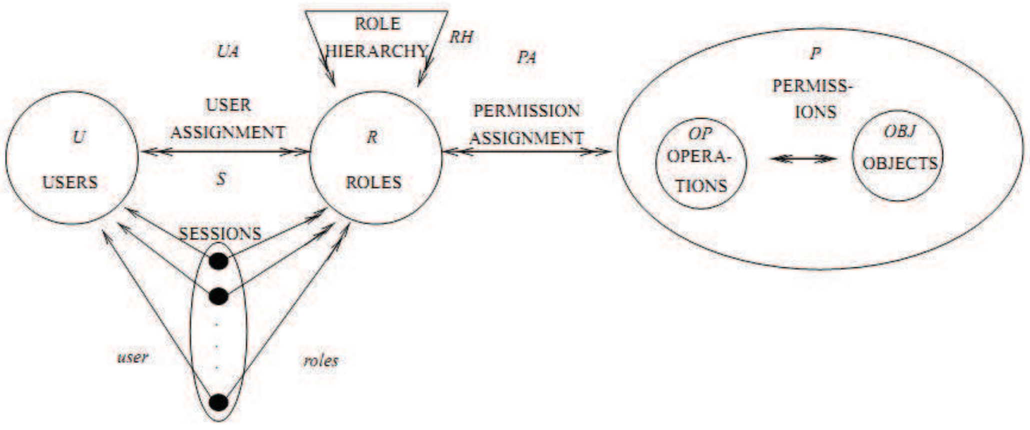


Figure 34: RBAC model.

In RBAC, permissions are based on the functional roles of users or subjects within the system. Since the roles are relatively of long duration in an organization, the administration of RBAC is simpler and more powerful.

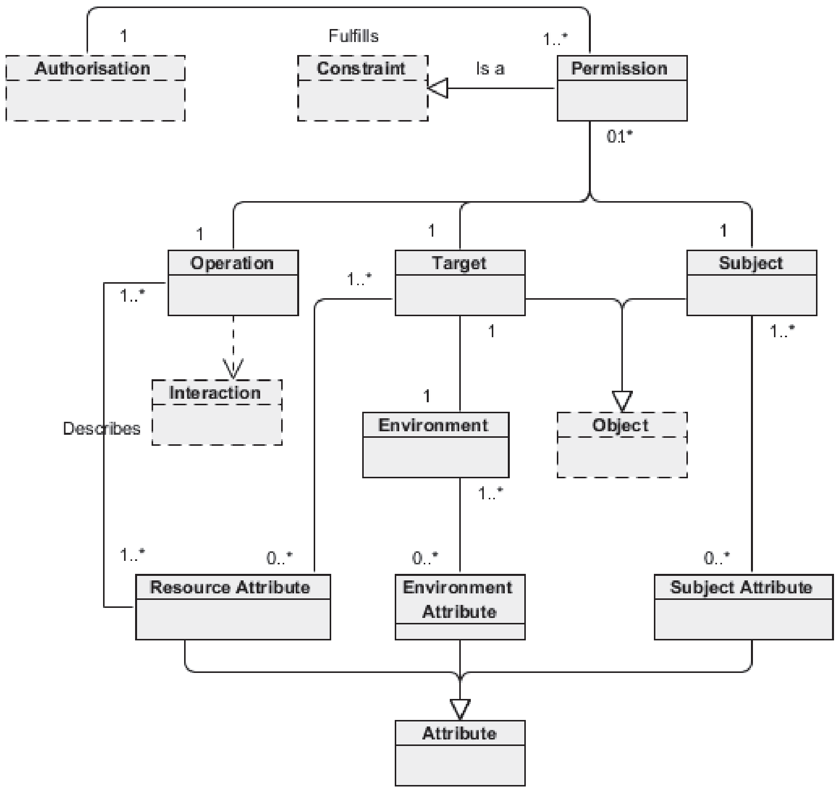


Figure 35: ABAC model [WMS+09]

In Web services environments, the traditional "identity-based access control models" such as RBAC, where subjects and objects are usually identified by identities are not relevant. Thus, Attribute-Based Access control Model (ABAC) [SH06] have been proposed (Figure

35), which can be seen as the most comprehensive access control model [WMS+09]. The access control policy in ABAC specifies what attributes should be used in order to access to a certain service.

Our access control DSL is built on top of ABAC model, and aims at offering an easy way for specification and generation of access control policies. Existing XML-based languages for specifying access control policies are expressive, but policy definition is cumbersome. Thus, in order to make the specification of access control policies easier, we propose an abstract graphical DSL, which is domain specific language with graphical notation like UML for access control. Access control policies specified using our DSL will be transformed into a low level language such as XACML or temporal logic.

We distinguish two users of our DSL: from a low level, a security expert creates a list of access control patterns, and creates a custom transformation module in order to generate access control policies in some desired language. From a high level, a business expert uses the access control elements defined within the low level DSL in order to specify authorization in abstract way, without worrying about technical aspects such as the targeted platform and language.

Figure 36 shows the model of our access control DSL, which defines several access control elements. Each access control element has a graphic shape and may be of different types: subject, operation, target, policy, transformation module, and target language.

Policy defines a set of rules in terms of access control patterns, different type of policies are predefined within our access control DSL:

- Deny overrides: If any authorization pattern evaluated to deny, then the final authorization decision is also Deny.
- Permit overrides: If any rule evaluates to permit, then the final authorization decision is also Permit.
- First applicable: The result of the first relevant authorization pattern encountered is the final authorization decision as well.
- One applicable: It should be only one applicable authorization pattern, which is the final authorization decision of the policy.

An access control pattern defines an authorization rule that determines the authorization decision i.e. permit or deny access to a given resource. It implies three security elements, which are: (1) the subject requesting the access to a resource, (2) the name of the requested operation and (3) the target of the request i.e. the requested resource by the subject. These three security elements may be specified as a target in an authorization request, in case a matching is established between the request and the pattern regarding these three elements, the pattern becomes applicable i.e. the rule defined within the pattern will be triggered.

