CESSA

Compositional Evolution of Secure Services using Aspects

ANR project no. 09-SEGI-002-01

Survey and requirements analysis

Abstract. The CESSA industrial research project will provide solutions for the evolution of secure SOAs by providing an aspect-oriented structuring and programming model that allows security functionalities to be modularized, even if they cross administrative and technological domains.

This deliverable offers motivation and basic requirements for the models and applications targeted within the project. It defines the basic characteristics of the service and aspect models that are used as part of the project, describes a use case scenario that will be implemented during the project, presents requirements for these models derived by the industrial partners of the project using the scenario, and presents an extensive analysis of the state-of-the-art of services and composition techniques in academia and industry, including standards.

Deliverable No.	D1.1
Task No.	1
Type	Deliverable
Dissemination	Public
Status	Final
Version	2.1
Date	2 July 2010
Authors	R. Douence, H. Grall, I. Mejía, J.-C. Royer, M. Südholt (EM Nantes); M. S. Idrees, Y. Roudier (Eurecom); J. Leroux, F. Rivard (IS2T); J.-C. Pazzaglia, G. Serme (SAP)

Contents

Chapter 1

Introduction

Service-oriented architectures (SOAs) constitute a major architectural style for large-scale infrastructures and applications that are built from loosely-coupled well-separated services. SOAs today are the major structuring principle of a multitude of commercial infrastructures and applications that consist of service compositions, in particular service orchestrations and choreographies. They may span a number of different organizations, and involve powerful servers as well as resource-constrained devices (e.g., mobile devices).

Such applications frequently are subject to stringent security requirements, for example, in order to protect company-internal data, avoid breaches of the right to privacy of clients, and provide tracing information for auditing purposes to official institutions. Security properties generally pervade software systems, in technical terms, security properties crosscut a service-based systems: security-relevant policies and implementations depend on (and affect) large parts of the underlying system.

Furthermore, service-based systems, notably enterprise information systems, are frequently subject to evolution because they cannot be shut down even for short interruptions. Such evolutions may, in particular, cross different administrative domains (that belong, *e.g.,*to different branches of a company that use different security policies) or cross different technological domains (that may support, more or less general service models).

The CESSA industrial research project will provide solutions for the evolution of secure SOAs by providing an aspect-oriented structuring and programming model that allows security functionalities to be modularized, even if they cross administrative and technological domains. Overall CESSA aims at four contributions:

1. The definition and implementation of a service model that enables modeling of service compositions within the administrative and technological domains of the CESSA partners, notably SAP's large-scale service infrastructures and ERP applications, and IS2T's infrastructures for resource-limited embedded devices.

2. The definition and implementation of an aspect model that allows service-based systems to be evolved flexibly by, at the same time, guaranteeing correctness properties of evolutions.

3. A security model for service-based systems, as well as corresponding analysis and en-

forcement mechanisms.

4. Two applications of these models: an application to a service infrastructure for ERP systems from SAP and an integration with IS2T's customized Java VMs for embedded devices.

This deliverable is the first of two deliverables (along with deliverable D2.1) that lay the foundations for work on these results. This document offers motivation and basic requirements for the models and applications targeted within the project. Concretely, we present the following information.

- Chapter 2 introduces the main concepts of service-oriented computing and aspect-oriented computing. It describes two models, for services and aspects respectively. It also defines the terminology that will be used in the CESSA project.

- In Chapter 3, we first present a loan negotiation scenario typical for the acquisition of loans in a business environment that involves customers, banks and credit bureaus. This scenario will be used over the full duration of the CESSA project to motivate, illustrate and validate the methods, techniques and tools developed as part of the project. Then, using this scenario, we sketch requirements for the service and aspect models derived by partners SAP and IS2T. To prepare the future developments, we also introduce a method that will be used to formalize requirements.

- Chapters 4 and 5 present an extensive analysis of the state-of-the-art of services, composition techniques, especially aspect-oriented software development. This analysis encompasses both, academic approaches and industrial systems, including existing industrial standards and practices.

Note that this document does not focus on any specific property of service-based systems. Approaches to the security of service compositions, the specification and implementation of security properties, as well as their evolution are not considered here but are the subject of the deliverable D2.1 that is entirely dedicated to the security of services and aspects within the CESSA project.

Finally, note that this deliverables defines the fundamental properties of the CESSA service and aspect models, the (full) definition of the models will be provided later, notably as part of deliverable D1.2.

Chapter 2

Service and aspect models

The growth of Internet has extended the scope of software applications, leading to network-based architectures. The main characteristic of these distributed architectures is that they restrict the communication between remote components to message passing. Service-oriented computing is a key solution to organize the exchange of messages in a network-based architecture, by using services as primitive components. Services provide autonomous operations that can be described, published, discovered and orchestrated using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries. There is a strong separation between the interfaces of the service operations and their implementation: interfaces are independent from the underlying platform used for implementing the operations.

As service technologies are advancing fast and are being extensively deployed in applications spanning different organizations, in order to hold their promise, it becomes crucial to ensure security and trust in security for these applications. Confidentiality, integrity, availability and digital identity management are now required [25].

Security functionalities, such as access control and monitoring for intrusion detection, are an example of cross-cutting functionalities. In an application, cross-cutting functionalities, which are scattered and tangled over large parts of the application, cannot be modularized in a well-separated module. Aspect-Oriented Software Development is an application-structuring method that addresses in a systemic way this problem of a lack of modularization facilities for cross-cutting functionalities. The partners of the CESSA project aim at developing solutions to secure service-oriented computing by providing aspect-oriented methods and techniques, allowing security functionalities to be modularized.

In the following sections, we introduce service-oriented computing and aspect-oriented computing. The purpose of the following sections is to provide the participants in the project CESSA with a common conceptual model for service-oriented and aspect-oriented computing. In particular, the aims are:

- to provide a terminological background and ease the integration between CESSA partners,
- to provide the concepts that have to be considered when designing service-oriented and aspect-oriented systems.

2.1 Service model

We present a model for service-oriented computing. Actually, there are two current models for service-oriented computing, which we briefly present now.

First, interoperability and integration issues has led to the development of the technology of "big" Web services [115], based on XML and Internet technologies. A service corresponds to a set of operations, implemented with any technical infrastructure, declared in a specific language, like the Web Services Description Language (WSDL), and accessed via a standard protocol like the Simple Object Access Protocol (SOAP). Processes, corresponding to the composition of operations, can be locally defined from an orchestration language like the Business Process Execution Language for Web Services (BPEL4WS), and globally specified with a choreography language like the Web Services Choreography Description Language (WS-CDL). Since processes are central in this model, we say that the model is process-oriented.

More recently, an alternative solution has been brought forward leading to RESTful web services. Returning to the original design principles of the World Wide Web, and its REST style [115], it proposes a form of automation: the user sending requests with a web browser then getting responses from a server is replaced with a computing and communicating process calling services attached to resources. Since the primitive entities are resources, we say that the model is resource-oriented.

2.1.1 Overall architecture and main concepts

We propose a model that unifies both models, by retaining their common structure. It is based on three layers, corresponding to collaborations, processes and services. It can be considered as an abstraction and a simplification of the technological stack for web services described in Sect. 5.1 and represented in Fig. 5.1.

Fig. 2.1 presents the main concepts of the CESSA service model. Multiple *peers* can interchange *messages* which activate *collaborations* between *processes* (represented by rounded rectangles in the figure). Processes invoke or provide services (ovals). *Services* are composed of different operations which encapsulate the access to *resources*. These concepts are defined in more detail below.

2.1.2 Collaborations

A *collaboration* consists of a set of *processes* that interact with each other by exchanging *messages*. It can be defined as the flows of messages exchanged among the processes at a global level. A collaboration is specified by a global *protocol*, called a *choreography*, that describes the set of allowable interactions for the collaboration. *Sessions* are a common mechanism of interaction for a collaboration. For instance, assume two processes willing to collaborate. They first establish a connection on a shared public channel. After agreeing on some private channel, they commit in a *conversation*, following a *protocol* describing the sequence of messages exchanged on the private channel. In general, a protocol does not specify a unique sequence, but a set of allowable sequences.

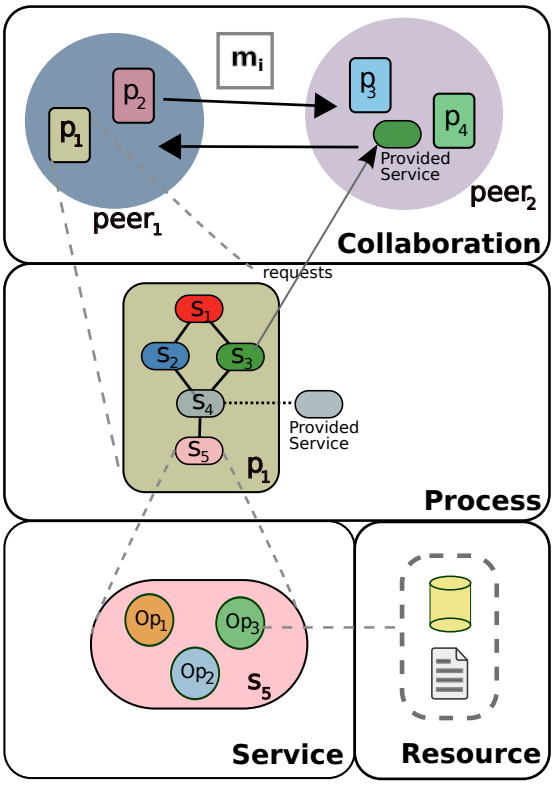


Figure 2.1: Service Model

A collaboration can involve many processes, and not only two. These processes can be split into groups, each group being controlled by a *peer*. Thus, a choreography can be considered as a contract between peers: for each peer, the choreography specifies the behavior of the processes under its control when they engage into a collaboration. Therefore, there is no centralized control. Furthermore, note that collaborations foremost are an abstraction mechanism: they may not exist as physical entities, in particular at execution time.

As a running example, we detail in section 3.1 a loan negotiation scenario with multiple actors. In this scenario, there is no centralized control but the actors act and react according to the steps of the scenario. In particular, the loan applicant and the real estate agency are more active in the early stages of the scenario. Later, the bank and government personnel become more active.

Technologies used for choreography are described in Sect. 5.1.1.

2.1.3 Processes

From a top-down perspective, a *process* can be considered as the projection of a collaboration. Just as a choreography specifies a collaboration, a *behavioral interface* specifies a process: it is a protocol defining the messages that the process can exchange during a collaboration. This interface abstracts away from the computations and the communications inside the process.

From a bottom-up perspective, a *process* invokes services that are thus required from other processes acting as servers and provides services that are invoked by other processes acting as clients. It is specified as an *orchestration* of services, resorting to the following operations to schedule service invocations and provisions.

- Communication

  A process can invoke external services. As it also provides services, it can receive a service request and reply to it. In other words, a process can act not only as a client but also as a server.

- Data manipulation

  Communication entails incoming and outgoing information, represented as structured data. The process needs to manipulate these data with a dedicated language.

- Sequentiality

  Processes can be organized in a sequence. The first process executes, after its termination, the second one executes, and so on.

- Control flow operations

  Standard operations allow choices between processes with conditional branches and iterations of processes with loops.

- Concurrency

  Processes can execute in parallel, and not only in sequence. Parallelism is efficient when the processes can execute without interfering. When some control dependency between processes in parallel is required, a mechanism provides the synchronizations that are needed.

- Exception handling

  When a process invokes a service that is currently unavailable, an error occurs. As this kind of errors may be frequent in a network-based context, a mechanism to detect and handle these errors is required. When an error occurs, an exception is thrown. It can subsequently be caught by an exception handler.

For instance, in our running example of the loan negotiation scenario, the application processing is decomposed into two sequential phases to be performed by two different clerks. Each phase requires concurrent operations: for instance, the post-processing clerk double checks the credit worthiness by querying in parallel the credit bureau, the government and the internal rating system.

The technologies used for orchestrating services are described in Sect. 5.1.1. As for data, they are usually represented as XML documents. A language like XPATH is then used to query documents.

2.1.4 Services

Services are the units processing messages. As they are distributed over the network, they are identified by their addresses. A service is a set of operations, with input and output parameters. Messages therefore correspond to operation calls and returns. Data exchanged are structured: generally speaking, they represent algebraic terms, that is trees built from raw data. Addresses can also be exchanged: hence, the service network may dynamically evolve. Given a service, the operations can be implemented by any technology. When the implementation does not entail further communications, we say that the operation is primitive. Otherwise, the operation is provided by some process: calling the operation corresponds to a process delegation. The process can be defined either in an orchestration language, or in a general-purpose language like Java, providing means to invoke and to provide services.

The most basic message is a request sent by a client to a server: it corresponds to the call of an operation declared in the service. The restriction to this purely asynchronous communication primitive is not a limitation since common communication modes for service invocation can be defined using this primitive. First, a simple protocol allows operation returns to be encoded: the request indicates as an extra input parameter the address of a service where the reply must be sent, called the continuation. Hence operations can declare not only input parameters but also output parameters. Then, common communication modes are easily defined.

- Asynchronous one-way invocation (most basic mode): the invocation immediately terminates without expecting a reply.
- Synchronous invocation: the invocation terminates only after the server replies.
- Asynchronous invocation with future: the invocation immediately terminates without waiting for the reply and the client deals with the reply when it is available via a future, the continuation of the invocation.

For instance, in the loan negotiation example, the credit bureau provides a service requested by the post-processing clerk in order to compute a ranking for the customer loan request.

The technology of Web services, which is the most widespread implementation of services, is described in Sect. 5.1.4 for the WS* stack, and in Sect. 5.2 for Restful web services. WS* web services emphasize the procedural aspect of services: a service invocation is akin to a remote procedure call using web standards. Restful web services are resource-oriented: they use some basic operations (CRUD operations, create, request, update, destroy) to manipulate resources.

2.1.5 Glossary

The glossary recalls the definitions of the basic concepts for service-oriented computing. The terms are alphabetically ordered.

Behavioral interface: protocol defining an enriched interface for a process.

Choreography: specification of a collaboration with a protocol.

Collaboration: exchange of messages between processes.

Conversation: multi-step interaction between processes or peers having state and duration.

Message: basic unit of communication corresponding to the call or return of an operation declared in a service.

Operation: computation unit declared in a service, with input and output parameters, either primitive, or corresponding to a process delegation.

Orchestration: specification of a process.

Peer: owner of one or more processes, which are under its control.

Process: a set of execution flows invoking services and providing services.

Protocol: specification of an exchange of messages.

Service: a set of operations.

Session: an interaction between processes or peers during a collaboration, managing the setting up and taking down of the conversation between the processes or peers involved in the collaboration.

2.2 Aspect model

Security is a functionality of service-based systems that is very difficult to specify and implement because it is not modular: modifications to one part of an application may interact strongly with the security properties of other parts of the same application.

Aspect-Oriented Software Development (AOSD) [6, 90] has emerged as the domain investigating and providing solutions for the systematic treatment of such non-modular functionalities. Aspect-oriented approaches provide aspects, a new programming abstraction for the modularization of such functionalities, see the aspect glossary (see Sec. 2.2.3). Aspects are typically defined in terms of pointcuts, abstractions defining the contexts where modifications have to be applied, and advices that define the modifications themselves.

While this approach, in principle, fits very well the problem of defining evolutions of the security model of service-based applications and infrastructures, services and service compositions impose several specific characteristics on an aspect model. In the following we first present basic properties that our aspect model should have and a set of corresponding major characteristics of the CESSA aspect model.

Note that the CESSA aspect model will be completely defined only later as part of D1.2 after the security properties, policies and standards that are considered as part of the project have been fixed (the CESSA security model will be presented as part of deliverable D2.1).

However, some basic properties for aspects can already be motivated by the specification, definition and implementation of services and their evolution.

- Aspects must be able to refer to and potentially modify all entities that form the CESSA service model as defined in Sec. 2.1.

  This requirement implies a number of more specific ones. As an example, one of the most important of these subsumed requirements concerns service compositions:

- The aspect model has to support the manipulation of horizontal and vertical service compositions. In particular, they have to support the modifications of choreographies and orchestrations of services.

- Aspects must be able to define implementations of evolutions implemented using the real-world infrastructures that are used by the industrial partners.

  While evolutions should be specified, defined and implemented based on the CESSA service model that is to be implemented by the industrial partners, the aspect model should accommodate particularities of the target platforms that have to be taken into account for important evolution tasks.

- Support for the evolution of service systems requires means for the precise definition of the semantics effects of aspects and the verification of their properties.

- Aspects must be able to refer to and interact with the CESSA security model, modify in particular security policies and properties.

As illustrated in Fig. 2.2, we have in mind a model where aspects may refer to and modify all entities at all levels of the service model. Furthermore, aspects may act on interfaces and implementations, as indicated in the figure only for the process abstraction.

We are now in the position to refine the above properties into concrete characteristics that the CESSA aspect model (that will be defined as part of deliverable D1.2) must meet.

