Network of Excellence

Deliverable D7.2

Initial Solutions for Secure Service Architectures and Design
Abstract
NESSoS WP7 focuses on methods and tools to model secure and adaptive architectures for software applications deployed over the Future Internet. A first set of contributions focuses on providing modeling formalisms and methods in order to precisely design secure web applications. The second set of contributions focuses on the use of abstract models to analyze the impact of security evolution and to drive the reconfiguration of the system according to these evolutions. This report synthesizes the initial results developed within WP7 during the first year of NESSoS. The scope of the deliverable corresponds to the timing in the DoW, this is why Task 7.4 (Reusable architectural know-how), which will start only in the 2nd year, is not covered in this deliverable.

Keyword List
Security, Future Internet, Design, Architecture, Model-based, Decomposition, Composition, Dynamic, Adaptation, Contract, Enforcement, Reusable Know-how
### Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Type of Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>initial setup</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA)</td>
</tr>
<tr>
<td>0.2</td>
<td>initial version</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
<tr>
<td>0.3</td>
<td>draft version</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
<tr>
<td>1.0</td>
<td>draft</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
<tr>
<td>2.0</td>
<td>draft before revision</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
<tr>
<td>2.1</td>
<td>after review</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
<tr>
<td>3.0</td>
<td>final version</td>
<td>(Benoit Baudry, Olivier Bendavid - INRIA; Marianne Busch-LMU; Javier Cubo, Marina Egea, J. Antonio Martin, Riccardo Scandariato, Isabel Vinagre Torres)</td>
</tr>
</tbody>
</table>

### Document Review

NESSoS - 256980
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Reviewer</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 26, 2011</td>
<td>2.0</td>
<td>Marinella Petrocchi</td>
<td>Sections 2.2, 3.1, and 3.2, resemble scientific papers, and, maybe, so many details let the reader not fully aware of the connections of such work with the NESSoS goals (this holds in particular for sections 3.1 and 3.2.)</td>
</tr>
<tr>
<td>Sep 30, 2011</td>
<td>2.0</td>
<td>Christoph Sprenger</td>
<td>Sect 3.1 is hard to understand (I guess even for an expert). The quality and accuracy is good in Sect 2, but needs improvement in Sect 3.</td>
</tr>
<tr>
<td>Oct 24, 2011</td>
<td>2.1</td>
<td>Marinella Petrocchi</td>
<td>I found that my main comments have been addressed.</td>
</tr>
<tr>
<td>Oct 24, 2011</td>
<td>2.1</td>
<td>Christoph Sprenger</td>
<td>I have checked the revised version of D7.2. I have found that most of my comments have been addressed.</td>
</tr>
</tbody>
</table>

(For the editor: please, list a resume of the most relevant reviewers’ comments)

Report Versioning

<table>
<thead>
<tr>
<th>Reports Versioning (x=1,2,3,...)</th>
<th>Outline (0.1)</th>
<th>Draft (1.0)</th>
<th>Under internal review (2.0)</th>
<th>Accepted (3.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2011-06-11</td>
<td>2011-09-23</td>
<td>2011-09-23</td>
<td>2011-10-24</td>
</tr>
</tbody>
</table>
Table of Contents

LIST OF FIGURES ........................................................................... 9

1 INTRODUCTION ........................................................................ 11
   1.1 Integration in WP7 ................................................................. 12

2 MODEL-BASED DECOMPOSITION OF SECURITY CONCERNS .......... 13
   2.1 ActionGUI – Model-based Development of Security-Aware GUIs for Data-Centric Applications ................................................................. 14
   2.2 UWE – UML-based Web Engineering with security features ...................... 16
      2.2.1 Web Engineering with UWE ................................................. 16
      2.2.2 Modeling Access Control for Web Applications ............................ 17

3 MODEL-DRIVEN ADAPTATION OF SECURITY CONCERNS ............... 21
   3.1 RBAC policy evolution ............................................................. 21
      3.1.1 Motivating scenario ......................................................... 21
      3.1.2 RBAC policy evolution : overview .................................... 23
      3.1.3 Conclusion ........................................................................ 25
   3.2 Service adaptation by contracts .................................................... 26
      3.2.1 Overview of contract-based adaptation ................................ 26
      3.2.2 Security adaptation contracts in action ............................... 28
      3.2.3 Conclusions .................................................................... 32

4 CONNECTION WITH OTHER WORKPACKAGES.............................. 35

5 CONCLUSION .............................................................................. 37

BIBLIOGRAPHY .............................................................................. 39
List of Figures

Figure 1.1: Modelling secured architectures: overview of WP7 activities .............................................. 11

Figure 2.1: Model-driven development of security-aware GUIs. .............................................................. 15

Figure 2.2: Address book: Basic rights model ......................................................................................... 17

Figure 2.3: Address book: Outermost navigation state model ............................................................... 18

Figure 2.4: Navigation states profile ....................................................................................................... 20

Figure 3.1: Policy example ....................................................................................................................... 22

Figure 3.2: Policy adapted with delegation ............................................................................................. 23

Figure 3.3: Policy adapted with assignments to a new permission .......................................................... 23

Figure 3.4: Process overview .................................................................................................................. 24

Figure 3.5: Policy enforcement architecture ............................................................................................ 25

Figure 3.6: A simple example of adaptation ............................................................................................ 26

Figure 3.7: Different deployments of service systems with WS-Security .............................................. 27

Figure 3.8: Behavior of services simulating different ways to perform secure shell operations .............. 28

Figure 3.9: Adaptation contract for services $a$ and $b$ ......................................................................... 30

Figure 3.10: Adaptation protocol for contract $C_0$ and services $a$ and $b$. ........................................... 31

Figure 4.1: Relationships from WP7 to other WPs .................................................................................. 35
1 Introduction

NESSoS WP7 focuses on methods and tools to model secure and adaptive architectures for software applications deployed over the Future Internet [14]. A major characteristic of Future Internet applications is “the fading boundary between development time and runtime” [13]. In the specific context of Future Internet architectures, this means that parts of architecture models that are built and analyzed at design time will continue evolving at runtime. The WP7 contributions address this new challenge through the development of secure modelling techniques at different moments of design, deployment and runtime.

Figure 1.1 summarises the relationships between the different contributions developed in WP7. A first set of contributions focuses on providing modeling formalisms and methods in order to precisely design secure web applications. These contributions are detailed in (section 2): ActionGUI and UWE are two UML-based approaches for model-driven security. As part of the integration activities developed in NESSoS, we emphasize the complementarity between both approaches and how they can benefit one from the other.

As mentioned earlier, a major characteristic of Future Internet is its highly dynamic nature. In order to address this characteristic, it is critical that NESSoS provides means to evolve and adapt software applications with respect to security. The second set of contributions developed in WP7 focuses on the use of abstract models to analyze the impact of security evolution and to drive the reconfiguration of the system according to these evolutions. Section 3 synthesizes model-driven approaches for security adaptation, considering the evolution of access control policies and the alignment of security protocols.

