Chapter 1
Introduction: The SERENITY vision

Antonio Maña, George Spanoudakis and Spyros Kokolakis

Abstract In this chapter we present an overview of the SERENITY approach. We describe the SERENITY model of secure and dependable applications and show how it addresses the challenge of developing, integrating and dynamically maintaining security and dependability mechanisms in open, dynamic, distributed and heterogeneous computing systems and in particular Ambient Intelligence scenarios. We describe the basic concepts used in the approach and introduce the different processes supported by SERENITY, along with the tools provided.

1.1 A new model of secure and dependable computational ecosystems

Traditionally, Security and Dependability (S&D) Engineers have been faced with complex but static and predictable systems. Existing tools for creation and analysis of S&D solutions are designed for predictable systems, but the emergence of computing paradigms that exhibit high degrees of distribution, heterogeneity and dynamism means that systems are not predictable anymore. In fact, because of the high degree of heterogeneity and the coexistence of many different devices, appli-
cations and users that interact and collaborate in order to achieve their goals, the term computing ecosystem is starting to be common when referring to the systems in these new paradigms.

One especially relevant example of these new paradigms is Ambient Intelligence. Defined by the EC Information Society Technologies Advisory Group (ISTAG), in the vision of Ambient Intelligence people will be surrounded by ubiquitous computers with intelligent and intuitive interfaces embedded in everyday objects around them, making the physical environments adapt and respond to users’ needs in an invisible way in order to provide anytime/anywhere access to information and services.

The most relevant features inherent to the realization of this vision are the increasing decentralization, high heterogeneity (of devices, applications, user needs, capabilities, etc.), dynamism, unpredictability, lack of predefined trust relations and context awareness. In fact, because of the high degree of heterogeneity and the coexistence of many different devices, applications and users that interact and collaborate in order to achieve their goals these systems have been defined as AmI ecosystems.

Due to the high heterogeneity, dynamism and lack of a central control of these ecosystems, it is not possible, even for the most experienced and skilled security engineers, to foresee all possible situations that may arise during the life of the applications in order to create solutions that can be used in these circumstances. Moreover, due to the highly distributed nature of these ecosystems, applications will no longer belong to or be under the control of a single entity, which would force the software engineers to deal with incomplete system descriptions. Therefore, not only devices but also applications must be ready to participate in dynamic collaborations with heterogeneous (in terms of capabilities, functional goals, security and dependability needs, etc.) and non-trusted external elements.

The main consequence of these characteristics is that the provision of S&D in the new computing ecosystems introduces the need for the dynamic application of the expertise of S&D engineers in order to automatically react to unpredictable and ever-changing contexts. This approach takes advantage of recent developments in technologies regarding security engineering, run-time monitoring, semantic description and self-configuration. In the research towards materialising this approach done in the SERENITY project, the concepts of S&D Patterns and Integration Schemes have proven to be very effective as tools to capture this expertise. While S&D Patterns represent independent security mechanisms, Integration Schemes represent solutions for complex S&D requirements achieved by the combination of various S&D mechanisms.

One additional aspect is that the overall security of a system not only depends on the security mechanisms used within the boundaries of a system but also on a variety of external factors including social context and human behaviour, IT environments, and even protection of the physical environment of systems (e.g. buildings). The actual source of security and dependability requirements lies in the real world. Consequently, SERENITY aims at providing solutions to capture these re-
requirements, to trace them and to validate solutions against these requirements. In addition to that, we took especially into account the computing context including heterogeneous communication systems, computing infrastructures and external actors and components dynamically interacting with the system.

Current methodologies and tools for engineering and development of secure applications are not well suited to be used in these new computing scenarios because they are not capable of dealing with characteristics like the heterogeneity, uncertainty and dynamism, lack of a central control, and increased needs for security and dependability.

1.2 Related work

The dynamic provision of appropriate S&D mechanisms for Ambient Intelligence ecosystems, supported both in the software engineering processes and during runtime has not received enough attention in the literature.

However, several approaches have been introduced in order to capture the specialized expertise of security engineers and make it available for automated processing providing the basis for automated synthesis and analysis of the security and dependability solutions of systems [5]. These approaches are supported by a wide range of technologies: components, frameworks, middleware, aspects and security patterns, being this last the most relevant from our work point of view.