Generally, aspect models [6] come in very different forms, concerning their basic concepts but also implementation strategies, suitability for the application of formal methods, etc. The CESSA aspect model will be based on the most popular model, the so-called pointcut-advice model for aspects, but requires some important extensions to be applied. In the following, the basic pointcut-advice model for aspects is introduced and the need for extensions motivated. We then present the main characteristics of the CESSA aspect model.

2.2.1 The basic model: an extension of the pointcut-advice model

The basic aspect model the CESSA aspect model is based on is the so-called pointcut-advice model that has already been introduced with the first dedicated aspect models [88] and is the basis of the most popular aspect systems, AspectJ [14, 89] and corresponding industrial models, such as SpringAOP [128] and JBoss AOP [84].

This model is characterized by three main abstractions: aspects, pointcuts and advice (cf. the glossary of basic aspect concepts in Sec. 2.2.3) that together provide means for the concise definition and efficient implementation of so-called crosscutting functionalities of a base application,

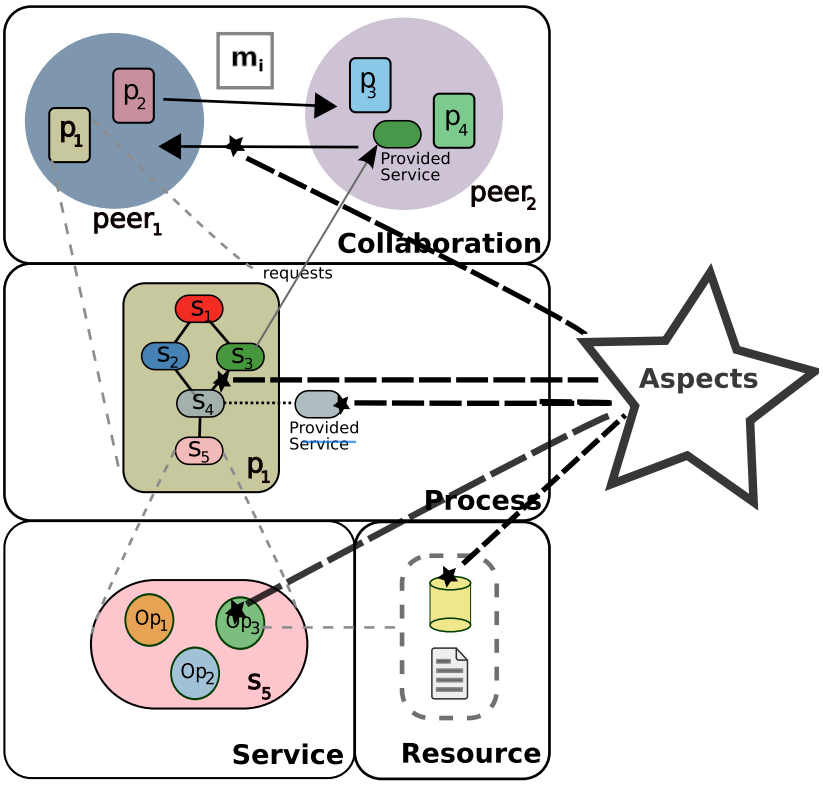


Figure 2.2: Service and Aspect Models

such as security, that cannot typically be modularized with existing structuring and encapsulation mechanisms, such as services or components.

- *Aspects* constitute the abstraction enabling the modularization of crosscutting functionalities. They contain, a minima, several bindings of pointcuts and advice but may also contain several other declarations for the definition, *e.g.,*of local state and the creation of fresh instances of aspects (*i.e.,*their local state).

- *Pointcuts* allow to define where (points in the source code of an application) or when (events during the execution of an application) aspects should apply modifications.

  Pointcuts are expressed in pointcut languages: their expressive power, their support for automatic verification of properties, etc, may significantly vary. Poincut languages often contain a large number of aspect-specific constructs that match specific structures of the language in which base applications are expressed.

- *Advice* is used to define the modifications that an aspect may perform. Advice is often expressed in terms of some general-purpose language with a small number of aspect-specific extensions, most notably the `proceed` construct that allows, in its basic form, the execution of the behavior of the base application that triggered the aspect application in the first place.

2.2.2 Fundamental characteristics of the CESSA aspect model

In order to apply AOP to the evolution of services defined according to the CESSA service model, aspects have to meet the basic properties introduced at the beginning of this section. We address these requirements by the following set of major characteristics that the aspect model has to fulfill. These characteristics are for most of them general in the sense that they apply to all the three basic aspect abstractions (aspects, pointcuts and advice); characteristics that only apply to some abstractions state this explicitly.

Characteristic: (Basic abstractions and relations) The pointcut language enables referencing all relevant abstractions of the service model and the concrete infrastructures; the advice language allows to manipulate these entities.

Furthermore, the pointcut language provides means to identify relevant relationships between these entities and the advice language allows to manipulate them. □

Concrete examples for such abstractions include collaborations, processes, services and resources. Relevant relationships between them include relations between adjacent abstraction levels or the ability to protect some of them using certain security mechanisms, such as access control, while others may not be modified by that security mechanism.

Evolution scenarios frequently cannot be realized using traditional black-box composition only, *i.e.,*by composing only interface-level entities of services or components. This is for instance the case if new requirements concern properties of implementations that have not been made explicit previously on the interface level. A concrete instance of such a requirement that is of prime interest to partner SAP are new requirements imposed by evolving legal environments: the introduction of the Sarbanes Oxley act in the U.S., for instance, required software editors to meet much more stringent traceability requirements that had not been (and could not be) anticipated during the interface design of existing ERP systems. The CESSA project therefore supports a more general composition model.

Characteristic: (Composition model) The CESSA aspect model provides a gray-box composition model, *i.e.,*aspects may access parts of service implementations. However, such access can be restricted by explicit fine-grained conditions on the structure and behavior of the underlying base system. □

The CESSA aspect model will therefore provide strong control over invasive composition. Corresponding conditions will be defined as part of evolution tasks through the aspects that realize them. The conditions may then be integrated before execution in the runtime representations of aspects or the underlying infrastructure, or enforced, possibly at execution time, on service implementations.

The next characteristic defines the applicability and generality of our aspect model with respect to the service lifecycle (design, implementation, assembly, deployment, execution).

Characteristic: (Dynamic application) Aspects are applied dynamically. Static application strategies may be used, however, if appropriate. □

Many current aspect models only support static or load-time application of aspects, which severely limits their applicability for many composition tasks. Our model therefore significantly

broadens the use of aspects to many real-world scenarios that involve highly dynamic service applications.

Another general characteristic of our model is that the model enables the aspect-based definition of service evolutions whose (security) properties can be formally analyzed.

Characteristic: (Formal properties) The aspect model includes explicit means to restrict aspects, pointcuts and advice, such that relevant formal properties of service evolutions defined using aspects can be specified precisely, formally analyzed and enforced on corresponding implementations. □

The following characteristic is motivated by the importance of protocols within the CESSA project.

Characteristic: (Protocol support) The pointcut language includes direct support for matching (parts of) protocols that govern the collaboration (choreography etc.) between entities of the service model. The advice language permits the manipulation of protocols. □

Local state of aspects may interfere with the behavior of the underlying base application, which is a potential problem for security properties of service-based systems that are evolved using aspects.

Characteristic: (Local state) Aspects may contain local state that can be used to modify state of the base application. Aspect definitions may, however, restrict the kind of state that can be defined and used. □

A restriction on aspect state is crucial to a model for the evolution of security properties of services, because unrestrained use of local state in aspects and for the modification of base applications forbids effective analyze analysis of the formal properties, in particular security properties, of the corresponding systems.

2.2.3 Glossary

This glossary introduces basic concepts of AOP. The terms are alphabetically ordered.

Advice: a piece of code to be executed when a pointcut matches a joinpoint. Advices can be inserted before or after the current execution events, and may also replace a joinpoint. In this latter case, the computation corresponding to the current execution event (*e.g.* a service call) is intercepted and the original computation is never executed.

Advice of advice: a program woven with an aspect generates new joinpoints (*i.e.,*the events corresponding to the advice execution, *e.g.,*when the advice calls services). These extra joinpoints can be matched by another aspect. An aspect could even match its own advice executions with the risk of infinite weaving. So, it is important to precisely define which joinpoints can be matched by which aspects. This can be seen as vertical composition of aspects.

Aspects: new modularization concept and corresponding program abstraction that typically defines one or several pointcut-advice bindings and possibly a local state.

Base application: the underlying application into which aspect are woven.

Binding (of pointcuts and advice): often a joinpoint exposes values of the base program (*e.g.,* a parameter of a service call). These values can be bound to variables in order to parametrize advices.

Composition or interaction (of aspects): when two pointcuts match the same joinpoint, two pieces of advice may be executed. In this case, the advice execution order can be important (for instance to determine whether a woven program should call a log service then abort, or first abort and never call a log service). In general, aspect composition goes beyond the ordering of advice. For instance, if an advice changes a color to blue and another advice to yellow, the composition of these two advices may be another color, not only one of the two. In other words, when the two advices call services, their execution must be orchestrated.

Concurrent aspect: in a concurrent context, the aspect-related concepts have to be adapted. For instance, a joinpoint could carry the identity of the thread that generates it. A history-based pointcut could match several joinpoints corresponding to a rendez-vous of processes. The base program and the advice executions could be parallel or sequential. When the joinpoint is a service, the corresponding advice could be executed sooner or latter according to whether the service was synchronously or asynchronously invoked.

Distributed aspects: distributed applications require extension to the standard concepts of AOP. For instance, a joinpoint could carry the identity of the host that generates it. A history-base pointcut could inspect a remote-call stack distributed across several hosts. A joinpoint could be matched based on host information and the corresponding advice could be executed remotely on several different hosts.

History-based pointcuts: a pointcut can select individual execution events, but it may also inspect the history of the computation that led to the current event. For instance, AspectJ provides pointcuts operators that inspect the call stack, and event-based AOP enables pointcuts that inspect the full execution trace.

Instantiation (of aspects): an aspect consists of at least one pointcut and its associated advice. The advice is executed each time a joinpoint is matched by the pointcut to which the advice is bound. An advice can thus be executed several times, and an aspect can have a local persistent state. For instance, a profiling aspect can declare an integer variable and the advice can increment it in order to count a particular kind of joinpoints. History-based aspects can also use a local persistent state. *e.g.,*to match a sequence of joinpoints. As soon as an aspect is stateful, aspects may be instantiated, *i.e.,*different instances of the same aspects with different (fresh) local states may be created. For instance, AspectJ propose to instantiate an aspect either once per virtual machine, once for each loaded class, once per object created or once per method called. It might be useful to instantiate an aspect each time a new service is discovered, or each time it is invoked, or each time and orchestration of services comes to an end.

Joinpoints: an execution event that an aspect can monitor and react to. In the context of CESSA a service invocation could naturally be a joinpoint.

Pointcuts: pointcuts denote joinpoints. Pointcuts can often be seen as matching individual or sets of joinpoints.

Properties (of correctness, security . . . :) in general an aspect can arbitrarily modify the semantics of the base program (for instance, it can replace the execution of `main` by its advice, or the `credit` service invocation could be replaced by the `debit` service invocation). Support for property verification and analysis is often provided by aspects that make explicit more information on the contexts in which they are applied, *e.g.,*history-based aspects.

Weaver: a compiler or interpreter that translates a base application and one or several aspects in an executable application.

Chapter 3

From use case scenarios to requirements

In this chapter, we detail a running use case: a loan negotiation scenario. Then we give some informal requirements drawn from the scenario and derived by the industrial partners SAP and IS2T. Generally speaking, the CESSA project first aims at providing conceptual solutions and at developing proofs of concepts, followed by an application and integration phase by the industrial partners. We can therefore distinguish two stages, the development of the solutions and the development of the applications using the solutions. More precisely, we will define a service and aspect infrastructure and will use the infrastructure in use cases (the major of which, a loan negotiation scenario, is presented in this deliverable), leading to a proof of concept. In the development process of the infrastructure and the use case, we will provide requirements that will be more complete and more formal than the requirements given in this first deliverable. That is the reason why we have also chosen to introduce a methodology that aims at easing the completion and the formalization of the requirements.

The chapter is organized as follows: the first section introduces the loan negotiation use case scenario, the second introduces the methodology, and the third one gives a first set of requirements derived from the use case and systems of the industrial partners.

3.1 Loan negotiation scenario

To represent the flexibility of CESSA and its advantages, we present a scenario that involves many actors in a distributed environment. After introducing the context and presenting the process of a loan origination workflow, we highlight the benefits of the CESSA platform describing what kind of evolution should be brought by CESSA.

During the project CESSA, we will build using this scenario a concrete use case, thereby providing a proof-of-concept prototype.

3.1.1 Introduction

John is a single man 25 years old. Recently, he was appointed as a teacher in a primary school on a permanent contract. His gross salary is about 25,000 euros per year. Before getting this job

John was a student and he was living at his parents' house. Though he gets well with his parents, he realizes it is time to move forward and to obtain his own independence. He contacts a real estate agency in order to find a house according to his needs. After few visits, John finds a flat which is a good deal. Despite the fact that affording this expense will require some efforts, John decides to go for it. Since he was a child John is saving money on a bank deposit account opened by his parents at the BBB bank. At the moment, John's account exhibits a positive balance of 10,000 euros.

The two figures 3.1 and 3.2 represent respectively the initial state of the workflow to request a loan, then the workflow modified by the CESSA platform. In the following section, we'll describe in detail the final workflow, where John request of loan through his mobile phone and is able to contact many organizations. But the real interest is to understand what kind of evolution is brought by CESSA to ensure security properties from initial workflow to the advanced one and keeping in account additional organizations and type of devices. We'll discuss about it in Section 3.1.5.

Actors from this scenario are :

- **John** The customer who request for a loan
- **The real estate agency** John signs a "promise of sale" with an agency
- **The BBB Bank**
  - **Peter** The bank pre-processing clerk who verifies early documents
  - **Gabriel** The bank post-processing clerk who launches internal processes
  - **Ted** Gabriel's manager
- **Government** Entity which provide financial aid

