CESSA

Compositional Evolution of Secure Services using Aspects

ANR project no. 09-SEGI-002-01

Language definition and aspect support

Extension of the Service Model for Security and Aspects

Abstract. In this milestone we present the current state of the integration of the service model defined in deliverable D1.2 and formal models for aspects and the definition of security properties. We provide an analysis of security requirements in the context of three service composition and evolution scenarios, and discuss related work on service collaborations, security and aspects. We then introduce a framework for the definition of security properties and provide a first discussion about which aspect mechanisms to include in the forthcoming aspect model.

Deliverable No.	D1.3
Task No.	1
Type	Milestone
Dissemination	Internal
Status	Final
Version	1.2
Date	31 March 2011
Authors	D. Allam, R. Douence, H. Grall, J.-C. Royer, M. Südholt (EM-Nantes)

Contents

Chapter 1

Introduction

In this milestone we present a point of advancement on the integration of the service model defined in deliverable D1.2 and formal models for aspects (to be used for service evolution) and the definition of security properties based on concrete requirements from application-driven scenarios and an analysis of the relevant state-of-the-art.

Since the CESSA project's main objective is the secure evolution of horizontal and vertical service compositions, we are striving, in fine, for evolution mechanisms at all levels of service compositions (including collaborations, processes and service implementations). In this milestone, however, we consider evolution properties, especially security properties, on the level of collaborations. We therefore discuss how to develop languages for expression evolution properties and for specifying collaborations. Concretely, we envision the use of session types as foundations for this extension.

In order to enforce evolution properties, we consider that monitors are sufficient. At the collaboration level, they act as membranes for processes, thus controlling communication between processes and the network. We therefore need to extend the service model with an aspect layer, allowing monitors to be woven, and a composition layer, allowing monitors to be defined as processes containing sub- processes, corresponding to the processes under control. These layers will be developed in an incremental way, mainly following our needs for security.

This document is structured as follows. In Chapter 2, we present three service composition and evolution scenarios that provide requirements for the security and aspect extensions. Chapter 3 presents related work on collaborations, security properties and mechanisms as well as work on the use of aspects for software security. Chapter 4, finally, presents a framework for the definition and use of security properties as well as a discussion of aspect mechanisms that may be included in our aspect model.

Chapter 2

Scenarios for Services Evolutions

We can distinguish three kinds of evolution for the Loan Scenario:

- Legislation evolution: Laws may change, national and European governments can propose new rules for loan agreements.
- Business evolution: Adding a new agent, changing the validation process, etc. This is the business of the bank.
- Technical evolution: Modifying the systems, changing a faulty protocol, adding cryptographic feature, etc. It concerns the design and implementation of the software system.

The following scenarios are extensions of the initial loan negotiation scenario described in Figure 2.1.

2.1 Data Confidentiality

Suppose that the customer's file is divided into two parts, one for the customer information at the bank and another one for his private information at the government. The bank and the government need to protect their sensitive data from unauthorized access. Only some data can be shared during a collaboration and should be secret to outside agents.

2.1.1 Adding a New Agent

The first evolution in our loan negotiation scenario is adding the European government as a new agent which will impose some European rules for loans. As it is represented in Figure 2.2, the evolution must add new services at the process level of the bank, the national government and European government. After receiving the government verification acknowledgement, the bank has to check the European government rules by sending a subset of the client data information and the loan details. From the European government rules, the bank must send the message firstly to the national government, who will forward the message by adding a subset of customer's confidential data stocked in his database.

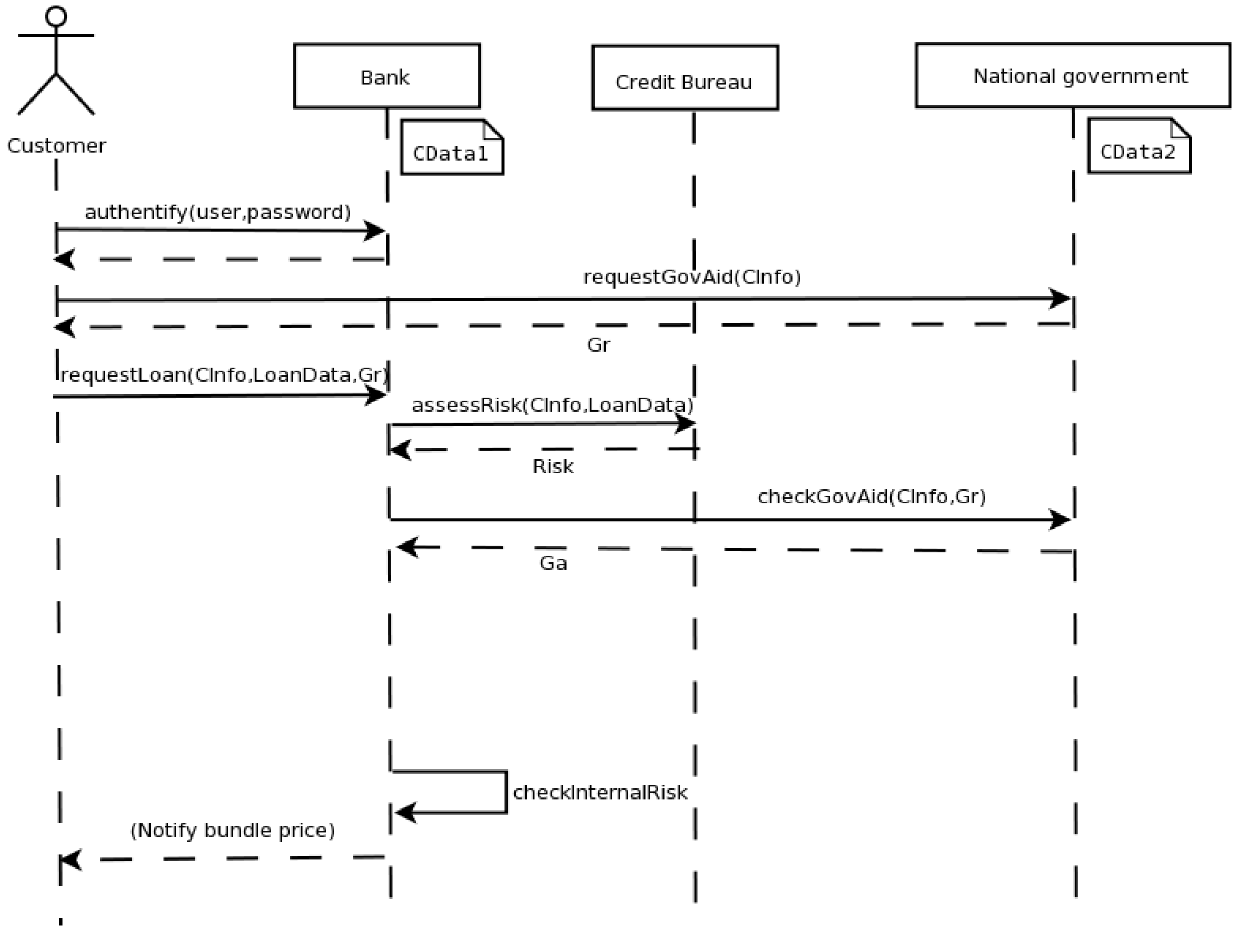

Figure 2.1: Initial Loan Negotiation Scenario

In this scenario, we impact the process orchestration by adding an evolution at the collaboration level (horizontal composition) to impose some new rules in the loan negotiation scenario, *i.e.*,, the contract between different involved agents was updated by adding the European government.