The concept of security pattern was introduced to support the system engineer in selecting appropriate security or dependability solutions. However, most security patterns are expressed in textual form, as informal indications on how to solve some particular security problem [6-9]. Some of them do use more precise representations based on UML diagrams, but these patterns do not include sufficient semantic descriptions in order to automate their processing and to extend their use [10].

Perhaps the first and the most valuable contribution as pioneer in security patterns as we know them at present, is the work from Joseph Yoder and Jeffrey Barcalow proposing to adapt the object-oriented solutions to recurring problems of information security [9]. In their own words, seven patterns were presented to be used when dealing with application security. A natural evolution of this work is the proposal presented by Romanosky in [11]. It takes into consideration new questions that arise when securing a networked application.

Going one step down in the abstraction scale, Eduardo B. Fernandez in his work about authorization patterns [10] combines for the first time the idea of multiple architectural levels with the use of design patterns. In [12] they propose the decomposition of the system into hierarchical levels of abstraction.

The same author et al. offer in [13] a good source to study the historical approaches that have been appearing in the scientific literature as pattern systems for security. Wassermann and Cheng present in [14] a revision of most of the patterns
from [9] and [12] and categorise them in terms of their abstraction level. In order to facilitate the reuse of security knowledge, variations of the design pattern template given in [15] are included in order to better suit the presentation of security specific information. In the field of web application protection, [16] is the source of some patterns for Application Firewall and XML Firewall.

The special needs of secure-ware systems and the constantly changing context in which the systems are designed nowadays arise some new problems that have a starting solution in works like the one presented by Cheng et al in [8]. The proposal consist of a template for security patterns that takes into account essential information that has not been necessary in the general design patterns but appears as mandatory in the new security context. On the basis of the security patterns described by Gamma et al. in [15], a new patterns library is created by the addition of new relevant information (e.g. Behaviour or Supported Principles are two new fields to describe the security pattern) and altering some existing fields (e.g. Consequences is altered to convey a new set of possible consequences including confidentiality, usability, integrity, etc.).

Some security patterns have also been proposed for multiagent and Web Services systems. Early efforts in this area were concentrating on the use of the object-oriented paradigm. In [17], for instance, an object oriented access control (OOAC) was firstly introduced as a result of consequently applying the object-oriented paradigm for providing access controls in object and interoperable databases. In [18], Fernandez proposes some specific solutions oriented to web services: a pattern to provide authentication and authorization using Role-based access control (the so-called Security Assertion Coordination pattern) and a pattern for XML Firewalls. The Security Assertion Coordination pattern takes as source the abstract security architecture defined by SAML (the main standard to provide security for web services). However, a minor work has been done in the field of agent-based systems. A good starting point is [19].

Some other authors have proposed ways to provide formal characterizations of patterns. The idea of precisely specifying a given class using class invariants and pre- and post-conditions for characterizing the behaviours of individual methods of the class is well known and provides the basis of the design by contract (DbC) approach [20]. Evolutions of that approach have appeared, notably in [21, 22]. More specifically, Eden et al. proposed to use logic formalism with an associated graphical notation to specify rich structural properties in [21]. Their approach, however, provides only limited support for specifying behavioural properties. The work in [22] had as its main goal to preserve the design integrity of a system during its maintenance and evolution phases. Also in [23] Mikkonen focused on behavioural properties. In his work, data classes model role objects, and guarded actions (in an action system) model the methods of the roles.
1.3 SERENITY Concepts

In SERENITY, the main pillar of building secure and dependable solutions is the enhanced concept of S&D Pattern. An S&D Pattern captures security expertise in a way that, in our view, is more appropriate than other related concepts. As explained in the previous section, components, frameworks, middleware, and patterns have been proposed as means to simplify the design of complex systems and to capture the specialized expertise of security engineers and to make it available for non-expert developers. However, all these approaches have important drawbacks and limitations that hamper their practical use. The approach taken in SERENITY aims at integrating the best of these approaches in order to overcome the problems that have prevented them to succeed individually. Furthermore, because secure interoperability is an essential requisite for the widespread adoption of the SERENITY model, trust mechanisms are an essential aspect of S&D patterns.