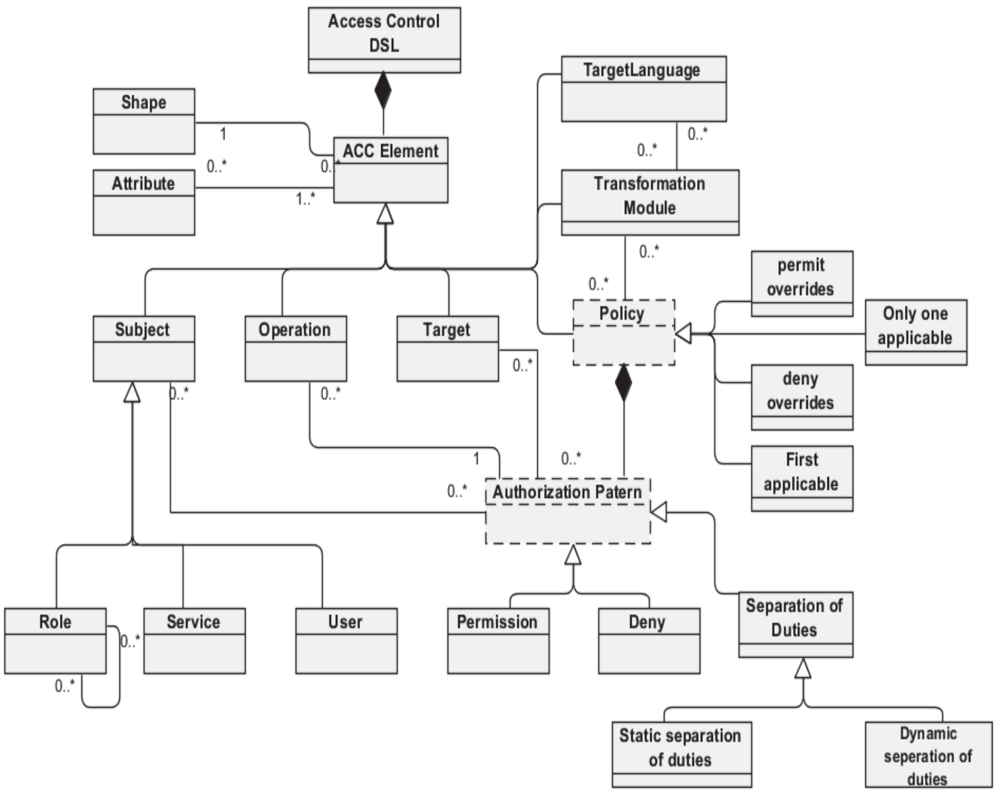


Figure 36: Access control DSL model

An access control policy may be associated with a transformation module, which specifies how to transform the policy with its connected elements into a specified target language. For example: a policy associated with an XACML transformation module indicates that this policy should be transformed into an XACML policy. Each access control has a graphical shape, as it will be presented further. An operation shape is graphically contained in the graphical shape of an authorization pattern. The graphical shape of authorization pattern is contained in the graphical shape of a policy.

The used graphical notation in our domain specific language is important in order to enhance the usability of DSL by making it closer to the application domain. In contrast to the graphical notation of other modeling languages as UML for instance, which propose a fixed graphical notation, we have a total freedom to choose the notation, which is more appropriate for the application domain and may change over time.

4.5 Access control specification

Our access control is a C# based tool that is deployed as a visual studio plug-in. It comprises a main interface (1) for creating different access control policies by instantiating the access control DSL, which can be done by a drag and drop of a set of graphical shapes from the toolbar (2) into the DSL main interface, which includes three categories of elements:

- Abstract security elements: e.g., policy, permission, subject, operation, etc.
- Application domain element: this category includes those elements related to the application domain, such as bank domain; these elements can be imported directly from the MORSE repository.
- Technical elements: Elements making a reference to an existing technical component, such as those responsible of code generation, e.g., XACML module, which transforms a specified graphical policy to an XACML policy. These kinds of elements can take place in the panel (3) of the main interface.

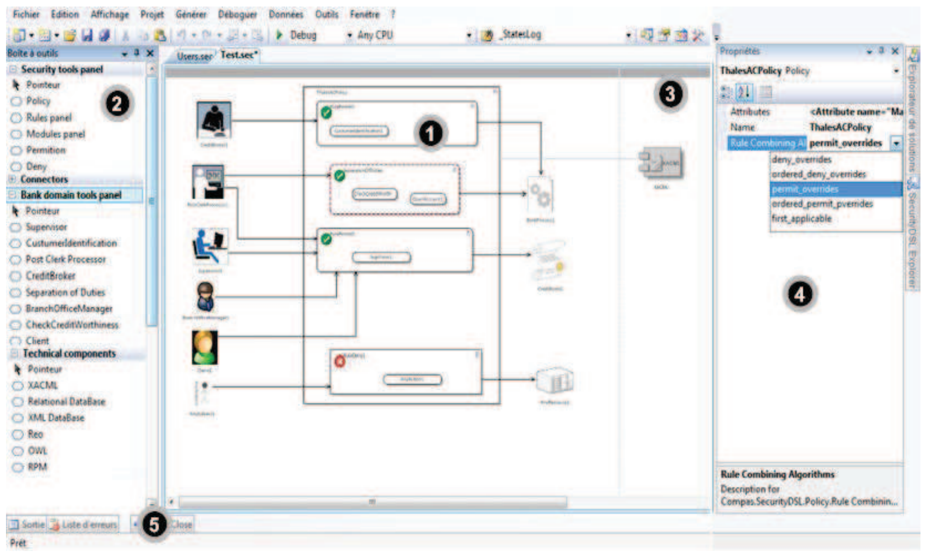


Figure 37: Access control DSL tool

There is also a possibility for creating a hierarchy of roles (Figure 38), and specifying how and where this hierarchy should be serialized and stored, by associating a transformation module.

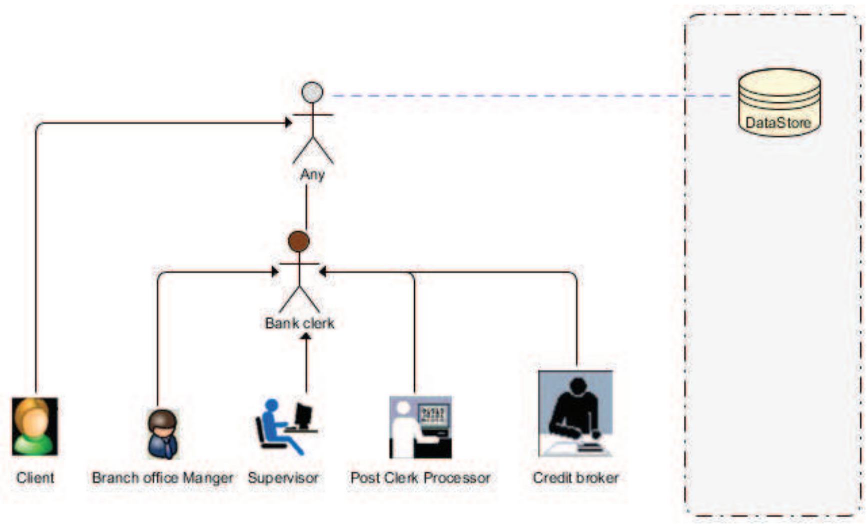


Figure 38: Role hierarchy interface

4.5.1 Access permission and deny

A permission pattern implies three types of security elements: subjects, operations and targets, this means that all specified subjects have the permission to activate all specified operations on all specified targets. Figure 39 illustrates a permission pattern, which authorizes a *Credit Broker* to activate a *customer identification* operation on a *Bank Process*. In the same way, a deny pattern (Figure 40), means that all specified subjects are not authorized to activate specified operations on the specified targets.