To realize this evolution, we can take one or other of following ways:

- Modify the bank and the national government processes to include new services.
- Add a filter for incoming and outgoing messages. This filter will contain a sub-process which accomplish the added job without modifying the implementation of the actual loan process.

2.1.2 Modifying some Access Rights

Completing the previous scenario, we now restrict the access rights of the national government. The customer's subset data at the bank will be shared only between the bank and the European

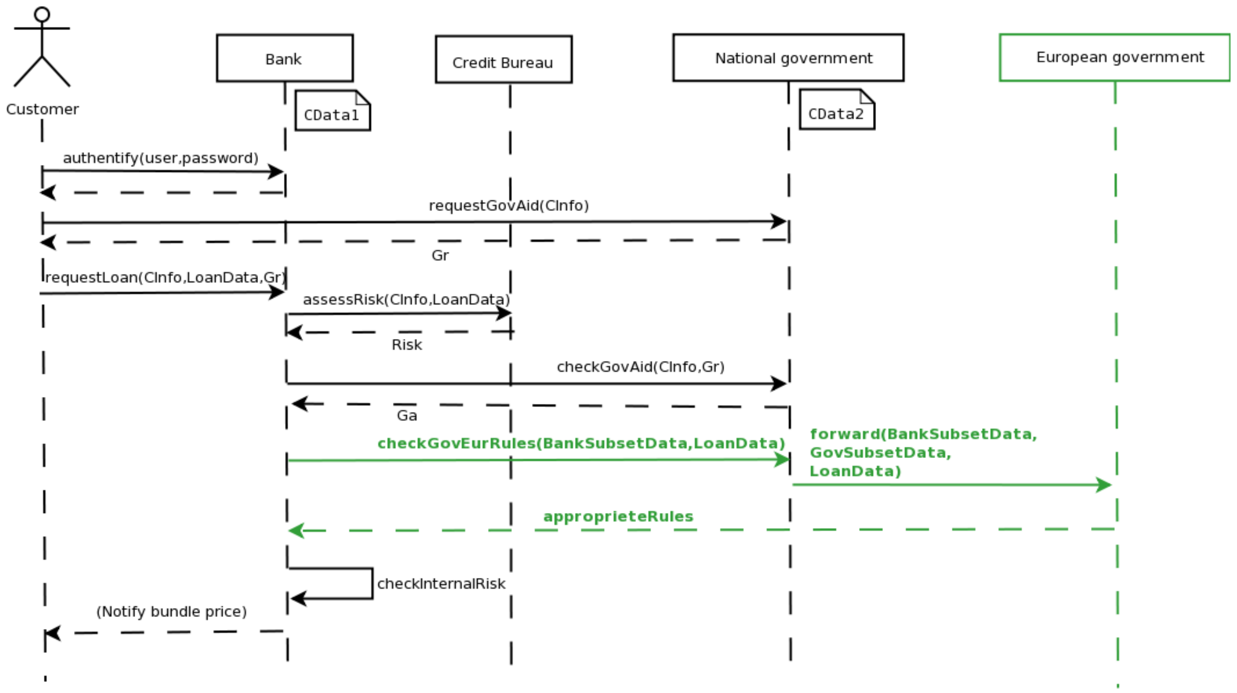

Figure 2.2: Loan Negotiation Scenario: Confidentiality

government, that is why they decide to evolve their processes by adding an encryption/decryption protocol for their in/out data. As the new encryption protocol is seen as a new contract between the bank and the European government, this evolution is at the collaboration level impacting the horizontal composition.

Once we are sure that the bank confidential data arrive to the destination without secret violation, we can wonder about what the European government will do with this data. For instance, he could stock the decrypted data or send it to another peer. To resolve that we have two solutions:

- Analyze data flow and information flow to be sure that all confidential data are encrypted when they are sent on the network.

- Modify the vertical composition by encrypting confidential data before sending a message.

We can provided all these modifications by simply adding a filter to decrypt incoming messages and to encrypt outgoing ones.

2.2 Integrity

In this scenario, the integrity property will assure that one peer receives the correct data. In our loan request example, we can suppose that the customer tries to ensure a better government support and a lower interest for his loan. That is why he sends different information to the government and to the bank. As it is illustrated in Figure 2.3, this cheat was not detected by the

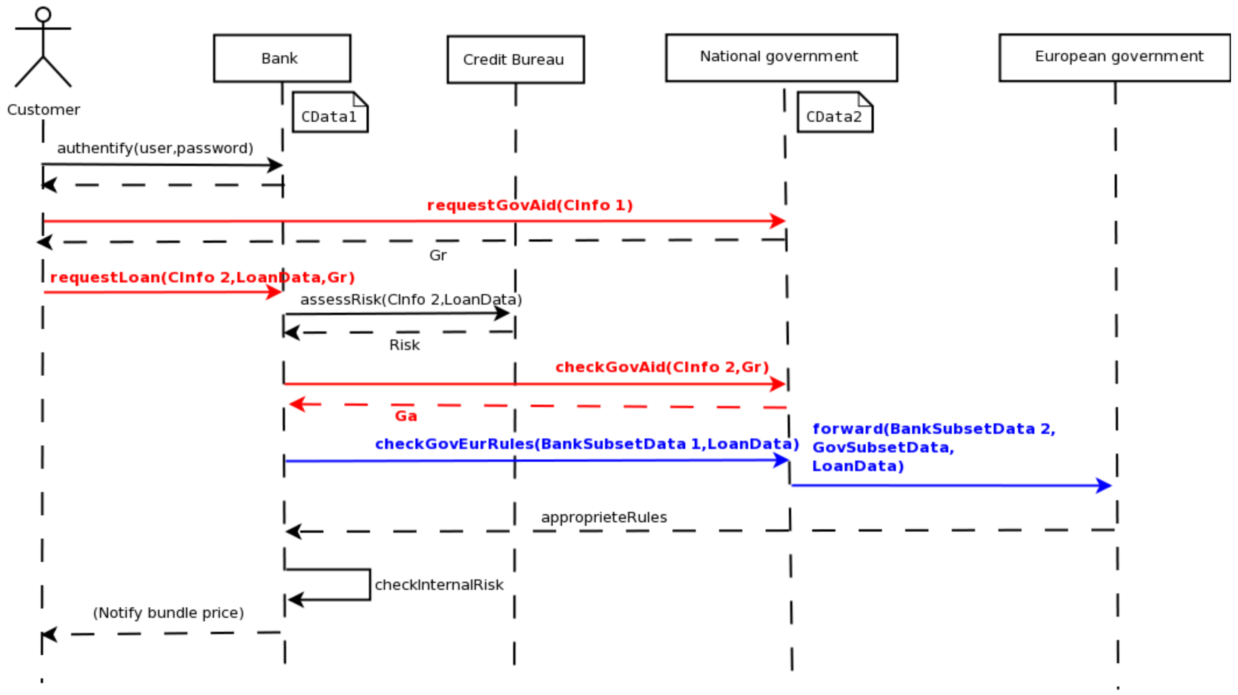

Figure 2.3: Loan Negotiation Scenario: Integrity

loan system because the government sends an acknowledgement to the bank that he gave the aid ”Gr” to the customer’s information ”CInfo2” while he did not.