Figure 3.1: Initial Loan Origination Workflow

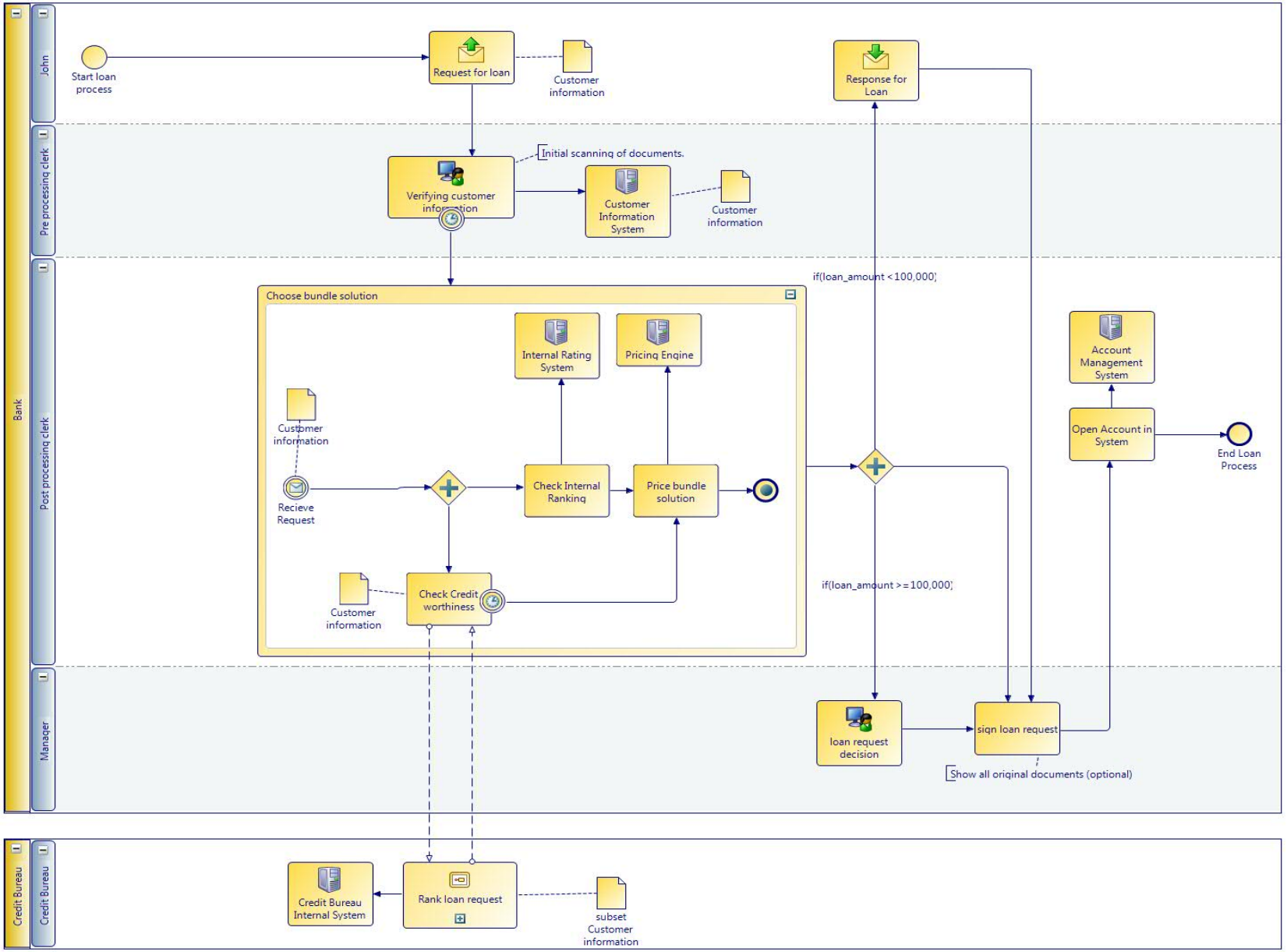
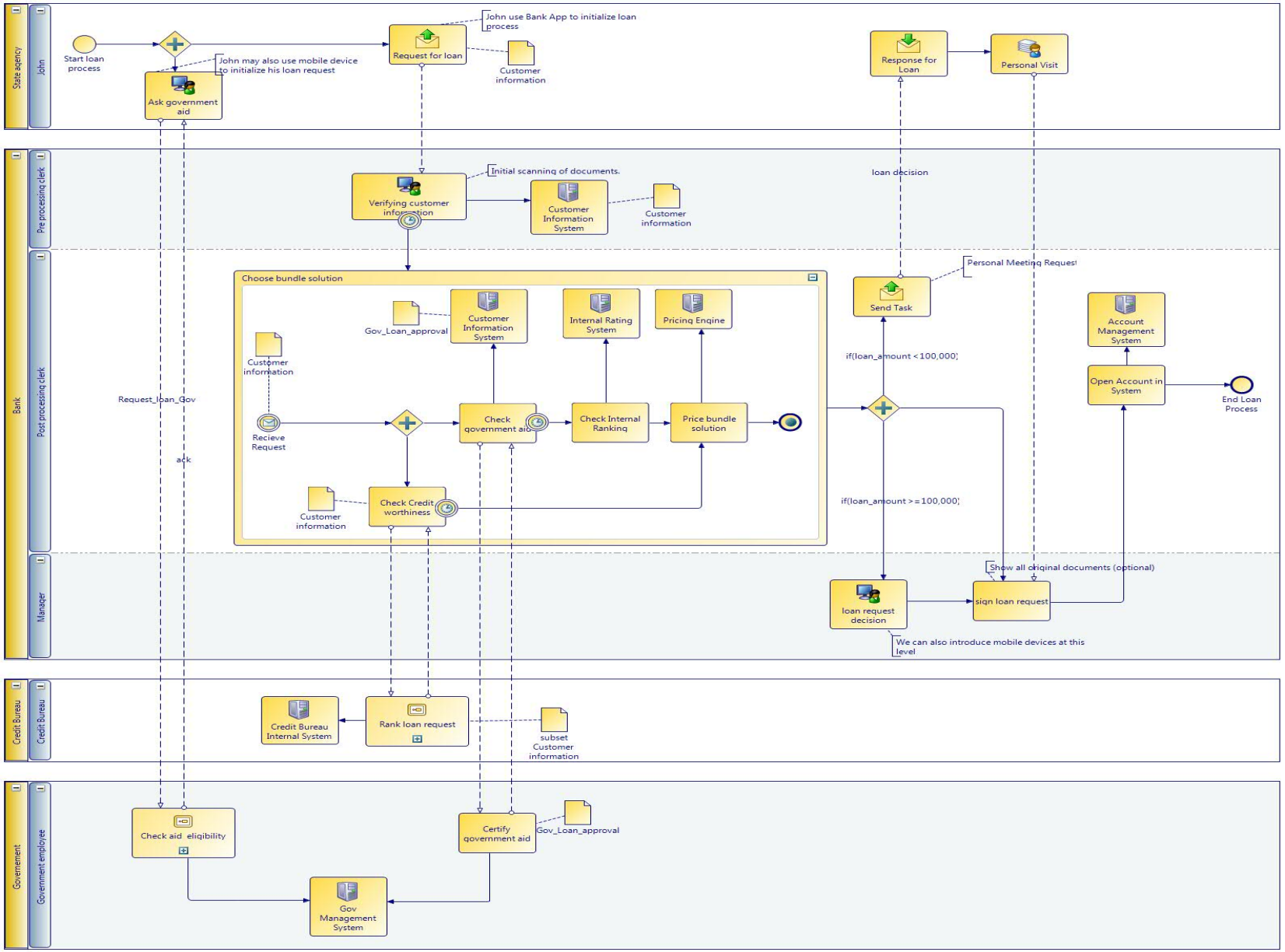


Figure 3.2: Evolution of the Loan Origination Workflow



Scene 1 John signs a ”promise of sale”

In the real estate agency, John is convinced he wants to buy the flat he is visiting, he wants to sign a ”promise of sale” to take the exclusivity. As he knows, there is a limited time frame once he signed the paper to gather documents in order to fulfill the contract. His bank, aware of this kind of regulation, provides a mobile application to accelerate the process. It's a trusted application issued from the bank, optimized to this specific usage and John use one of the first release. This kind of application is known as ”Mobile banking” and as it is run in restricted device, lot of limitation apply [36]. The state agency also owns an embedded device to interact with both John and the Bank. This application is able to request and send early documents such as, ”promise of sale”, loan application form etc. to the bank in order to prepare the first estimation and schedule an appointment. John provides the authorization by means of an electronic signature which will allows bank to retrieve or verify John personal information.

Also, in regards of his profile, John is eligible to government aids. First of all, he uses his mobile device to ask which amount he can get from the government, for how long and with which condition, then he advises the bank he asked for aid.

Scene 2 Bank verifies John's request

At the BBB bank, Peter (the bank pre-processing clerk) checks the Customer Information File sent by the mobile application during the initial phase. Peter processes John's confidential data. The history of John's bank account is definitely regular. Moreover John has been a BBB's customer for a long time and his parents too. In addition to this good saving profile, the most important credential for John relies in the permanent and stable job position he has recently obtained within a public administration.

As a positive point, John also claims he is eligible to government aid for a certain amount. As it is not Peter's role to verify this information, Peter checks the consistency of the case and agrees government will provide support. He also verifies he got all documents needed for the next phase, included the digital signature which certify John allows that personal information and payment behavior may be accessed from and supplied to third parties for risk management purposes. In conclusion, Peter's feedback on John's credentials is extremely positive at this stage. As required by CESSA these comments are opportunely signed by Peter and reported in both the Customer Information File and the process log. The loan origination process can move to the next phase.

Scene 3 The bank double checks the credit worthiness of John

The post processing clerk is responsible to double check the credit worthiness of John by means of a careful analysis of a more comprehensive risk analysis involving a larger set of data including sums of liabilities, sums of assets, third-party loans, reasons for rating, etc. It must be noted that the majority of this data are collected and maintained by different institutions than BBB. As stated above john will authorize the bank to verify his data. The access to this information is regulated by appropriate collaborative services. These services behave accordingly to precise security policies defined and

enforced by the CESSA framework through an authorization infrastructure that relies on pre-existing trust relationships between the data owner and the BBB bank.

In selecting the post-processing clerk the CESSA framework arises a separation of duty requirement imposing a mutual exclusion constraint on the tasks of pre and post processing. In our specific case, this results in preventing Peter to be involved in the post processing phase. Gabriel is chosen for this task and due to the fact that the amount of the loan inquired by John does not exceeds 1 million Euros, Gabriel's adviser is not required to intercede. Gabriel proceeds in the post-processing phase by querying, in parallel :

- the credit bureau in order to have ranking for john loan request (see section 3.1.2)
- the government in order to verify aid for john loan request (see section 3.1.3)
- the internal rating system (see section 3.1.4)

3.1.2 Assessing the loan risk using a Credit Bureau

The Credit Bureau is a third party business partner of financial institution that processes, stores and safeguards credit information of physical individual and industrial companies. Credit Bureaux gather data from various sources and cross-check and match the data for accuracy. Some of these sources include publicly available records (Courts and Deeds Offices) and Credit account details (from Credit Grantors or subscribers). Credit Grantors are companies such as banks, retailers and any other organization that needs to manage their risk when extending credit to the public. They are also called 'subscribers' because they subscribe to the Credit Bureau in order to collect, submit, use and share the information held in the database libraries. They use the information from the Credit Bureau to make decisions on whether or not to grant credit, in terms of their own credit granting policies. It basically performs three tasks: it gathers data from various sources (mainly banks and credit institutes), it cross-checks these data, it sells on demand this data to its clients to allow them for checking the credit worthiness of their customers. Of course all these tasks required a formal and signed authorization from the customer for treating his data [1].

All the credit Bureau institutes must comply with national legislation such as the national transcription of the European Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data. They may also adopt a code of conduct for ensuring the well functioning of the credit information industry in terms of maintaining proper mechanisms to ensure data processing and safety, ensuring high data safety standards to avoid the costs and consequences of loss of data or unauthorized access, and monitoring/auditing their systems regularly.

The role of the CESSA framework in the interaction between the BBB bank and the Credit Bureau is manifold: it offers mechanisms for guarantying the compliance of the Credit Bureau to the regulation and code of conduct, it ensures a secure inter-operation of the services running on the bank side and on the Credit Bureau side, it protects the privacy of the data exchanged, it takes care of managing and checking all the customer authorizations, and (for auditing purposes)

it keeps a record of all the operations performed. The Credit Bureau must especially balance the rights of the Credit Grantors to access detailed information on past credit behavior with the rights of Credit Receivers not to be prejudiced by out-of-date data. The CESSA framework may therefore provides schemes to insure the respect of obligations such as maximal specific data retention (for example 3 years for Account performance), to insure that only factual data are stored (excluding racial or rumors), and to grant the right for opting out.

In the case of John, the Credit Bureau does not return any negative information and Gabriel moves to query the internal rating application.

3.1.3 Government aid checking

In his loan request, John asks for governmental aid, and provide information on what he is suppose to get. As for the estimation the bank needs to be certain of what John will get, the bank contacts the government to verify that a process has been launched and to certify they'll support part of the loan with conditions.

In parallel of Credit Bureau assessment process, Gabriel launches the government aid certification process. For John's loan request, the government certifies John will receive a no interest loan up to 30% of the loan with a 20 years delay to reimburse it, and with maximal aid of 32 500 euros. The government also specifies conditions John should comply with. For example, the government aid is only for people who ask their first real estate loan, with limited resources. In order to get this money, the bank have to contact the government back when John will sign the loan.

At this stage, the bank is able to determine how the government will support John for his loan, thus evaluate the risk. If John isn't eligible to any aid, it's not a point of failure in the process. It only means there is less guarantees for this specific request, and the global ranking system may be less favorable.

The government has enough reputation to guarantee he will give aid to the bank once the loan has been accepted, but the bank still need electronic evidences. A certificate should provide non repudiation for example. The CESSA platform will help adding such technology.

3.1.4 Assessing the loan risk using an internal information

The CESSA framework assumes the control of this internal interaction for what concerns the security aspects (e.g., the data exchanged must be kept confidential, authorization policies need to be considered for accessing to John private data, etc).

The internal scoring application assigns a low risk level for John's application and, once Gabriel has stored this information, the loan origination process move to the third phase. In the circumstance of a negative scoring result, the application would have enforced the completion of the request assessment to Gabriel's manager.

Scene 4 The bank calculates the price for the bundled product (loan)

At this stage, the post processing is finished. It is now time to propose a bundle solution to John. It could be the first time John physically goes to the bank as everything before was done through electronic communication.

Gabriel has to choose the most appropriate bundled product for John in the database products. He queries the Pricing Engine service to compute a price for the bundled product (notice that this query does not need the real identity of the customer to be executed). The result in terms of, e.g., original price, customer segment special conditions, customer company special conditions, asset limit for price, is then returned to Gabriel and proposed to John.

In doing these operations the CESSA framework (i) ensures the provision and treatment of anonymous data on the side of the pricing engine service, (ii) protects the communication between the inquiring service used by Gabriel and the Pricing Engine service to preserve the integrity of the data, and (iii) provides appropriate log mechanisms to guarantee the transparency of the price calculation process.

Scene 5 The bank and John sign the form

John is now introduced to Ted, a manager of the BBB bank, to discuss the bundled product in more details and to finalize the process. After some negotiations, John and Ted finally come to an agreement on the loan conditions. The contract is digitally signed using the respective secure signature creation device (SSCD) of John and Ted. A copy of the contract is printed for John's documentation. In doing this the CESSA framework exploits an appropriate security protocol to enforce the non-repudiation of the signature, it should also ensure that an adequate certification authority and a time-stamping system are used during the signature process.

The contract is signed; Ted updates the BBB information system with the remaining data and provides to John a formal document stating that the amount of the loan will be transferred on John's bank account in one week (at most).

3.1.5 Evolution

One of the main advantage of CESSA is to provide tools to make evolve an application while ensuring secure property preservation. As state above, the two figures 3.1 and 3.2 represent respectively the initial state of the workflow and the evolved one modified by the CESSA platform.

On this scenario, the loan origination process is modified to take in account new possibilities. First of all, two new actors are involved namely a Real Estate Agency whose role is to facilitate procedures for a customer. The state agency may have agreement with some banks, and is able to initiate a loan process. The second new actor is the National Government which provide financial aid to customer depending on specific conditions.

The bank has to reconfigure its internal process to accept loan origination from a state agency, and should communicate with the government to acknowledge that part of a loan for a customer is covered by it. CESSA needs to allow this flexibility : reconfigure part of the process to integrate new organizations, with specific security policies and constraints.

One last thing, is the ability to integrate various devices, such as a mobile phone or embedded devices. They are used to accelerate the time to process a file, and thus reduce the response for the loan request. On the example of resource constraint devices, CESSA should provide a way to address constraints and express correct security preservation (*e.g.,*encryption on mobile device is resource and power consuming, but necessary for signature exchange. Thus CESSA shall provide an encryption scheme that balance the strength and the consumption).

3.2 Methodology

Before giving requirements, we describe the method that we will follow to formalize the requirements in our future developments. We consider the two stages, infrastructure development and application development. For each stage, the development process involves different roles, different kinds of requirements and different abstraction levels.

First, we have identified four main roles.

Specifier The specifier defines the properties that an artifact must satisfy. These properties express requirements in specification languages, which can be more or less formal.

Developer Given a specification, the developer implements an artifact conforming to the specification, tests it, then deploys it and finally maintains it. Actually, the development phase involves different sub-roles, corresponding to the development cycle, from implementation to maintenance.

Verifier Given a specification and an implementation, the verifier checks their conformance. Verification can lead to certification: a certificate asserts the conformance with some level of trust, which is no more implicit. Verification involves verification languages, allowing conformance relations to be formalized and measured, leading to a dependability evaluation.

User The user manipulates the artifact developed in order to complete the needs that has been captured into the requirements. The user can estimate the trust in the completion by resorting to certificates.

These roles, except the verifier role, are standard, as they follow the standard development process. We introduce the verifier role to fulfill the objective advertised by the project CESSA, namely improving trust in service-oriented infrastructures.

Two main kinds of requirements will be considered.

Functional requirements Functional requirements define functions to be implemented. A function is described with its inputs, its outputs and its behavior, defining computations and communications to be provided.

Non-functional requirements Non-functional requirements specify global properties for the behavior of the whole artifact, in addition to the functional requirements, dedicated to

functions composing the artifact. These properties corresponding to constraints can be split into two main categories: execution properties, like safety and security, and evolution properties, like maintainability and extensibility.

Each role and each requirement can be described at different abstraction levels, which need to be related in a refinement process.

Abstraction levels Abstraction is a way to analyze an artifact by changing the level of detail to be considered, and thus allowing information that is not relevant to be forgotten. An artifact can be considered at different abstraction levels, each one corresponding to a specific analysis.

Refinement Refinement is the relation between two abstraction levels, the former abstract, the latter more concrete. While the structures at the different levels may significantly differ, the behaviors and the constraints are generally preserved.

In the following, in the description of the requirements, we will adopt the preceding classification. First, two classes of requirements will be introduced, the former for the technological solutions, the latter for the proofs of concept. Second, for each requirement, we will determine:

- its nature: functional or non-functional,
- its features, taking the different roles successively,
- its abstraction level.

For instance, suppose we have to describe a simple functionality, represented by a function. At the proof of concept level, we define the following requirements, according to the different points of view.

- Specifier's point of view: definition of the function with a precondition and a postcondition in some Hoare logic [120, chap. 3]
- Developer's point of view: possibility for the function to be implemented in a language allowing contracts to be defined
- Verifier's point of view: possibility for the precondition and the postcondition to be checked by a tool
- User's point of view: documentation of the function integrating the definition of the precondition and the postcondition

At the technological level, we need to define the logic used and the associated tools.

- Specifier's point of view: definition of some Hoare logic allowing preconditions and postconditions to be expressed

- Developer's point of view: implementation of a contract layer in some given language, à la Eiffel [100, chap. 11]
- Verifier's point of view: formal verification that the contract layer allows preconditions and postcondition to be checked
- User's point of view: use of a human-oriented logic for expressing preconditions and postconditions

For the preceding requirements, we can consider that they deal with the same abstraction level, corresponding to the programming language used.