SERENITY’s S&D patterns are precise specifications of validated security and dependability mechanisms (referred to as “S&D Solutions” in the SERENITY terminology) materialised as files containing models that can be described using formal or non-formal languages (e.g. XML and logic-based languages) to capture the expertise of security engineers with the objective of being used by automated means. SERENITY S&D patterns include a precise functional description of the mechanisms they represent, references to the S&D properties provided by these mechanisms, constraints about the context that is required for deployment of the mechanisms, specifications of how to adapt and monitor the mechanisms. They also include trust mechanisms, which have a dual purpose; on the one hand, they guarantee the origin and integrity of the descriptions contained in the different SERENITY artefacts; on the other hand they also support the evolution of artefacts and mechanisms. S&D patterns, along with the formal characterisation of their behaviour and semantics, are the basic building blocks of S&D solutions that will enable the provision of S&D over a wide range of heterogeneous AmI ecosystems. SERENITY’s integration schemes represent ways for systematically combining S&D patterns into systems composed of dynamically collaborating elements that operate in mobile, heterogeneous, and highly dynamic ICT infrastructures.

As will be described in other chapters along the book, the concept of S&D Pattern has been materialized in a series of modelling elements that we call S&D Artefacts. The main reason for having different modelling artefacts is that they correspond to different abstraction levels and serve different purposes. Consequently, the representation of S&D Solutions in SERENITY is supported by three main artefacts: S&D Classes, S&D Patterns and S&D Implementations. A fourth element, called ExecutableComponent is part of the SERENITY artefacts, but in this case it is not a modelling artefact but an operational one. Let’s briefly describe each of them now:
• **S&D Patterns** represent abstract S&D solutions. These solutions are well-defined mechanisms that provide one or more S&D Properties. There is a special type of S&D Pattern that represents the combination of several S&D Patterns. This type of S&D Patterns is called Integration Schemes. The popular Needham-Schroeder public key protocol is an example of an S&D solution that can be represented as an S&D Pattern. A special type of S&D Pattern that is used to represent complex S&D Solutions that are realized by combining other S&D Solutions is called Integration Scheme. Although, the concepts are slightly different, the modelling artefact used for both types is the S&D Pattern. One important aspect of the solutions represented as S&D Patterns and Integration Schemes is that they can be statically analysed using S&D Engineering Tools. However, the limitations of the static analysis tools and the assumption that perfect security is not achievable, introduce the need to support the dynamic validation of the behaviour of the described solutions by means of monitoring mechanisms.

• **S&D Classes** represent S&D services (abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and being compatible with a common interface). This artefact is mainly used at development time by the SERENITY Development Tools. An example of an S&D Pattern Class could be a Confidentiality Class, which would define an interface including for instance, an abstract method `SendConfidential(Data, Recipient)`. S&D Patterns that belong to an S&D Class can have different interfaces, but they must describe how these specific interfaces map into the S&D Class interface. The main purpose of introducing this artefact is to facilitate the dynamic substitution of the S&D solutions at runtime. This is a basic pillar to support the idea of the SERENITY model: first, select an abstract definition at development time (i.e. abstract methods from S&D Classes); second, have several patterns complying to this definition (by means of their Class Adaptor); and third, at runtime, the patterns will be selectable and interchangeable because (though having different interfaces) they all comply with the same abstract one. Given that interoperability is a key issue in the scenarios that are the focus of SERENITY, with this approach it is possible to create an application bound to S&D Class, as this artefact defines the high-level interface (i.e. the set of functions, calls, or methods that form the functionality offered by an artefact). Thus, given that S&D artefacts have a reference to the higher level artefact they belong to, it is always possible to track back from an Executable Component to its S&D Class. In conclusion, all S&D Patterns (and their respective S&D Implementations) belonging to an S&D Class will be selectable by the framework at runtime.

• **S&D Implementations** are the representation of operational S&D Solutions, which are in turn called ExecutableComponents. It is important to note that the expression “operational solutions” refers here to any final solution (e.g. component, web service, library, etc.) that has been implemented and tested. These solutions are made accessible to applications thanks to the SERENITY
Runtime Framework (SRF). The description of either a specific dynamic library providing encryption services or a web service providing timestamping services (both including a reference to its corresponding Executable Component), are examples of S&D Implementations. At this stage, it is important to note that the physical implementation (either software or hardware) of an S&D Patterns corresponds to an Executable Component pointed by an S&D Implementation, and not to the S&D Implementation itself. In fact, an S&D Implementation describes not just an implementation of the S&D Solution, but describes an implementation of an S&D Pattern. This means that all S&D Implementations of an S&D Pattern must conform directly to the interface, monitoring capabilities, and any other characteristic described in the S&D Pattern. However, they may have differences, such as the specific context conditions that must be met before applying one specific S&D Implementation, their performance, target platform, programming language or any other feature not fixed at pattern’s level.