To resolve this problem, we must verify that the government’s function which associates an aid to some client information, is a reversible function. That is not the case in this scenario because we have:

$$(f(CInfo1) = Gr \text{ and } f(CInfo2) = Gr) \Rightarrow \text{(f is not reversible)}$$

Another example of integrity problem is presented in the Figure 2.3, when the bank sends a subset of customer’s information to the national government who falsifies it before forwarding it to the European government. One possible solution to let the European government know that he receives the right data is that: The bank and the government define a reversible function *h*, hard to be find by a hacker. When the bank sends his data he will add a key verifying the property:

$$h(key) = BankSubsetData1$$

at receipt, the European government detects that the bank data was falsified because of the following result:

$$h^{-1}(BankSubsetData2) \neq key.$$

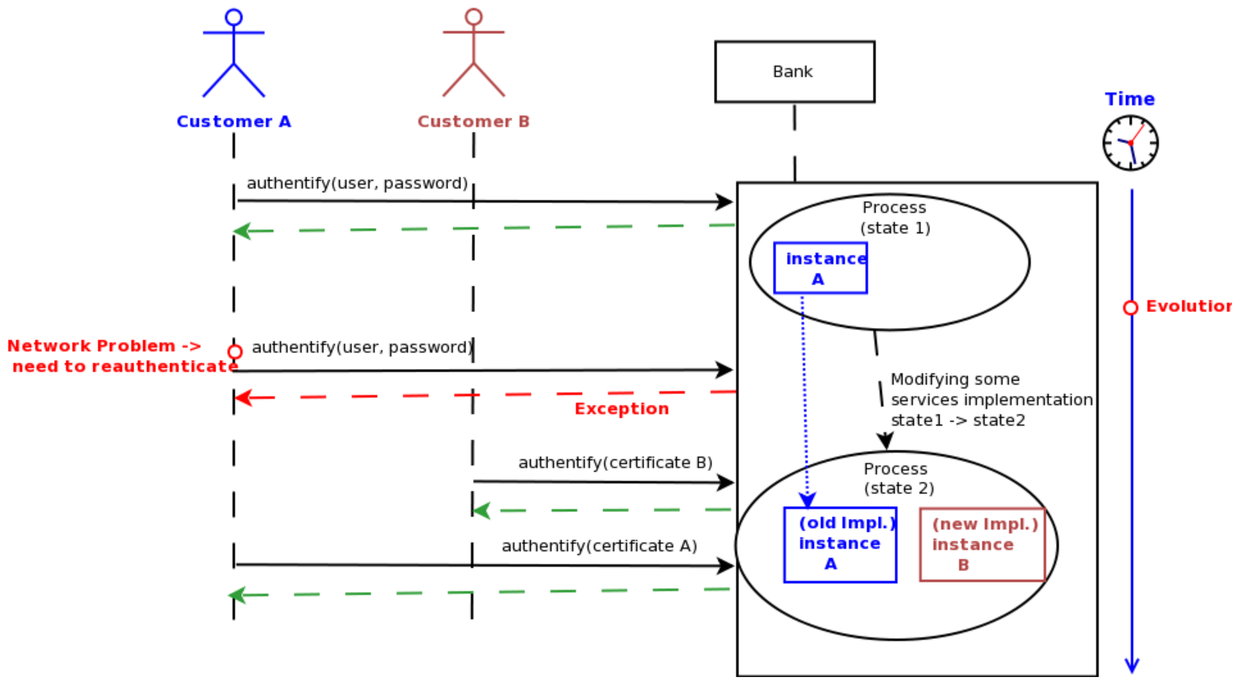

Figure 2.4: Loan Negotiation Scenario: Authentication

2.3 Authentication: Modifying the Access Protocol From Password to Certificate

One point we should have in mind is that a loan process is a long living process, at least one month. There are also many clients at different steps in this process. Thus in case of a required change we cannot simply stop all the loan processes. To go from the password authentication system to the certificate one we have two strategies:

- The best solution is to migrate progressively the different services. Thus we will have two versions of the authentication which can cohabit during a period of time. Then we modify or add some new services which handle mutual authentication on the bank side, having both password-based and certificate-based during a period

- The bank will no more accept an authentication by a password. If a customer still use it, the bank process will generate an exception by sending a message to the client indicating the different steps to follow in order to adapt the service calls on the mobile application to use the certificate.

In Figure 2.4, we illustrate the second strategy. This evolution presents some modifications at the horizontal and vertical compositions:

- Horizontal composition: Modifying services at the customer and the bank process level.

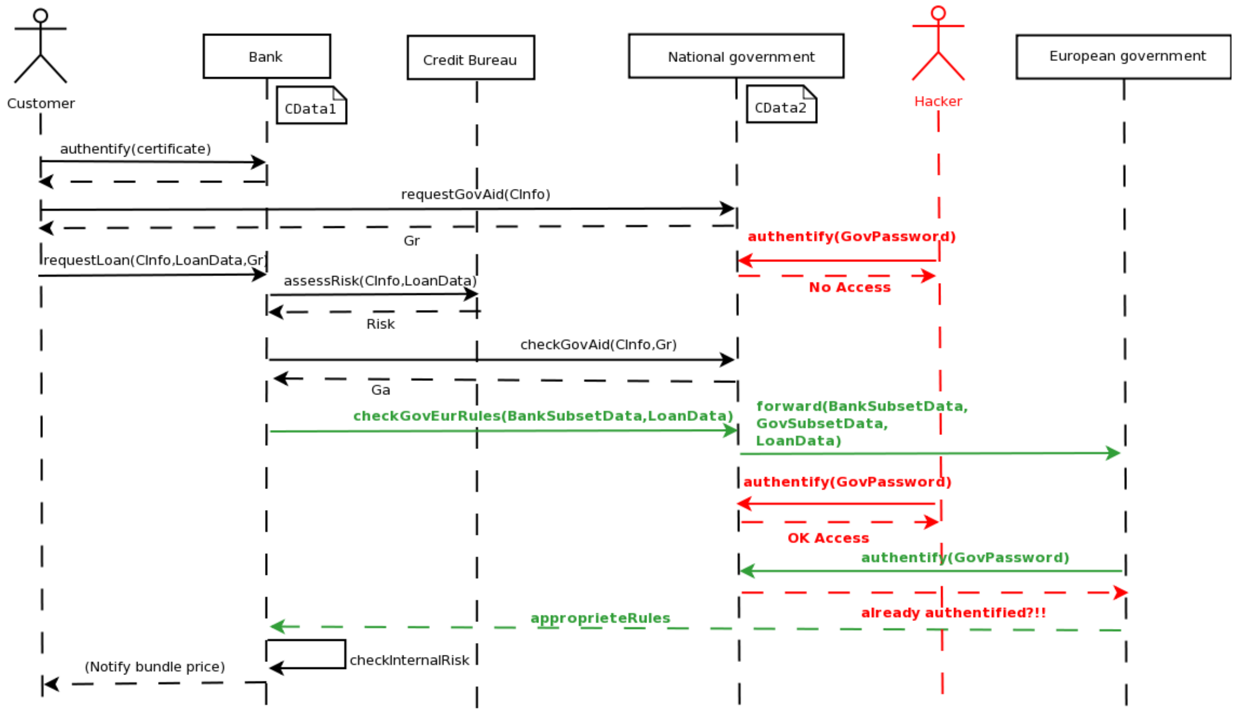

Figure 2.5: Loan Negotiation Scenario: Accountability

- Vertical composition: Adding some new services to generate some certificate for customers and store them in the database.