All contributions developed within WP7 are related through the use of abstractions to reason about and analyze security concerns. Also, in order to deal with complexity of applications developed for the Future Internet (scale, heterogeneity, adaptability), all works developed in WP7 promote the separation of concerns principle for design, as well as the use of domain specific languages that are the most fit to reason about the different security concerns (access control, secure GUI, privacy, secure protocols, etc.). The use of specific modeling languages is meant to

- reduce the gap between security requirements and the enforcement mechanisms.
- ease the verification of security properties on the right level of abstraction.
• improve the maintainability of the system with respect to security.

Consequently, the different contributions that are presented in this document use different formalisms according to the type of security property they deal with and according to the intention (design or adaptation).

This report synthesizes the initial results developed within WP7 during the first year of NESSoS. The scope of the deliverable corresponds to the timing in the DoW, this is why Task 7.4 (Reusable architectural know-how), which will start only in the 2nd year, is not covered in this deliverable. The initial results for task 7.2 are detailed in section 2, while section 3 reports on results for task 7.3.

1.1 Integration in WP7

As far as integration and collaboration is concerned, it is worth noting the following:

• ETH and IMDEA have tight connections that manifest in the common development of the ActionGUI framework for the UML-based development of secure GUIs. This work has also led to collaborations with LMU and the UWE approach developed there. The first year of NESSoS concludes with solid perspectives for complementary usage of both approaches, as detailed in section 2.

• During this first year, collaborations have been initiated between INRIA and KUL in order to systematically drive the dynamic reconfiguration of security concerns from the requirements [34]. Also UMA and CNR have collaborated on the rigorous adaptation of security protocols [21].

• A large amount of the information captured in design models is expressed in requirements. Consequently, several participants of WP7 have established connections with WP6, which should lead to tighter integration in the second year.
2 Model-based decomposition of security concerns

Model-driven software development emphasizes abstract modelling and separation of concerns at each step in the development. Design of secure systems can benefit from such an approach by decoupling the architecture of services from the mechanisms that will ensure the security (and privacy) of those services. For instance, this decoupling in models allows architects to reason about security early in the development, in isolation from other concerns. They can study tradeoffs among different security mechanisms, simulate security policies, and test security protocols before they are integrated with functional services. Then, through model composition (see Task 7.3), security models can be integrated with other concerns. Finally, models are leveraged to connect both upwards the requirements phase, and downward to the implementation.

Model-driven security The initial realization of model-driven security in Basin et al. [4] is built upon the ideas of model-driven development, also called model-driven architecture [27, 15], which has 3 parts: users create (1) system models in highlevel modeling languages like UML; (2) tools are used to perform automatic model transformation; and the result is (3) a target system architecture. Model-driven security specializes this idea to model secure requirements and generate security infrastructures.

Model-driven security has enormous potential not because it tackles the deep problem of synthesizing “business logic” but rather the shallow yet often extremely wide problem of generating security infrastructure. This infrastructure can be built from standard APIs and assertions and its complexity lies, essentially, in getting the deployment information right, despite the numerous details that must be considered. Security-design models provide a clear, declarative, high-level language for specifying these details. The strength of security-design models also lies in their well-defined semantics.

In our view, the role of model-driven security for Future Internet applications is to propose a methodology for rigorously modeling, analysing, and validating applications’ specific security policies, and for automatically enforcing and monitoring these security policies when these applications are running in the Internet. In particular, security policies that aim at protecting privacy are all-pervasive in Future Internet applications.

Model-based development of secure web applications Secure web applications are becoming increasingly important due to rising cybercrime as well as the growing awareness of data privacy. Besides authentication and confidential connections, both data access (and usage) control and navigational access control are the most relevant security features in this field. Adding such security features, however, to already implemented web applications is an error-prone task. Existing approaches, such as OOHRIA [24], OOWS [33], UWE [16], WebML [25], or ActionGUI [5] already provide well-known methods and tools for the design and development of web applications. Most of them follow the principle of “separation of concerns” using separate models for views, such as content, navigation, presentation, business processes, etcetera.

Within the NESSoS Network of Excellence, we plan to define a tool-supported, model-based approach that will enable web engineers to model security issues in an early phase of the development process. Our initial solution is based on the integration of two of the aforementioned approaches: namely, the UWE approach by Koch et al. [16] and the ActionGUI approach by Basin et al. [5], developed respectively at LMU and IMDEA Software, both partners of NESSoS. Moreover, the ActionGUI approach is an offshoot of the long-term collaboration between IMDEA Software and ETH Zürich, also a NESSoS partner. ActionGUI and UWE are briefly introduced in Sections 2.1 and 2.2, respectively.

There are good reasons for being optimistic about the outcome of the integration of the UWE and ActionGUI approaches: first, both strongly support the principle of “separation of concerns”; second, both provide well-defined metamodels for each of the languages that they support for modeling the different concerns; and third, both support a similar language, in terms of expressiveness, to specify access control rules for domain concepts. However, the challenges ahead are non-trivial: basically, the languages that each approach provides for modeling the different concerns needs to be mapped/translated to each other in a meaningful way. While this may be simple for some concerns (like security), it will require complex transformation rules (in fact, refinement rules) for other concerns (like navigation and presentation) since the corresponding models are specified within each approach at different abstraction levels.
Interestingly, both UWE and ActionGUI will also greatly benefit from this integration. On the one hand, ActionGUI will be able to adopt UWE's higher-level modeling languages to specify navigation, presentation, and business process concerns. On the other hand, UWE will be able to borrow ActionGUI's code-generator to produce web applications directly from its models. The latter particularly implies that UWE will be able to use the ActionGUI model-transformation rules to automatically align the navigation state model with the access control mechanisms in the security models. This alignment is crucial for secure web applications, e.g., a user who is not allowed to access a function of a class should be disallowed to navigate to a node that uses this functionality and vice versa. Finally, this integration will allow the three partners involved, namely, LMU, IMDEA, and ETH, to work together in this workpackage. This collaborations will support the following tasks: model not only access control policies but also usage control policies; and provide model-analysis tools able to cope with security aspects in multi-tier architectures.

2.1 ActionGUI – Model-based Development of Security-Aware GUIs for Data-Centric Applications

Model building is at the heart of system design. This is true in many engineering disciplines and is increasingly the case in software engineering. Model-driven engineering (MDE) [27, 15] is a software development methodology that focuses on creating models of different system views from which system artifacts such as code and configuration data are automatically generated. Proponents of model-driven engineering have in the past been guilty of making overambitious claims: positioning it as the Holy Grail of software engineering where modeling completely replaces programming. This vision is, of course, unrealizable in its entirety for simple complexity-theoretic reasons. If the modeling languages are sufficiently expressive then basic problems such as the consistency of the different models or views of a system become undecidable. However, there are specialized domains where MDE can truly deliver its full potential: in our opinion, security-aware GUIs for data-centric applications is one of them.

Data-centric applications are applications that manage information, typically stored in a database. In many cases, users access this information through graphical user interfaces (GUIs). Informally, a GUI consists of widgets (e.g., windows, text-fields, lists, and combo-boxes), which are visual elements that display and store information and support events (like “clicking-on” or “typing-in”). A GUI defines the layout for the widgets, as well as the actions that the widgets’ events triggered either on the application’s database (e.g., to create, delete, or update information) or upon other widgets (e.g., to open or close a window).