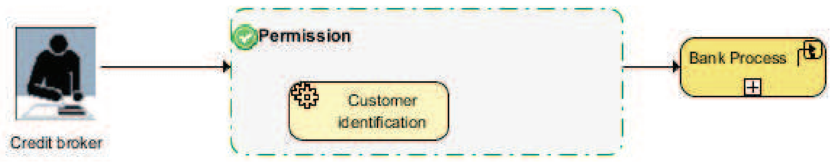


Figure 39: Permission pattern

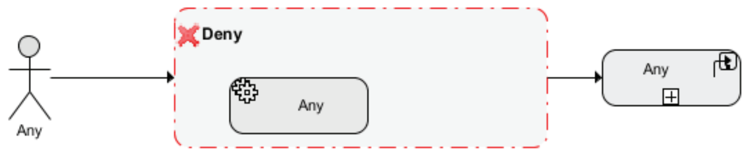


Figure 40: Deny pattern

When access control patterns are evaluated, they can produce different authorizations decisions, as it is the case when the two previous patterns are evaluated. The first one shown in Figure 39, allows a *credit broker* to perform a *customer identification* operation on the *bank process,* but the second one in Figure 40, prohibits to any subject whatever it is, to activate any operation on all resources. This decision conflict can be resolved by referring to the type of the policy containing these two patterns, which determines which will be the final authorization decision. For instance, as shown in Figure 41, the final decision is to permit *customer identification* on the *bank process* when the requester is a *credit broker.*

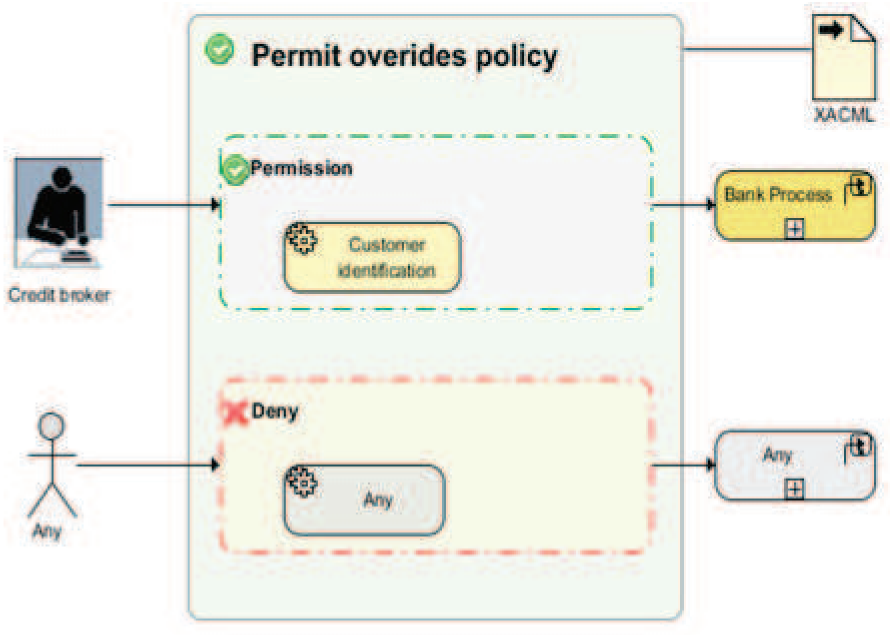


Figure 41: Permit overrides policy

The policy is linked with an XACML transformation module, which means that this policy should be automatically transformed to an XACML policy document.

4.5.2 Separation of duties

Separation of duties is a design principle for the protection of information in computer systems and it is a mechanism to control fraud and error and ensure the consistency of the data objects. It can be either static or dynamic.

4.5.3 Static separation of duties

Static separation of duty enforces conflict of interest policies. Conflict of interest arises as a result of the simultaneous assignment of two mutual exclusive permissions or roles to the same subject.

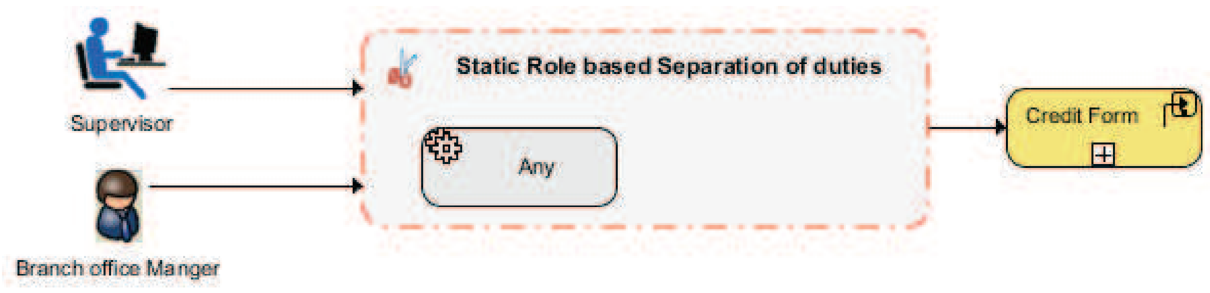


Figure 42: Static separation of duties

Static based separation of duties is ensured by a static validation of the designed access control DSL instance, for example: only a *supervisor* or *branch office manager* will be allowed to sign a *credit form*, and the DSL tool should deliver a message error when a *business expert* tries to authorize both *supervisor* or *branch office manager* to sign a form as it is shown in Figure 43.