To conclude the methodological section, we mention that will adhere for expressing requirements to the convention described in the document `RFC 2119`[1], published by the Internet Engineering Task Force (IETF). This document gives the precise meaning of several words used to signify requirements in a specification, like "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL".

3.3 Requirements derived by the industrial partners

This sections intends to give some requirements derived from different sources. Some requirements are driven by the use case, thus involve actors from this scenario, while others are driven by the underlying infrastructure. Also, we can specify requirements for two kinds of models : the service model (2.1) and the aspect model (2.2).

3.3.1 Large-scale business infrastructures

Requirement: (Migration scheme) CESSA allows the evolution of an application while preserving security properties. A business process is a flow of tasks and services that are chained together, and there can be many "instances" at different stages already launched. The question with an evolution is how to know which instances at a specific stage can switch to the new business process. We certainly need a migration scheme whenever possible, and a version management in order to avoid breaking the application. □

Requirement: (Standards) CESSA shall use distributed and cross-organizational deployment standards supported by the industry to be well integrated with all solutions (like SAP's NetWeaver). □

Requirement: (Collaboration) CESSA should be able to operate among many different business entities at the same time. □

Requirement: (Policy consistency) CESSA should take into account local policy in each entity to avoid incompatibility while performing an evolution. □

[1] See http://tools.ietf.org/html/rfc2119.

Requirement: (Tools for evolution) The CESSA platform should provide tools to ease evolution. For instance, at each stage, it should display the current state, highlight steps where security properties stands and give a list of corresponding possible changes. □

Requirement: (Infrastructure dependencies) The set of non-functional properties available depends on the infrastructure. It means that the specifier and developer have to know what is the underlying infrastructure to know what is available. Likewise, some non-functional properties may have restrictions depending on the infrastructure. It means that to cover all platforms, different implementations of the same aspect exist. This will happen for example when you want to do encryption. A resource-constrained device will have a subset of capacities of other platforms in other platforms. □

Requirement: (Security properties) CESSA will focus on a limited set of security properties (non repudiation, separation of duty, monitoring, access control, etc.). The complete list will be provided - with details - in the deliverable 2.1. □

Requirement: (Dynamic evolution) An evolution should involve a dynamic reconfiguration without service interruption. □

The following requirements are driven by the loan negotiation use case (3.1). We are describing few examples of the service model and the aspect model applied to the use case.

Requirement: (Separation of Duty) The scenario requires a separation of duty at the bank, to avoid that one employee bypass the four-eyes principle. □

Requirement: (Peer trust) A trust should already be established between the Bank and the other actors (Real estate agency, government, credit bureau). □

Requirement: (Peer agreement) An evolution involving more than one peers should imply an agreement between peers. □

Requirement: (Monitoring) The CESSA platform should enforce monitoring to raise an alert in case of bad behavior. For instance, the pre-processing clerk has limited access to the customer file. □

3.3.2 Infrastructures for embedded devices

Partner IS2T contributes several requirements specific to their hardware and software environment that should (also) be targeted by the CESSA project.

Requirement: (Target devices) Support a representative high volume production product, like an Embedded Artists' LPC2478 board that is based on NXP ARM7 microcontroller. Other devices, such as a more powerful CPU like an ARM9, are acceptable devices if they meet stringent price limits. □

Requirement: (Communication) Hardware devices have to support communication connections, *e.g.,*for database access, via a link (ethernet probably). Furthermore, suitable communication protocols with other infrastructures (*e.g.,*by SAP) need to be available along with corresponding system integrity properties. □

Requirement: (OSGi Me) CESSA should support OSG Me as a target platform. □

Embedded software is not just "software on small machines." Embedding applications usually is about optimizations that strive to create a software specific for some hardware. System integration is one of the most important aspects in this context and heavily impacts testing and debugging. This is a big difference between PC-based systems and embedded systems. For PC-based systems, the development machine is the machine that will execute the application (or a fairly similar machine). For embedded systems, there are actually two systems: one used to design the application (a PC-like desktop) and one to run the application (the device). As OSGi-R4 brings benefits in terms of software architecture for PC-like systems, OSGi ME brings as much benefits to the embedded world. The philosophy of OSGi ME may be summarized by: "A faulty service cannot cause the failure of the entire platform."

OSGi ME distinguishes bundle designers from application designers. The first ones provide services while the second ones make use of the registered services. This is very much in sync with the CESSA role model (Specifier, Developer, User, Verifier). The emphasis is on the fact that while the bundle designer constructs a bundle with its set of services, he does not control the life cycle of the services his bundle provides: this is at the discretion of the application designer. Moreover, there may be several application designers that make use of services, possibly unregistering old services and registering news ones during the whole live cycle of the device that runs the OSGi ME framework.

Requirement: (Security support) Devices should provide (either in hardware or software) basic support for security, a minima, for authentication. □

Appropriate mechanisms range from specific hardware support to full software solution like the NTX software solution. A software solution allows to get rid off hardware which lowers the device prices and allows for a potentially much larger diffusion if integrated in a Java application platform.

Requirement: (REST) The CESSA access model to services must support the REST web server protocol (because of its low resource requirements). □

Requirement: (Device AOP) The aspect model should enable fine-granular access at the device level, probably at the level of individual bytecodes. Aspects have to be written in Java, or at least have a binary representation in a Java classfile. □

Requirement: (Device migration scheme) One of the main issues with migration is about collaborations of processes that may share some global resources. Migration must have a well-defined semantics in order to allow for a stable system that must never become inconsistent.

The migration scheme should be transactional in order to maintain the system in a consistent state after a migration (either the migration has succeeded, either it had no effect). □

This is a refinement of Req. 3.3.1.

Requirement: (Transactional VM) Provide transactional support on the VM level. □

Interactions between transactional context in the JAVA world and the transactional context in SAP databases should be made explicit in order to maintain the transactional context during a whole transaction call from Java down-to a single database atomic operation. This will ease guaranteeing the liveness of the transactional context during a full context of execution, even if it crosses platform and languages barriers.

Chapter 4

State of the Art: academic approaches

In this chapter, we mainly deal with academic approaches to service-oriented computing and aspect-oriented computing. We have also found useful to describe related work around interaction protocols, a notion used not only in service-oriented computing, but also in component-oriented computing.

4.1 Service-oriented computing

Web services appear under two different flavors: Web services and semantic web services. The latter is an extension of web services with more semantic information about services, and operations. This information is required to automatic services discovery and to enable automatic and dynamic composition (at run-time) of services. However, they are only few proposals for semantic web services and dynamic composition [33]. The static composition (at design time) of web services is possible in two manners called *orchestration* and *choreography*.

Orchestration defines the local view of service interactions while choreography addresses a global view of these interactions. The orchestration describes the interactions from one service to other services it is linked. An orchestration makes explicit a central coordinator which is responsible for invoking and combining the web-services realizing the composition.

A choreography do not assume a central coordinator, it defines the collaboration as the set of messages and the interaction rules between the services. Choreography, contrary to orchestration, describes a global view of the observable behavior (message exchanges) between all the services involved in the composition. This distinction between orchestration and choreography could seem subtle, [65] states they should both merge in the future. The expectation is that a choreography could describe the external visible behavior of a service while an orchestration could define how the involved services are cooperating to realize the service composition. It is a common practice in formal protocols which distinguishes between a protocol description and its observational behavior and usually the formal languages are not the same. BEPL4WS is an orchestration language while WS-CDL is a choreography language. Following the survey in [33], from a tool support point of view, most of the commercially existing approaches are dedicated to BEPL4WS. A more in depth analysis and comparison of existing platforms for service

composition was done in [66].

RESTful services [121] has gained widespread acceptance in the Web community as a simpler alternative to SOAP and WSDL based services. Regarding the specification of the workflow process or service protocol, the situation for RESTFul web services seems much simpler than SOAP based web services. A comparison of the SOAP and REST standard is available in [138] as well as discussion about standard and workflow proposals. As far as we know, there is no adopted standard to describe protocols, workflow, or composition of RESTful services. However, there are still some academic proposals like the extension of BEPL for REST services [114].

They are already various attempts to provide formal models for protocol services and to define their composition. One first piece of information is the survey of Beeker et al. [129]. Quoting this paper: "Neither of the industrial approaches offer any direct support for the verification of service composition at design time". However, they are many attempts to formally specify and analyze the behavior of BEPL. The above paper collects approaches to formally describe orchestration and choreography. They are using mainly state machines (timed automata, Petri nets) or process algebras like CCS, the π-calculus and LOTOS. The authors note that process algebras like the π-calculus and LOTOS are better due to the need of a value passing mechanism. Another survey is [118] which explores automatic service composition using workflow techniques or artificial intelligence planning. The latter is often used to automatically generate the process model in case of automated composition.

Various calculi for web services exist focusing on different features (distribution, session, choreography, orchestration, etc) and targeting different goals (formal specification, verification, formal properties expression, etc). Behavioral types, type and effect systems, have been used to check security properties in [21]. The use of spatial logic to express, in addition to behavioral properties, resource usage and ownership is presented in [37]. [27] introduces a service calculus, it is a process calculus making explicit notions of service definition, service invocation and session handling. Session types provide a programming style for structured interaction, and static checking for the composition of service protocols [137]. The work of [94] proposes a calculus for service-based systems, suitable to describe both orchestration and the conversation. This paper presents two equivalent semantics as well as an illustration of a static type analysis, bisimulation and deadlock-avoidance. [132] is a calculus addressing distribution, process delegation, communication and loose coupling. It is based on a fragment of the π-calculus and ensures the congruence of bisimulation.

4.2 Interaction protocols

Service-oriented computing is a new technology which promotes the decomposition of software applications into a set of collaborating services. Services are atomic elements to build applications and support a loosely coupling. A service can be viewed as a component: An independent unit of computation exporting its interfaces. But in addition services are published in repositories which can be browsed for service discovery.

Notions of protocols In order to successfully interact, entities (objects, agents, components, services, etc) need to conform to a certain form of contract. Interaction protocols, generally, describe the entity behavior in terms of allowed message sequences that must be exchanged with other entities in order to perform the system intended global behavior. A protocol is a language dedicated to the description of interactions, controls, and communications between several entities. These formalisms can be extended to describe the synchronizations, collaborations and possible data exchanges between several partners in a composite system. They are often used to check for some important properties like safety or liveness of the compound system but also to predict the performance or bottlenecks in a complex assembly. Of course, a first variation is the language to describe entities (objects, agents, components, services, ...) and the context (distribution, mobility, network dynamicity, real-time, ...). There are various ways to describe these protocols, we classify them below.

State machine This is surely the most important family and the most used to describe protocols. Basically it consists in states and transitions between these states [12]. Transitions as well as states can be decorated. Generally, transitions are completed with label events, variables, guards and emission/receipts, among others. One of the simplest instance is the Labeled Transition System (or LTS) which is a graph of states and transitions where labels of transition denote events to go from one state to the next. There are used for the specification and verification of many dynamic systems. Many extensions of this exist, the statechart of UML is one of the most complex, timed automata [10], symbolic transition systems [38, 79, 98], counter automata [81], communicating machines [30] and so on. Concurrency is generally modeled using specific construction like the synchronous product of automatas.

Another important sub-family is Petri Nets [105] where concurrency is directly embedded in the state machine. They are many variants of these machines with very different properties in term of expressiveness and verification [63, 64, 112, 125, 72].

Process Algebra The term process algebra refers to a family of specification techniques well suited to describe concurrent and communicating systems. Process algebras describe the process interactions in terms of calculus based on few primitives like choice, sequence, or parallel composition of processes. Processes are defined as assembly of atomic actions and other processes using the composition operators of the language. There are many process algebra formalisms, the most important ones are CCS [103, 104] and CSP [80] as they stand at the base of other algebras like the π-calculus [102] and the mobile agents [39], integrating primitives for the expression of distribution and mobility. Many variants of these calculus exist, one example is LOTOS [26, 83] which is a process algebra approach completed with an algebraic specification data language and normalized as an ISO standard for telecommunications. But many others exist introducing time or stochastic behavior.

Behavioral types model abstractly and precisely the intended behavior of a system, they are particular process algebra formalisms. O. Nierstrasz in [108] proposes regular types for active objects and an associated notion of substituability.

Modal logic Modal logic is a family of formal logic based on modalities qualifying generally the time but it can be some other domains. The logic extends some classic logics with new operators, for instance the *ALWAYS* or *UNTIL* operators. One familiar modal logic is linear temporal logic, more advanced logics are CTL and CTL* which allow branching time, thus to reason on more than one time line [67]. An alternative is the μ-calculus which is an extension of the propositional calculus with an explicit least fix point operator [91, 119].

Coordination Language In comparison with more usual languages in which the interaction part of compositions (*i.e.,*communication, synchronization) is embedded within the computation part, coordination languages [113] promote separation of concerns hence the definition of interaction as a first-class entity, described separately from computation. Coordination languages are split in two categories. Data driven languages propose expressive but low level communication mechanisms based on shared data spaces, such as Linda tuple spaces and their Java implementation, javaspaces. On the contrary, event driven languages such as Reo promote more abstract coordination patterns based on events corresponding to the coordinated entities input and output ports.

These various means have differences regarding their user readability, expressiveness, effectiveness of the verification methods, and existing tool supports. State machine is surely the family widely used by engineers, while process algebras or temporal logics are often preferred for theoretical analyzes.

Protocols for components There are numerous approaches which discuss protocols and component programming [9, 96, 97, 13, 55, 24, 16, 73, 92, 122, 18, 98, 78, 85, 31, 23, 28, 19, 22, 34, 117, 116, 35, 11, 15, 76, 77, 40, 70, 69]. Components introduce some constraints on the use of protocols. Component-based software engineering (CBSE) relies on defining primitive and composite components with provided and required interfaces. Communications can be based on direct bindings or the use of connectors. In the context of services only binary connections are relevant, for proposals considering richer connections see [22, 69]. With binary messages, the binding mechanism is rather simple, since a port is, either connected or not connected, and either hidden or visible outside the current scope.

We will here refer to component types rather than component instances. We can note that the formalism used at the level of a primitive component may be different from the one of the composite levels. For instance, WRIGHT uses process algebras everywhere, while Korrigan defines primitive components with state-machines and a modal logic at the composite level. Both primitive and composite component types declare their externally visible events. The notion of hierarchy is related to the visibility of events outside the context of a composite. Inside the context of a composite, only visible events of contained components (those that appear in their interface) can be synchronized. Events can be hidden or exported outside a component context depending on the type of the event exportation. A hierarchy allows some internal events (made visible by contained components) to be hidden for outside of the current hierarchical level (composite context) while exported events may be synchronized. The protocol language at the composite level is often called a *glue language*. To get more flexible interactions it is important to pay attention

to the design of this language. To improve components and protocols reusability, partners of communications should be defined at the glue level only. For instance, in Korrigan, the partners of a communication are defined in the composite which assembles the communicating components. Thus there is nothing in the protocol of one component denoting a communication with a precise partner.

Finally, both services and components shared common features and common issues. They are both based on primitive elements (primitive component, primitive service) and allow hierarchies of elements (composite component, process, business service). They are both specifying required and provided interfaces and using a notion of scope. They are also proposing various synchronous and asynchronous communication modes. However, they are still some important differences, for instance CBSE consider interaction with two or more partners while web services rely on point-to-point communications. Services consider transactions and resources as important aspects to deal with, this is generally not covered by component languages.

4.3 Aspect-oriented software development

Modularity is a key concept of software development. When a piece of software gets big and complex, it gets difficult to design the right architecture: an architecture that enables to program each concern modularly. Aspect-Oriented Software Development [88] aims at providing support for a better separation of concerns in such a system. In the following, we discuss four sets of related work:

- Aspects for the definition and evolution of (web) services
- History-based aspects
- Formal semantics and properties for aspect-based systems
- Aspects for the evolution of protocols

4.3.1 Aspects and (web) services

Four kinds of related work are presented in the following: approaches using AOP in the context of de/centralized web service composition, distributed web service composition infrastructures, and approaches for AOP in distributed systems.

AOP and decentralized web service composition. There are only very few approaches applying AO techniques to distributed web service composition. A notable exception is Aspect-Sensitive Services (CASS) [52], which provides a distributed aspect platform that targets the encapsulation of coordination, activity life-cycle and context propagation concerns in service-oriented environments.

AOP and centralized web service composition. Some more recent AOP approaches are explicitly targeted at Web services. With Padus [29] and Ao4BPEL [42], aspects can be (un)plugged into BPEL composition processes. Since BPEL processes consist of a set of activities, joinpoints in Padus and AO4BPEL are well-defined points in the execution of the processes: each BPEL activity is a possible joinpoint. The attributes of a business process or certain activity can be used as predicates to choose relevant joinpoints. BPEL, Padus and AO4BPEL realize centralized compositions.

