

Towards Precise and Certified Security Patterns

Antonio Maña, Daniel Serrano
{amg,serrano}@lcc.uma.es
Computer Science Department
Univ. of Malaga, 29071 Malaga, Spain

Athanasios-Dimitrios Sotiriou
d.sotiriou@atc.gr
Athens Technology Center (ATC)
Rizariou 10, 15233 Athens, Greece

***Abstract.** This paper presents a new application development process for ambient intelligence scenarios that considers security and dependability issues as core elements. It centres on the precise description of reusable security and dependability solutions (S&D solutions) stored in the form of S&D patterns. One of the main motivations of this approach is the practical impossibility for programmers to develop sound complex security services for their applications.*

1. Introduction

The technologies and characteristics of the emerging adaptive, distributed and heterogeneous computing paradigms such as ubiquitous computing, ambient intelligence, and autonomic computing, introduce new challenges, especially from the security and dependability points of view, for the development of software applications. These emerging paradigms need sophisticated approaches to help determine the security requirements and provide adequate security solutions. One of the key issues is to develop applications that are able to deal with different dynamically-changing contexts. Context awareness and adaptation is one of the cornerstones of future ambient intelligence applications. Once an application is running, changing from one context to another usually implies reconsidering security requirements and selecting appropriate solutions.

This paper presents a new application development process for ambient intelligence scenarios. Our approach considers security and dependability issues as core elements of the software development process. It centres on the precise description of reusable security and dependability solutions (S&D solutions) stored in the form of S&D patterns. One of the main motivations of this approach is the practical impossibility for programmers to develop sound complex security services for their applications. As the development of security software is a highly specialized task, programmers are usually not able to satisfactorily address all the S&D requirements. Applications would benefit from improved security if these programmers could rely on S&D solutions that have been tested, certified and precisely described. In the development process we propose, a number of S&D Solutions are provided to the programmers through corresponding S&D libraries. Each S&D solution is developed at different levels of abstraction, ranging from very abstract levels, where only interfaces and security properties are specified, to low level, where specific implementations of a solution are provided.

The rest of the paper is organized as follows. The next section provides a detailed description of our concept of S&D solutions and how these are captured through

the concept of S&D patterns. Section 3 presents the main ideas of this paper, introducing our development process, including in particular, the selection of patterns from the S&D library. Finally, we summarize our work and provide the readers with some conclusions.

2. Capturing security expertise through S&D Patterns

Our approach proposes the use of a library of precisely-described and formally verified security solutions that programmers can search to access available security and dependability solutions. In this paper we will refer to these solutions as *S&D Solutions*, defined as well-defined mechanisms that provide one or more S&D Properties such as confidentiality, integrity, availability, etc. In order to appropriately represent these solutions we have developed the following artefacts: *S&D Classes*, *S&D Patterns*, *Integration Schemes*, *S&D Implementations* and *Executable Implementations*. These artefacts represent S&D Solutions using semantic descriptions at different levels of abstraction. The main advantage of using different artefacts, each one covering a different abstraction level, is that it allows us to thoroughly cover the whole application life-cycle including both the development and run-time phases. These artefacts are described in detail below:

S&D Patterns are precise descriptions of abstract S&D Solutions. These descriptions must contain all the information necessary for the selection, instantiation and adaptation, and dynamic application of the solution represented in the S&D Pattern. S&D Patterns are composed by different elements which describe the security pattern's functionalities and how to use them. Since S&D Patterns are one of the main concepts in this paper we provide a description of the data conforming a S&D Pattern.

- The *SolutionDescription* is a brief description of the solution itself.
- The *ProvidedProperties* element connects each S&D Pattern to the S&D Properties provided. One S&D Pattern can provide one or more properties.
- *InterfaceDefinition*. This element describes the interface of the S&D Pattern.

- The *PatternClass* references to the classes where the pattern belongs. The description of the S&D Class artefact can be found below in this section. All S&D Patterns belongs at least to one S&D Class. Since the S&D Pattern interface has to conform to the S&D Class interface, the PatternClass element includes a description of how to adapt the S&D Pattern interface to the S&D Class interface, this sub element is called *ClassAdaptor*.
- The *Preconditions* element describes the conditions under which the S&D Pattern is applicable.
- The *Parameters* element describes some parameters that could be used for pattern configuration at run-time.
- The *PartDescription* element is used to describe external components, either software or hardware, that the S&D Patterns needs to work properly.
- *MonitoringRules*. This element describes external monitoring mechanisms to be fulfilled during each S&D Pattern use.
- The *TestsPerformed* element specifies the proofs that have been applied in order to prove the S&D Properties that this S&D Pattern provides.