There is an important, but little explored, link between visualization and security: when the application data is protected by an access control policy, the application GUI should be aware of and respect this policy. For example, the GUI should not display options to users for actions (e.g., to read or update information) that they are not authorized to execute on application data. This, of course, prevents the users from getting (often cryptic) security warnings or error messages directly from the database management system. It also prevents user frustration, for example from filling out a long electronic form only to have the server reject it because the user lacks a permission to execute some associated action on the application data. However, manual encoding the application’s security policy within the GUI code is cumbersome and error prone. Moreover, the resulting code is difficult to maintain, since any changes in the security policy will require manual changes to the GUI code.

ActionGUI is a model-driven engineering approach for developing security-aware GUIs for data-centric applications. The backbone of this approach, illustrated in Figure 2.1, is a model transformation that automatically lifts the access control policy modeled at the level of the data to the level of the GUI [3]. More precisely, given a security model (specifying the access control policy on the application data) and a GUI model (specifying the actions triggered by the events supported by the GUI’s widgets), our model transformation generates a GUI model that is security-aware. The key idea underlying this transformation is that the link between visualization and security is ultimately defined in terms of data actions, since data actions are both controlled by the security policy and triggered by the events supported by the GUI. Thus, under our approach, the process of modeling and generating security-aware GUIs has the following parts:

1. Software engineers specify the application-data model.
2. Security engineers specify the security-design model.
3. GUI designers specify the application GUI model.

4. A model transformation automatically generates a security-aware GUI model from the security model and the GUI model.

5. A code generator automatically produces a security-aware GUI from the security-aware model.

Figure 2.1: Model-driven development of security-aware GUIs.

The other key components of this approach are the languages that we propose for modeling the data (ComponentUML), the access control policy (SecureUML), and the GUI (ActionGUI). These languages are defined by their corresponding metamodels and support the rigorous modeling of a large class of data models, security models, and GUI models. For data models, the main modeling elements are entities, along with their attributes and associations; for security models, these elements are roles, permissions (possibly constrained at runtime to satisfy given properties), and the actions associated to these permissions. For GUI models, these elements are widgets, the (possibly conditional) events associated to these widgets, and the (possibly conditional) actions associated to these events. The constraint language OCL [28] is used in all of these models. For security models, OCL is used to formalize the constraints on the permissions. For GUI models, it is used to formalize the conditions on the actions, as well as to specify the information to be displayed in widgets, updated in the database, or passed from one widget to another.

To support a full model-driven engineering development process, we have built a toolkit, named the ActionGUI Toolkit. This features specialized model editors for data, security, and GUI models, and implements the aforementioned model transformation to automatically generate security-aware GUI models. Moreover, our toolkit includes a code generator that, given a security-aware GUI model, automatically produces a complete web application, ready to be deployed in web containers such as Tomcat or GlassFish. A key component of this code generator is our translator from OCL to an SQL-based query language [12], which handles the OCL expressions appearing in the security-aware GUI models.

Overall, we see the full generation of security-aware GUIs from models for data-centric applications as a very promising application for model-driven engineering. By working with models and using code-generators to produce the final products, GUI designers can focus on the GUI's layout and behavior, instead of wrestling with the different, often complex, technologies that are used to implement them. Moreover,
by using model transformations, the problem of establishing the link between visualization and security is successfully addressed.

To appreciate this last point, consider the standard alternative: the default, “ad-hoc” solution of directly hard-coding the security policy within the GUI. This is clearly disadvantageous. First, the GUI designer is often unaware of the application data security policy. Second, even if the designer is aware of it, manual hard-coding the application data security policy within the GUI code is cumbersome and error-prone. Finally, any changes in the security policy will require manual changes to the GUI code that implements this policy, which again is cumbersome and error-prone.

2.2 UWE – UML-based Web Engineering with security features

We demonstrate the integration of security features for the UML-based Web Engineering (UWE) method, which enables web engineers to model security issues in an early phase of the development process. The approach supports the engineer by providing means to model navigational security with a plugin in a UML modeling tool that is integrated in the Service Development Environment (SDE). The SDE serves as software workbench for secure services. Various security-related tools, methods and techniques are being integrated into the SDE workbench in order to enhance the quality of secure software and to accelerate the development process. The SDE is further described in D2.2 [1]. Additionally – as described in D9.2 [2] – our UWE models can be used for the verification of web systems and security properties, such as reachability of navigation nodes in general and of those that are restricted to authorized users. Further information can be found in [8].

2.2.1 Web Engineering with UWE

UWE strongly supports the principle of “separation of concerns” selecting the appropriate UML diagram type and elements for each web concern and using the UML extension mechanisms, i.e. defining a set of stereotypes and tagged values in a profile. In the following, the most important UWE models are introduced:

- The content model is used to represent the domain concepts that are relevant for the web application to be built and the relationships between them.
- A user or context model can be used to collect information needed for adaptation. A role model is a special case of user model, in which characteristics of the user groups are defined with the purpose of authorization and access control.
- The navigation model is used to represent navigable nodes of the hypertext structure and the links between nodes. Navigational nodes are constrained by navigational access control rules, which means that users without permission can see only an error message or an advertisement for a less restrictive account. Furthermore, information accessed through navigation may have specific requirements on the confidentiality, integrity and freshness of data of the web application content.
- A presentation model sketches the rough layout of the pages and not only represents the static GUI widgets but also features of Rich Internet Applications (RIAs) like auto-completion in search fields, live validation of input fields, or drag & drop functionality.
- The process model aims to represent the workflows which are invoked from certain navigation nodes. The same security considerations apply as for the navigation model.

Concepts of the content and user model and their relationships are shown as classes and associations in a UML class diagram. For the navigation model UWE provides two different graphical representations: a structural visualization as UML stereotyped class diagrams and a behavioral form using UML state machines, which eases the specification of security features.

UWE’s presentation model is visualized like a mockup, using composite structure diagrams in which composition is visualized as nested classes and properties. Stereotypes are used for GUI widgets, and tagged values to represent RIA functionality. The process model comprises two views: first the process structure model that describes the relations between the different process classes, which are related to
the navigation, and second the process flow model that shows the workflow for each process. They are represented by UML class diagrams and UML activity diagrams, respectively.

In the following, we describe UWE’s model for access control and its behavioral representation of the navigation model.

### 2.2.2 Modeling Access Control for Web Applications

We integrate the modeling of access control, both for data and navigational access, into the modeling of (rich) Internet applications. This integration enables the modeler to address security aspects right from the early phases of the development life cycle. We rely on standard UML modeling techniques and the definition of a UML profile extending UWE. The profile enhances class diagrams to specify a basic rights model building on role-based access control (RBAC) and state machines for modeling controlled navigation. We illustrate our approach by a secure address book application [18]. Due to lack of space, not all features of our security extension are used in this example, for further information the reader is referred to [7].

The web application of our case study should allow registered users to create and navigate several address books and to add and retrieve contacts in them. Non-registered visitors can only read an introduction and the terms of service until they register or authenticate themselves. Administrators cannot use the address book functionality, but they are allowed to search for users and to delete their accounts including all address books and contacts.

#### Basic Rights Model

The basic rights model is used to specify access control rules for domain concepts which are represented as UML classes and class instances. The Role instances from UWE’s user model and content (or user model) classes, their attributes, and methods are connected with stereotyped dependencies. These dependencies, on the one hand, specify create/read/update/delete (CRUD)-rights; on the other hand, an execution dependency between a role and a method grants execution rights for the method.