Singh et al. [126] present a software architecture for web services: Aspect-Oriented Web Services (AOWS). It is targeted at describing crosscutting concerns between web services to give more complete description of Web services, supporting richer dynamic discovery and seamless integration. An implementation is made on the .NET platform and all AOWS subsystems and their relationships have been formally modeled. While aiming to achieve similar goals as the WSML, AOWS does not support third-party independent services as services need to be modeled in an AOWSDL language, and registered in a dedicated AOUDDI registry.Multiple services are bundled in a centralized fashion in an AOComposite. The aspectual features of the AOWS framework are used to provide more efficient and effective dynamic description, discovery and integration.

Decentralized web service composition. A few approaches have recently been put forward for decentralized web service composition that do not employ AOP techniques. Most notably, Chafle et al. [41] propose techniques for the partitioning of web service compositions and error handling mechanisms for distributed web service compositions.

4.3.2 History-based aspects

AspectJ [89] is the seminal proposal for AOP. This language supports pointcut and advice definitions. A pointcut is an expression that denotes execution points. An advice is a piece of code to be executed before, after or around (*i.e.,* instead of) an execution point. An aspect definition can contain several pointcuts and advices in order to implement a given concern. An aspect is quite similar to an execution monitor: each time an execution event is matched by a pointcut, its associated advice is executed. Pointcuts can be defined independently, or they can be related with one another. In particular, AspectJ provides an operator `cflow` in order to select a method call within the control flow of another method call. This mechanism can be formalized as (call) stack inspection [124].

Other mechanisms have been proposed to relate pointcuts with each other according to an history. For instance, AspectJ also offers a static operator for control flow, `pcflow` (for predicted control flow), which can be formalized as inspection of the static call graph. An operator `dflow` (data-flow) has been proposed in order to relate pointcuts related by an information flow [99]. Event-based AOP [62] and trace matches [8] both can be formalized as inspection of the execution trace. Event-based AOP has been adapted to a concurrent context in order to inspect the interweaving of the concurrent execution traces [58]. This line of work has also been extended to a distributed context, in order to match control-flow and sequence across several hosts [106].

The challenges of these work is to provide high-level yet efficient pointcuts. Their implementation can require static analyses for efficiency concerns [111].

4.3.3 Formal semantics for and properties of aspect-based systems

AOP has first been supported by tools such as AspectJ, then came semantics. Numerous semantics have been proposed for AOP. Some of them describe an implemented system such as AspectJ, or one of its subset for these systems are often big and complex. Others are more generic but they require translations of existing systems. As for any other language, semantics can be either denotational or operational. The more abstract ones are presented as calculus. We detail here a few representative of them.

First, Wand *et al.,* [134] propose a denotational semantics for an idealized AspectJ-like system. Their language offers joinpoints, pointcuts and advices. The base language is a subset of scheme with first order function only. The keyword `proceed` and `cflow` are supported, but dynamic pointcuts that dynamically check a predicate and exceptions are not supported. This first semantics have provided a better understanding of some core notions of AOP, however it is complex and it has not led to formal study of properties.

Clifton and Leavens [44] propose a small step operational semantics that describes each execution steps (hence implementation) in more detail. It is based on a subset of AspectJ and on a subset of Java. This language offers the main mechanisms of an object oriented language: class, object instantiation, method call, late binding, etc. Only `around` aspects can be defined but their advice can modify the parameter of `proceed`, in particular it can change the receiver of the intercepted method call. This detailed semantics operationally describes the method call interception but it does not cover more advanced features such as dynamic pointcuts based on a predicate, exceptions, or dynamic instantiation of aspects.

Jagadeesan *et al.,* [32] propose a aspect calculi that does not depend on a given programming language. In their calculus, each computation emits an execution event. These events can be seen as messages transmitted from a sender to a receiver. In this context, an `around`-like advice can intercept and modify message passing. The authors have demonstrated how to encode a functional language and an object-oriented language in their calculus. This semantics does not enable history-based pointcuts.

Last but not least, the Common Aspect Semantics Base (CASB) [56], is a generic framework for defining small step operational semantics of AOP systems. It does not rely on a particular programming language but it offers different constructs and rules according to the kind of language to be formalized. This enables us to keep the semantics as simple as possible by introducing only the required features. This framework has been successfully used to model a subset of Java, as well as several specialized aspect languages that preserve families of properties [57].

Semantics are interesting for they provide a simple, abstract definitions of the main mechanisms of a language. And they are yet more interesting when they enable to study formal properties. We now detail these studies.

The generalization and formalization of the call stack inspection-based aspects [124] models a pointcut by a regular expression (of the stack). This formalization enables to statically detect (by computing the intersection of two regular expression) when two aspects execute their advice

at the same joinpoint. The conflict of aspect is one of the most studied property. In [59, 60] we studied how to statically detect conflicts of stateful aspects (*i.e.,* aspects with history based on the execution trace). This analysis is based on the product of the two automata that represents the sequences of expected events by each aspect. Latter [56], we studied some classes of aspects that preserve some families of properties for the base program. In general, any safety or liveness property of the base program can be violated by weaving an aspect. In our study, the less powerful the aspect is, the more it preserves properties. We have identified families of properties to be preserved defined by subsets of the temporal logic LTL. And we have identified the corresponding classes of aspects. This study led us to design restricted aspects languages [57] that preserve by construction a family of property (*i.e.,* any aspect that can be defined in the class preserves all properties of the family). This has been formally proved in the context of our Common Aspect Semantics Based (CASB) framework for an imperative base language. Our families of properties are based on previous work by Katz [87].

Other works focus on specific aspect classes. They use different techniques such as typing ([54],[53]), Hoare Logic ([43]), proof carrying code ([20]). Some work propose a modular verification techniques [93, 75, 86]. All these work are devoted to general classes of aspects (for instance, the class of aspects that do read but do not write the variables of the base program). Such a large class of aspect is of few use to deal with security properties. One notable exception is the work of Fradet and Hong Tuan Ha [74] that defines an aspect language to prevent the denials of service such as starvation caused by resource management.

4.3.4 Aspect-based evolution of protocols

There are some approaches which consider aspect languages that explicitly support the manipulation of protocols, most notably [7, 61, 133]. Approaches [7, 61] feature *regular aspect languages* and a framework for static analysis of interaction properties. The language introduced by Walker and Viggers[133], one of the very few approaches providing non-regular (but not turing-complete) pointcut languages, proposes tracecuts which provide a *context-free pointcut language*. However, all of the above approaches do not use the language for an integration of aspects and components or explore the problem of property-preserving for systems that have protocols being modified by aspects. Farías [68] has proposed a regular aspect language for components that admits advice modifying the static structure of protocols and considered *proof techniques* for the resulting finite-state based aspects.

More recently, Nguyen and Südholt [107] have presented the first exploiting formal methods to investigate the preservation of compositional properties such as compatibility and substitutability for component-based systems that are subject to evolution by protocol-modifying aspects. There approach is based on a pointcut language that allows matching of VPL (visibly pushdown language) expressions, a language class strictly larger than regular ones but also strictly smaller than context-free ones. VPLs have all effective (but less efficient) decision procedures and closure properties that are enjoyed by regular languages.

Chapter 5

State of the Art: industrial approaches

In this chapter, we first describe implementations of service-oriented infrastructures. There are two mainstream implementations, using the Web, the WS* stack, corresponding to a process-oriented model, and the restful web services, corresponding to a resource-oriented model. Second, we describe cloud computing, a new field where service-oriented computing is intensively used. Indeed, with cloud computing, software, platforms and infrastructures are provided over Internet as services. Finally, we review existing infrastructures and standards of our industrial partners.

5.1 The WS* stack

This section provides an overview of established specification languages and infrastructures for Service-Oriented Architectures (SOA). The Organization for the Advancement of Structured Information Standards (OASIS) defines a `service` as *a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description* [47]. The SOA properties, such as loose coupling, interoperability, re-usability, etc., provide a good computer system architectural style for creating and using business processes, packaged as services, throughout their life-cycle. Nowadays, SOA is typically implemented by Web Services, such that services are made accessible via Web interfaces using XML. It has to be noted that a service-oriented architecture is not tied to a specific technology. It may be implemented using a wide range of technologies, including SOAP, RPC, DCOM or Web Services. A SOA can be implemented using one or more of these protocols. In this respect, several industrial standards are specified (see figure 5.1). As a starting point, we begin with a short summary of most important and extensively used standards, to emphasize the main characteristics of service orientation: their heterogeneous, distributed, and dynamic nature.

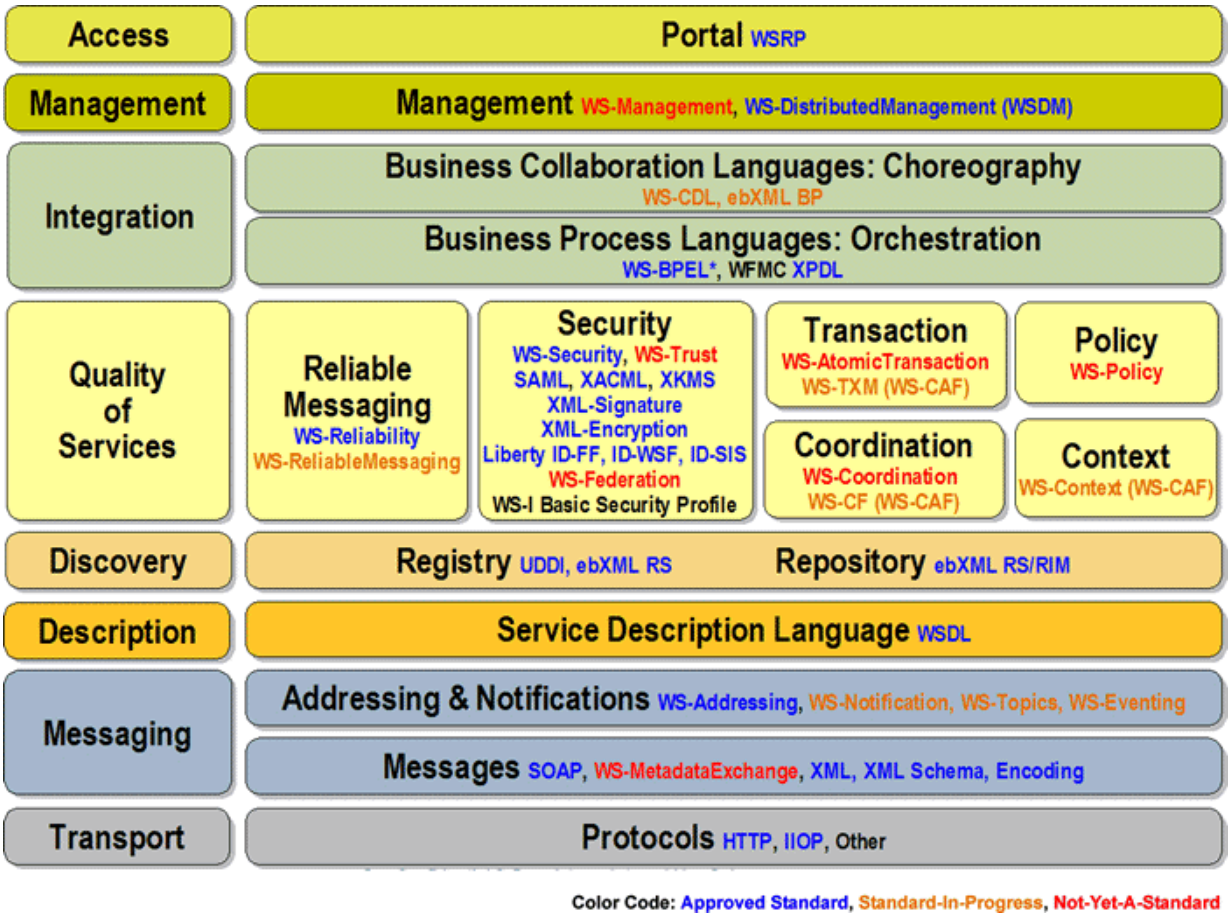


Figure 5.1: SOA Industrial Standard Stack

5.1.1 Integration Layer

The integration layer corresponds to the two upper layers of our service model, collaborations and processes.

Business Collaboration Languages : Choreography There are two main proposals for choreography.

Web Service Choreography Description Language - WS-CDL The Web Services Choreography Description Language (WS-CDL) is a W3C candidate recommendation since 2005 [136]. Many concepts are similar to WSCI, and, once more, the philosophy of WS-CDL is not the definition of an executable description or a new web-service, but to describe the common collaborative observable behavior of a system in terms of abstract business processes. This may also be regarded as a protocol between autonomous parties without a central point of control. When a service does show a behavior violating these constraints, this is considered as an error in the sense that it is not compliant with its WS-CDL description.

WS-CDL is related to formal methods, in particular pi-calculus by which some concepts have been inspired. However, the designers have not defined a formal semantics so far and many questions remain open, in particular the mapping of constructs to lower levels of the stack, in particular the link to WSDL and BPEL descriptions [17].

Web Services Choreography Interface The Web Service Choreography Interface (WSCI) was proposed in 2002 by BEA Systems, Intalio, SAP, and Sun and has the status of a note of the W3C since then [130]. Apparently, it never made it to a recommendation so far. WSCI works in conjunction with WSDL so that one can dynamically find web services and use them in ones business processes which requires a definition of the possible collaboration and interaction of partners. A WSDL description provides basically static information about a service; WSCI complements the static interface details provided by the WSDL file by describing the choreography of operations. Thus, WSCI describes the observable flow of messages of (stateful) web services. The goal is to enable users to understand how to interact with a service in a meaningful way. More in detail, the additional information about the dynamic behavior of web services that a WSCI can provide and that are not covered by a WSDL description are the following.

- Sequence of messages and processes, e.g. the operation of placing an order must occur before the actual payment operation is performed.
- Correlation of messages, e.g. correlating a reply (and subsequent communication) to a request using an order-ID.
- Workunits: what triggers a message exchange/process, when is it finished? This includes also the ability to either execute the entire unit or restore a consistent state prior to execution (compensation).
- Handling of exceptions like a time-out. WSCI also supports catching exceptions, allowing for recoverable operations.
- Defining Contexts in which variables are shared or to which exceptions are related.

There are basic activities like request and response messages or invocation of external services, and structured activities like sequential and parallel processing.

Business Process Languages : Orchestration Business processes can be described at two levels, a modeling one and an execution one. Sometimes, during its execution, a business process mixes automatic tasks and human tasks. Here are the corresponding specifications.

Business Process Modeling Notation - BPMN BPMN is a standardized graphical notation for drawing business processes. BPMN provides the capability for defining and understanding business processes through a BPD, which gives the ability to communicate business procedures1 in a standard manner. BPMN is based on graphical notations which are used to symbolize business processes. BPMN supports modeling concepts that are applicable to business-to-business processes, it facilitates communication between all users/roles/groups

of complex business processes. BPMN was developed by Business Process Management Initiative (BPMI), and is now being maintained by the Object Management Group. There are 53 current implementations and 4 planned implementations of BPMN.

The primary goal of BPMN is to provide a standard notation that is readily understandable by all business stakeholders 2. These business stakeholders include business analysts who creates and refine processes, technical developers responsible for implementing the processes, and business managers who monitor and manage processes. Consequently BPMN is intended to serve as common language to bridge the communication gap that frequently occurs between business process design and implementation.

Business Process Execution Language for Web Services -BPEL4WS BPEL4WS is an XML schema based abstraction that enables the composition of multiple synchronous and asynchronous web services into an end-to-end business flow. BPEL4WS provides a formal mechanism for business process management systems to define and execute business process and to inter operate with each other.

BPEL4WS is an OASIS standard which was renamed WS-BPEL. BPEL4WS represents the uniting of two previously competing standards.

1. Web Services Flow Language (WSFL) from IBM, and Microsoft(support for graph oriented processes).
2. XML business process language in BizTalk Server(XLANG) (structural constructs for processes).

The WS-BPEL standard is based on other WS-* specifications, more precisely the WS-BPEL process model is layered on top of the service model defined by WSDL 1.1, that is a stateless model of correlated request-response (or solicit-response) interactions or uncorrelated one-way interactions (one-way or notification) and adds support for business transactions. Note that, e.g., the standard WSDL SOAP binding comprises one-way and request-response primitives only, thus providing an even weaker service model. Other bindings may support the full WSDL process model.

As defined in the abstract of BPEL specification [109], WS-BPEL is a language for specifying business process behavior based on Web Services. Processes in WS-BPEL export and import functionality by using Web Service interfaces exclusively. BPEL processes are executed by an execution engine, which publishes BPEL processes through a Web Service interface. Thus, every BPEL-process composed of Web Services is a Web Service itself and can be used as a component of higher-level BPEL-processes.

BPEL provides a language for the specification of both, executable and abstract business processes composed of Web Services. Abstract business processes are partially specified processes that are not intended to be executed and, thus, only describe the observable behavior of a process (*behavioral interface*). This interface captures constraints on the ordering of messages to be sent to and received from a service. Using abstract processes, concrete operational details of a service can be hidden from, e.g., a business partner. An

executable process on the other side augments an abstract process with all of the required concrete operational details so that the process can be executed by a BPEL execution engine. This is achieved by defining the execution order of a set of activities, the partners involved in the process, the messages exchanges between the partners, and reactions to specific events, exceptions or faults.