S&D Classes represent abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and complying with a common interface. This is one of the most interesting artefacts to be used at development time by system developers. The main purpose of introducing this artefact is to facilitate the dynamic substitution of the S&D solutions at run-time while facilitating the development process. At run-time all S&D Patterns (and their respective S&D Implementations, see below) belonging to the same S&D Class will be selectable by the SERENITY Run-time Framework automatically.

S&D Implementations represent the components that realize the S&D Solutions. S&D Implementations are not real implementations but their representation/description. These components are made accessible to applications throughout the SERENITY Run-time Framework. An S&D implementation describes an implementation of an S&D Pattern and as thus a S&D Pattern may have more than one S&D Implementation.

Finally *Executable Implementations* are the actual implementations of the S&D Implementations. These elements are not used at development time, but they are the realization of the selected S&D Solution during run-time. An Executable Implementation works as a stand alone executable S&D Solution ready to provide its services to applications. At run-time the SERENITY Run-time Framework is able to activate, deactivate and adapt Executable Implementations on the basis of the S&D Classes or S&D Patterns that the applications request and the current context. Figure 1 presents a class diagram showing the relationships between all aforementioned artefacts.

There are two relevant points regarding S&D Patterns that deserve to be further explained:

On the one hand, there is a special type of S&D Solution called Integration Scheme. An Integration Scheme is a S&D Solution at the same level as S&D Patterns. These artefacts represent S&D Solutions that are built by combining other S&D Patterns. At application development time, Integration Schemes are used in the same way as S&D Patterns. They differ however in their development process, because S&D Patterns that are created to represent monolithic isolated S&D Solutions need to include all details of such solutions while Integration Schemes include only the integration-relevant start with the selection of several S&D Patterns that must be combined. As thus, in this paper we do not cover the development process of S&D Solutions but only their usage. For this reason, during the rest of this paper we use the notion of S&D Patterns to refer to S&D Patterns and Integration Schemes indistinctly.

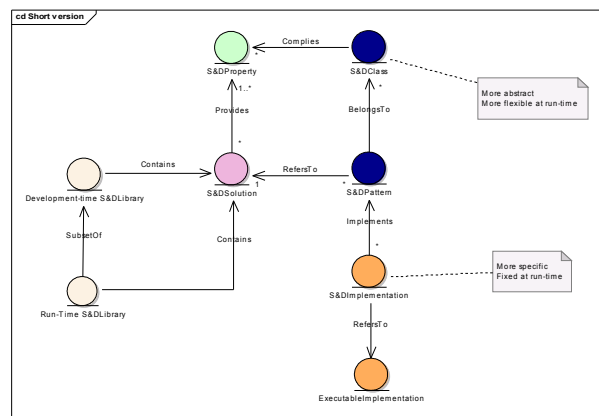


Figure 1: Conceptual model representing security solutions at different abstraction levels

On the other hand, each S&D Pattern provides different functionalities depending on the role the S&D Pattern plays in an application. Therefore, the description of the S&D Pattern interface indicates which services are available for each of the possible roles of an S&D Pattern. This fact is particularly important taking into account that many S&D Solutions can be represented as protocols that describe the communication between several parties. However, due to the fact that it is also frequent that every party is playing a different role, S&D Patterns provide means for using the same solution from the different point of views (roles). This approach presents several advantages including: ease of use (because developers do not need to deal with the interoperability of S&D Patterns), and robust design (because verification is done for all roles encapsulated in the S&D Pattern).

Regarding the security solutions library, our approach foresees two different kinds of libraries. The first one, called *Development-time S&D Library*, is used at development time, and is populated with a very large catalogue of security solutions. These libraries can be public or private and we propose them to be hosted in on-line repositories. During development time,

programmers query these libraries for appropriate S&D Solutions, either through stand-alone applications or by accessing the corresponding web service. The second one, called *Run-time S&D Library*, is a smaller and more specific S&D Library, used to manage the security solutions installed in a system or device during run-time. This run-time library contains the artefacts that represent the S&D solutions that can be used by the applications running in the system or device. The system administrator is responsible for updating the run-time S&D Library during the installation of new solutions or applications in the system.

Our current implementation of the development-time S&D Library is based on a relational database running on MySQL. The different artefacts presented in the previous figure have been mapped to tables and relations between them. Figure 2 provides a view of the tables and relations used in the implementation of this library.

Our S&D library provides two alternative access methods: On the one hand, we have developed an application that allows users to connect to the repositories in order to search or browse the library. On the other hand, we also provide access through Web Services. The first method allows developers to perform queries in order to select specific patterns based on different properties and in particular, on the security properties provided by the S&D Patterns, as well as visually browse through the catalogue of existing S&D Patterns in the repository, as shown in figure 3.