An important question appears for this kind of evolution: As it is shown in the Figure 2.4, before the evolution, some processes could be launched to serve some clients. After the evolution, we must keep all these old processes, thus we have dynamic evolution. That is the case for the client A, but it is not for the client B who had authenticated after that the evolution finished, so the corresponding process instance already takes in consideration all evolution steps.

2.4 Accountability

It means to be able to trace which is the responsible of an action. This is usually done by some logging system.

2.4.1 Adding some Logging Capabilities

The objective is to add logging for some actions in a local process. For instance, the security expert needs to analyze what happens after detecting a security problem. He suspects that some actions are responsible of this problem and wants to trace them. This is vertical composition:

- Modify the process in order to capture all the necessary information.

- Store the log in secure devices at the low level of resources.

In the Figure 2.5 we suppose that we add a new task for the European government, now he must authenticate at the national government to have an access right to the data base. But to avoid a possible attack, the national government will add a log at calling the service ”checkGovEurRules” to stock at which time he forwards a message from the bank to the European government. So on, when he will receive a call on the authentication service for the European government access, he will check the log to verify if there was a previous call from the bank for checking the European loan rules. In this case, the national government authorizes the access to the data base, else he supposes that a hacker claims to be the European government. In the Figure 2.5 we see clearly how an intelligent hacker can break this security strategy by listening on the channel between the national government and the European one.

The next scenario impacts other processes in the collaboration, it shows horizontal composition.

2.4.2 Logging a Complex Interaction

To solve the problem from the previous scenario, the expert still need to trace a more complex and distributed interaction between several partners to identify who is the spy or the origin of the problem. He wishes to log a set of actions on several distributed process.

This issue can imply several monitoring policies to be conflicting in the sense that one peer my wish to log sensitive information owned by a second peer. Therefore, means to preserve the confidentiality of part of the data need to be provided. Still one question that should deserve our attention: How the security expert can ensure that some partners do not log confidential data to resend it to a malicious?

2.5 Conclusion

In the previous examples, we can observe that: Resolving one security issue can involve to add some new means which are either violating other security properties or introduce new possible security flaws in the system. Another important observation is related to the enforcement of the security properties. All the considered properties can be expressed at the level of the collaboration. Usually an attack involves at least two agents which are collaborating but in an unexpected manner. For instance, to enforce confidentiality in one previous scenario we must encrypt the data thus becoming secret to non trustful agents. Controlling at the global level, that means at the choreography level, the messages we can envision to control many (if not all) the security properties. Obviously there are still work to achieve the enforcement of all the security properties. We need to formally reason, thus a formal model based on this global point of view is required. But means to enforce at the process/choreography level the projection of the secure collaboration are also critical. The next section discussed related work which seems relevant for these purposes.

Chapter 3

Related Work: Collaboration Specification, Security and Aspects

All the security properties considered in the previous chapter about security scenarios stand at the collaboration level. We first study the different methods used to specify collaborations, and therefore security properties. Then, in order to enforce a security property, we also need to control the messages exchanged over the network. As no centralized control is possible, the control mechanisms have to be decentralized: in a first approximation, they will act over the message boxes, by filtering communication between processes and the network. Since these control mechanisms will be implemented by using aspects, we also study some related work about security and aspect-oriented programming.

3.1 Specification of Collaborations

In our model, collaborations are specified by a protocol, also called a choreography, defined as the sequence of the messages exchanged over the network. Each message owns an origin and a target, two processes, and conveys some argument and possibly a return channel for the reply. A protocol can describe a collaboration in an extensional way, without any approximation. However, this description is generally awkward and unreasonable, since a protocol can be parametric, each collaboration corresponding to an instantiation of the parameters. For instance, a client can send in its request a string and the server replies by sending an integer. At the protocol level, values may not matter.

To define protocols, we can adopt three points of view.

Internal Point of View: Via a process calculus

External Point of View: Via a transition system

Logical Point of View: Via a logic

We now detail these points of view.

3.1.1 Three Points of View: Internal, External and Logical

The three points of views are inter-related. Indeed, a process defined in a process calculus can be semantically interpreted as a transition system. A transition system can then be considered as a logical model of some formulas expressed in the logic used. We now review the formalisms associated to the three points of view.

Process Calculi

Process calculi have been used to specify collaborations, with two main proposals: session types [18] and contracts [15]. These proposals are strongly related, since encodings exist [30]. Generalizations are also available, for instance a generic type system for processes [27] and conversation types [10]. In the following, to simplify, we subsume session types and contracts, or their generalizations, by the general term *collaboration types*.

Collaboration types are defined as processes in variants of process calculi like CCS, values being abstracted as value types, corresponding to sets of values. A conformance relation associates a collaboration type to a collaboration: see for instance the formalization of this relation for a calculus of global interaction [12]. A subject reduction property then holds: in its strongest form [1][27, Th. 4.1.1], it asserts that if collaboration c with collaboration type t reduces to collaboration c', then there exists collaboration type t' such that c' has type t' and t reduces to t'. Progress properties can then be deduced [1]. A relation of safe replacement is also often defined [22, 15]: it corresponds to the usual subsumption rule allowing to convert from a subtype to a supertype. This relation entails interesting composition properties. Given a collaboration type t, can we compose a collaboration with type t from two collaborations with respective collaboration types t' and t''? The answer is positive if the composition of t' and t'' is a subtype of t. A generalization of this problem is as follows: given a collaboration type t, can we realize a collaboration with type t? The solution is to project over processes involved in the collaboration. The solution is valid under certain conditions [12]. For instance, assume four processes A, B, C and D. Consider as a collaboration a sequence of two messages, m from A to B followed by n from C to D. Then the four projections are defined as follows.

- A: send m
- B: receive m
- C: send n
- D: receive n

The parallel composition of the four projections does not generate the intended collaboration: a notification between B and C is needed. If B is equal to C, then the projection over B defined as "receive m then send n" generates the collaboration: a connectedness condition does matter.

Transition Systems and Logics

Transition systems have also been used to specify collaborations. It is not surprising as the standard semantics of process calculi resorts to labeled transition systems. Thanks to the definability equivalence between classes of transition systems and logics like temporal logic, these results can also be formulated by using logic, therefore in a declarative manner instead of an operational one. Actually, these frameworks, either operational with transition systems or declarative with logics, allow the same questions as with process calculi to be dealt with.

For instance, Bultan *et al.*. [8] propose a solution for the previous realizability problem using the same top-down approach: starting from a specification of a collaboration as a set of sequences of messages, they show how to project the specification to get local processes realizing the specification. Processes are modeled as transducers, generating sequences of output messages from sequences of input messages. More generally, the relationship between collaborations and processes realizing them have been studied not only in an operational setting [9] using process algebras and labeled transition systems, but also in a logical setting [32] using a spatial and temporal logic.

3.1.2 Example: Session Types

We now detail a recent approach, following the first point of view based on process calculi. It should provide guidelines to develop collaboration types.