Human Interaction in Business Processes In June 2007, Active Endpoints, Adobe, BEA, IBM, Oracle and SAP tried to fill this gap and published WS-HumanTask (Web Services Human Task, Version 1.0) [4] and BPEL4People (WS-BPEL Extension for People, Version 1.0) [5] specifications. These specifications describe how human interaction could be implemented in BPEL processes. BPEL4People is not a new version of BPEL but extends BPEL using WS-HumanTask. Both specifications went into OASIS specification process quite recently.

The BPEL specification focuses on business processes the activities of which are assumed to be interactions with Web services, without any further prerequisite behavior. But the spectrum of activities that make up general purpose business processes is much broader. People often participate in the execution of business processes introducing new aspects such as interaction between the process and user interface, and taking into account human behavior. Different standardization efforts are performed in order to define human interaction in business process. Two specifications are introduced; Business Process Execution Language extension for People (BPEL4People) and Web Services for Human Task (WS-HT) to integrate human interaction in BPEL4WS. These specification introduces a BPEL extension to address human interactions in BPEL as a first-class citizen. It defines a new type of basic activity which uses human tasks as an implementation. The goal of these specification is to enable portability and interoperability:

1. Portability - The ability to take design-time artifacts created in one vendor's environment and use them in another vendor's environment.
2. Interoperability - The capability for multiple components (process infrastructure, task infrastructures and task list clients) to interact using well-defined messages and protocols. This enables combining components from different vendors allowing seamless execution.

Business Process Execution Language extension for People (BPEL4People) integrates human behavior in a Services Oriented Application (SOA). According to the BPEL4People specification A new type of basic activity which uses human tasks as an implementation, and allows specifying tasks local to a process or use tasks defined outside of the process definition. This extension is based on the WS-Human Task specification. Web Services Human Task is an activity performed by humans and considered as part of the business process. A human task might be as simple as only Approval and as complex as Delegation of task. The WS-Human Task specification introduces the definition of human tasks and notifications, including their properties, behavior and a set of operations used to manipulate human tasks. A coordination protocol is introduced in order to control autonomy and life cycle of service-enabled human tasks in an interoperable manner. Web Service

for Human Task (WS-HT) provides business analyst the ability to define human behavior, this can be done by services offered by a task and a person assigned to the task to perform the activity i.e. Approve Document. At a task level generic human roles are task initiator, task stakeholder, potential owner, actual owner, excluded owner, business administrator and notification recipients. People assignments allows a business analyst to assign a certain role for a human task. When defining these kinds of services we consider who should be permitted to play a certain role on the task and how this service is performed. During assignment of these tasks a modeler can specify access controls and define policies for a specific activity, these policies are inherited as WS-Policy 1.5.

XML Process Definition Language - XPDL XML Process Definition Language (XPDL) is standardized by the Workflow Management Coalition (WfMC) since 2002. Formerly working on WPDL (Workflow Process Definition Language), WfMC soon changed their language name to XPDL when it was decided to use XML for its syntax. Current version is XPDL 2.1 .

XPDL is designed to answer the Process Definition Interchange question, which is, according to WfMC, one of the key features a workflow management system must have. Since 2.0, XPDL is even explicitly designed to allow store and exchange of BPMN diagrams. To quote WfMC: XPDL is the Serialization Format for BPMN. This means there is a one to one matching between a BPMN diagram and its XPDL translation. That includes the graphical part of BPMN: the coordinates of the different elements, of the lines linking those elements, all this is included in XPDL.

5.1.2 Quality of Service Layer

This layer defines extra specifications for services in order to ensure non-functional properties like security for instance.

WS-Security WS-Security defines a SOAP Security Header format containing security related information. A SOAP message may include multiple security headers. Each header is targeted at a specific SOAP actor/role that may be either the ultimate recipient of the message or an intermediary. Security headers may encapsulate one or many elements of the following types:

- Security tokens
- Signatures
- Encryption elements
- Timestamps

By providing a common syntax and a flexible processing model for security headers, this specification accommodates a large variety of security models and encryption technologies. Moreover, incorporating security features in the application level ensures end-to-end security.

WS-Trust This specification provides a framework built on WS-Security for managing security tokens. In the WS-Trust trust model, a requester examines the policy associated with a Web Service to identify the claims it needs. If the policy statements require security tokens that the requester does not possess, WSTrust specifies a way of obtaining them: contacting a Web Service referred to as Security Token Server (STS). A STS may also be used to renew, cancel and validate security tokens.

WS-Trust defines abstract formats of the messages used to manage security tokens. To each usage pattern corresponds a specific binding providing concrete semantics to the general security token requests and responses. For complex scenarios, WS-Trust describes flexible mechanisms for trust establishment. In fact, different STS may get involved to broker, exchange or delegate security tokens issuance. A general model for negotiation/challenge extensions is specified to support multi-messages exchanges for security tokens management.

The flexibility and extensibility of the specification allows interfacing with a large number of security models, including legacy protocols. In fact, increasing interoperability between trust domains is one of the purposes of this standard.

WS-Policy The WS-Policy is a policy expression language for describing the capabilities and requirements of a Web Service, i.e. representing whether and how a message must be secured, whether and how a message must be delivered reliably or whether the request must follow a transaction flow. Such requirements are translated into machine-readable policy expressions that are usually provided by the web service developer for the client component to automatically apply the requirements.

Basically, WS-Policy is a simple language that defines four elements (Policy, All, ExactlyOne, PolicyReferences) and two attributes (Optional, Ignorable) that suffice to express generic policy expression by combining individual assertions. The policy assertions syntax are outside the scope of WS-Policy specifications. Thus, WS-Policy can be viewed as a meta policy composition language that can express any kind of requirements as long as the policy-aware clients (Web Services endpoints and relays) are capable of understanding the specific syntax of the unitary assertions. An individual policy assertion expresses one requirement, behavior or capability related to messaging (how the message must be built), security (how the message must be secured through authentication or encryption), reliability (how to ensure that the message has been sent/received) and transaction (what transaction flow must be followed to ensure transaction commit).

XACML XACML (eXtensible Access Control Markup Language) is a declarative XML-based access control policy language used to describe the access control restrictions to actions on objects. Usually, access control models involve a subject (that is, either a user, a user on behalf of another user, a service, or a service on behalf of a user) making some access request and the system either authorizes this access request or denies it. XACML also defines a processing model, which describes how to operationally interpret the policies.

XACML defines both an access control policy language (to express the access control

conditions) and a canonical XML language to communicate with a Policy Decision Point (PDP), to send to the PDP decision requests and obtain decision responses. This canonical form or language is called the XACML context. The current version of XACML, Version 2.0, was released by OASIS in February 2005. Version 3.0 is in preparation at the time of preparation of this document (June 2008). It is chartered to add generic attribute categories for the evaluation context and policy delegation profile (administrative policy profile). There are already some prototypical implementations of XACMLv3.0.

WS-Federation The WS-Federation specification [48] defines mechanisms to support federation between different security realms, *i.e.,*the authorized access for principals of one realm to resources managed by another. The WS-Federation framework builds on the specification for WS-Security and WS-Trust. Specifically, WS-Federation relies on the Security Token Service (STS) model defined by WS-Trust, and a protocol (involving Request Security Token and RST Response messages) for handling such tokens, which contain information described by WS-SecurityPolicy [82]. The STS is used to broker an establishment of a trust relationship between resource providers / relying parties and other service providers. The goal is to simplify the development of federated services by reusing the WS-Trust STS model and protocol. Different federation services can be developed as variations of the base STS. Processing in WS-Federation is kept independent of the security token format and the type of token being transmitted. WS-Federation defines a metadata model and a document format describing how services can be discovered and combined, as well as their access policies.

SAML The Security Assertion Markup Language (SAML) is an XML standard for assertions regarding identity, attributes and entitlements of a subject [49]. It allows to exchange authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The primary example for the use of SAML is is the Web Browser Single Sign-On (SSO) problem. The service provider relies on a SAML assertion from the identity provider about the principal to make an access control decision. This setup requires the existence of local authorization services from an identity provider, however, SAML provides a level of abstraction, since it does not specify how they are implemented. SAML is a dialect of XML, uses the XML signature and encryption facilities, and relies on HTTP, specifying the use of SOAP.

Generally, a SAML assertion is a statement made at a given time by an issuer regarding a subject provided that certain conditions hold. SAML assertions can contain three types of statements: authentication, attribute and authorization decision statements. Each of these corresponds to a type of query which forms part of a SAML request-response protocol. The structure of an assertion is described by a SAML profile, which may be defined dependent on the desired application. At the implementation level, SAML messages admit bindings to several standard message types and protocols [95]. SAML provides XML formats for transmitting security information, defines how they work with underlying protocols, and specifies message exchanges for common use cases. In addition, it supports several privacy

protection mechanisms (providing means to determine security attributes without revealing identity), and specifies a schema that allows systems to communicate the SAML options they support. SAML is linked to the WS family of standards: SAML assertions are one supported security format for WS-Trust.

WS-Reliability WS-Reliability is a SOAP-based specification for reliable messaging requirements [110]. Subsequently to its standardization in 2004 (v. 1.1), the WSReliable Messaging specification was developed by the same OASIS Technical Committee. WS-Reliability separates reliable messaging issues into a protocol (wire) aspect which deals with the horizontal contract between sender and receiver (e.g., message headers, choreography) and a quality of service aspect which deals with the vertical contract between service provider and service users. The latter defines a set of abstract operations on messages (such as Deliver, Submit, Respond and Notify). The specification assumes transparency of SOAP intermediaries and support for message integrity (e.g., as in WS-Security).

5.1.3 Discovery, Registry, and Publishing Layer

A typical service-oriented architecture includes partners acting as service requesters, service providers and service brokers. Initially, the provider publishes a description of its services to the broker. When the requester wants to use a service, it needs first to discover its location with the broker and then to bind to the provider. Two specifications deal with discovery and publishing.

Universal Description Discovery and Integration - UDDI The current version of UDDI, version 3.0.2, was released by OASIS in February 2005 [131, 46]. It provides the infrastructure required to publish and discover services in a systematic way. UDDI specifies an XML-based registry wherein service providers can register their available Web Services and whose content can be browsed by clients. A common usage is the description of every particular service in WSDL and their registering in a UDDI registry. The UDDI data model is an XML schema that describes services, using the following structures:

- businessEntity: represents the provider of Web Services. It contains information about the company, including contact information, industry categories, business identifiers and a list of services provided.
- businessService: represents an individual Web Service provided by the business entity. Its description includes information to bind to the Web Service, its type and taxonomic categories.
- bindingTemplate: represents the technical implementations of the Web Service represented by the business service structure.
- tModel: represents metadata used for more detailed information about a service.
- publisherAssertion: represents the association between some businessEntity structures according to a specific type of relationship, such as subsidiary or department.
- subscription: is used to subscribe to events about changes of a list of entities.

As a registry is useless without some way to access it, UDDI specifies two interfaces for service consumers and service providers to interact with the registry. Service consumers use the Inquiry API to find a service, and service providers use the Publisher API to register a service.

WS-Discovery The WS-Discovery standard [101] provides a mechanism for finding the specific addresses of web-services at run-time. It relies on a multicast protocol to will respond. When a web-service provider joins the network, it sends an announcement message to the multicast group in order to minimize the need for polling.

Various security threats have been identified for this service [101], including message alteration, replay, and denial of service attacks. To prevent these, the standard recommends, but does not enforce, using standard cryptographic techniques such as digital signature schemes and employing timeouts.

5.1.4 Description Layer

Services needs to be described in order to be used.

Web Service Description Language - WSDL The Web Service Description Language (WSDL) [50] is responsible for describing Web Services, especially the interface a Web Service exposes to other applications. Means for expressing service interfaces are at the core of all service models, and WSDL provides very flexible, highly-extensible, and well designed methods for doing this. As most other Web Service standards, WSDL is based on XML. WSDL documents contain all information required for the use of a Web Service, including data types, message patterns, method descriptions, and service location. As a consequence, programming frameworks that are based on Web Services - such as Windows Communication Foundation (WCF) [45] - provide tools that consume a WSDL document and dynamically create the proxy code necessary for the use of a Web Service. In WSDL, Web Services are expressed as collections of endpoints that exchange messages. WSDL also contains information of how these messages are mapped to a concrete network protocol - a so-called binding - so that these messages can be exchanged in an interoperable fashion.

5.1.5 Messaging and Transport Layer

This layer is not integrated in our conceptual service model, since it is essentially at the infrastructure level.

TCP, UDP / HTTP, SMTP Transport protocols are required to facilitate message delivery. The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) belong to the core protocol stack of the Internet. TCP, as a session-based, reliable and in-order delivery transport protocol, is suitable for applications like file transfer, e-mail, exchanges between Web servers and clients. UDP is a fast and efficient protocol that is stateless and unreliable; datagrams (short messages) may arrive out of order, duplicated or missed without notice.

Unlike TCP, UDP is compatible with broadcasting (sending to all on local network) and multicasting (sending to all subscribers).

The most common communication protocols are the Hypertext Transfer Protocol (HTTP) (or its secured variant HTTPS) and the Simple Mail Transfer Protocol (SMTP), thus by sending XML requests, and getting XML responses over the transport protocol. HTTP is a request/response standard between a client (an end-user) and a server (a Web site). HTTP is not constrained to using TCP/IP and its supporting layers, it only presumes a reliable transport. SMTP is a relatively, text-based protocol. It is the de facto standard for e-mail transmissions across the Internet.

XML-RPC XML-RPC [135] is a Remote Procedure Call protocol that uses XML to encode the messages and HTTP to handle them. A network node (the client) sends a request message to another node (the server), which sends a response message to the client. XML-RPC is a precursor to SOAP. It is sometimes preferred to SOAP because of its simplicity, minimalism, and ease of use.

Simple Object Access Protocol - SOAP The Simple Object Access Protocol (SOAP) [51] defines a Remote Procedure Call using an XML messaging protocol for basic service interoperability. SOAP once stood for Simple Object Access Protocol, but this acronym was dropped with the version 1.2 of the standard, as it is not simple anymore and it is not only used to access objects. It is a messaging framework for transferring information between an initial SOAP sender, optionally some intermediate receivers and an ultimate SOAP receiver.

This protocol is non-proprietary (it became a W3C Recommendation in 2003) and platform and language independent. It can be run over a simple transport protocol (e.g., HTTP or SMTP). There are examples of the usage of SOAP services over the transport protocols in [127]. For instance, one can also explore sending and receiving service-oriented requests over the Simple Mail Transfer Protocol (SMTP). In fact the nature of the service-oriented architecture enables one to expose services over any protocol, even beyond those described in official bindings such as TCP, named pipes, UDP and custom transport protocols.

5.2 Alternative: Restful web services

As already mentioned, there is an alternative solution for web services: RESTful web services. In this model, which is resource-oriented, any information that can be named is abstracted as a *resource*. and resources are manipulated using a fixed set of four CRUD (create, read, update, delete) operations. In a context analogous to databases, CRUD languages for RESTful web services have been developed as variants of the SQL language: see for instance the language `YQL` from Yahoo. But, contrary to the process-oriented model, there is no standard like BPEL, which has led to the current diversity of CRUD languages in use.

The Representational State Transfer (REST) [71] is a network architecture paradigm relying on standard transport protocols like HTTP, without the use of an additional messaging layer. A

service call is handled via its URI. The term REST was coined by Roy Fielding in his PhD dissertation [71]. "REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems". Fielding describes the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, contrasting them to the constraints of other architectural styles. HTTP provides the standard CRUD operations (GET, POST, PUT, DELETE) as procedure calls. Each method has clear defined semantics that can be relied upon. Security is handled as for a standard Web application (using SSL, sessions, cookies, etc). REST is a client-server, stateless, cacheable and layered network paradigm. The World Wide Web is the key example of a REST design.

5.3 Cloud computing

Cloud computing provides three kinds of services that we detail now.

5.3.1 Software as a Service (SaaS)

Many platforms nowadays proposes Software on-demand, also known as Software as a Service. SaaS vendors provide applications to end users on demand. Instead of purchasing software for on-premise use, customers license use of applications as a service over the web, and may pay based on number of users accessing applications or by utilization.

See section 5.4.2 for more details about ByDesign.

5.3.2 Platform as a Service (PaaS)

PaaS provides an additional level of abstraction, thus emulating a virtual platform on top of the infrastructure. Service Providers (like VMware) might be caring for databases or servers and provide a consistent service-oriented abstraction for programmers to access to these, without the developer needing to care for their maintenance or management. Mashups are providing a consistent view over publications originating from different service providers acting as publishers. In a completely different fashion, systems like Hadoop are for instance combining any number of nodes into a single virtual computer abstraction. Not all PaaS systems are actually service oriented, but there's an increasing tendency to provide access to such functions through a service-oriented API though. It should also be understood that such services are also providing a form of coordination over IaaS services. PaaS generally features a form of intermediation to underlying services akin to middleware in traditional communication stacks.