![Figure 2.2: Address book: Basic rights model](image)

For the address book example, the basic rights diagram in Figure 2.2 specifies execution rights on methods with dependencies stereotyped «execute» and «executeAll» (the CRUD support using «read» / «readAll» etc. is not shown in this example). The dependencies connect the role instances shown on the right in Figure 2.2 to the methods of the content model, like Contact and AddressBook, or of the user model, such as the class User, on the left. In particular, a non-registered visitor has no execution permissions. The {except} tag for «...All» stereotypes allows the modeler to avoid the creation of too many dependencies.
For instance, a registered visitor can execute all methods of a user object except delete. Further restrictions are added in comments stereotyped by «authorizationConstraint» in the Object Constraint Language (OCL): A registered visitor shall only be allowed to delete his own contacts and address books; an administrator shall have the permission to delete all users except other administrators. The corresponding restrictions on «execute» for Contact::delete(), AddressBook::delete(), and User::delete() use attributes like AddressBook.owner and User.roles from the content model. The pre-defined currentUser refers to the user of the current session.

UWE’s basic rights model offers a compact notation for access specifications where permissions and prohibitions can be readily read off. This is in contrast to approaches like SecureUML [19] where all permissions have to be specified separately in association classes, and exceptions cannot be expressed. However, transformations between SecureUML and our basic rights model are possible.

**Navigation State Model**

A navigation state model describes the navigation structure of a web application and its behavior according to the different states. In UWE, navigation can be represented by a UML state machine: States, possibly hierarchical, represent navigational nodes, whereas UML transitions represent the navigational links between the nodes. The UWE security profile allows to integrate navigational access control, but also session management and secure connections into the state machines specifying navigation. In particular, the navigational state model should be aligned with the access control mechanisms in the basic rights model, as e.g., a user who is not allowed to access a function of a class should be disallowed to navigate to a node that uses this functionality and vice versa.

**Figure 2.3: Address book: Outermost navigation state model**

Figure 2.3 depicts the main navigation state diagram for the address book example. All states are navigational nodes; when the system is in a particular state, the information and behavior offered by this state is accessible.

Navigation starts inside ExternalArea (pointed to from the outermost initial state) which is a sub-state of AddressBookApplication. Here, the sub-state machine LoginViaPasswordForm (indicated by ↖) and state ShowIntro are entered simultaneously (as ExternalArea shows two regions separated by a dashed line). ExternalArea is the starting point of the web application, tagged by {isHome}. The tag {navigationMenu=ExternalMainMenu} tells that when inside ExternalArea and whichever sub-states, the user can access the actions from the navigation menu ExternalMainMenu, which include showLoginNode, showRegisterNode, and developerInfo (we omit a UML representation as a class diagram). When
showLoginNode is selected, LoginViaPasswordForm is entered, when showRegistrationNode is chosen SecureBasicRegistration is entered. Both sub-state machines are, in fact, instances of the UWE security patterns offered for recurring security issues.

Patterns are a common approach to tackle the problem of repetitive tasks. “A security pattern describes a particular recurring security problem that arises in a specific security context and presents a well-proven generic scheme for a security solution.” [31] We use security patterns specified as state machines that can be easily included as sub-state machines in navigation state diagrams. Typical examples are registration, authentication (login mechanisms), credential recovery (lost password), or profile configuration.

After successful login or successful registration (leaving the two success exit points), two types of internal areas can be reached in our example: one for the administrators and one for the registered users who want to manage their contacts. However, the subsequent area depends on the role from the role model the user takes on during login (we assume that role visitors is the default role): The guards on the transitions targeting the internal areas check the access rights. In these guards, currentUser.role is abbreviated to role.

The areas ExternalArea, InternalAreaRegisteredVisitors, and InternalAreaAdmin, as well as the super-state AddressBookApplication are distinguished navigational nodes: Their stereotype «navigational» (8) shows that context information on the navigating user is kept. The areas also restrict navigational access to them to particular roles by using the tag (*roles*. . .), such that, e.g., InternalAreaRegisteredVisitors can only be entered by registeredVisitors. This restriction not only protects against access through navigational transitions which should show appropriate guards, but also prohibits direct unauthorized access via a URL. Additionally, the tag (*unauthorizedAccess*=. . .) specifies which state is entered when the access rule is violated; for both internal areas this state is Error. The tag transmissionType="cif" for the session state AddressBookApplication sets the overall type of data transmission during the session to cif, providing for confidentiality, integrity, and freshness: The implementation should prevent eavesdropping, replaying, or altering of transmitted data. The transmission type is sustained also in the sub-states.

InternalAreaAdmin can be left explicitly by choosing logout from its navigation menu InternalMainMenu-Admins; it will also be left when the user stays idle for more than 20 time units after which the application will transit into the navigational node LogoutMessage. Finally, the «externalLink» (8) developer can be reached from ExternalArea. When this external web page is opened in a new browser window or tab, the system will still be inside ExternalArea, otherwise the web application is left.

The excerpt of UWE’s navigation states profile in Figure 2.4 summarizes the integration of navigation and security we have illustrated for the address book example.

The basic state and state machine stereotype for navigational state models is «navigationalNode>. Here, “navigational” refers to the view and the granularity of the state machines, because not all states and transitions need to represent navigational behavior. In particular, an «externalLink» is not a proper navigational state. The initial node of a web application is marked by isHome. A «navigationalNode» can also be set as isModal, meaning that no other navigational node of the navigation model can be accessed as long as the modal node is active. Each «navigationalNode» can also refer to «navigationMenu»s containing the operations which can be chosen from within the node. The «session» stereotype is derived from «navigationalNode» and thus inherits navigationMenu and isHome; a session additionally keeps sessionData and specifies role access restrictions (roles, unauthorizedAccess, and rolesExpression for more fine-grained rules) and a transmission type.

The UWE profile offers some further features for modeling navigation (cf. [7]): The tag *(goBack)* of the stereotype «target» (8) allows to navigate to the state which previously had been active. Access to collections, e.g., to lists, is specially supported. Also, large menus can be integrated on transitions, which is particularly useful if each user can be associated with a set of roles. Due to the fact that stereotyped diagrams can be transformed into plain UML (without UWE stereotypes), there is no need to change the UML semantics itself.

Furthermore, the UWE profile is constructed in a way that transformation to code can be achieved in the near future. Such a transformation can utilize navigational access control offered by several web frameworks. First ideas for transforming UWE models to parts code for the Scala-based Lift framework [17] are presented in [8]. However, regarding aspects like code generation LMU and DUE will make use of the NESSoS mobility program to elaborate on the possibilities of joint efforts.
3 Model-driven adaptation of security concerns

Software applications for the Future Internet will have to manage highly dynamic environments. In Future Internet settings, the execution environment will change (e.g., available resources will change dynamically, physical computation nodes will come in and out), usages will change (e.g., new users will use the devices in different ways), applications will be accessed everywhere at anytime. This nature of Future Internet requires that software applications must dynamically adapt to many different sorts of changes. In particular, the security concerns of the applications have to adapt to changes.