The rationale for introducing these three modelling artefacts is based on the following reasons:

- S&D Patterns can be verified using the SERENITY S&D Engineering Tools, while S&D Classes and S&D Implementations cannot. Therefore, it is wise to separate their definitions, since all information referring to the provided properties and the available proofs concern only the abstract solution (i.e. the S&D Pattern) and not the interface (i.e. S&D Class) or the specific implementation (i.e. S&D Implementation).
- S&D Patterns are verified by S&D experts (usually by means of formal methods) while the S&D Implementations are tested by their producers or third parties designated by them. In contrast to the case of S&D Patterns, which will be frequently produced by people who did not created the S&D Solutions described in such S&D Patterns, the creators of S&D Implementations will frequently be the creators of the corresponding Executable Components. Finally, S&D Classes are mainly groups of patterns that provide the same properties and are characterized by their interface definitions. The main goal of S&D Classes is to facilitate application development.
- S&D Classes will be defined by entities mainly interested in interoperability (e.g. industry associations, standardization bodies). S&D Patterns will be produced by independent entities interested in security and dependability (e.g. S&D Companies and Experts, but maybe standardization bodies as well). However, patterns will not only enhance security and dependability, but also interoperability, as all implementations of an S&D Pattern will be required to conform to the pattern specification. Finally S&D Implementations will be produced by entities interested in the creation of working solutions (commercial solution providers, open source communities, etc).
All these artefacts are represented in the following figure, along with their composing elements and their interrelations.

Fig. 1. Modelling elements and their relations

These new concepts can be useful in two different ways: at design/deployment time and at run time. In the first case, we must consider that today’s large applications are built by integrating solutions from different sources and at different levels of abstraction. These applications must face the existence of multiple and heterogeneous user interfaces and access devices and the need to respond to the presence of individuals and the context in an intelligent and invisible way. In this setting, the abovementioned concepts can facilitate the integration of components from different sources in a controlled and secure way, because components will be described by associated S&D Patterns and Integration Schemes that will contain all necessary information in order to allow system engineers to take an informed decision. Furthermore, the SERENITY Development-time Framework (SDF) helps these engineers in the process of searching and browsing a large catalogue of S&D Artefacts, in order to find the most appropriate one to fulfil the S&D Properties required by the
application under development, and to analyse the resulting applications in order to check that the application of the components has been done in a proper way (for instance, an Integration Scheme can be used to ensure that a “digital signature pattern” and an “asymmetric encryption pattern” can be safely used together by taking care that the keys used on each pattern are different) and that compliance to relevant S&D policies is achieved (for instance, by checking that all information related to the company accounting is always transmitted using a solution belonging to the “ConfidentialCommunication” S&D Class).

In the second case (i.e., during runtime) S&D Patterns and Integration Schemes are used in order to support automated adaptation of the S&D mechanisms to the changing context conditions. To achieve this, it is necessary to have a framework supporting the management of a pattern library and the constant evolution of such patterns, taking into account the context in which they are applied. The SERENITY Runtime Framework (SRF) is able to select the most appropriate S&D solution among the ones available in the S&D Library, based on the end-user requirements and the actual context.

The SRF provides a way to secure interconnected and heterogeneous systems because it provides means for systems to interact and collaborate (each SRF provides a negotiation interface) and to manage the selection, control and monitoring of S&D Solutions represented as S&D Artefacts.

As mentioned above, the key to success in these scenarios is based on capturing security expertise in such a way that it can be supported by automated means. In particular, the assumption that S&D experts will not be able to foresee all possible situations for the operation of the system introduces the need to complement the static analysis performed by the security experts with mechanisms for dynamic supervision that can monitor that the conditions established by the S&D experts are fulfilled during the operation of the system. SERENITY provides support for the dynamic supervision and adaptation of the security of systems to the changes in ever-changing ecosystems. In this way, SERENITY’s integral model of S&D considers not only static aspects but also dynamic aspects by means of the monitoring mechanism.