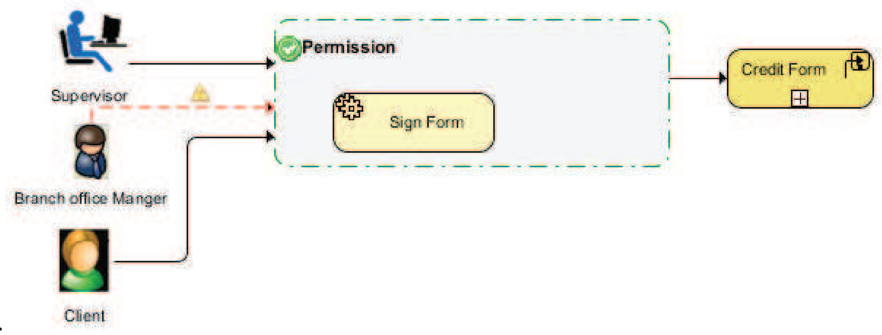


Figure 43: Violation of static separation of duties

4.5.4 Dynamic separation of duties

Dynamic role based separation of duty defines that two mutual exclusive roles must never be activated by the same subject simultaneously at runtime for a given operation, and a resource. This means that two dynamically mutual exclusive roles may be assigned to the same subject. However, a given subject will be allowed to activate at most one of its dynamically mutual exclusive roles at the same time during runtime for the specified operations and resources. So, dynamic separation of duty provides more flexible rules regarding separation of duties.

For instance, the dynamic separation of duty pattern shown in Figure 44 allows a user to play a *Supervisor* or a *Post Clerk Processor* role, but not both, in the same instance of the *Bank process* for opening an account or checking credit worthiness.

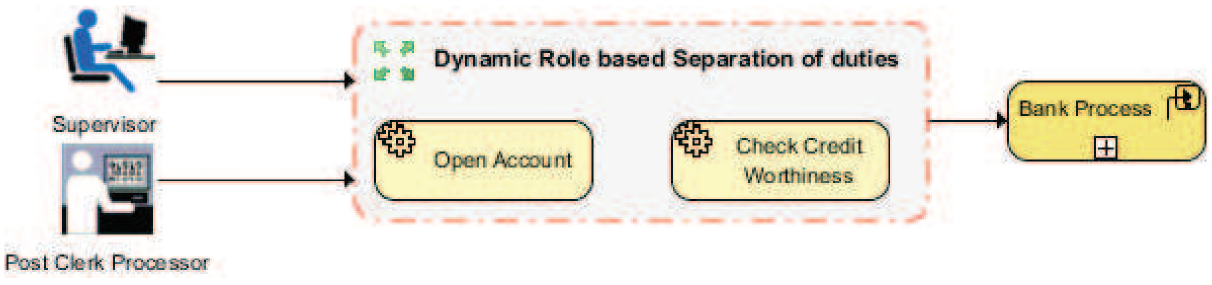


Figure 44: Dynamic role based separation of duties

4.6 XACML generation

In order to enforce an access control policy designed with our access control DSL, it should be translated into an existing language that can be evaluated by some access control framework. This translation can be done automatically by means of a transformation module. The transformation proceeds in two steps: in the first step, the access control DSL model is parsed using the T4 language, in order to generate an XML document capturing the security elements and their relationships as defined within the DSL tool. Then, in the second step, by applying an XSLT template, the resulting XML document may be transformed into various access control languages.

We have implemented a transformation module in order to transform our graphical DSL to XACML. The generation of XACML policies can be done according to a predefined mapping between our access control model and XACML model as shown in Figure 45. This mapping establishes a correspondence between elements defined within our access control DSL model and those of the XACML model, and most of these correspondences are of type one-to-one i.e. they can be translated directly into theirs corresponding XACML elements. However, the translation of an access control pattern implies several XACML elements, which can be specified through an XSLT template.

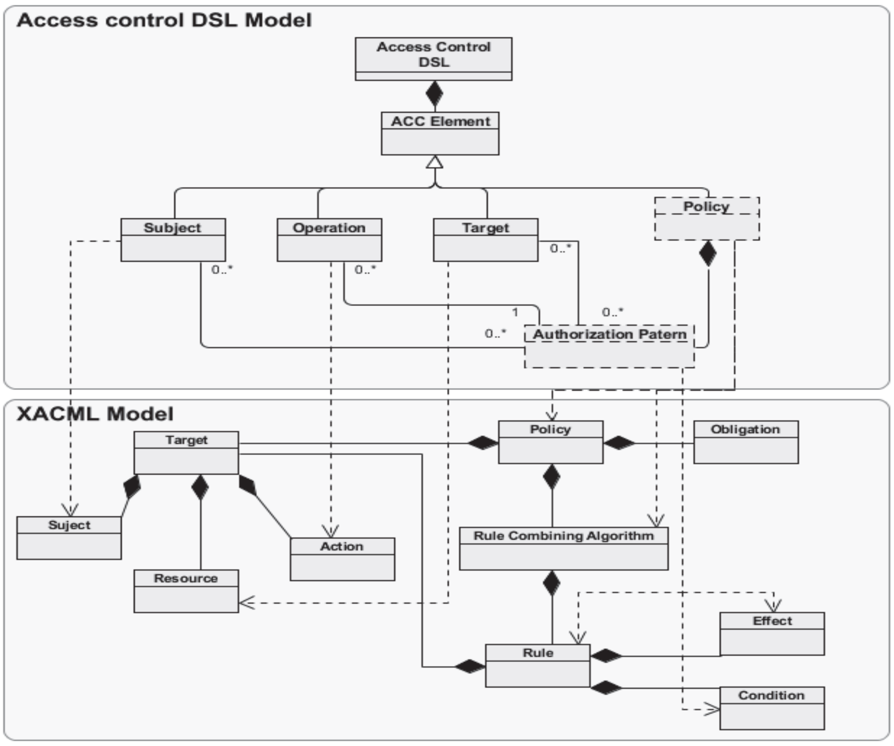


Figure 45: Mapping between access control DSL and XACML

. Listing 11 shows an XACML policy fragment which is the result of transforming the access control pattern (dynamic separation of duties) shown in Figure 44. This generated XACML policy will be deployed in an XACML repository, and then evaluated when receiving an authorization request. In fact, we encode a dynamic separation of duties as an XACML rule ‘*Rolebasedseparationofduties*’, with a condition that makes reference to a custom function ‘*CheckRSoD*’.

```
<?xml version="1.0" encoding="utf-8"?>
<Policy RuleCombiningAlgId="permit-overrides" >
  <Description>Security DSL</Description>
  <Rule RuleId="Rolebasedseparationofduties" Effect="Deny" >
    <Target>
     ...
    </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="CheckRSoD">
        <Apply FunctionId=" function:string-one-and-only">
          <SubjectAttributeDesignator AttributeId="subject-id" />
        </Apply>
        <Apply FunctionId="function:string-one-and-only">
    <SubjectAttributeDesignator AttributeId="urn:compas:attributes:role" />
        </Apply>
    <Apply FunctionId="function:string-one-and-only">
     <ResourceAttributeDesignator
AttributeId="attributes:resourceinstance" />
            </Apply>
    <AttributeValue DataType="string">Supervisor</AttributeValue>
    <AttributeValue DataType="string">PostProcessingClerk</AttributeValue>
    </Apply>
   </Condition>
  </Rule>
</Policy>
```