As illustrated in Figure 1.1, WP7 addresses two different types of security adaptation: the adaptation of the system’s architecture to dynamically reflect changes in role-based access control (RBAC) policies and behavioral adaptation to resolve compatibility issues between security protocols. In order to perform safe and correct adaptation, we rely on abstract models to reason about the consequences of adaptations on the security policies and to drive the actual adaptation.

We keep a model of the policy and the architecture at runtime to reason about adaptation of RBAC policies. This allows us to check that the policy evolution satisfies separation of duty properties and to reconfigure the architecture accordingly. In order to reason about the impact of protocol adaptation on the policy, we model the behavior of each service protocol as well as the contract that specifies how to control them. Based on these two models we can synthesize a correct adaptor that enables the orchestration of the services.

3.1 RBAC policy evolution

Current role-based access-control formalisms allow the static definition of policies as a set of rules. Each rule in the policy associates a role to a set of permissions. A permission determines the operation that can be performed on an object. Users can be assigned to one or many roles. Once a policy has been defined, it can be analyzed and verified. Then it is possible to automatically generate a security mechanism that enforces the policy in the business logic. However, there is currently no support to modify the policy and the security mechanism after deployment. This is a major limitation of current model-driven security approaches. In the following we illustrate the need for dynamic adaptation of access-control policies (Section 3.1.1) and then present our approach for analysis and dynamic enforcement of role-based access-control policies. In the last part we summarized future work that will integrate space and time in the policies and enforcement mechanisms.

3.1.1 Motivating scenario

As a motivating example, let us consider an access-control policy to manage authorization for the address example application presented in section 2.2. Figure 3.1 displays the policy. We distinguish three roles: employee, relationshipManager and rescuer. Roles relationshipManager and rescuer are associated by a SSoD (static separation of duty) relationship, which means that a user can’t be assigned both of these roles. Each role is associated one permission: employee is allowed to read employeeAddressBook; relationshipManager is allowed to read and update employeeAddressBook and externalContactAddressBook; rescuer is allowed to create, read, update and delete emergencyAddressBook. In addition, the policy model of Figure 3.1 presents four users: Gary and Mary can connect as employees, Alicia can connect as employee and relationshipManager, while Bob can connect as employee and rescuer. This policy can be enforced using model-driven security approaches such as ActionGUI or UWE presented in Section 2.1, 2.2.

Now, let us imagine that we want to change the policy after deployment in order to let Gary, Mary and Alicia access the emergencyAddressBook, which can currently only be accessed by Bob. This dynamic evolution requires two steps: change the policy in a consistent way; dynamically change the system to enforce this new policy. Both steps are challenging:

- Consistent evolution. As discussed in the following, there are several strategies to evolve the policy. Some of these strategies lead to violation of access-control consistency properties. Thus, it is necessary to check the consistency of the changes before enforcement.
According to our scenario, first step should consist in applying changes to the policy such Gary, Mary and Alicia could access emergencyAddressBook. In this case, multiple strategies of changes can be performed to cope with the objective. A first strategy could consist in adding delegation relationships between Bob and the others users. This adaptation is presented in Figure 3.2, through delegation relationship, Gary, Mary and Alicia are assigned to Bob's roles including rescuer role, which allows them to perform CRUD operations on emergencyAddressBook. However this adapted policy violates the SSoD property. Violation is due to the SSoD relationship between relationshipManager and rescuer roles, Alicia should not be assigned to the rescuer role. This adaptation strategy illustrates the need of analysis. In this example, the violation of SSoD is highlighted and can be understood, in larger policies, when considering more properties, checking their combination becomes more complex to realize.

Another possible solution of changes could consist in adding a new permission, which allows to associate, read operation to emergencyAddressBook. Then this new permission has to be assigned to the roles of Gary, Mary and Alicia. The result of applying those changes is presented in Figure 3.3. Gary, Mary and Alicia can read emergencyAddressBook through the newPermission and these changes do not violate access-control properties.

Once the access-control policy is consistently changed, the second step aims at updating the security mechanism without interrupting the running system. This would consist in dynamically reconfiguring the security mechanism in the running system architecture. Recent middleware platforms offer the reflexive and adaptation capabilities through safe mechanisms to by add or remove components, connectors and bindings between components. Both mechanisms are necessary in order to determine what to adapt and how to reconfigure the system as such as a precise description of the enforcement policy mechanism architecture. However, there is currently no technique that leverages these middleware capacities to enforce and dynamically update RBAC policies.

This example illustrates two major requirements for adaptive RBAC policies in Future Internet:

- efficient, online verification of RBAC policies to check the consistency of a change in the policy
- establish a safe bridge between RBAC policy specifications and software architecture in order to leverage reflexive and adaptive capacities of middleware for runtime evolution of the policy enforcement mechanism

Figure 3.1: Policy example

- Dynamic system evolution. There is currently no support for the dynamic reconfiguration of access-control policies. Model-driven security approaches would require stopping the system, re-generating and re-deploying the security mechanism.
3.1.2 RBAC policy evolution: overview

Our process of (i) evolving the policy and (ii) evolving the enforcement mechanism in the system consists of three steps:

- generate an adapted policy,
- verify the adapted policy,
- transform the adapted policy into its corresponding enforcement architecture.

Figure 3.4 introduces the flow of activity of our approach. The current policy is first copied, then we apply the changes on this copy to produce the adapted policy. The purpose and the nature of these changes is out of the scope of our current work. Then, the adapted policy is checked against access-control properties (e.g. static and dynamic separation of duties [11]).
If the adapted policy is not valid, then we go back to the current policy and we apply another strategy of change. If the adapted policy is valid it is transformed into an enforcement architecture, which is then dynamically deployed to replace the former enforcement security mechanism.

**Figure 3.4: Process overview**

In the following we detail the main techniques that support this global process.

**The verification of a policy model** is based on two strategies:

- **Before-check.** In the case we know exactly the type of change that is applied to a RBAC policy, it is possible to verify that this change does not violate access control properties prior to its application on the policy. We have defined specific verification operations for each concept that can be present in a RBAC policy. Thus if we know the exact list of elements impacted by a change, we can run the minimal set of verification operations that is necessary to validate the change.

- **After-check:** is meant for a global verification of a policy model. This verification consists in checking all access-control properties that have been defined as invariants over the structure of RBAC policy.

**Classical policy enforcement mechanisms** rely on the communication between two components, a PEP (Policy Enforcement Point), which is usually integrated into the client code application and a PDP (Policy Decision Point), from the server side, in charge of evaluating a PEP request against the policy. Once evaluated, the PDP returns the request answer to the PEP.

We choose the enforcing mechanism proposed by Morin et al [26], which defines an approach based on components to enforce the policy. The main benefit is that this approach clearly separates the business logic from the enforcement mechanism. This facilitates the modification of the enforcement mechanism since there is no impact on the business logic.

**To generate our enforcement mechanism** we rely on a model transformation from the policy model to a component architecture model. The transformation is a straightforward mapping between policy...
elements and their corresponding components defined in the component architecture. The result of this transformation from policy Figure 3.1 is presented in Figure 3.5.