Session types have been introduced in the end of the nineties as an approach to bilateral synchronous interaction protocols [24]. Since then the expressivity of session types and their support for composition properties has been widely enlarged, in part following the development of Service-Oriented Computing and the associated standards. As a result, session types are currently one of the most expressive formal framework for interaction protocols that provide static guarantees on type safety, refinement and progress for interactions. Recently, Castagna *et al.*. [14] propose an overview of the foundations whereas Dezani-Ciancaglini *et al.*. [18] presents the concepts and a state of the art.

In the following we briefly present the features that set apart session types from less expressive previous approaches and their applications, notably to software security.

3.1.2.1 Advanced features of session types

Multiparty and asynchronous interactions Honda *et al.* [25] have extended the original approach of session types by support for multiparty protocols that may use synchronous but also asynchronous communication. Both of these extensions are critical to collaboration protocols in our context: First, multiparty protocols are notoriously difficult to represent in terms of bilateral collaboration protocols because many synchronizing events have to be made explicit in the bilateral collaboration protocols that do not need to be included in the multiparty protocol. Second, today's service-oriented applications typically rely on synchronous and asynchronous communication.

A safe type systems ensures that multiparty protocols may be defined using a global protocol that can projected on per-site protocols. Technically, this is ensured using causality relations between sequences of asynchronous interactions to ensure the coherence of multiparty interactions.

Event-based programming Event-based programming can be integrated with session types as introduced by Hu *et al.* [26] using three primitives: (i) for asynchronous (non-blocking) input, (ii) for the dynamic inspection of session types, and (iii) collection of heterogeneous session types. These primitives allow the selection of an incoming message based on its protocol, *i.e.,* session protocol, and arrives on one of several channels that have different protocols.

This extension has been added to a Java-based prototype called SJ ("Session Java") and applied in the context of a real-world SMTP server.

Dynamic multi-roles Daniélou and Yoshida [17] have shown how a notion of roles that individual participants can join and leave dynamically can be integrated with session types. This extension enables protocols to be defined in terms of roles that are automatically instantiated for all participants that belong to the roles mentioned in the protocol. Furthermore, a participant may belong to several roles. Communication between roles are automatically broadcast to all participants to a role.

Technically, the handling of roles with several participants requires the introduction of universal types on the level of session types. The coherence of protocols is ensured, as usual, through a safe type systems that ensure coherence between a global protocol type and its projections to individual sites. Two progress properties are ensured: (i) a message send will always progress to a receiver with the correct session type, and (ii) a participant may always join to on-going sessions. However, this last property is restricted in that late joiners in a session cannot interact during the on-going run of the session; they become active only on start of the following run of the session or iteration of the current session.

3.1.2.2 Applications of session types.

Session types have been applied in several domains, notably to the software security, software components, and distributed applications.

Sessions and software security. Session types have been applied to problems of access and information flow, as well as the definition of cryptographic protocols (see, *e.g.,* [6]).

Capecchi *et al.* [11] studied whether session types can be used to enforce access control and information flow properties. (They also provide references on previous approaches that ensure access control on bilateral session types.) As the underlying interaction model, they use multiparty session types with delegation. Delegation here means that a message sent to one participant is sent (implicitly and, in a sense, securely) to another participant. Security properties can be expressed in terms of security levels for participants and data, the security levels of both may be disclassified. Capecchi *et al.* show how these session types can be used to avoid divulging of information above a given security level in the presence of access control and a limited form of declassification (where declassification only takes place during value communication).

Session types for software composition. Session types have been applied to different problems in software composition and service interactions.

Carbone *et al.* [13] point out the link between session types and the web service choreography description language (WS-CDL). Similarly, Vallecillo *et al.* [38] discuss the relation of session types to the interactions of software components.

Sivaramakrishnan *et al.* [35] present how session types can be used for the optimization of distributed applications. Because of the more precise definition of the recursive interaction protocols between distributed participants, opportunities for data aggregation and redirection of service calls can be identified that permits, *e.g.,* to decrease the number of messages sent over the network and thus speed up distributed computations.

3.2 Security as Aspect

There are few related works devoted to AOP and security.

Aspects for enforcing trace properties AOP can be used to define execution monitors that dynamically forbid some execution traces. In particular, Hamlen et al. [23] propose SPoX (Security Policy XML) a declarative aspect oriented specification language. This language enables to define execution monitors as automata. The transitions are defined by pointcuts denoting the execution events of interest. Each automaton defines a sink-like state that enables to specify forbidden transitions. In this case, the monitored program is halted before it executes the forbidden action. A bytecode weaver has been implemented: it in lines the execution monitor in the program. For the sake of security, an aspect should enforce a security policy and should not itself breach a security policy. So, the aspect language is restricted to effect-free pointcuts and advices.

Conflict detection of Aspects Enforcing Trace Properties However, this specialized language still allows to define ambiguous security policy [28]. A security policy is ambiguous when it denotes a non deterministic security automaton: a transition in the automaton authorizes an action, while another transition in the same automaton forbids the very same action. The security policy programmer should be warned in this case. This can be statically analyzed. The analysis is program agnostic. It has two phases: first, it determines which edges have a common source but distinct destinations. This is computed by translating the automaton to a linear programming problem. Second, it determines if the pointcuts of those edges have a non empty intersection. This is computed by translating the pointcuts to a boolean satisfiability problem (SAT). If the analysis detects a policy is ambiguous, it returns a witness security state and joinpoint to help the programmer to debug his policy. It also enables the programmer to check if a combination of two policies is ambiguous. A tool has been implemented.

History-based Security Property for Securing AOP In general AOP is very invasive. For instance, in AspectJ an aspect can read or modify any variable of the base program. An aspect can even skip calls to a method in the base program or replace it by another method. This makes it easy for an aspect to skip security related checks or to introduces non secure computation. Even when an aspect is less invasive it can breach a security policy. Indeed, in Java the security is based on principals and dynamic call stack inspection, but the advices of a before or after aspect do no occur in the call stack. De Borger et al [7] have proposed a permission system for secure AOP. Their approach is based on a full history-based (rather than stack-based) access control. This ensures the right are correctly updated when an advice is entered and exited. If a security aware aspect needs to increase the current rights, this must be done explicitly with *grant* and *accept*. This enables to execute a piece of code with higher rights and to increase the trust in the result of a piece of code. The developer must annotate a program to be secured with statements to trigger the AOP run time system. A weaver has been implemented by modifying the AspectJ weaver, but not the Java Virtual Machine. It introduces computations that maintain the security information at run time. This code accesses at runtime a security policy and checks if aspects have sufficient rights before an advice is executed.

Formal Semantics of AOP for Security The previously described work deal with AOP and security but the user has to trust the approach and the tools because they are not formally based. Lambda-SAOP [3] proposes such a basis. It is an aspect-oriented calculus for security (it provides a data flow pointcut). It is based on the lambda-calculus extended with imperative sequence and side effects (variable assignment and dereference). The pointcuts considered are: function call, variable assignment and dereference, and data-flow. The advice language is the imperative lambda-calculus extended with a proceed construction. Advices can be woven before, after or around a joinpoint. The weaving process is static: it is type-based and the type system used is effect-based (it infers for a given program red and written regions which are abstractions of memory locations). This calculus models the main features of AspectJ (e.g., static weaving, before/after/around advice, zero or multiple occurrences of proceed) but it does not model concurrency or distribution.