Listing 11: XACML Policy Fragment for separation of duty

Formula 10: Check history of authorizations

```
Boolean CheckRSoD (RequestSebject, RequesetResourceInstance,
RequestRole, SepratedRoleSet){

Bolean result= BPath.Check(
   "P(
   (@Subject=RequestSebject and
   @Resourceinstance = RequesetResourceInstance) and
   @role!=RequestRole  and
   ( Contains( SepratedRoleSet, @Role) and
Contains(SepratedRoleSet, RequestRole );
   )"

   Return result;
}
```

The *CheckRSoD* function (**Formula 10**) is offered by our access control framework, which is based on Sun's XACML implementation [SUN]. This function allows exploring the history of authorization decisions, in order to check whether the authorization of a requested target violates the separation of duties. Concretely, this function uses our BPath API to check the history of authorizations over the authorizations trace. The *CheckRSoD* function takes four parameters: the requester name *RequestSebject*, the role attribute *RequestRole*, the resource instance *RequestResourceInstance*, and a list of roles *SepratedRoleSet*, which should be dynamically separated. The authorization trace is a state trace storing the history of authorization decisions, where each state corresponds to an authorization decision, and all information of the authorization request or decision response can be accessed as dynamic attributes.

Once an XACML policy is generated, it should be enforced at runtime in order to make authorization decision and allowing access to resources. Figure 46 shows our runtime access control framework which can be described as following:

- Access control DSL is used to create and generate access control policies and to deploy them in an XACML repository.
- When a requester (user or service) invokes a service, it sends necessary information for authentication.
- A context interceptor intercepts an invocation message before achieving its destination (service or process), extracts necessary authentication information, then creates an XACML request by adding necessary information such as the role of the requester, this component is a part of a Policy Enforcement Point (PEP).
- The PEP sends the XACML request to a Policy Decision Point (PDP), which loads policies from the XACML repository, then evaluates the request and sends back a response. The response can be either a permit or a deny.
- Depending on the PDP response, the PEP component allows the invocation message to achieve its destination, or sends a non-allowed access response to the requester.

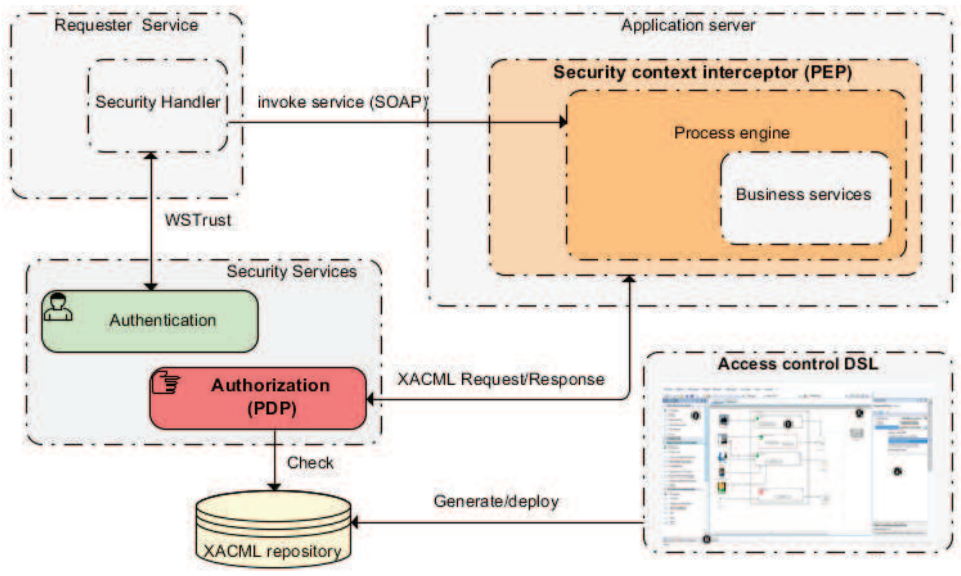


Figure 46: Access control system architecture

The Listing 12 shows an example of an XACML request and its corresponding XACML response in Listing 13.

```
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute AttributeId="subject:subject-id" >
      <AttributeValue>Steve</AttributeValue>
    </Attribute>
   <Attribute AttributeId="urn:compas:attributes:role">
      <AttributeValue>Supervisor</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="resource:resource-id" >
      <AttributeValue>Bank Process</AttributeValue>
    </Attribute>
    <Attribute AttributeId="resourceinstance">
      <AttributeValue>1089283476889798232</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
   <Attribute AttributeId="action-id" >
      <AttributeValue>Open account</AttributeValue>
   </Attribute>
  </Action>
</Request>
```

Listing 12: XACML request

```
<?xml version="1.0" encoding="UTF-8"?>
  <Response xmlns="urn:oasis:names:tc:xacml:1.0:context" >
    <Result>
     <Decision>Permit</Decision>
    </Result>
  </Response>
```

Listing 13: XACML response

4.7 Related work

Several model driven based research approaches for compliance to security policies have been proposed recently. The SecureUML approach [LBD02] proposes a modeling language for model-driven development of secure, distributed systems based on the Unified Modeling Language (UML). This approach is based on role-based access control with additional support for specifying authorization constraints. This approach uses UML profile to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures. This approach aims at improving productivity during the development of secure distributed systems and the quality of the resulting systems.

We think that embedding security specification within a system specification may be useful in small size systems, but in case of large systems, there is a need of separation between the two kinds of specification in order to improve readability of each specification, and especially for security specification.

Authors in [WMS+09] addressed the issue of specifying security requirements in the context of business processes. They propose an MDA based approach to express different security requirements at the business process level. they provide a generic and an abstract meta-model for security; specifying security goals, policies, and constraints, and defined mapping to some technical specification, as the mapping from a confidentiality model to Axis rampart confidentiality policy [RAM], and the one from authorization model to XACML specification. There are three mains differences with our approach. The first difference is the level of the specification of security policies, in our approach we made a choice to completely separate between the processes definition and security specification interfaces, which allows both security and business experts to have a clear and global view on the specified security policies. Second, information needed for security specification is obtained from a process repository, which allows making abstraction on the used language to specify business process: BPMN, BPEL, etc. Third, our approach proposes a domain specific language rather than a generic language, by the use of security patterns, which are designed by a security expert, to deal with a specific security needs, using concepts from the application domain.