![Policy enforcement architecture](image)

**Figure 3.5: Policy enforcement architecture**

### 3.1.3 Conclusion

We presented an approach allowing to adapt a policy and its corresponding enforcement mechanism without interrupting the running system. This approach consists of three steps, modifying a policy, verifying it and transforming it into an enforcement mechanism architecture which is used for the reconfiguration of the running system. This approach clearly establishes a separation between the business logic and the access-control concern. Our approach could be applied to different kind of policies and architecture. We aim to continue this work in three directions: (i) evaluate the relevance of this approach compared to off-the-shelf ones from the performance point of view; (ii) introduce location and time in RBAC policy (iii) investigate reasoning strategies in order to provide a comprehensive design technique for specifying a policy evolution.
3.2 Service adaptation by contracts

Security is considered to be one of the main challenges as regards the widespread application of Service Oriented Architectures across organizations. WS-Security and its successive extensions [32], have emerged to fulfil this need, but these approaches hinder the loose coupling among services, therefore constraining their reusability and replaceability.

Software adaptation is a sound solution to overcome the incompatibilities in interface, behavior and security constraints among stateful services. However, programming adaptors from scratch is a tedious and error-prone task where special care must be given to concurrency and security issues.

We propose to use security adaptation contracts [20] that allow us to express and adapt the security requirements of the services and their orchestration. Given a security adaptation contract and the behavioral description of the services, such as BPEL processes (Business Process Execution Language), WF workflows (Windows Workflows Foundation), or UML secured service designs - as is presented in Figure 1.1, we can generate the protocol of the adaptor. An adaptor is a kind of orchestrator that complies with the security requirements, such as confidentiality, integrity and authenticity, while overcoming incompatibilities at the signature, behavior and security QoS (Quality of Service) levels. The formalization behind of security adaptation contracts has other applications such as security policy negotiation and automatic security protocol verification. Additionally, in [21] the adaptor protocol is refined to preserve the secrecy properties considered in the service composition.

In Section 3.2.1 we describe an overview of our approach, which argues that security adaptation contracts not only addresses incompatibilities between services but also covers several security-related WS-* specifications in a high-level and integrated manner, hence reducing the effort required from the system architect. Section 3.2.2 presents an example to motivate and illustrate how security adaptation contracts are used to orchestrate incompatible services in behavior and security QoS, and describes our security adaptation methodology. In Section 3.2.3, some conclusions about our approach are drawn.

3.2.1 Overview of contract-based adaptation

An adaptor is a third-party service in the middle of the communication, that is in charge of coordinating all the services involved in the system with respect to a set of interactions defined in a contract. Thus, the adaptor receives, recomposes and forwards every messages it receives in such a way that all the services can interact properly and reach a stable state.

Consequently, all the services communicate through the adaptor as illustrated in Figure 3.7, where a service \( W_{s1} \) wants to activate a \( W_{s2} \). The contract for this simple example is given by the vector \( \langle w_{s1}: on!, w_{s2}: activate? \rangle \). We emphasise that the adaptor interacts with the services using the same name of messages but the reversed directions, e.g. communication between \( on! \) in \( w_{s1} \) and \( on? \) in the adaptor. Furthermore, the adaptor always starts a set of interactions formalised in a vector by the receptions (\( on? \)), and next handles the emissions (\( activate! \)).

![Figure 3.6: A simple example of adaptation](image)

Deriving adaptors is a complicated task since, in order to avoid undesirable behaviors, the different behavioral constraints of the composition must be respected, and the correct execution order of the messages exchanged must be preserved while mismatch situations are corrected.
We use security contract terms to represent the input and output actions (or messages) in the adaptor of WS-Security enabled messages coming from/to the services. These terms are used to receive and process messages, or to compose and send messages from the adaptor, and they are combined in a security adaptation contract.

Figure 3.7: Different deployments of service systems with WS-Security

Figure 3.7 shows several ways of deploying Web Services which use WS-Security SOAP (Simple Object Access Protocol) messages. The most common way (WS 1) involves the definition of the service without security (considering the security is included internally within the service provider), only its signature, by means of WSDL (Web Services Description Language), and behavior are described. If the service provider offering that WS has a WS-Policy defined over the SOAP messages exchanged by that service, the provider must have WS-Security descriptions to comply with that WS-Policy. These WS-Security descriptions of the SOAP message are part of the configuration of security interceptors that capture all SOAP messages and recompose them according to those WS-Security descriptions. These security interceptors require security token services (either local to the service provider or as a third party service) which provide them with WS-Trust mechanisms. The first services that were deployed with WS-Security SOAP messages did not have engines with security interceptors, therefore WS-Security was hardwired within the service logic (WS 2). Finally, there are services without any security (WS 3) that might want to cooperate with security enabled engines.

By using security adaptation contracts, we can express different adaptations required over the behavior of the services for them to interoperate properly, as well as the security requirements that must be met during the communication. The purpose of the security requirements in adaptation contracts is to specify: i) the security checks that must be satisfied by every received message and their sequence in secure conversations, and ii) the transformations which must be effected on the security policy of a service so that it adapts to the security constraints of the system.

Therefore, security adaptation contracts allow us to: i) express the security checks that must be performed by the adaptor; ii) describe how to decompose and recompose the messages that must be transformed by the adaptor while preserving the security restrictions of the services; iii) define security constraints among sequences of messages, such as secure sessions or security protocols; iv) perform analysis over the security of the resulting orchestration against several attacks; and v) retain the ability to adapt signature and behavioral incompatibilities.

Behavioral adaptation involves receiving messages and sending replies with the structure expected by the intended recipient. This recomposition of messages is achieved by symbolic parameters, which specify how the data is received and how the data must be restructured before being sent. However, in security-enabled WS, reception and emission of the WS-Security messages is more complex as these
messages may have to provide certain integrity, confidentiality, or authentication guarantees. Therefore, the adaptor must be capable of verifying that the messages received comply with the security policy of the sender; it must decrypt those parts of the messages that must be recomposed to match the structure accepted by the receiver; and finally send the messages encrypted and authenticated as expected by the partner. Security adaptation contracts are versatile enough in order to perform the required adaptation either in a transparent way or by using the privileges to know secret keys in case of being an entity know by the system (see Figure 3.7).

3.2.2 Security adaptation contracts in action

Web services with security requirements are tightly coupled to the message format, cryptographic algorithms, and protocols used in their communications. In this section, we first illustrate a scenario where such restrictions prevent the proper communication among services and, in this way, we motivate the need for security adaptation contracts. Second, we present the advantages of using adaptation contracts. Finally, we describe the methodology of our security adaptation process.

Motivating Example

![Labeled transition systems](image)

**Motivating Example**

**Example 3.1** In Figure 3.8, we present the behavior of four services for performing secure shell operations (similar to the SSH protocol). Services \(a\) and \(a'\) try to get access to the functionality provided either by service \(b\) or \(b'\), but incompatibilities prevent proper communication. These behaviors are represented by labeled transition systems (LTS, [29]) where operation names are quoted and prefixed with ‘!’ and ‘?’ in output and input actions, respectively. Operations also have a list of arguments including cryptographic messages. Internal operations (i.e., transitions without external communication) are represented by \(\tau\). In this example we shall focus on services \(a\) and \(b\).