Formal Semantics of AOP in a Concurrent and Distributed Context A theory of distributed aspects [37] proposes a formal model for AOP in a distributed context. It is based on the join calculus (a formal calculus for declarative concurrent and distributed computation). First, the author defines the syntax and the (static and dynamic) semantics of an idealized object oriented language : the core objective join calculus. Second, he extends this language with aspect oriented features: pointcuts (call, cflow, host, etc.), around advices with proceeds, aspects with pairs of pointcut and advice. The original semantics is extended accordingly. Third, a translation process from the aspect language to the base language is defined and proven correct. This translation can be understood as a constructive definition of a weaver. Translation of more sophisticated features (such as sequence of synchronous aspects or distributed control flow) is discussed.

As shown by these different work, there is still work to be done in order to propose a formal framework for AOP for security.

Chapter 4

Extension of the Service Model for Security

We present a framework designed in order to deal with security issues in the service model, as defined in Deliverable D1.2 [4]. The framework not only allows security properties to be expressed and enforced, but also paves the way for a generalization, beyond security towards general evolution. As it relies on an aspect layer, we end by enumerating different possible features of the aspect layer.

4.1 A Framework: From Security Enforcement to General Evolution

Since we want to enforce security properties at the level of collaboration, we need to extend the service model with:

- a language to express security properties,
- a language to specify collaborations,
- a satisfaction relation, asserting when a collaboration satisfies a security property,
- an enforcement mechanism.

We now detail these different items.

4.1.1 Security Properties

A collaboration involves processes. With respect to security, the processes can be split into two parts: the trusted part and the untrusted one. Consider a collaboration and the two associated sets of processes. We can assume that the projection of the collaboration over the trusted part is known: it corresponds to an intended behavior. As for the untrusted part, this assumption is not reasonable: indeed, some untrusted processes may exhibit malicious behaviors, which are not intended. That is the reason why we describe an insecure collaboration as follows, by considering collaborations such that a part of their projection over trusted processes is fixed.

Definition 4.1.1 (Unsecure Collaboration) *Assume that a security property φ and a collaboration projection b involving trusted processes are given. A collaboration c is* insecure *with respect to φ and b if*

- *the projection of c over the trusted processes entails projection collaboration b,*
- *collaboration c does not satisfy property φ.*

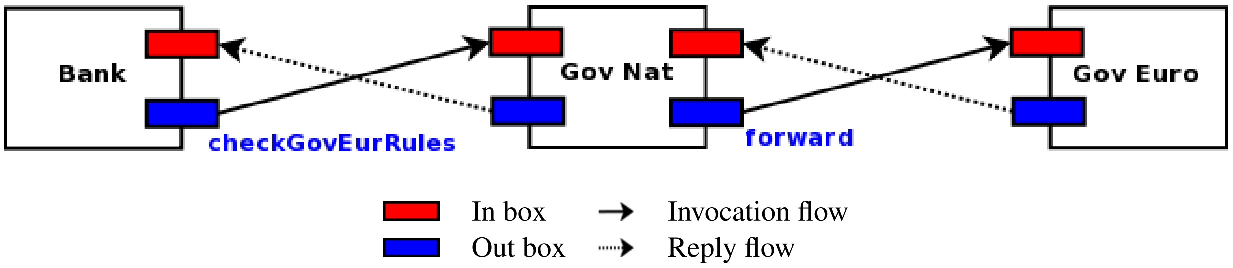

Figure 4.1: Loan Negotiation Scenario: Process Interactions

Example: Insecure Collaboration An example of an insecure collaboration is illustrated in Figure 4.1. Here we come back to the loan negotiation scenario extended in Chapter 2, particularly to the confidentiality evolution, described in Section 2.1.2. In this sub-scenario, we suppose that the bank data sent to the European government is confidential and must be shared only between these two peers. According to our definition, we express as follows the security property.

1. *c* is a collaboration between the bank and the European government, which are trusted, and the national government, which is not trusted. Functionally, it represents a data sending of a message from the bank to the European government, via the national government.
2. *b*, the projection of *c*, is defined as follows, given a confidential value *s*.
   - At the bank: data sending of message *s* to national government
   - At the European government: data receiving of message *s* from the national government
3. The security property φ is defined as follows:

   for all collaboration *c* with projection *b*, no untrusted process can read value *s*.

Thus, the projection of *c* over the trusted bank and the trusted European government entails collaboration *b*, but collaboration *c* does not satisfy property φ because the confidential bank data is sent to an untrusted peer: the national government.

4.1.2 Security Enforcement

How to avoid insecure collaborations? A solution is to add a reference monitor to each trusted process in order to control the message boxes, in other words the input and output messages. We can consider reference monitors with different control powers: for instance, they could add or remove messages, or modify the content of messages. Since a reference monitor mediates the communication between the trusted process under its control and the network, it can be considered as a wrapper of the trusted process, and expressed as a specific process containing the trusted process as a sub-process. We are now able to formally express security enforcement.

Definition 4.1.2 (Security Enforcement) *Assume that a security property* φ *and a collaboration projection b involving trusted processes are given. Reference monitors controlling trusted processes enforce property* φ *under collaboration projection b if for any collaboration c such that*

- *the projection of c over the trusted processes under the control of reference monitors entails projection collaboration b,*

we have:

- *collaboration c does satisfy property* φ.

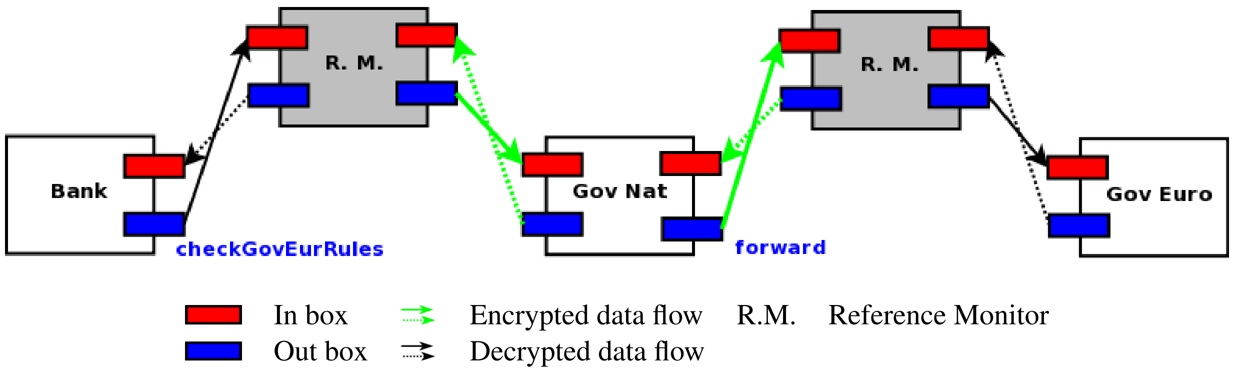

Figure 4.2: Loan Negotiation Scenario: Protect Confidential Data by Encryption–Decryption

Example: Security Enforcement To explain how reference monitors could be used to ensure a security property in a collaboration, we present in Figure 4.2 a solution to protect confidential data in the previous example. Since we want to make the collaboration evolve without modifying the existing processes, we add two reference monitors as wrappers for both trusted processes. The reference monitors encrypt outgoing messages and decrypt the incoming ones. Thus, we have enforced the confidentiality property.