1.4 SERENITY Processes

SERENITY covers all aspects of the development and management of security and dependability components and applications. To achieve such coverage, it includes processes for: (i) the engineering, analysis and characterization of S&D solutions, (ii) the development of applications that incorporate statically or dynamically S&D solutions, (iii) the runtime management of applications that incorporate SERENITY S&D solutions; and (iv) the management of wider organization security and dependability issues by means of S&D Properties and S&D Policies. The main characteristics of these properties are overviewed below, whilst a more de-
tailed account of their methodological and technical aspects is provided in the subsequent chapters of the book.

1.4.1 S&D Solution engineering and development

Regarding the lifecycle of S&D Solutions, SERENITY addresses both (i) the creation of new solutions and their characterization as S&D Patterns; and (ii) the description of their real executable implementations (i.e. Executable Components) as S&D Implementations.

In the first place S&D engineers design solutions ranging from cryptographic primitives, to protocols, to complete mechanisms, and up to workflows and organizational structures. These solutions are then analysed in order to gain knowledge about them, which is finally captured in S&D Artefacts. Therefore, S&D Artefacts contain representations of the knowledge that S&D experts have about these solutions. This process is the missing link in the security engineering chain. Security solutions are normally thoroughly analysed and tested. Different techniques such as formal methods, code inspection, etc. provide rich knowledge about the solutions to those performing those analysis activities. However, after all that effort, the results of the analysis are rarely made available in a carefully organized, precise, complete and usable way. On the contrary, we tend to make the results available as textual descriptions, by means of technical papers or reports. The consequence of this common practice is that a large part of the security expertise is lost, hidden in the minds of the analysts or in difficult to find and even unreachable documents. The concept of Security Pattern [9, 11, 12] has been an important step towards the solution of this situation. Unfortunately, the formats and description means for security patterns are still too varied. Furthermore, with all previous approaches the security expertise is captured in formats that are not suitable for being exploited by automated means. In SERENITY a framework for the precise description of security solutions covering both the needs of software engineers during application development and system administrators is provided. Moreover, our solution is specifically designed to support the automated management of security solutions at runtime.

On the other hand, once solutions have been developed and characterized by means of S&D Artefacts, real implementations have to be produced. It is worth noting at this point that Executable Components are not just implementations of the S&D Solution, but more precisely implementations of S&D Patterns. This means that all Executable Components of an S&D Pattern must conform directly to the interface, monitoring capabilities, and any other characteristic described in the S&D Pattern. Of course, they may have differences, such as the specific context conditions that must be met before applying one specific Executable Component, their performance, target platform, programming language, or any other feature not fixed at pattern’s level. These specific characteristics are captured in S&D
Implementations. Therefore, an existing realization of an S&D Solution such as the OpenSSL implementation of the SSL protocol does not qualify “as is” as a SERENITY Executable Component of the SSL_Channel.umaker S&D Pattern. The main reason is that every Executable Component has to implement specific interfaces to interact with the SRF and with the application using it. Furthermore, all SERENITY Executable Components must provide support to monitoring by providing specific event capturers defined in the S&D Pattern specification. However, SERENITY provides an API that facilitates the creation of Executable Components. In most cases, an existing implementation of an S&D Solution can be transformed into an Executable Component simply by constructing a wrapper around it.

1.4.2 Application development

In addition, SERENITY supports the application development process assisting developers in the selection and use of the most appropriate solution fulfilling their requirements. The SERENITY application development process is compatible with most of the current software and system engineering processes. Therefore, it does not require application developers to change their current development practices. The SERENITY process only assumption on the underlying development process is that the system developers know the security and dependability requirements of their applications and can relate these to S&D Properties.

This process focuses on the mechanisms for search and selection of S&D Solutions at development time. Prior to giving a complete description of the proposed 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 application and they face the challenge of integrating S&D Solutions that can adequately solve their S&D requirements in every possible run-time context.
- Programmers are not security experts. For instance, they lack enough expertise about cryptography in order to implement security primitives such as encryption, key management, etc. required for securing a specific communication.
- Fortunately, in our model, the previous drawbacks can be overcome thanks to the existence of libraries of well-defined and formally verified S&D Solutions online. They are maintained by different entities, such as security software development companies, open source communities, etc.

Following our approach, the development process of adaptive secure applications, 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 devel-
opers 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 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, developers 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 Executable Components of all S&D Patterns belonging to the selected S&D Class that was fixed by the developers. If at runtime there is no applicable S&D Pattern found, the application will be informed so it can react appropriately.