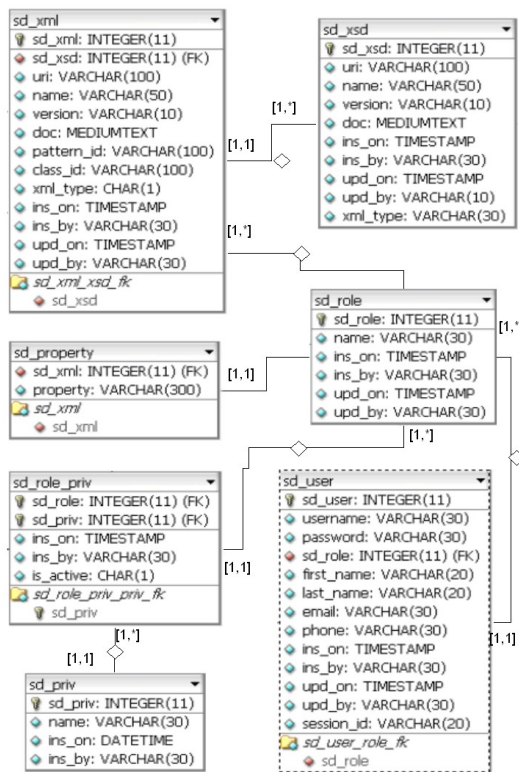


Figure 2: Relationship Diagram of the Development-time Library Database

Web Services allow software agents and software components in general to access the development-time S&D library, thus facilitating the development of new application-development tools and also providing means for querying this library during run-time, which is interesting in some scenarios such as mobile computing.

Since this paper is centred on the use of all these artefacts at development time, we will skip the information regarding run-time procedures and Run-time S&D Libraries. A detailed description of SERENITY run-time processes can be found in [1].

3. Developing applications using S&D patterns

This paper focuses on the mechanisms for search and selection of S&D Solutions at development time. Prior to giving a complete description of the proposed development process, it is worth providing a description of the scenario that we intend to address. This scenario can be described as follows:

- Programmers are developing an ambient intelligence application and they face the challenge of integrating S&D Solutions that can adequately solve their S&D requirements in all possible runtime context. Because of the nature of Ambient Intelligence, these run-time contexts are unpredictable.
- Programmer are not security experts. For instance, they do not have enough expertise about cryptography in order to implement security primitives such as encryption, key management, digital signatures, etc. required for securing a specific communication.
- Fortunately, in our model, the previous drawbacks can be overcome thanks to the existence of well-known libraries of precisely described and formally verified S&D Solutions on-line. They are maintained by different entities, such as security software development companies, open source communities, etc.

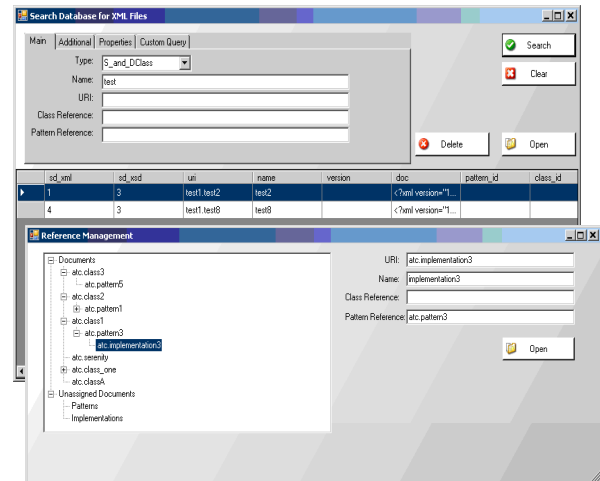


Figure 3: S&D Library GUI

Following our approach, the development process of adaptive secure applications, depicted in the UML activity diagram shown in figure 4, includes the following steps:

Firstly, developers design their application following any existing development process. During this phase the development team must identify all security requirements of the application. For each security requirement identified the developers must analyse which S&D Properties must be fulfilled and the appropriate restrictions and static (i.e. known at development time) context conditions for it.