4.1.3 Implementation of Reference Monitors

How to implement reference monitors? Our solution is to extend the service model with an aspect layer. An aspect make processes evolve by adding membranes to them. A *membrane* can be defined as a process mediating the communication between the process encapsulated in the membrane and the network. Hence, the process encapsulated has to be modified: instead of directly communicating with the network, it communicates after aspect weaving with the membrane, which communicates with the network. With this view, an extension to the service model is required: in addition to processes, we need sub-processes. Indeed, the reference monitor corresponds to a process, containing a sub-process, the process that is monitored. Actually, this extension is useful, generally speaking, as it allows a greater modularity. A process can now be either distributed or encapsulated in another process: collaborations now correspond to global interactions and local interactions.

Example: Evolution for Horizontal and Vertical Compositions We now deal with another example to justify the usefulness of the proposed extension and to assess its impact on security. We take again the loan negotiation scenario, but this time with an evolution enforcing the integrity property described in Section 2.2. There are still three processes, corresponding to the bank, the national government and the European government. But now, the latter contains two sub-processes, *A* ans *B*, as well as a database, also corresponding to a specific sub-process. Both processes *A* and *B* use the database. We assume that the national government and process *B* are not trusted whereas all other processes are trusted.

Suppose that in our collaboration, we must satisfy the following integrity property φ for a message *m*:

> an untrusted process cannot modify message *m*.

To enforce this property, no encryption–decryption mechanism is required. Instead, we just need to add to message *m* its hash value with respect to some hashing function, which cannot be guessed by untrusted processes. More precisely, we need to add reference monitors in order to control the usage of message *m*: every time the message flows outside the trusted processes, a reference monitor adds to the message its hash value, whereas every time the message flows inside the trusted processes, a reference monitor checks that the hash value corresponds to the message. Hence, if an untrusted process modifies the content of message *m*, it is not able to compute the new hash value, so that the reference monitors can detect this forbidden modification. In Figure 4.3, three monitors have therefore been added, at the frontier between trusted processes and untrusted ones. The evolution has an impact not only on the horizontal composition, around the national government, but also on the vertical composition, around process *B*.

To conclude, the framework, as stated, is not limited to security issues: indeed it can be extended to allow general evolutions to be enforced. Here is a summary of its intended features, for this generalization:

- a language to express evolution properties,

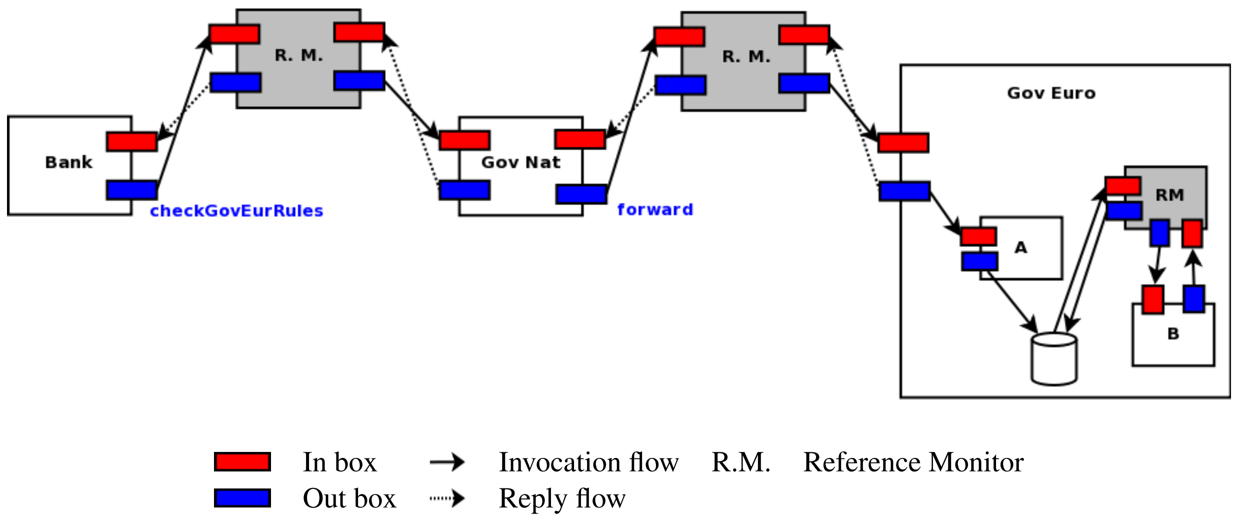

Figure 4.3: Loan Negotiation Scenario: Integrity property for Horizontal and Vertical Compositions

- a language to specify collaborations,
- a satisfaction relation, asserting when a collaboration satisfies an evolution property,
- an aspect layer, allowing membranes to be added to control collaborations.

4.2 Aspect Mechanisms

A large set of classes of aspect mechanisms can potentially be included in the CESSA aspect model. In the following, we present a list of individual aspect mechanisms and classes thereof that we currently evaluate, together with, if already available, first information on the pertinence of their inclusion based on the CESSA service model and the need for the expression and manipulation of security-related functionalities and properties. We also provide concrete examples illustrating uses and dangers of aspect mechanisms in the context of the enforcement of security properties of service compositions.

The following (classes) of aspect mechanisms are currently being included or evaluated in our model for the secure evolution of service compositions.

1. **Atomic aspect mechanisms** consist in mechanisms that enable the manipulation of individual execution

   (a) **Service calls:** redirection, modification of arguments, suppression (etc.) of individual calls (including methods calls that are used to implement services).

   (b) **State manipulation:** direct modification of security-related state (*e.g.*,, databases) and state in service implementations (*e.g.*,, assignment to state variables).

(c) **Exception throwing and handling:** fault handling in service choreographies and orchestrations, exception handling in OO or imperative service implementations etc.

Inclusion into the CESSA aspect model: all of these mechanisms are basic mechanisms on which many service evolutions have to be built. They also constitute a major cornerstone for the injection of many security-related functionalities. Furthermore, adding these mechanisms to our service model and implementation frameworks should not pose major problems. Their unrestricted use is, however, likely to destroy relevant security properties (an argument that is less relevant if aspect are used to enforce security properties by construction).

They are therefore likely to be included in the CESSA service model but will be equipped with mechanisms to guard them in order to provide explicit support for property preservation. Such guards may take the form of application conditions or expressed through effect-mediating aspect interfaces, see item 3.

2. **History-based aspect mechanisms**

   (a) **Control flow:** AspectJ's `cflow` [29], Douence et al.'s `path` [20]

   (b) **Data flow:** Masuhara's et. al.'s `dflow` [31]

   (c) **Finite-state systems:** stateful aspects [21], tracematches [5], etc.

   (d) **More expressive (non-regular) languages:** Viggers et al.'s tracecuts [39], Ha et al.'s VPA-based aspects [33] etc.

   Inclusion into the CESSA aspect model: History-based approaches allow to directly express many security-related functionality, *e.g.,*, information-flow properties, use of cryptographic protocols, and the monitoring for intrusion detection. Furthermore, history-based pointcuts allow to express modifications directly in terms of the collaboration structure (including choreography and orchestration) of our service model. However, several history-based mechanisms, notably data flow pointcuts and pointcuts defined in terms of non-regular structures, are notoriously difficult to implement efficiently if applied to fine-grained frequent execution events.

   History-based mechanism are therefore likely to be included in the CESSA aspect model, notably control flow, data flow (probably with restrictions on the grain of application), and regular pointcuts. The use of non-regular languages will, however, probably be excluded due to their weak support for the insurance of properties.