- **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 Artefacts. 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 [16]. If the query is successful, the library returns all S&D Artefacts 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 aforementioned, the SRF can select amongst all Executable Components (represented by S&D Implementations) that correspond to any S&D Pattern belonging to that S&D Class. If developers select an S&D Pattern, the SRF can select amongst all Executable Components 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 changing some of its parameters, or by notify-
ing the user.

The result of the application development process is an application that con-
tains calls to Executable Components that are unknown at development time. This
is possible thanks to the services offered by the SRF for selection and activation of
appropriate Executable Components.

SERENITY provides an API that facilitates the creation of SERENITY-aware
applications. This API provides the necessary support for applications to interact
with the SRF and with Executable Components. For instance, the API provides
means for an application to request an Executable Component from the SRF, to
access the Executable Component’s operations and to receive notifications from
the SRF about the functioning of the Executable Component.

1.4.3 Runtime management

The runtime management process of SERENITY is concerned with the configura-
tion, automated dynamic selection, use and control of S&D Implementations, ac-
cording to the requirements of applications and the context conditions which arise
during their operation.

After an application is developed, the next step is to deploy it. To ensure cor-
rect deployment, an application must be installed on a platform supported by the
SERENITY Runtime Framework (SRF). Note that installation of the application
includes the installation in the SRF Library of all artefacts describing solutions
that could be selected at run-time, along of course, with the corresponding Exec-
utable Components.

Once the application has been created and deployed in the target environment,
it can interact with the SERENITY Runtime Framework. The interaction between
the SERENITY-aware application and the SRF starts when the former sends a re-
quest for an Executable Component to the SRF. Of course, the request can follow
one of the development strategies described in section 1.4.2.

The SRF supervises the context of the system and controls the solutions that are
used to provide S&D services to applications. On the one hand, in order to select
the most appropriate solution to fulfill a request by an application, the SRF first
identifies those solutions (Executable Components) that conform to such request.
Then, for each of the possible Executable Components, the SRF analyses the its
preconditions, which are contained in the different SERENITY artefacts related to
that Executable Component, and refer to characteristics of the target system, con-
text and other conditions necessary for the correct deployment and operation of
the Executable Component. Once an Executable Component has been selected to
fulfill the application request, it is activated (loaded in memory and initialized),
and its handler is returned to the application. From this point onwards, the Execut-
An Executable Component is effectively linked to the application, which can access the services provided by such Executable Component.

On the other hand, after an Executable Component has been selected and activated by the SRF, the context and other conditions that need to be checked at runtime are also specified within the S&D Patterns and S&D Implementations that have been activated for the application. Following the activation of an S&D Pattern and S&D Implementation and the linking of an Executable Component to an application, the SRF also starts the process of monitoring the deployed solution. The monitoring process is based on monitoring rules, which are specified in the S&D Patterns that have been activated for the application. These monitoring rules are extracted by the SRF and sent to a monitoring service. This service starts immediately a continuous check of the submitted rules and keeps a record of any violations that it detects for them. The checks are based on events that the SRF collects from different sources. Typically, these events are generated and sent to the SRF by the Executable Components that have been activated for the given system or by some event captor which is part of the deployment infrastructure of a component (e.g. in cases where an executable component is a web-service communicating with its clients via the SOAP protocol, events that represent invocations and responses from the service may be captured by a SOAP message captor embedded in the server within which the service is deployed).

To obtain the violations of the monitoring rules that it has sent to the monitoring service, the SRF polls this service periodically and extracts descriptions of any violations that may have occurred as well as diagnostic information about them (the latter type of information indicates the likely causes of the violation). The reaction to violations of the monitoring rules depends on actions that have been specified for them within S&D Patterns. In many instances, the reaction to a violation will be the immediate cease of the execution of the Executable Component that has been activated to realize the pattern. Other actions are however possible including for example, the dispatch of notifications to the administrator of the SRF, the application (see Chapter D3 for more details). The realization of the actions associated with violations may be the responsibility of the SRF, the application, or a human user. When the conditions associated with a particular implementation of an S&D Pattern are violated, for example, but the conditions associated with the pattern itself are satisfied, the SRF can try to find alternative S&D Implementation and Executable Components for the pattern and activate them. However, in cases where the conditions of the S&D Pattern are violated and the SRF is unable to find alternatives the operation of the application may need to be aborted following or not confirmation of the application’s user. In scenarios in which dependability and continuous operation is essential, developers and SRF authorities must ensure the availability of complementary solutions that can cover all context conditions. In particular, “manual operation” solutions can be described using the SERENITY artefacts, with the advantage over purely manual solutions that the system can still supervise (by means of the monitoring capabilities) the operation of such solutions and advise operators about how to proceed.
1.4.4 Managing S&D Properties and Policies