Once developers have identified all S&D Properties, they can use two development strategies:

- *Fully-delegated S&D*: Developers may fully delegate the provision of the S&D Solutions to the Run-time Framework. Doing this, the SERENITY Run-time Framework, SRF, can apply the most suitable S&D Pattern in each situation. Therefore, developers fix the minimum amount of details for their application to work. These details are the S&D Property required and the interface used to access the services related to the property. For doing this developer have to identify which S&D Class provides the required S&D Properties with a suitable interface. Later, at run-time, the SRF is able to choose amongst all S&D Patterns belonging to the selected S&D Class that was fixed by the developers. If at run-time there are no applicable S&D Pattern found, the application will be informed and, probably, stopped.
- *Partially fixed S&D*: Developers can limit the range of S&D Solutions that can be selected by the SRF at runtime for a particular requirement. In order to do this, developers fix an S&D Pattern or an S&D Implementation for the security requirement at development time. In this scenario, developers have at their disposal development time

S&D Libraries where they can search for S&D Solutions. The result of a query for S&D Solutions matching a particular set of S&D Properties and constraints is a list of applicable S&D Solutions. Of course, in some cases it is possible that the list is empty. It is important to note that the search for a particular S&D Property is very flexible and allows the selection of artefacts providing not only the requested property, but also all other equivalent S&D Properties. For more details on description and reasoning about S&D Properties, the reader can consult reference [3].

If the query is successful, the library returns all S&D Solutions that fit the developer query. That is, those artefacts that can be used to fulfil the requirement according to the constraints expressed by the developer. Moreover, in the case of abstract artefacts (S&D Classes and S&D Patterns) the result will include the constraints that need to be checked at run-time.

The type of artefact selected determines the flexibility allowed to the SRF. If developers select an S&D Class, as already mentioned, the SRF can select among all S&D Implementations that correspond to an S&D Pattern belonging to that S&D Class. If developers select an S&D Pattern, the SRF can select among all S&D Implementations that correspond to that S&D Pattern. Finally, if developers select an S&D Implementation, they do not allow the SRF to do any dynamic selection. However, it is important to note that even in this case there are important advantages of using this approach instead of “hard-coding” the solution in the application. In particular, this approach allows the SRF to monitor the execution of the S&D Implementation and to react to events, for instance, by disabling the S&D Implementation, changing some of its parameters or notifying the user.

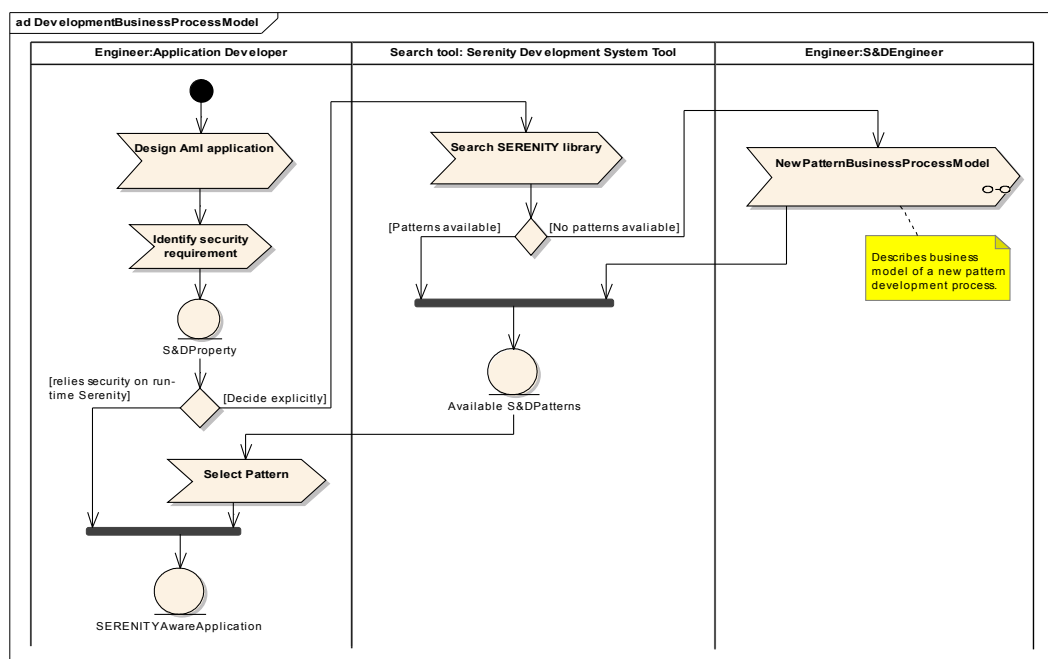


Figure 4: Activity diagram showing application development process

If no artefact is appropriate for fulfilling the query, new artefacts can be created and added to the library. However, this process is out of the scope of this paper. Interested readers can find a detailed description of the S&D artefact development process in [2].