Enhancing UML to model custom security aspects is addressed in [DPM07] by presenting an approach to compose features from different access control schemes. This includes an expansion of UML to include role-based, discretionary, and mandatory access controls, via new UML diagrams for roles, user authorizations, and delegation. As a result, designers are able to specify access control policies with UML-based diagrams and an underlying scheme that combines RBAC, MAC and DAC. Because our access control model is based on ABAC model, the resulting access control DSL can also support the RBAC, MAC, and DAC models. For this, it is sufficient to develop the appropriate transformation module, necessary to transform a graphical access control DSL according to a given access control model, which is completely transparent from the access control specification in the high level DSL.

Authors in [HMB07] proposed a domain specific language for securing distributed systems. They have developed a domain-specific, high-level and a declarative language, named PPL, for specifying security policies. PPL language is a textual DSL that allows specifying security policies in a concrete human-readable form. Security policy expressed in the DSL can be scrutinized, split, combined, shared, published, put under release control, printed, commented, and even be automatically generated by other applications. PPL can be used to write a variety of typical security policies as well as special-purpose ones. We offered a graphical DSL rather than a textual DSL, and we targeted a specific security requirement for SOA based systems.

4.8 Summary

In this chapter, we have introduced security for SOA based system, and concluded on the necessary adoption of SOA security standard as a key to the success for any solution built for SOA. Then, we presented the issue of making the specification of security policies easier, thus we explored the MDA based approach.

We proposed a security architecture based on domain specific language, by focusing on access control requirements. Thus, we presented an access control DSL, which encompasses two types of DSL, for both business expert and security expert. We presented in detail our graphical access control patterns for specifying access controls, with the process of generating XACML based policies, Finally, we presented our runtime framework in order to evaluate and enforce the generated XACML policies.

Part 5. Conclusion And Perspective

Outline

In this chapter, we summarize our work and contributions and anticipate on some future research directions.

5.1 Summary of contributions

We have presented a view-based monitoring approach, offering the capability to monitor business processes, services choreography, and their related views. Also, we addressed a specific security goal, which is access control. Two main issues have guided our work in this thesis: assuring the usability of the specification language, and facilitating the integration with SOA environment, both for monitoring and for access control.

5.1.1 View-based monitoring approach

The first of our contributions is the view-based monitoring approach, which covers monitoring of business processes, service choreography, and monitoring on the basis of some view from the perspective of a related set of concerns. Three monitoring systems have been developed:

- Internal Business process monitoring system, which observes the internal activities during execution of an instance of a business process or on a set of process instances, and report aggregated information about their execution.
- A Business protocol monitoring system observing and checking a business process when messages are received from or sent to external services.
- A choreography monitoring system which observes the exchanged messages between partner services and performs verification and statistical measurement. Our choreography monitoring system is characterized by the ability to perform monitoring without a need to store choreography messages.

These monitoring systems are based on our monitoring language called BPath, which offers a single expressive language based on XPath for both evaluating behavioral and statistical properties, and aims to be:

- An expressive monitoring language: offering the possibility to use hybrid logic, and linear temporal logic (LTL) with past and future operators.
- An easy to learn language: because BPath is based on XPath, which is a widely used language in SOA, consequently BPath is easy to learn for users with a

background on XPath. Additionally, BPath is a single language for checking both behavioral and statistical properties.

- Easy to use: offers the ability to express monitoring properties over different views.
- Easy to integrate verification language: closer to Web services environment (XPath based language)

Also, we have presented an algorithm and developed a tool for extracting business protocols from business processes implemented in BPEL language. The extracted business protocol is used as a view of the business process in our monitoring system. This tool can be reused by other approaches based on business protocol for different purposes.

5.1.2 Access control DSL

Access control is a very important aspect for SOA based systems due to the open environment in which SOA based systems interactions occur. Our second main contribution is an access control DSL tool offering a graphical language to express access control policies in an abstract way, and came in the same spirit of BPath language by providing an easy to use language that fits different abstraction levels existing in a company's organization, and implements a an SOA based language for access control, which is XACML. Our access control DSL is composed of two levels:

- A low level DSL allows a security expert to extend a list of predefined access control patterns defined within our DSL tool, and to create a custom transformation module in order to generate access control policies in some desired language,
- A high level DSL allows a business expert to specify security requirements in an abstract way by reusing predefined security patterns, without worrying about technical aspects, such as the targeted platform and language.

Also, we have developed a runtime framework to enforce access control policies generated by our access control DSL, We made a choice of an SOA language for encoding our access control policies, which is XACML, in order to enhance the integration of our framework within SOA environment.

By being a graphical domain specific language, several advantages from our access control are expected:

- A clear view of access authorization policies.
- Policies are directly specified using domain concepts.
- Only valid relations between the domain concepts exist.
- Separation between business and technical levels
- The productivity and the software quality can be improved because they are easier to maintain and the generated code does not contain bugs.
- The generated code can be tailored for the particular technology.
- Technical aspects are not shown to business experts.

5.2 Discussion and future work

Companies are increasingly concerned by their compliance to different compliance sources, and about the reputational risk from failing to comply with. Thus, companies increasingly place compliance in the heart of their concerns.

Two mains issues have guided our work in this thesis: assuring the usability of the specification language, and facilitating the integration with SOA environment, in case of both monitoring and access control. Our work is divided into two main contributions; each of them covers different aspects.

The interest of using XPath or XQuery to perform monitoring is relatively new, the work done by Beeri et all [BEM+07, BE07] has inspired us in the beginning of our thesis, by turning our attention to the importance of using an SOA based language for the monitoring of business processes. The work done in [HV09] is the closest one to ours, in terms of implementing temporal logic by using an XPath based language.

BPath proposes interesting features to express monitoring properties, but it remains a low level language compared to our access control DSL. We believe that it will be interesting to explore the possibility to provide a high level language to express compliance requirements, and to automatically generate monitoring properties in BPath.

Actually, our implementation doses not support the evaluation of BPath expression in a streaming way, which is due to a technical limitation of the used XPath processing engine. We have started working on this issue by extending an XPath engine, especially to process state log document in a streaming way. The simplicity of the structure of the state log was an advantage, but some technical issues have occurred, and we have decided to keep this extension as a future work.

In terms of execution performance, a lot of things can be made in the future; starting from the enhancement of our extended XPath engine. Thanks to the separation between the execution state trace and the execution data. The defining of new strategies of preloading dynamic attributes during the evaluation of a BPath expression will be a considerable improvement.

Our monitoring systems operate asynchronously, which leads to a certain distance between the real state of a monitored business process, and what is observed by our monitoring systems, which may be an obstacle if some actions should be made on the monitored business process. A solution to tackle this issue is through the instrumentation of the monitored business process, by adding synchronization activities that force a business process to wait until a verification task is completed. For instance, this can be achieved by adding invocation activities to the monitored business process, which forces the business process to wait until a response is received by our monitoring systems. But this solution is not without drawbacks, as the fact of polluting business processes with new activities that are not related to the business, and disrupting the execution of the business process, in case where temporal constraints should be respected by the execution. More suitable strategies should be investigated.