On one side, the service \(a\) (Figure 3.8(a)) can perform several requests (with its credential and request as arguments), which can be refused or followed by replies. It is important to highlight that the values for name and pass must remain constant throughout the session, so the same values must be sent in every iteration of the loop. Additionally, parameter nonce is used to correlate requests and replies while avoiding replay attacks. Therefore, the nonce is freshly generated for each request.

On the other side, the service \(b\) (Figure 3.8(b)) begins by notifying its availability with a proceed message. Then it must receive a login message with the credentials, which can be either accepted (another proceed)
or denied. If the login is accepted, several requests can be made (with their results), and results can be exchanged. Additionally, the service allows users to upload files.

These services are incompatible with respect to their signature (e.g., “refused” with “denied”), behavior (e.g., unexpected “proceed”) and security levels (e.g., service a uses an encrypted password, requires the digest of the data, and correlates requests and replies with the argument nonce). Services a and b present complementary functionality and semantics but, due to these incompatibilities, security adaptation is required to make them to cooperate successfully.

Behavioral adaptation of a set of services is achieved by deploying an adaptor. The size of the LTS representing the adaptor’s behavior can grow exponentially with the complexity of the services involved and their design requires taking into account all the possible interleavings between the messages exchanged (see dashed lines in Figure 3.10). We emphasize that although some process algebra could avoid the infinite-state problem, using contracts allow us to express in a more appropriate way the solution of the adaptation required. Thus, first, we reduce infinite-state adaptors with a bounded generation of branches, and second, we use the contract (based in a mapping with correspondences between messages, i.e., counterparts) to coordinate and synchronize in a correct way the communication of services to be composed.

Therefore, we propose to describe adaptors with security adaptation contracts, which abstract away from concurrency issues and focus on the mapping between the operations, arguments and security of the services. This mapping is expressed as a set of vectors which correlate the operations of the services. These vectors use symbolic parameters in place of arguments and they contain security expressions to process, analyze and recompose the messages.

In addition, we might want to enforce some additional requirements over the adaptation such as “a particular message must not be sent more than x times” or “an operation A will be (un)available until the operation B is called”. These requirements constrain the application order of the interactions expressed by vectors. In order to represent such high-level requirements, adaptation contracts also include a vector-LTS (or VLTS, for short) which is a LTS with adaptation vectors as labels. If such restrictions are not required, the vector-LTS is considered to be a single initial and final state with all the transitions looping on it.

**Definition 3.2 (Security Adaptation Vector)** A security adaptation vector (or security vector, for short) for a set of security-enabled services is an element of $SVector = VTerm \cup (VTerm \times VTerm)$ where $VTerm = SId \times \{?,!\} \times CTerm$. Such a vector is noted $\langle s : T \rangle, \langle s : ? T \rangle, \langle s : ? T ; s' : ! T' \rangle$, or $\langle s : ! T ; s' : ? T' \rangle$ where $s \neq s'$ are service identifiers and $T, T'$ are contract terms for any security specification in their respective interfaces.

**Definition 3.3 (Security Adaptation Contract)** A security adaptation contract (or security contract, for short) of a set of services is a triple $\langle V, vLTS, E_0 \rangle$ where $V$ is a set of security-enabled services, $vLTS$ is a vector-LTS over the vectors of $V$, and $E_0$ is the initial environment of the adaptor which contains the initial security tokens required to perform the adaptation.

**Example 3.4** There are several incompatibilities between services a and b (Figure 3.8) which are solved by the adaptation contract $C_0$ in Figure 3.9. First, the “proceed” messages sent from b are received by the adaptor due to vector $v_p$. Operations are prefixed with the service identifier. Vector $v_l$ maps the login request at service b with the appropriate arguments coming from the first request of a. Symbolic parameters will be bound to the received values (e.g., I, P, R and N on the left hand side of $v_l$). Parameters with a superscript $^\wedge$ will be replaced by a value already known by the adaptor, therefore $enc(K^\wedge, P)$ indicates that the value that will be bound to parameter P is encrypted with a known key $K^\wedge$. This key, which must be known at the beginning of the session, is given in an initial adaptor environment $E_0$. This environment states that parameter K is a key, and x is its run time value (Figure 3.9). Parameters I and P are used to compose the "login" message to be sent to serve b. Vector $v_q$ processes the req argument (in R) of the initial request. Once the first request has been fully received, subsequent requests can be mapped directly with vectors $v_q$ and $v_\delta$. Arguments name and pass are checked to be always the same due to superscript $^\wedge$. Nonces are updated and reused accordingly in vectors $v_1, v_\nu, v_\delta$ and $v_f$. Vector $v_q$ can conflict with $v_q$ therefore we use the VLTS$\delta$ (Figure 3.9(b)) to enforce that $v_q$ is triggered only the first time. Vector $v_q$ is guaranteed to be triggered first because of the need to send b : ?"login" at the beginning of
\( V_0 = \{ \langle a:\text{"request"}, I, \text{enc}(K^\wedge, P), R, N ; b:\text{"login"}, I^\wedge, P^\wedge), (v_i) \rangle \\
\langle b:\text{"proceed"}, (v_p) \rangle \\
\langle b:\text{"request"}, R^\wedge), (v_q) \rangle \\
\langle a:\text{"request"}, I^\wedge, \text{enc}(K^\wedge, P^\wedge), R, N ; b:\text{"request"}, R^\wedge), (v_r) \rangle \\
\langle a:\text{"reply"}, D^\wedge, \text{hash}(D^\wedge), N^\wedge ; b:\text{"result"}, D), (v_d) \rangle \\
\langle a:\text{"refused"}, N^\wedge ; b:\text{"denied")} \} \) (v_f) \)

(a) Set of security vectors for the contract

(b) \( \text{VLTS}_{V_0} \) for the contract

\[ C_0 = \langle V_0, \text{VLTS}_{V_0}, E_0 \rangle \]

where

\[ E_0 = \langle \{\text{key}/K\}, \{x/K\} \rangle \]

(c) Adaptation contract \( C_0 \)

Figure 3.9: Adaptation contract for services \( a \) and \( b \)

\( b \). Figure 3.10 shows an adaptation protocol which complies with \( C_0 \) and services \( a \) and \( b \). The transition labels have been reduced to the characters underlined in \( C_0 \) and prefixed with the identification of the corresponding service. Note that the adaptation contract given by the vector \( V_0 \) refers to the messages of the services represented in Figure 3.8(a) and Figure 3.8(b), and that the adaptor (presented later in Figure 3.10) interacts with the services using the same name of messages but the reversed directions by means of the adaptation contract.

The intuition behind contracts is that adaptation vectors are enabled by the current state of the VLTS and triggered by synchronization of an operation of one of their sides with a service. Once a two-sided vector is triggered by one of its sides, the operation on the other side must eventually be synchronized before the behavior of the adaptor reaches a final state. Vectors with only one operation can be used independently. Interleaving between vectors is allowed and the current state of the VLTS is updated as soon as a vector is triggered by any of its sides.

Once we have an adaptation contract and the behavioral description of the services, we use state-of-the-art techniques [23, 9] for the automatic generation of the adaptor protocol. These techniques explore all the required interleaving among the messages enabled by the contract, prune those branches which lead the system to deadlock or livelock situations, and reduce the state-space of the protocol by removing duplicated and unnecessary paths.