The last of the SERENITY processes is related to the definition of S&D Properties and S&D Policies. The definition of S&D Properties is based on the assumption that there will be different interpretations of the concepts that are the core of the S&D Properties, which in turn requires the ability to allow different definitions that express subtle differences of interpretation. There are many reasons supporting this assumption; for instance, legislation, cultural differences etc.

Regarding the development-time phase, we assume that SERENITY-aware system developers have identified and selected the S&D Properties that their applications must fulfil. These S&D Properties result in a set of S&D Solutions to be applied in the target context. This set of S&D Solutions is composed by those S&D Artefacts explicitly related to the requested properties.

In order to provide the developer not only with the exact results matching her queries, but also with those results that, though not explicitly related, implicitly provide the same S&D Properties, reasoning mechanisms have been defined. For instance, given an S&D Property \( p \), it is desirable to find those S&D Properties (e.g. \( p', p'' \)) that semantically imply \( p \) as well as those S&D Properties that considered jointly (e.g. \( p' \land p'' \)) imply \( p \). Following this process, not only those solutions providing \( p \) will be taken into consideration but also those solutions providing \( p', p'' \) and \( p' \land p'' \).

This approach expands the universe of possible solutions to include those catalogued as semantically implied.

The same reasoning holds for S&D Policies, which in SERENITY are defined as a set of rules specifying the application of different S&D Properties to different system elements. Based on the property reasoning mechanisms, we can also define reasoning mechanisms for S&D Policies, which would allow the automated verification of their relations. By verifying that an S&D Policy \( P_1 \) implies another policy \( P_2 \), we demonstrate that \( P_2 \) complies with \( P_1 \). If we think of \( P_1 \) as a policy corresponding to a regulation (e.g. HIPAA) and \( P_2 \) as the corporate policy of the ACME Company, we have a simple but powerful way to verify compliance.

Moreover, we can also verify that applications comply with a policy. This is achieved by checking that the protection of the computational elements in the application is managed through calls to the SERENITY framework. The checking of the use of these calls can ensure the use of solutions that fulfill specific S&D properties on specific computational elements, which in turn can ensure the compliance with an S&D policy.
1.5 SERENITY tools

SERENITY provides a suite of tools to support the processes described in the previous section. In the following we briefly overview them and a more detailed description can be found in the subsequent chapters of this book.

Solution analysis. There is a set of tools designed to support the analysis of S&D Solutions at different levels of abstraction. These tools are loosely coupled with the rest of SERENITY tools. The main reason is that the output of these tools is knowledge about the solutions, which is precisely the input to the solution description tools. Additionally, the loose-coupling model has the advantage of allowing external tools such as AVISPA or CAPSL to be used for this purpose. It is also possible to use combinations of these tools because they have specific characteristics that facilitate the analysis of different aspects of the studied solution.

Solution description. The tools for describing solutions support S&D Experts in the expression of their knowledge about the S&D Solutions in a way that is suitable for automated processing, but is also useful for humans (application developers).

SERENITY Development-time framework (SDF). The SDF is designed to support application developers in finding appropriate solutions and artefacts to use in their applications. Opposed to the SRF, the SDF accepts requests based on S&D Properties. The SDF contains a large catalogue of S&D artefacts (called SDF S&D Library) enabling developers to choose the best ones to fit the requirements of their applications. One important process performed by the SDF when a request is received is the identification of semantic relations between the S&D Property required by the application and other S&D Properties that can be used to fulfil the application requirement. The Operational S&D Properties Language (OPL) and the corresponding infrastructure of Domain Properties Servers (DPS) explained in Chapter 4 support this process.

Other application development support tools. These tools cover a wide range of objectives that go from low-level tools to conceptual tools. The low-level tools operate at code level and help application developers to manage the calls to the Executable Components providing the S&D services required by the application. At this level of source code, an Eclipse plug-in is provided for this purpose. Additionally, other tools like the SI* tool operate on a higher level allowing the automatic application of S&D patterns to the system model.