5.3.3 Infrastructure as a Service (IaaS)

The "Infrastructure as a Service" approach touted by cloud computing aims at sharing the infrastructure in order to reduce the cost of operating it. Services in that case relate to the management

and customization of infrastructure mechanisms. Virtualized execution environments and distributed data storage are common examples of such services. They are being supported by an increasing range of companies (Amazon, Google, Canonical, etc.) and brought to the programmer through an increasing number of service oriented APIs, notably REST ones, like for instance Amazon EC2 for execution environments, or Amazon S3 for distributed data storage. Although IaaS is often likened to cloud computing by many, one can observe that embedded system infrastructures (virtual machines, communication frameworks) are increasingly service-oriented. The main drawback of today's IaaS APIs is that they exhibit only an informal contract with the infrastructure, which can only be made more explicit through examples since the infrastructure's internal organization and behavior are not exposed. Service Level Agreements (SLAs) may be complementing these APIs and describe for instance the expected quality of service.

Vertical composition is perfectly outlined by these three types of services and how they suggest to perform compositions . In particular, while SaaS offers a complete application as a service (which might be reused in more complex composite services), and PaaS provides new primitives for developing applications (that is a set of abstractions devoted to distribution, fault tolerance, transactionality, etc. as building blocks), IaaS is transparent or even orthogonal to the application in itself. Customizable virtual execution environments of the infrastructure might however interact with applications by modifying the semantics of software that runs on top of them, and for instance provide additional transactional, fault-tolerance, or security properties . Furthermore, security constraints are likely to arise at the infrastructure level due to the deployment of applications and in particular the geolocation of a particular execution environment or of the application data. These different points of view on services also have a certain mapping to the concepts identified in the SOA model defined previously: SaaS corresponds to collaboration and processes, PaaS to primitive services and resources, and IaaS to resources).

5.4 Infrastructures and standards of the CESSA industrial partners

5.4.1 Challenges for service-based business applications

SAP has proposed the notion of enterprise service-oriented architecture [123] as a business driven approach to SOA that expands the concept of Web services into an architecture that supports an enterprise-wide, service-enabled business architecture (see Fig 5.2).

In a nutshell, enterprise services are highly-integrated web services combined with business logic and semantics that can be accessed and (re)used to support a particular business process. The main characteristic that differentiates enterprise services from regular Web Services is their integration with the underlying business semantics: enterprise services are structured according to a harmonized enterprise model based on business objects, process components, and global data types (GDTs).

The dynamic evolution of such integrated enterprise SOAs, in particular under the constraint to enforce unanticipated new security requirements, has however not yet been investigated.

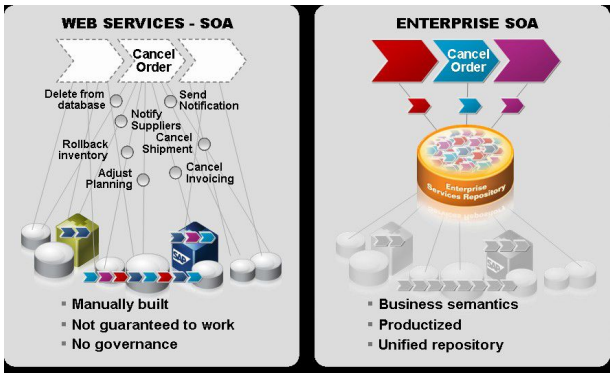


Figure 5.2: The SAP Vision of Enterprise SOA

5.4.2 Business ByDesign, Netweaver

5.4.2.1 SAP Business ByDesign

SAP Business ByDesign [2] is delivered on-demand as a software-as-a-service (SaaS), to address small and medium enterprises market with a rapid development and adoption of solutions. The goal is to focus on business needs rather than in software development for most of concepts that are addressed everyday.

Midsize companies with big plans to grow understand that in addition to great products and services, they need business processes that contribute to their overall operational excellence. For starters, they need full-function business applications to advance visibility and control over key areas of their business. On-demand applications - also known as ”software as a service” - offer midsize companies these advantages along with reduced operational complexity, reliable security, privacy protection, and high availability.

The SAP Business ByDesign solution is an integrated, adaptable, on-demand business solution that is managed, monitored, and maintained by SAP experts. On-demand access to the solution and business data for SAP Business ByDesign requires only a standard Web browser, thereby simplifying IT requirements.

SAP Business ByDesign provides the flexibility (see Fig 5.3) to change, grow, and extend the reach of your business across your entire ecosystem. Employees can reconfigure SAP Business ByDesign on the fly - without disrupting critical business processes - so you can respond immediately to new opportunities, streamline your operations, and adjust to market requirements. Because SAP Business ByDesign is engineered to accommodate the individual roles of your employees, it reflects organizational changes as soon as you notify the software. Flexible and easy-to-use reports and dashboards help you manage your business in real time, while built-in support for regulatory compliance makes it straightforward to address changing legal and reporting requirements.

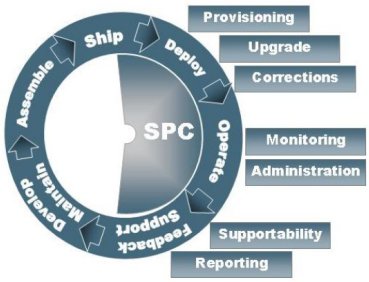


Figure 5.3: SAP Business ByDesign Life cycle Management

5.4.2.2 SAP NetWeaver

The SAP NetWeaver technology platform enables the composition, provisioning, and management of SAP and non-SAP applications across a heterogeneous software environment. That let standardize business processes across technological boundaries, integrate applications for employees as needed, and access and edit simple information easily and in a structured manner. SAP Netweaver provides features depending on the underlying application server used. SAP support two application servers. The first above an ABAP stack and the other one in a JAVA stack. Most of features are provided for both technologies, but it may have few differences. Netweaver covers a large spectre of technology. There is seven topics covered by this solution , namely :

- **User Productivity** : IT organizations can help users and groups improve their productivity through enhanced collaboration, optimized knowledge management and intuitive search in business objects as well as unstructured content. Personalized access to mission-critical applications and data is accomplished using portals, desktop clients and mobile interfaces. Flexible UI technology enables IT organizations and partners to build their own state-of-the-art applications.

- **Business Intelligence** : BI solutions provide comprehensive business intelligence functionality that can empower users to make effective, informed decisions based on solid data and analysis. All users, from the high-end analyst to the casual business user, have access to the information they need. With these solutions, users throughout enterprise can access, format, analyze, navigate, and share information across the organization.

- **Business Process Composition** : It enables customers to provide design, model, implement, run, monitor, operate and improve business processes and focuses on model-driven development of composite applications as an alternative to the traditional code-based approach

- **Enterprise Information Management** : It's the business activity of creating, cleansing, integrating, managing, governing, and archiving structured and unstructured data used by an organization. By managing information assets effectively, organizations can minimize

data integration efforts, streamline end-to-end business process execution, and gain well-founded business insight.

- **SOA Middleware** : It enables IT organizations to use standards-based Web services to quickly form new, innovative business solutions that meet their changing business needs. In particular, SAP NetWeaver provides service-oriented architecture (SOA) middleware that facilitates communication between disparate applications. From a logical view, SOA middleware consists of an enterprise services repository and registry, an enterprise services bus (ESB), and SOA management tools.

- **Custom Development** : IT professionals can extend or enhance existing applications and create custom applications using either the *ABAP* or *Java* programming language.

- **Security and Identity Management** : Organizations face a difficult challenge in today's security-conscious world having to support transparent enterprise boundaries, location-independent users, and the growing demands for regulatory compliance. IT organizations can enable safeguards that protect their business, while helping users - and the business processes they rely on - proceed unhindered by security operations. We develop this section (see section 5.4.2.3) as it is of interest for CESSA.

- **Application Life-cycle Management** : It provides processes, tools, services, and an organizational model to manage SAP and non-SAP solutions throughout the complete application life cycle. Instead of just focusing on the individual phases, SAP provides a holistic approach

5.4.2.3 SAP NetWeaver Security

In today's world of collaborative business processes and open system environments, security no longer means just adding a firewall and using passwords to log on. It requires a complete approach that not only applies to the IT landscape, but also to issues that arise beyond borders, in which even simple organizational measures can have a significant impact. The infrastructure of the SAP NetWeaver technology platform supports the customer by delivering comprehensive security features for heterogeneous environments [3].

User Authentication and Single Sign-On There are several mechanisms available for authenticating users on the SAP NetWeaver platform. In addition, if you have many systems in your system landscape, then a Single Sign-On environment is also desirable to reduce the number of passwords that users have to remember. For example, SAP Netweaver provide the user and password mechanism for user authentication and single sign on, and also SSL and X.509 client certificates. It also use Security Assertion Markup Language (SAML) but only for Single Sign On.

Identity Management SAP systems within the SAP NetWeaver technology platform perform authorizations using a role-based identity management approach. This means that you assign authorizations to users based on the job they perform using the particular system.

It uses the User Management Engine (UME) for user management and the integration of user accounts on SAP systems with user accounts maintained on a directory server.

Network and Transport Layer Security You can provide for security at the transport layer for securing connections between SAP NetWeaver system components. When using transport layer security, the data transfer not only protected against eavesdropping by using encryption, the communication partners can be authenticated as well. Protection is provided in two forms, depending on the type of communication that is being used. For connections that use Internet protocols such as HTTP, you can use the quasi-standard Secure Sockets Layer (SSL) protocol. For SAP protocols such as RFC or dialog, you can use Secure Network Communications (SNC).

WS Security SAP Netweaver provides WS-Security standards to build secure services both to consume and expose them. Depending on which standards you combine, you can reach different kind of security (e.g single sign on, end-to-end security, etc.).

System Security SAP Netweaver is also cover topics that apply to system security, for example: key, key pair and trust management, logging of security-related events using the Security Audit Log, and virus detection using the virus scan interface.

Digital Signatures and Encryption SAP NetWeaver has *Secure Store and Forward (SSF)* mechanisms which provide the means to secure data and documents in SAP Systems as independent data units. By using SSF functions, it's possible to "wrap" data and digital documents in secure formats before they are saved on data carriers or transmitted over insecure communication links. If you save the data in a secure format in the SAP System, it remains in its secured format even if you export it out of the system. SSF mechanisms use digital signatures and digital envelopes to secure digital documents. The digital signature uniquely identifies the signer, is not forgeable, and protects the integrity of the data. Any changes in the data after being signed result in an invalid digital signature for the altered data. The digital envelope makes sure that the contents of data are only visible to the intended recipient(s).

Chapter 6

Conclusion

In this deliverable we have laid part of the foundations for the CESSA project. This document has provided motivation and basic requirements for the CESSA service and aspect models, as well as the applications targeted within the project.

First, we have introduced the main concepts of service-oriented computing and aspect-oriented computing, and the associated terminology.

Second, we have presented a loan negotiation scenario typical for the acquisition of loans in a business environment that involves customers, banks and credit bureaus. This scenario will be used over the full duration of the CESSA project. We have presented sets of requirements for the service and aspect models derived by partners SAP and IS2T using the use case scenario. To prepare the future developments, we have also introduced a method that will be used to formalize requirements.

Third, we have provided an extensive analysis of the state-of-the-art of services, composition techniques, especially aspect-oriented software development. This analysis encompasses both, academic approaches and industrial systems, including existing industrial standards.

This document provides a basis for the analysis and definition of the security properties that will be studied as part of the CESSA project and presented in deliverable D2.1. Finally, it paves the way for the definition of the CESSA service and aspect models, whose (full) definitions will be provided in deliverable D1.2.

Bibliography

[1] Code of conduct - the office of the credit information ombud. http://www.creditombud.org.za/code.htm.

[2] Sap bydesign. https://www.sme.sap.com/irj/sme/.

[3] Sap security. http://www.sdn.sap.com/irj/sdn/security.

[4] A. Agrawal et al. Web Services Human Task (WS-HumanTask), Version 1.0. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf`, June 2007.

[5] A. Agrawal et al. WS-BPEL Extension for People (BPEL4PEOPLE), Version 1.0. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf`, June 2007.

[6] Mehmet Akşit, Siobhán Clarke, Tzilla Elrad, and Robert E. Filman, editors. *Aspect-Oriented Software Development*. Addison-Wesley Professional, September 2004.

[7] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, et al. Adding trace matching with free variables to AspectJ. In Richard P. Gabriel, editor, *ACM Conference on Object-Oriented Programming, Systems and Languages (OOPSLA)*. ACM Press, 2005.

[8] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie J. Hendren, Sascha Kuzins, Ondrej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to aspectj. In Ralph E. Johnson and Richard P. Gabriel, editors, *OOPSLA*, pages 345–364. ACM, 2005.

[9] Robert Allen, Rémi Douence, and David Garlan. Specifying and Analyzing Dynamic Software Architectures. In *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*, volume 1382 of *Lecture Notes in Computer Science*, pages 21–37. Springer-Verlag, 1998.

[10] R. Alur and D. Dill. The theory of timed automata. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 45–73, Berlin, Germany, June 1992. Springer.

[11] Pascal André, Gilles Ardourel, and Christian Attiogbé. Defining Component Protocols with Service Composition: Illustration with the Kmelia Model. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[12] André Arnold. *Finite Transition Systems*. International Series in Computer Science. Prentice-Hall, 1994.

[13] André Arnold, G. Point, Alain Griffault, and Antoine Rauzy. The AltaRica Formalism for Describing Concurrent Systems. *FUNDINF: Fundamenta Informatica*, 34:109–124, 2000.

[14] AspectJ home page. `http://www.eclipse.org/aspectj`.

[15] Christian Attiogbé, Pascal André, and Gilles Ardourel. Checking component composability. In *Proceedings of the 5th International Workshop on Software Composition (SC'06)*, volume 4089 of *Lecture Notes in Computer Science*, pages 18–33. Springer-Verlag, 2006.

[16] S. Balsamo, M. Bernardo, and M. Simeoni. Combining Stochastic Process Algebras and Queueing Networks for Software Architecture Analysis. In *Proc. of the Int. Workshop on Software and Performance (WOSP'2002)*, 2002.

[17] A. Barros, M. Dumas, and P Oaks. A Critical Overview of the Web Service Choreography Description Language. *BPTrends Newsletter*, 3, 2005.

[18] Tomás Barros, Rabéa Boulifa, and Eric Madelaine. Parameterized models for distributed java objects. In David de Frutos-Escrig and Manuel Núñez, editors, *FORTE*, volume 3235 of *Lecture Notes in Computer Science*, pages 43–60. Springer, 2004.

[19] Tomás Barros, Luc Henrio, and Eric Madelaine. Behavioural Models for Hierarchical Components. In *Proc. of SPIN'05*, volume 3639 of *LNCS*, pages 154–168. Springer-Verlag, 2005.

[20] Gilles Barthe and César Kunz. Certificate translation for specification-preserving advices. In Curtis Clifton, editor, *FOAL*, pages 9–18. ACM, 2008.

[21] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. Types and effects for secure service orchestration. In *Computer Security Foundations Workshop*, pages 57–69. IEEE Computer Society, 2006.

[22] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3–12, 2006.

[23] Hubert Baumeister, Florian Hacklinger, Rolf Hennicker, Alexander Knapp, and Martin Wirsing. A Component Model for Architectural Programming. In *Proc. 2nd Int. Wsh. Formal Aspects of Component Software (FACS'05)*, volume 160 of *ENTCS*, pages 75–96, 2005.

[24] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Detecting Architectural Mismatches in Process Algebraic Description of Software Systems. In *Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA'2001)*, pages 77–86, 2001.

[25] Elisa Bertino, Lorenzo Martino, Federica Paci, and Anna Squicciarini. *Security for Web Services and Service-Oriented Architectures*. Springer Verlag, 2009.

[26] Tommaso Bolognesi and Ed. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–29, 1988.

[27] Michele Boreale, Roberto Bruni, Luís Caires, Rocco De Nicola, Ivan Lanese, Michele Loreti, Francisco Martins, Ugo Montanari, António Ravara, Davide Sangiorgi, Vasco Thudichum Vasconcelos, and Gianluigi Zavattaro. SCC: A service centered calculus. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *Web Services and Formal Methods, Third International Workshop, WS-FM, Proceedings*, volume 4184 of *Lecture Notes in Computer Science*, pages 38–57. Springer, 2006.

[28] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74(1), 2005.

[29] Mathieu Braem, Kris Verlaenen, Niels Joncheere, Wim Vanderperren, Ragnhild Van Der Straeten, Eddy Truyen, Wouter Joosen, and Viviane Jonckers. Isolating process-level concerns using Padus. In *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria, September 2006. Springer-Verlag.

[30] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983.

[31] Lubos Brim, Ivana Cerná, Pavlína Vareková, and Barbora Zimmerova. Component-Interaction Automata as a Verification-Oriented Component-Based System Specification. In *In Proceedings of SAVCBS 2005*, pages 31–38. Department of Computer Science, Iowa State University, 2005.

[32] Glenn Bruns, Radha Jagadeesan, Alan Jeffrey, and James Riely. microabc: A minimal aspect calculus. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2004.

[33] A. Bucchiarone and S. Gnesi. A survey on services composition languages and models. In *International Workshop on Web Services Modeling and Testing (WSMaTe '06) Proceedings*, pages 37–49, 2006.