3. **Interfaces mediating the effects of aspects**

   (a) **Controlling aspect application:** Douence et al.'s applicability conditions [19], Aldrich's open modules [2], Skotinitiois et al.'s aspect interfaces [36]

   (b) **Explicit event types**: Leavens et al.'s Ptolemy [34] [, AOSD'11], EScala [AOSD'11]

Inclusion into the CESSA aspect model: approaches for interfaces mediating the application of aspects to base programs provide effective and flexible means for controlling the effects of aspects. Since such control is crucial for the enforcement and analysis of (security) properties, the CESSA aspect model will include a notion of aspect-aware interface, probably consisting of a more primitive notion, similar to applicability conditions, and a declarative mechanism using explicitly typed events.

Chapter 5

Conclusion

In this milestone we have presented a point of advancement on the integration of the CESSA service model with formal models for aspects and the definition of security properties

Starting from extensions of the loan negotiation scenario with respect to security, we have presented a framework for extending the service model, with its two levels, the collaboration one and the process one.

We consider that evolution properties, especially security properties, are expressed at the collaboration level. We therefore need to develop languages for expression evolution properties and for specifying collaborations. We envision to use session types as a foundation for this extension.

In order to enforce evolution properties, we consider that monitors are sufficient. They act as membranes for processes, thus controlling communication between processes and the network. We therefore need to extend the service model with an aspect layer, allowing monitors to be woven, and a composition layer, allowing monitors to be defined as processes containing sub-processes, corresponding to the processes under control. These layers will be developed in an incremental way, mainly following our needs for security.

Bibliography

[1] Lucia Acciai and Michele Boreale. A type system for client progress in a service-oriented calculus. In Degano et al. [16], pages 642–658.

[2] Jonathan Aldrich. Open modules: Modular reasoning about advice. In Andrew P. Black, editor, *ECOOP*, volume 3586 of *Lecture Notes in Computer Science*, pages 144–168. Springer, 2005.

[3] Dima Alhadidi, Nadia Belblidia, Mourad Debbabi, and Prabir Bhattacharya. lambda_saop: A security AOP calculus. *Comput. J*, 52(7):824–849, 2009.

[4] D. Allam et al. Model and formal architecture specification. Deliverable D1.2, CESSA ANR project, no. 09-SEGI-002-01, January 2011.

[5] Pavel Avgustinov, Julian Tibble, and Oege de Moor. Making trace monitors feasible. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *OOPSLA*, pages 589–608. ACM, 2007.

[6] Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *CSF*, pages 124–140. IEEE Computer Society, 2009.

[7] Wouter De Borger, Bart De Win, Bert Lagaisse, and Wouter Joosen. A permission system for secure AOP. In Jean-Marc Jézéquel and Mario Südholt, editors, *Proceedings of the 9th International Conference on Aspect-Oriented Software Development, AOSD 2010, Rennes and Saint-Malo, France, March 15-19, 2010*, pages 205–216. ACM, 2010.

[8] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 403–410. ACM, 2003.

[9] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration: a synergic approach for system design. In *In Proc. of 4th International Conference on Service Oriented Computing (ICSOC*, pages 228–240. Springer Verlag, 2005.

[10] Luís Caires and Hugo Torres Vieira. Conversation types. In *Proceedings of the 18th European Symposium on Programming Languages and Systems*, ESOP '09, pages 285–300. Springer, 2009.

[11] Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session types for access and information flow control. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2010.

[12] Marco Carbone, Kohei Honda, and Nobuko Yoshida. A calculus of global interaction based on session types. *Electronical Notes in Theoretical Computer Science*, 171:127–151, June 2007.

[13] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[14] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *PPDP '09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 219–230, New York, NY, USA, 2009. ACM.

[15] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '08, pages 261–272, New York, NY, USA, 2008. ACM.

[16] Pierpaolo Degano, Rocco De Nicola, and José Meseguer, editors. volume 5065 of *Lecture Notes in Computer Science*. Springer, 2008.

[17] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 435–446. ACM, 2011.

[18] Mariangiola Dezani-Ciancaglini and Ugo De'Liguoro. Sessions and session types: an overview. In *Proceedings of the 6th international conference on Web services and formal methods*, WS-FM'09, pages 1–28. Springer, 2010.

[19] 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.

[20] Rémi Douence and Luc Teboul. A pointcut language for control-flow. In Gabor Karsai and Eelco Visser, editors, *GPCE*, volume 3286 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2004.

[21] RÃ©mi Douence, Pascal Fradet, and Mario SÃ¼dholt. Trace-based aspects. In Mehmet Aksit et al., editor, *Aspect-Oriented Software Development*. Addison-Wesley, 2003.

[22] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42:191–225, 2005.

[23] Kevin W. Hamlen and Micah Jones. Aspect-oriented in-lined reference monitors. In *Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, PLAS '08, pages 11–20, New York, NY, USA, 2008. ACM.

[24] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems—ESOP'98, 7th European Symposium on Programming*, volume 1381 of *LNCS*, pages 122–138, Lisbon, Portugal, 1998. Springer.

[25] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008.

[26] Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, and Kohei Honda. Type-safe eventful sessions in java. In Theo D'Hondt, editor, *ECOOP 2010 - Object-Oriented Programming, 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings*, volume 6183 of *Lecture Notes in Computer Science*, pages 329–353. Springer, 2010.

[27] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311:121–163, 2004.

[28] Micah Jones and Kevin W. Hamlen. Disambiguating aspect-oriented security policies. In Jean-Marc Jézéquel and Mario Südholt, editors, *Proceedings of the 9th International Conference on Aspect-Oriented Software Development, AOSD 2010, Rennes and Saint-Malo, France, March 15-19, 2010*, pages 193–204. ACM, 2010.

[29] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, London, UK, UK, 2001. Springer-Verlag.

[30] Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In Degano et al. [16], pages 681–700.

[31] 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.

[32] Carlo Montangero and Laura Semini. A logical view of choreography. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages*, volume 4038 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2006.

[33] Dong Ha Nguyen and Mario Südholt. Property-preserving evolution of components using vpa-based aspects. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 613–629. Springer, 2007.

[34] Hridesh Rajan and Gary T. Leavens. Ptolemy: A language with quantified, typed events. In Jan Vitek, editor, *ECOOP*, volume 5142 of *Lecture Notes in Computer Science*, pages 155–179. Springer, 2008.

[35] K. C. Sivaramakrishnan, Karthik Nagaraj, Lukasz Ziarek, and Patrick Eugster. Efficient session type guided distributed interaction. In Dave Clarke and Gul A. Agha, editors, *Coordination Models and Languages, 12th International Conference, COORDINATION 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings*, volume 6116 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2010.

[36] Therapon Skotiniotis, Jeffrey Palm, and Karl J. Lieberherr. Demeter interfaces: Adaptive programming without surprises. In Dave Thomas, editor, *ECOOP*, volume 4067 of *Lecture Notes in Computer Science*, pages 477–500. Springer, 2006.

[37] Nicolas Tabareau. A theory of distributed aspects. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD '10, pages 133–144, New York, NY, USA, 2010. ACM.

[38] Antonio Vallecillo, Vasco Thudichum Vasconcelos, and António Ravara. Typing the behavior of software components using session types. *Fundam. Inform*, 73(4):583–598, 2006.

[39] Robert J. Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In Richard N. Taylor and Matthew B. Dwyer, editors, *SIGSOFT FSE*, pages 159–169. ACM, 2004.