Additionally, some aspects are not addressed in this thesis, such as root cause analysis, in order to find the origin source leading to an unexpected behavior of the monitored business process or services choreography. An interesting technique to explore could be the use of a social network in order to graphically show the relation between involved elements from the execution context leading to the violation of some monitoring properties. For instance, when some elements are involved together many times in the violation of a given property, they should be shown closer to each other in the social network, etc.

The idea of using domain specific language for security seems to be interesting, as we tried to demonstrate in this thesis through our access control DSL. We have made a choice of separating between the tools for specifying access control policies from those for designing business process, in contrast to some approaches using annotation of business processes. This choice aims to avoid cluttering business process specification and to allow better visibility of the specified security policies. We also made a choice on spearing between technical and abstract levels in the specification of access control, which enable the specification of access control policies in an abstract way, and transforming it to a technical language in a transparent way. Through our approach, we have supported the idea of using patterns that are domain specific and extensible as needed to fit the application domain, rather than using generic access control policies for security policies specification.

An interesting research direction to pursue is to investigate other kinds of security requirements, and supporting more than one access control language, such as temporal logic for either design time or runtime verification. Finally, security remains a wide area, characterized by complexity and the use of many specification, languages, and technologies. We recommend for any possible extension of our work to address each security requirement one by one, because using only one DSL to cover all security requirements will not be efficient, and will be even contrary with the spirit of domain specific languages, which have the main goal to bring closer languages and development tools to application domains.

BIBLIOGRAPHY

[AAB+05] A. Arkin, S. Askary, B. Bloch , F. Curbera, Y. Goland, N. Kartha, C. Liu, S. Thatte, P. Yendluri, A. Yiu, "Web Services Business Process Execution Language Version 2.0," Working Draft, WS-BPEL TC OASIS, 2005.

[AMQ] http://activemq.apache.org

[AO08] P. Ammann and J. Offutt, "Introduction to Software Testing," 1st ed. Cambridge University Press, 2008.

[BBG+07] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "A Timed Extension of WSCoL," in IEEE International Conference on Web Services, 2007. ICWS 2007, 2007, pp. 663-670.

[BCT+03] B. Benatallah, F. Casati, F. Toumani, and R. Hamadi, "Conceptual modeling of web service conversations," Proceedings of the 15th international conference on Advanced information systems engineering, pp. 449–467, 2003.

[BCT04] B. Benatallah, F. Casati, and F. Toumani, "Web Service Conversation Modeling: A Cornerstone for E-Business Automation," IEEE Internet Computing, vol. 8, no. 1, pp. 46-54, 2004.

[BCT06] B. Benatallah, F. Casati, and F. Toumani, "Representing, analysing and managing web service protocols," Data & Knowledge Engineering, vol. 58, pp. 327–357, Sep. 2006.

[BE07] C. Beeri and A. Eyal, "Monitoring business processes with queries," IN VLDB, 2007.

[BEM+07] C. Beeri, A. Eyal, T. Milo, and A. Pilberg, "Query-based monitoring of BPEL business processes," in Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China, 2007, pp. 1122–1124.

[BFH+03] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation specification: a new approach to design and analysis of e-service composition," Proceedings of the 12th international conference on World Wide Web, p. 403–410, 2003.

[BG05] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in Service-Oriented Computing - ICSOC 2005, 2005, pp. 269-282.

[BKF+05] C. Barreto, N. Kavantzas, T. Fletcher, D. Burdett, G. Ritzinger, and Y. Lafon, "Web Services Choreography Description Language Version 1.0.," W3C, 2005.

[BLS08] A. Bauer, M. Leucker, and C. Schallhart, "Model-based runtime analysis of distributed reactive systems," in Software Engineering Conference, 2006. Australian, 2006.

[BM02] P. Blackburn and M. Marx, "Tableaux for Quantified Hybrid Logic," in Automated Reasoning with Analytic Tableaux and Related Methods, 2002, pp. 259-286.

[BM06] B. Benatallah and H. R. Motahari-Nezhad, "ServiceMosaic project: modeling, analysis and management of web services interactions," Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling - Volume 53, pp. 7–9, 2006.

[BMH07] S. Benbernou, H. Meziane, and M. S. Hacid, "Runtime Monitoring for Privacy-Agreement Compliance," in Service-Oriented Computing – ICSOC 2007, vol. 4749, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 353-364.

[BPEL] Web Services Business Process Execution Language (WSBPEL) 2.0, OASIS Standard, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[CD99] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0.," W3C, 1999. http://www.w3.org/TR/xpath

[Cla01] J. Clark, “XSL Transformations (XSLT) Version 1.1., ” W3C, 2001.

[Che02] A.M.K. Cheng, “Real-Time Systems: Scheduling, Analysis, and Verification,” Wiley-Interscience, 2002.

[CSR+09] D. Chamberlin, J. Snelson, J. Robie, and M. Dyck, “XQuery 1.1: An XML Query Language,” W3C, W3C Working Draft, Décembre 2009.

[DCA+09] F. Daniel, F. Casati, V. D’Andrea, E. Mulo, U. Zdun, S. Dustdar, S. Strauch, , D. Schumm , F. Leymann, S. Sebahi, F. de Marchi, M.-S. Hacid, “Business Compliance Governance in Service-Oriented Architectures,” in Advanced Information Networking and Applications, International Conference on. pp. 113-120 IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[DDH72] O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare, “Structured Programming,” APIC Studies in Data Processing, no. 8, Academic Press, (1972)

[DGR04] N. Delgado, A. Q. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” IEEE Transactions on Software Engineering, vol. 30, no. 12, pp. 859-872, Dec. 2004.

[DSL] http://www.microsoft.com

[EC80] E. A. Emerson and E. M. Clarke, “Characterizing Correctness Properties of Parallel Programs Using Fixpoints,” in Proceedings of the 7th Colloquium on Automata, Languages and Programming, 1980, pp. 169–181.

[EFH+03] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. Mcisaac, and D. V. Campenhout, “Reasoning with Temporal Logic on Truncated Paths,” in Computer Aided Verification, 2003, pp. 27–39.

[Eme90] E. Allen Emerson, “Temporal and modal logic,” in the Handbook of theoretical computer science (vol. B): formal models and semantics (MIT Press, 1990), 995-1072.

[FK92] F. Ferraiolo and R. Kuhn, "Role-Based Access Control," Proceedings of the 15th NIST-NSA National Computer Security Conference, Baltimore, MD, 13-16 October 1992, 554563.