Finally, once the ambient intelligence application is ready, the last step is to deploy it. In order to do a correct deployment, the application must be installed in a device with a SERENITY Run-time Framework. Note that installation of the application includes the installation of all possible artefacts that can be used at run-time, along of course, with the corresponding executable implementations.

The proposed development process model, depicted in Figure 5, integrates the results of the security engineering work into the engineering/development phase of the generic software engineering process. It is at this point where our tools/methods will be used to support developers of secure software. For instance, automated tools will support the developers in the selection of a particular security solution, the adaptation of the solution to the context where it will be finally applied, the analysis of the consequences of changes and the coherence between models at different levels of abstraction.

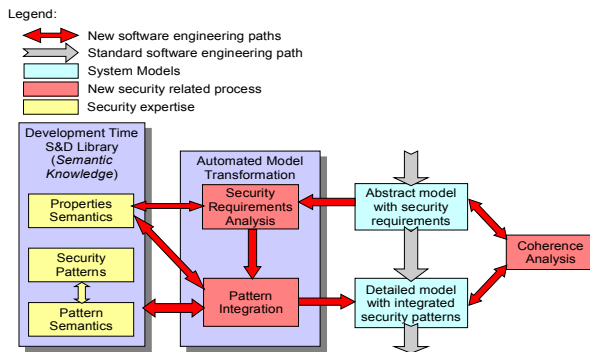


Figure 5: Generic software development process with integrated security engineering

Throughout our proposed development process it is clear that the Development time S&D Library is one of the most important components. The design of this S&D Library involved three main issues:

1. *What and how to store.* This library is able to store different kinds of artefacts (S&D Classes, S&D Patterns, Integration Schemes and S&D Implementations) along with their relationships. Although these elements have different descriptions and need to be stored separately, their relations need to be maintained. From this point of view, the intuition tends to suggest that these elements could be stored as trees, where solutions at the same level of depth in the tree would be described at the same level of abstraction. However, trees are not enough to capture the complex relations between artefacts, mainly because:
 - 1.1. Each S&D Pattern may belong to several S&D Classes, transforming our tree into a graph.

1.2. While Integration Schemes are at the same level with S&D Patterns, they combine several items of this level. This breaks the assumption of the relations between abstraction levels and the tree levels, and also indicates that the relationships will form a graph.

2. *How to search.* Usually, developers will look for different types of artefacts, sometimes requesting S&D Patterns while other times searching for S&D Classes or S&D Implementations. Additionally, they will add some constraints that the artefact must met. Because of this required flexibility, the interface needs to be rich enough to capture all kinds of searches.
3. *Where to search.* We envision that development time S&D Libraries will be available to developers via the Internet (or intranet). Likewise, the SRF will normally search only the local runtime library. However, in some scenarios and for some devices with limited storage capacity such as mobile phones it is quite usual to dynamically download software before execution. For these cases we believe that it is possible to allow the SRF to access a remote development-time library in order to select and dynamically download the artefacts required to fulfil a request. However, this strategy requires the development of suitable security mechanisms for the selection and download procedures.

4. Conclusions

The results of our work will enable the use of a library of security patterns in order to include security related aspects in the software development process. This library is not a simple set of “best practices” or recommendations (mostly at the operational level), like those proposed in the literature as “security patterns”, but a precise, well-defined and automated-processing-enabled repository of security mechanisms. The most relevant feature is the incorporation of rich and precise semantic descriptions of the patterns. The extensive use of semantic descriptions will enable the use of automated reasoning mechanisms capable of solving problems such as pattern composition and adaptation. Patterns will be categorized into distinct abstraction categories, to different types of security measures (technical, organizational, etc.) and to different levels of formalism (e.g. formal patterns for security services that implement formally specified requirements and informal patterns that address security issues in the organizational level).

Acknowledgements

This work has been partially supported by the E.U. through SERENITY project (IST-027587) and by Junta de Castilla La Mancha through the MISTICO-MECHANICS project (PBC06-0082). We want to express our gratitude to Pedro Soria, Ana Piñuela, Domenico Presenza and Francisco Sánchez-Cid for their support and for their contributions to the presented approach.

References

- [1] A. Maña, A. Muñoz, F. Sanchez-Cid, D. Serrano, K. Androutsopoulos, G. Spanoudakis, L. Compagna. SERENITY (IST-27587) A5.D2.1 Patterns and Integration Schemes Languages.
- [2] A. Maña, D. Presentza, A. Piñuela, D. Serrano, P. Soria, D. Sotiriou. SERENITY (IST-27587) A6.D3.1. Specification of SERENITY Architecture.
- [3] A. Maña, G. Pujol. Towards Formal Specification of Abstract Security Properties. Submitted to the The 5th ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code.