These techniques, however, do not support the adaptation of security concerns, yet they can be enabled by the security adaptation contracts presented in this work.

The Role of Contracts

Security adaptation contracts concisely represent the mapping of the operations, arguments and security requirements among services. Adaptation contracts abstract away from concurrency issues, leaving them to the adaptor generation phase. This is especially important because adaptation protocols grow exponentially with the complexity, incompatibilities and interleaving among services. In fact, for finite service behaviors, adaptation protocols are potentially infinite in states and transitions whereas security adaptation
contracts are not. For instance, Figure 3.10 has several dashed transitions which correspond to additional requests received before processing the previous one. These transitions represent a potentially infinite

Figure 3.10: Adaptation protocol for contract $C_0$ and services $a$ and $b$
stack of pending requests, in contrast to the small size of the contract (Figure 3.9) and services \((a\) and \(b\) in Figure 3.8) which resulted in this adaptor.

Additionally, adaptation contracts support high level restrictions expressed in vector-LTS. The level of abstraction of the contracts makes them versatile enough to cope with small changes in the behavior of the services. Let us note that the same contract with different services can generate different adaptation protocols. For instance, any combination of services \(\{a, a'\} \times \{b, b'\}\) is supported by the same contract \(C_0\) but not by the same adaptor (i.e., the adaptor in Figure 3.10 does not work for \(\{a, a'\} \times b\)). Finally, security adaptation contracts are subject to be negotiated\(^1\) among service providers, but this is out of scope in this work.

**Methodology for Security Adaptation**

Our methodology for behavioral Web Service adaptation involves the following steps:

1. First, we abstract the behavior of the services to our formal model. This model allows us to represent the sequence of messages sent and expected by a service and the order in which they must occur. This is done automatically from the public description of the behavior of the services written in abstract BPEL or Windows Workflow. Security requirements can be extracted from WS-Security messages or WS-Policy specifications.

2. Then, taking the service behavior into account, we must design (either assisted with a CASE tool [10] or automatically [22]) an adaptation contract able to solve all of the incompatibilities among the services. At this stage, we can do static validation of the contract (i.e., the contract is well defined) and, although there are not any data nor the concrete adaptor to test the adaptation, we still can perform symbolic simulation over the contract by hand-picking which adaptation vectors to apply at every moment. In addition, using this simulation exhaustively in an automatic way, we can do symbolic model-checking based on the contract and the model of the services to verify LTL and secrecy properties.

3. Using the contract and the behavior of the services, the protocol of the adaptor is generated while taking into account all possible interleaving among messages. The resulting adaptor conforms to the given contract. It is encoded into a particular implementation language (currently BPEL), and it can finally be deployed as a single adaptor [23] or distributed [30] in a set of wrappers over the services.

The goal of this methodology is to generate correct adaptors which are: i) secure, i.e., they enforce the security policies expressed in the security adaptation contract and security-enabled services; ii) non-intrusive, since adaptors do not alter the internal behavior of the services and, in fact, services can be oblivious to adaptation; and iii) transparent, because adaptors should support every possible interaction which complies with the security policies of the contract and services and do not cause deadlock or livelock situations.

It is worth observing that the whole methodology is supported by our toolbox called ITACA [9], which is enhanced with hierarchical adaptation, simulation and verification capabilities over the entire composition. ITACA supports behavioral adaptation but it is yet to be extended with security concerns.

### 3.2.3 Conclusions

The approach presented in this section contributes to this deliverable by providing a software adaptation methodology with respect to security. Specifically, this approach performs the adaptation of Web Services with stateful behavior and security requirements in order to make them cooperate within an orchestration while overcoming their initial incompatibilities at signature, behavior and security QoS levels. More details about the adaptation process can be found in [20]. That paper presents the contribution described in this section, where is explained how is achieved the adaptation through *security adaptation contracts* which allow the security requirements of the services to be concisely represented and managed, where services can be specified in BPEL processes, WF workflows or UML designs. The formal model applied to security adaptation contracts is based on a set of basic security primitives that support a wide range of security

\(^1\)See [6] for a discussion about the negotiation of adaptation contracts among behavioral services without security.
protocols and can be analyzed by automatic cryptographic protocol verifiers. Security adaptation contracts allow us to combine several WS-* security specifications in a single abstract notation and, at the same time, to solve possible incompatibilities among services. Such incompatibilities are common due to the tight coupling of WS-Security enabled services and their security restrictions.
4 Connection with other workpackages

As illustrated in Figure 4.1, work in WP7 has several relationships with work developed in workpackages 6, 8 and 9. WP6 focuses on the definition and formalization of security requirements. WP7 contributions use input from WP6 in the following way: requirements captured in use cases, access control and privacy policies are integrated in detailed UML design models; architecture patterns used to drive deployment and runtime architecture reconfiguration take access control policies into account. WP7 also relies on programming environments for secure and composable services developed in WP8. In particular, the synthesis of secured adaptors relies on security interceptors and the runtime reconfiguration of architecture models is based on a reflexive middleware. WP7 activities relate to WP9 through the use of model checking for the synthesis of adaptors, and runtime verification for runtime reconfiguration.

Figure 4.1: Relationships from WP7 to other WPs
5 Conclusion

The different activities that have been going on in WP7 during the first year of NESSoS have enabled the participants to agree on a common understanding of the different roles that models can play for security in Future Internet applications. We have identified two main roles: design and adaptation. These different activities address various security concerns (access control, privacy, communication) as well as different tasks (code generation, analysis, verification, architecture synthesis).

The UML-based design of secure web applications focuses on two complementary approaches:

- **ActionGUI** provides three modeling formalisms to model data, security and GUI as well as an automatic model transformation that generates a security-aware GUI model.

- **UWE** provides an environment based on a UML profile for the early integration of security concerns in data and navigation models of rich internet applications.

Because of the dynamic and open nature of Future Internet applications, model-driven engineering techniques developed in the NESSoS network of excellence have to consider the evolution and adaption of security concerns. WP7 focuses on two complementary directions for this problem:

- **models@runtime** for the evolution and the runtime reconfiguration of access control policies

- **contract-based adaptation** for the secured service composition

The initial results synthesized in this report are in line with the time frame established in the DoW. All partners have achieved a clear representation of their contributions and have started establishing connections for integration. As expected, there are also many research challenges that still need to be addressed in future years:

- The integration UML-based model-driven security approaches requires mapping the specific languages used by ActionGUI and UWE. While this may be simple for some concerns (like security), it will require complex transformation rules (in fact, refinement rules) for other concerns (like navigation and presentation) since the corresponding models are specified within each approach at different abstraction levels.

- Runtime adaptation of RBAC policies is still at its early stages and needs to be developed in the next years. One major challenges comes from the need to integrate different technologies that are not usually mixed: access control policy and analysis, model transformations and compositional software adaptation. The other challenge for this activity will be to integrate space and time in access control policy specification language, verification and runtime enforcement.

- Service adaptation by contract is a sound and safe mechanism to integrate secured services in global orchestrations. However, the current techniques presented in this report operate off-line. Thus the major challenge that will tackled in the next years is the efficient adaptation of contract-based adaptation techniques for runtime synthesis of adaptors.
Bibliography