[GKP03] G. Gottlob, C. Koch, and R. Pichler, "The complexity of XPath query evaluation," in Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York, NY, USA, 2003, pp. 179–190.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn, "Dynamic Logic," 1st ed, The MIT Press, 2000.

[HM05] P. Hallam-Baker and S. H. Mysore, "XML Key Management Specification (XKMS 2.0)," W3C, 2005.

[HMB07] H. Hamdi, M. Mosbah, and A. Bouhoula, "A Domain Specific Language for Securing Distributed Systems," in Second International Conference on Systems and Networks Communications, 2007. ICSNC 2007, 2007, pp. 76-76.

[Hoa69] C. A. R. Hoare, "An axiomatic basis for computer programming," Communications of the ACM, vol. 12, p. 576–580, Oct. 1969.

[HR01] K. Havelund and G. Rosu, "Testing Linear Temporal Logic Formulae on Finite Execution Traces," 2001.

[HV08] S. Hallé and R. Villemaire, "XML Methods for Validation of Temporal Properties on Message Traces with Data," in On the Move to Meaningful Internet Systems: OTM 2008, 2008, pp. 337-353.

[HV09] S. Hallé and R. Villemaire, "Runtime monitoring of web service choreographies using streaming XML," in Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, 2009, pp. 2118-2125.

[HZD09] T. Holmes, U. Zdun, and S. Dustdar, "MORSE: A Model-Aware Service Environment," 4TH IEEE ASIA-PACIFIC SERVICES COMPUTING CONFERENCE, p. 470--477, 2009.

[IEEE83] IEEE Standard Glossary of Software Engineering Terminology. Ieee Computer Society, 1983.

[Kam68] H. W. Kamp. Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles, 1968.

[LBD02] T. Lodderstedt, D. A. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," in Proceedings of the 5th International Conference on The Unified Modeling Language, 2002, pp. 426–441.

[LS09] M. Leucker and C. Schallhart, "A brief account of runtime verification," Journal of Logic and Algebraic Programming 78, n°. 5 (2009): 293-303.

[MDA] http://www.omg.org/mda/

[MNP05] O. Maler, D. Nickovic, and A. Pnueli, "Real Time Temporal Logic: Past, Present, Future," in Formal Modeling and Analysis of Timed Systems, 2005, pp. 2-16.

[MP95] Z. Manna and A. Pnueli, Temporal verification of reactive systems: safety. Springer, 1995.

[MP09] A. Metzger and K. Pohl, "Towards the Next Generation of Service-Based Systems: The S-Cube Research Framework," in Advanced Information Systems Engineering, 2009, pp. 11-16.

[MS04] K. Mahbub and G. Spanoudakis, "A framework for requirents monitoring of service based systems," Proceedings of the 2nd international conference on Service oriented computing, pp. 84–93, 2004.

[NMS] NMS API, http://activemq.apache.org/nms/nms-api.html

[ODE] Apache ODE project (http://ode.apache.org),

[Pet99] P. Pettersson, "Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice," Thesis, http://www.uppaal.com

[Pnu86] A. Pnueli, Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In J.W. de Bakker, W.P de Roever, and G. Rozenberg, editors, Current Trends in Concurrency: Overviews and Tutorials, number 224 in Lecture Notes in Computer Science, pages 510-584. Springer-Verlag, 1986.

[PTDL07] M. P. Papazoglou, P. Traverso, S. Dustdar, et F. Leymann, “Service-Oriented Computing: State of the Art and Research Challenges,” Computer, vol. 40, n°. 11, p. 38-45, 2007.

[QS82] J.-P. Queille and J. Sifakis, “Specification and verification of concurrent systems in CESAR,” in Proceedings of the 5th Colloquium on International Symposium on Programming, 1982, pp. 337–351.

[RAM] http://axis.apache.org/axis2/java/rampart/

[RE09] J. Reagle and D. Eastlake, XML Encryption Syntax and Processing Version 1.1. W3C, 2009.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual. Addison-Wesley Professional, Jan. 1999.

[SAML] Security Assertion Markup Language (SAML) v1.0, OASIS Security Services TC, http://www.oasis-open.org/standards#samlv1.0, November 2002.

[SFH+06] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns: Integrating Security and Systems Engineering, Wiley, 2006.

[SH06] H.-bo Shen and F. Hong, “An Attribute-Based Access Control Model for Web Services,” in 2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT’06), Taipei, Taiwan, 2006, pp. 74-79.

[SOAP] SOAP Version 1.2 Part 1: Messaging Framework, W3C Working Draft, http://www.w3.org/TR/soap12-part1/

[SSE] http://msdn.microsoft.com/fr-fr/express/aa718378

[STM+09] C.M. Sperberg-McQueen, H.S. Thompson, M. Maloney, H.S. Thompson, D. Beech, N. Mendelsohn, S. Gao, W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C (2009).

[SUI] www.soapui.org

[SUN] http://sunxacml.sourceforge.net/

[TOM] http://tomcat.apache.org/

[Tra09] H. TRAN, View-based and Model-driven Approach for Process-driven, Service-Oriented Architectures, PhD Thesis, Vienna University of Technology, Dec 2009.

[Tur07] A. M. Turing Award – 2007, http://awards.acm.org/homepage.cfm?awd=140

[UDDI] Universal Description, Discovery and Integration, Ariba, IBM and Microsoft, UDDI.org. http://www.uddi.org

[W3C] http://www.w3.org/TR/ws-gloss/#webservice

[Whi04] S.A. White, Business Process Modeling Notation (BPMN) Version 1.0. Business Process Management Initiative, BPMI.org, (2004)

[WMS+09] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel, "Model-driven business process security requirement specification," Journal of Systems Architecture, vol. 55, no. 4, pp. 211-223, Avril. 2009.

[WSDL] Web Services Description Language (WSDL) 2.0, W3C Proposed Recommendation, June 2007 http://www.w3.org/TR/wsdl20/

[WSS] WS-Security Core Specification 1.1, OASIS Web Services Security (WSS) TC, 2006, http://www.oasis-open.org

[XACML] Extensible Access Control Markup Language (XACML) v1.0, OASIS Extensible Access Control Markup Language TC,http://www.oasis-open.org/standards#xacmlv1.0,February 2003

[XML] Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds., W3C, 6 October 2000, http://www.w3.org/TR/REC-xml

[YRH+09] K. Roessler, F. Hirsch, E. Simon, D. Solo , P. Datta , B. Fox, J. Reagle , B. LaMacchia, M. Bartel, J. Boyer, D. Eastlake, XML Signature Syntax and Processing Version 2.0. W3C (2009).