[34] Tomas Bures, Petr Hnetynka, and Frantisek Plasil. SOFA 2.0: Balancing advanced features in a hierarchical component model. In *SERA*, pages 40–48. IEEE Computer Society, 2006.

[35] Tomas Bures, Petr Hnetynka, and Frantisek Plasil. Sofa 2.0: Balancing advanced features in a hierarchical component model. In *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 40–48. IEEE CS, 2006.

[36] Stephan Buse, Mobile Commerce, In Collaboration, Rajnish Tiwari, and Stephan Buse. The mobile commerce prospects: A strategic analysis of opportunities in the banking sector.

[37] Luís Caires. Spatial-behavioral types, distributed services, and resources. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *Trustworthy Global Computing*, volume 4661 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2006.

[38] Muffy Calder, Savi Maharaj, and Carron Shankland. A Modal Logic for Full LOTOS Based on Symbolic Transition Systems. *The Computer Journal*, 45(1):55–61, 2002.

[39] L. Cardelli and A. Gordon. Mobile ambients. In M. Nivat, editor, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS), European Joint Conferences on Theory and Practice of Software (ETAPS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, Lisbon, Portugal, 1998. Springer-Verlag, Berlin.

[40] Ivana Cerná, Pavlína Vareková, and Barbora Zimmerova. Component substitutability via equivalencies of component-interaction automata. *Electr. Notes Theor. Comput. Sci*, 182:39–55, 2007.

[41] Girish B. Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 134–143, New York, NY, USA, 2004. ACM Press.

[42] Anis Charfi and Mira Mezini. Aspect-oriented web service composition with AO4BPEL. In Liang-Jie Zhang, editor, *Web Services, European Conference, ECOWS 2004, Erfurt, Germany, September 27-30, 2004, Proceedings*, volume 3250 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2004.

[43] Curtis Clifton and Gary T. Leavens. Observers and assistants: A proposal for modular aspect-oriented reasoning. Technical Report 02-04a, Iowa State University, Department of Computer Science, April 2002.

[44] Curtis Clifton and Gary T. Leavens. Minimao: An imperative core language for studying aspect-oriented reasoning. *Sci. Comput. Program.*, 63(3):321–374, 2006.

[45] Windows Communication Foundation. WCF. `http://msdn2.microsoft.com/en-us/netframework/aa663324.aspx`.

[46] Oasis Consortium. Universal Description, Discovery, and Integration specification. `http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf`.

[47] OASIS Consortium. Oasis: Reference model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, 2006.

[48] OASIS Consortium. Web Services Federation Language (WS-Federation) V1.1. http://specs.xmlsoap.org/ws/2006/12/federation/ws-federation.pdf, December 2006.

[49] OASIS Consortium. Security Assertion Markup Language V2.0 Technical Overview. http://wiki.oasis-open.org/security/Saml2TechOverview, March 2008.

[50] The World Wide Web Consortium. Web Services Description Language (WSDL) version 2.0. http://www.w3.org/TR/2006/WD-wsdl20-rdf-20060327/, 2006.

[51] The World Wide Web Consortium. Simple Object Access Protocol 1.2. http://www.w3.org/TR/soap12-part1, Apr 2007.

[52] Thomas Cottenier and Tzilla Elrad. Dynamic and decentralized service composition: With contextual aspect-sensitive services. In José Cordeiro, Vitor Pedrosa, Bruno Encarnação, and Joaquim Filipe, editors, *WEBIST 2005, Proceedings of the First International Conference on Web Information Systems and Technologies, Miami, USA, May 26-28, 2005*, pages 56–63. INSTICC Press, 2005.

[53] Bruno C. d. S. Oliveira, Tom Schrijvers, and William R. Cook. Effectiveadvice: disciplined advice with explicit effects. In Jean-Marc Jézéquel and Mario Südholt, editors, *AOSD*, pages 109–120. ACM, 2010.

[54] Daniel S. Dantas and David Walker. Harmless advice. In J. Gregory Morrisett and Simon L. Peyton Jones, editors, *POPL*, pages 383–396. ACM, 2006.

[55] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proc. of ESEC/FSE'01*, pages 109–120. ACM Press, 2001.

[56] Simplice Djoko Djoko, Rémi Douence, and Pascal Fradet. Aspects preserving properties. In Robert Glück and Oege de Moor, editors, *PEPM*, pages 135–145. ACM, 2008.

[57] Simplice Djoko Djoko, Rémi Douence, and Pascal Fradet. Specialized aspect languages preserving classes of properties. In Antonio Cerone and Stefan Gruner, editors, *SEFM*, pages 227–236. IEEE Computer Society, 2008.

[58] Rémi Douence, Didier Le Botlan, Jacques Noyé, and Mario Südholt. Concurrent aspects. In Stan Jarzabek, Douglas C. Schmidt, and Todd L. Veldhuizen, editors, *GPCE*, pages 79–88. ACM, 2006.

[59] Rémi Douence, Pascal Fradet, and Mario Südholt. A framework for the detection and resolution of aspect interactions. In Don S. Batory, Charles Consel, and Walid Taha, editors, *GPCE*, volume 2487 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2002.

[60] Rémi Douence, Pascal Fradet, and Mario Südholt. Composition, reuse and interaction analysis of stateful aspects. In Gail C. Murphy and Karl J. Lieberherr, editors, *AOSD*, pages 141–150. ACM, 2004.

[61] Rémi Douence, Pascal Fradet, and Mario Südholt. Composition, reuse and interaction analysis of stateful aspects. In *Proc. of 3rd International Conference on Aspect-Oriented Software Development (AOSD'04)*, pages 141–150. ACM Press, March 2004.

[62] Rémi Douence, Olivier Motelet, and Mario Südholt. A formal definition of crosscuts. In Akinori Yonezawa and Satoshi Matsuoka, editors, *Reflection*, volume 2192 of *Lecture Notes in Computer Science*, pages 170–186. Springer, 2001.

[63] Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, July 1998.

[64] Catherine Dufourd, Petr Jančar, and Philippe Schnoebelen. Boundedness of reset P/T nets. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer.

[65] Schahram Dustdar and Mike P. Papazoglou. Services and service composition - an introduction (services und service komposition - eine einführung). *it - Information Technology*, 50(2):86–92, 2008.

[66] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. of Web and Grid Services*, 1:1–30, August 03 2005.

[67] E. Allen Emerson. *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 997–1072. Elsevier, 1990. J. Van Leeuwen, Editor.

[68] Andrés Farías and Mario Südholt. Integrating protocol aspects with software components to address dependability concerns. Technical Report 04/6/INFO, École des Mines de Nantes, November 2004.

[69] Fabrício Fernandes, Robin Passama, and Jean-Claude Royer. Event strictness for components with complex bindings. In Kiran Deshpande, Pankaj Jalote, and Sriram K. Rajamani, editors, *ISEC'09: Proceedings of the 2nd conference on India Software Engineering Conference*, pages 47–56. ACM, February 2009.

[70] Fabricio Fernandes and Jean-Claude Royer. The stslib project: Towards a formal component model based on sts. In Markus Lumpe and Eric Madelaine, editors, *Proceedings of the 4th International Workshop on Formal Aspects of Component Software (FACS 2007)*, volume 215, pages 131–149, June 2008.

[71] Roy T. Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000.

[72] Alain Finkel, Pierre McKenzie, and Claudine Picaronny. A well-structured framework for analysing Petri nets extensions. *Information and Computation*, 195(1-2):1–29, November 2004.

[73] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *ASE*, pages 152–163, 2003.

[74] Pascal Fradet and Stéphane Hong Tuan Ha. Aspects of availability: Enforcing timed properties to prevent denial of service. *Sci. Comput. Program.*, 75(7):516–542, 2010.

[75] Max Goldman and Shmuel Katz. Maven: Modular aspect verification. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2007.

[76] Gregor Gößler, Susanne Graf, Mila Majster-Cederbaum, Moritz Martens, and Joseph Sifakis. An approach to modeling and verification of component based systems. volume 4362, pages 61–70, 2007.

[77] Susanne Graf and Sophie Quinton. Contracts for bip: Hierarchical interaction models for compositional verification. In *FORTE*, pages 1–18, 2007.

[78] Florian Hacklinger. Java/A - taking components into java. In *IASSE*, pages 163–168. ISCA, 2004.

[79] Thomas A. Henzinger and Rupak Majumdar. A classification of symbolic transition systems. *Lecture Notes in Computer Science*, 1770:13–34, 2000.

[80] C.A.R. Hoare. *Communicating Sequential Processes*. C.A.R Hoare Series. Prentice-Hall International, 1985.

[81] Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tevfik Bultan, and Richard A. Kemmerer. Counter machines and verification problems. *Theoretical Computer Science*, 289(1):165–189, October 2002.

[82] IBM and Microsoft Corporation. Understanding WS-Federation. `http://msdn.microsoft.com/en-us/library/bb498017.aspx`, May 2007.

[83] ISO/IEC. LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, International Organization for Standardization, 1989.

[84] JBoss. `http://www.jboss.org`.

[85] Pavel Jezek, Jan Kofron, and Frantisek Plasil. Model Checking of Component Behavior Specification: A Real Life Experience. *Electronic Notes in Theoretical Computer Science*, 160:197–210, 2005.

[86] Emilia Katz and Shmuel Katz. Modular verification of strongly invasive aspects: summary. In Mario Südholt, editor, *FOAL*, pages 7–12. ACM, 2009.

[87] Shmuel Katz. Aspect categories and classes of temporal properties. 3880:106–134, 2006.

[88] Gregor Kiczales. Aspect-oriented programming. *ACM Comput. Surv.*, 28(4es):154, 1996.

[89] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In Jørgen Lindskov Knudsen, editor, *ECOOP*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[90] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *11th Europeen Conference on Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer-Verlag, 1997.

[91] D. Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[92] J. Kramer, J. Magee, and S. Uchitel. Software Architecture Modeling and Analysis: A Rigorous Approach. In *Proc. of SFM'03*, volume 2804 of *LNCS*, pages 44–51. Springer-Verlag, 2003.

[93] Shriram Krishnamurthi, Kathi Fisler, and Michael Greenberg. Verifying aspect advice modularly. In Richard N. Taylor and Matthew B. Dwyer, editors, *SIGSOFT FSE*, pages 137–146. ACM, 2004.

[94] Ivan Lanese, Francisco Martins, Vasco Thudichum Vasconcelos, and António Ravara. Disciplining orchestration and conversation in service-oriented computing. In *SEFM*, pages 305–314. IEEE Computer Society, 2007.

[95] Harold Lockhart. Demistifying SAML. `http://dev2dev.bea.com/pub/a/2005/11/saml.html`, September 2005.

[96] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying distributed software architectures. In *Proceedings of ESEC '95 - 5th European Software Engineering Conference*, pages 137–53. IEEE, 1995.

[97] Jeff Magee, Jeff Kramer, and Dimitra Giannakopoulou. Behaviour Analysis of Software Architectures. In *First Working IFIP Conference on Software Architecture (WICSA1)*, volume 140 of *IFIP Conference Proceedings*, pages 35–50, 1999.

[98] Olivier Maréshal, Pascal Poizat, and Jean-Claude Royer. Checking Asynchronously Communicating Components Using Symbolic Transition Systems. In R. Meersman, Z. Tari, and al, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *Lecture Notes in Computer Science*, pages 1502–1519. Springer-Verlag, 2004.

[99] Hidehiko Masuhara and Kazunori Kawauchi. Dataflow pointcut in aspect-oriented programming. In Atsushi Ohori, editor, *APLAS*, volume 2895 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2003.

[100] Bertrand Meyer. *Object-Oriented Software Construction, second editon*. Prentice-Hall, 1997.

[101] Microsoft and All. Web services dynamic discovery (WS-Discovery. `http://specs.xmlsoap.org/ws/2005/04/discovery/`, April 2005.

[102] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, (100):1–77, 1992.

[103] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[104] Robin Milner. Calculi for synchrony and asynchrony. *TCS: Theoretical Computer Science*, 25:267–310, 1983.

[105] Tadao Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[106] Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, Bruno De Fraine, and Davy Suvée. Explicitly distributed aop using awed. In Robert E. Filman, editor, *AOSD*, pages 51–62. ACM, 2006.

[107] Dong Ha Nguyen and Mario Südholt. Property-preserving evolution of components using vpa-based aspects. In *Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA'07)*, LNCS. Springer Verlag, November 2007.

[108] Oscar Nierstrasz. Regular types for active objects. In Andreas Paepcke, editor, *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 1–15, Washington, DC, USA, September 26October –1 1993. ACM Press.

[109] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, 11 April, 2007.

[110] OASIS Web Services Reliable Messaging TC. WS-Reliability 1.1. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf, November 2004.

[111] Klaus Ostermann, Mira Mezini, and Christoph Bockisch. Expressive pointcuts for increased modularity. In Andrew P. Black, editor, *ECOOP*, volume 3586 of *Lecture Notes in Computer Science*, pages 214–240. Springer, 2005.

[112] Julia Padberg. Abstract Petri Nets as a Uniform Approach to High/Level Petri Nets. In J. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques, Selected Papers of the 13th International Workshop on Algebraic Development Techniques (WADT'98)*, volume 1589 of *Lecture Notes in Computer Science*, pages 241–260, Lisbon, Portugal, 1999. Springer-Verlag.

[113] George A. Papadopoulos and Farhad Arbab. Coordination Models and Languages. In *The Engineering of Large Systems*, volume 46 of *Advances in Computers*, pages 329–400. Academic Press, August 1998.

[114] Cesare Pautasso. Bpel for rest. In *7th International Conference on Business Process Management (BPM08)*, pages 278–293, Milan, Italy, September 2008.

[115] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th International World Wide Web Conference (WWW 2008)*, pages 805–814, 2008.

[116] Pascal Poizat and Jean-Claude Royer. A Formal Architectural Description Language based on Symbolic Transition Systems and Modal Logic. *Journal of Universal Computer Science*, 12(12):1741–1782, 2006.

[117] Pascal Poizat, Jean-Claude Royer, and Gwen Salaün. Bounded Analysis and Decomposition for Behavioural Description of Components. In Springer Verlag, editor, *FMOODS*, number 4037 in Lecture Notes in Computer Science, pages 33–47, 2006.

[118] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Jorge Cardoso and Amit P. Sheth, editors, *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004.

[119] J. Rathke and M. Hennessy. Local Model Checking for Value-Passing Processes (Extended Abstract). In Martín Abadi and Takayasu Ito, editors, *Third International Symposium on Theoretical Aspects of Computer Software TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 250–266. Springer-Verlag, 1997.

[120] John Reynolds. *Theories of programming languages*. Cambridge University Press, 1999.

[121] Alex Rodriguez. Restful web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/, November 2008.

[122] Jean-Claude Royer and Michael Xu. Analysing Mailboxes of Asynchronous Communicating Components. In R. Meersman, Z. Tari, D. C. Schmidt, and al., editors, *On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 1421–1438. Springer-Verlag, 2003.

[123] SAP. Service oriented architecture. http://help.sap.com/content/documentation/esoa/.

[124] Damien Sereni and Oege de Moor. Static analysis of aspects. In *AOSD*, pages 30–39, 2003.

[125] C. Sibertin-Blanc. Cooperative objects : Principles, use and implementation. In *Concurrent Object Oriented Programming and Petri Nets*, volume 1973 of *LNCS*, pages 216–246. Springer-Verlag, 2001.

[126] S. Singh, J. Grundy, J. Hosking, and J. Sun. An architecture for developing aspect-oriented web services. In *Proceedings of the third European Conference on Web Services (ECOWS)*, pages 72–82. IEEE Computer Society, 2005.

[127] Simple Object Access Protocol v1.2. `http://www.w3.org/TR/2003/REC-soap12-part0-20030624/`, 2003.

[128] SpringSource home page. `http://www.springsource.org`.

[129] Maurice H. ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web service composition approaches: From industrial standards to formal methods. In *Second International Conference on Internet and Web Applications and Services (ICIW'07)*, page 15. IEEE Computer Society, 2007.

[130] The World Wide Web Consortium. Web Service Choreography Interface, 2002. Version 1.0, `http://www.w3.org/TR/wsci/`.

[131] UDDI. Introduction to UDDI: Important Features and Functional Concepts. `http://uddi.org/pubs/uddi-tech-wp.pdf`, 2004.

[132] Hugo Torres Vieira, Luís Caires, and João Costa Seco. The conversation calculus: A model of service-oriented computation. In Sophia Drossopoulou, editor, *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2008.

[133] Robert J. Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-12)*, pages 159 – 169. ACM Press, 2004.

[134] Mitchell Wand, Gregor Kiczales, and Christopher Dutchyn. A semantics for advice and dynamic join points in aspect-oriented programming. *ACM Trans. Program. Lang. Syst.*, 26(5):890–910, 2004.

[135] Dave Winner. XML-RPC specifications. http://www.xmlrpc.com/spec, 1999.

[136] World Wide Web Consortium. Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, 2005.

[137] Nobuko Yoshida and Vasco Thudichum Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electr. Notes Theor. Comput. Sci*, 171(4):73–93, 2007.

[138] Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web services choreography standards - the case of REST vs. SOAP. *Decision Support Systems*, 40(1):9–29, 2005.