S&D Property and S&D Policy management. These mechanisms support the publication and management of S&D Properties and S&D Policies, which are used by other tools like the SDF. In particular, an infrastructure for the semantic interoperability of S&D Properties is provided, which could be considered the backbone of the process described in section 4.4.

Serenity Runtime Framework (SRF). The SRF is the tool that enables the dynamic configuration, binding, monitoring and replacement of S&D mechanisms into applications, thus realising the core of the SERENITY approach. The SRF is
implemented as a service running in a device, on top of the operating system, and
listening to applications requests. Applications send requests in order to fulfil their
security requirements. These requests, once processed by the SRF, are translated
into S&D Solutions that may be used by the application. As a result of the selec-
tion process, the SRF instantiates S&D Patterns or Integration Schemes by means
of Executable Components and makes them accessible to the applications.

SRFs are specific for each device or system. The SRF has a generic architec-
ture that can be valid for the wide variety of target devices where the SRF will
run. From an external point of view, every instance of the SRF provides interfaces
in order to allow interaction with other systems or other SRFs instances. The SRF
provides two main interfaces: On one hand, it provides a negotiation interface,
used to establish the configuration of the interacting SRFs when two applications
supported by different SRFs need to be communicated using the same S&D Solu-
tion. On the other hand, SRFs offer a monitoring interface. External systems inter-
acting with an instance of the SRF will be able to monitor that the behaviour of the
framework and the executable components running in it is correct. With these in-
terfaces, we provide support for the dynamic supervision and adaptation of the
system’s security to the transformations in ever-changing AmI ecosystems.

**Monitoring Framework.** The monitoring framework of SERENITY is called
EVEREST (EVENT RESoning Toolkit). EVEREST provides the rule verification
services that are required in order to inform the processes of selecting, activating
and deactivating S&D mechanisms for applications at runtime. This monitoring
framework is implemented as a service that is accessible to the SRF. As discussed
earlier, the SRF provides EVEREST with the rules that need to be monitored at
runtime for the different S&D Patterns that have been activated and forwards to it
events generated by the different Executable Components that are linked with the
application and/or its operational environment, to enable the rule checks.
EVEREST checks these events against the monitoring rules and records property
violations. The recorded violations are retrieved by the SRF through a periodic
polling process.

EVEREST has been implemented as a generic reasoning engine that can check
rules expressed in a special form of Event Calculus, called EC-Assertion. This en-
gine is capable of reasoning with events coming from distributed sources and
time-stamped on separate time lines. In addition to basic rule checking capabili-
ties, EVEREST can perform diagnostic analysis. The objective of diagnostic anal-
ysis is to identify the possible causes of rule violations. This analysis is based on
abductive and belief based reasoning. Furthermore, EVEREST provides threat de-
tection analysis. The objective of this analysis is the prediction of potential vi-
lations of monitoring rules and the estimation of the likelihood of their occurrence.
Threat detection analysis is required for certain types of monitoring rules where it
is possible to take early action to prevent violations which seem likely but have
not occurred yet.
1.6 Conclusions

SERENITY is addressing the issue of security and dependability in the complex world of Ambient Intelligence. It provides an infrastructure for the development and application of adaptable S&D solutions in dynamic and continuously changing AmI ecosystems. It also provides a set of concepts, methods, processes, and tools that underpin this infrastructure and support S&D experts, S&D engineers, application developers, and system administrators in their tasks.

The benefits of the SERENITY approach have been demonstrated through prototype systems that use the artefacts and infrastructure of SERENITY to solve real S&D problems. The SERENITY artefacts and some of the prototype applications are presented in the following chapters.

The work that has been carried out in SERENITY has already made significant contributions to the dynamic configuration, deployment, monitoring and adaptation of security and dependability solutions in AmI ecosystems, opening a new path of possibilities. In this path, there are also several new challenges that have emerged.

The first such challenge is to bring together companies and academia in an effort to standardise several aspects of SERENITY. Another challenge is to investigate into effective ways of embedding the SERENITY runtime framework in different micro devices with diverge characteristics. The development of elaborated coordination mechanisms between different instances of the SERENITY runtime framework that would enable the delegation of tasks across devices and the possibility of dynamic sharing of S&D mechanisms that may be available in different frameworks is also one of the opened challenges of the work carried out in the project.

References

2. Astalavista.com, One of the world's most popular and comprehensive computer security web sites http://www.astalavista.com/
Future Directions


