Towards Secure Ambient Intelligence Scenarios*

Antonio Maña, Francisco Sanchez-Cid, Daniel Serrano, Antonio Muñoz
Computer Science Department
University of Málaga, 29071 Málaga, Spain

Abstract. Ambient Intelligence (AmI for short) represents a promising concept based on the ubiquity and intelligence of computational devices embedded into all types of objects. The realization of this concept entails many important challenges, such as the provision of intelligent and adaptive interfaces, the dynamic assembly of components, the automated negotiation, and many others. However, one of the most important barriers to the realization of Ambient Intelligence is the lack of adequate support for security. In particular, the lack of security-aware development processes is a very negative factor influencing this situation. In this paper we present a new model for addressing the security issues in the development of distributed applications and systems for very dynamic and heterogeneous scenarios such as Ambient Intelligence. The model is based on the concept of Security and Dependability (S&D) Pattern. S&D Patterns are precise, machine-understandable and user-meaningful representations of validated security and dependability solutions and mechanisms.

I. INTRODUCTION

Security is becoming one of the most critical concerns for the companies developing software in these days. New vulnerabilities and security bugs are published everyday in specialized web pages as Security Focus [1] or Astalavista [2], and the tendency is towards an increment on the number of reported vulnerabilities. The trend towards open environments, the crescent number of devices working online and connected to some kind of network, and the demand of mobile technology to connect from anywhere at anytime, has caused the apparition of new threats and new security concerns. Attacks based on buffer overflow, SQL injection, or DNS Spoofing have been identified and largely discussed by the security community, but those attacks still as the main breaches on security for the current applications. Designers continue making the same mistakes, permitting the same attacks to be successful again and again, as similar security breaches continue unsolved.

During this time, Software Engineering techniques have advanced providing more automated means to allow us developing more correct software. Unfortunately, the development of correct software does not imply the development of secure software. It is not enough providing the desired functionality: the intended functionality (and nothing more) has to be provided, assuring the absence of hidden side effects. It is proved that little bugs that do not affect in a visible way the functionality of the application are seeds for the grown of a future security breach. Therefore, when the Software Engineer is designing a new application he has to keep in mind the chain effects of the possible failures of the application as well as the hidden effects of some intended functionality. For example, in an old version of a popular email client, an email with a very long file name attachment (e.g. more than 300 characters) caused a crash when trying to download the message from the email server, causing the same error every time it tried to download mail from this account [3]. The system was properly designed, and the usual attachments were managed properly, but a malicious attacker using the attachment feature was able to make the system complete unusable, breaking the availability of the system.

The Software Engineer also has to take into account the possible external security threats or those coming from the interaction with other foreseen or unforeseen elements of the system. In the new environments, the concepts of system and application as we know them nowadays will disappear, evolving from static architectures with well-defined pieces of hardware, software, communication links, limits and owners, to architectures that will be sensitive, adaptive, context-aware and responsive to users’ needs and habits. These ecosystems (such as Ambient Intelligence –AmI) will offer highly distributed dynamic services in environments that will be heterogeneous, large scale and nomadic, where computing nodes will be omnipresent and communications infrastructures will be dynamically assembled. The combination of heterogeneity, mobility, dynamism, sheer number of devices, along with the growing demands for security, is going to make the provision of security for these ecosystems increasingly difficult to achieve with existing security solutions, engineering approaches and tools.

The conclusion of those facts is that the average Software Engineer has to be assisted in his lack of necessary knowledge on security to analyse the threats of the system and expertise to apply sound solutions to avoid those threats, in order to improve and secure the final design.

From the late 90’s, the security research community has proposed several approaches based on security patterns to bridge the gap on security knowledge between the average Software Engineers and the Security Engineers. Security patterns describe a proved solution to some common security problems that arise in specific scenarios. These patterns represent a standard mechanism to design more secure systems as they embed the knowledge and the expertise of Security Engineers ready to be applied on the new application designs [4]. Nevertheless, far from being simple, the application of these well-proven generic schemes requires some experience on the application of design patterns as well as a previous analysis of the potential threats faced by the system. Furthermore, each pattern has some constraints and consequences that arise when it is applied. Therefore, it may require the simultaneous application of some other security pattern or significant changes in the design of the system.

* Work partially supported by the E.U. through project IST-027587
Furthermore, dynamic environments (such as AmI) create the necessity of building systems capable of adapting their security mechanisms to the constantly changing context. Then, the semantic description of the patterns, including their side effects when applied and their exact relationship with other patterns of the system, has to be considered and monitored for correctness using information extracted from the problem and the evolving context. The inclusion of the SERENITY framework that we propose in the software engineering process will allow designers not to secure systems, but to build secure systems.

II. RELATED WORK

Currently, the provision of appropriate security and dependability (S&D) mechanisms, and its integration into the software engineering procedures, for Ambient Intelligence ecosystems remains an unsolved issue.

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 Romanovsky 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. offers 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 categorize 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 an 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 field 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. From the very beginning, the tendency has been to use the object oriented paradigm: in [17] 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. Fernandez proposes in [18] 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 start 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 is the basis of the design by contract (DbC) [20]. Evolutions of that approach have appeared and [21, 22] are some examples of that. Eden et al. [21] proposes to use logic formalism with an associated graphical notation to specify rich structural properties. However, it provides only limited support for specifying behavioural properties. In this sense, [22] has as main goal to preserve the design integrity of a system during its maintenance and evolution phases. Also Mikkonen in [23] focus his approach on behavioural properties. Here, data classes model role objects, and guarded actions (in an action system) model the methods of the roles.

III. THE SERENITY APPROACH

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.

The provision of S&D in AmI ecosystems requires the dynamic application of the expertise of security engineers in order to dynamically react to unpredictable and ever-changing contexts. Due to the high heterogeneity, dynamism and decentralization of AmI 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. Therefore, devices and 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.

In order to cope with the specific needs of these environments, a possible approach could be to try to create an “intelligent” system able to analyze the requirements and the context in order to synthesize new solutions. Unfortunately, given the state of the art in both security engineering and intelligent systems, this approach is not a promising one in the foreseeable future. On the other hand, a more realistic approach which can take advantage of recent developments in technologies regarding security engineering, run-time monitoring, semantic description and self-configuration is based in capturing the expertise of security engineers and making it available, by means of automated tools, to the dynamic ecosystems. It is our belief that this approach, supported by the enhanced concept of S&D Patterns and Integration Schemes provide the best means to capture this expertise. While S&D Patterns describe independent security mechanisms, Integration Schemes describe 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 requirements, to trace them and to validate solutions against these requirements. In addition to that, our proposal takes especially into account the computing context including heterogeneous communication systems, computing infrastructures and external actors and components dynamically interacting with the system.

In SERENITY, the main pillar to build solutions is the enhanced concept of S&D Pattern, which can capture security expertise in a way that is more appropriate than other related concepts. 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 will be provided for patterns.

SERENITY’s S&D patterns are precise specifications of validated security mechanisms materialised as files containing models that could be described using formal or non-formal languages (e.g. XML and logic-based language) to capture the expertise of security engineers with the objective of being used by automated means. S&D patterns include a precise behavioural description, references to the S&D properties provided, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. S&D patterns, along with the formal characterisation of their behaviour and semantics, are the basic building blocks of S&D mechanisms that will enable the provision of S&D over a wide range of heterogeneous AmI ecosystems. SERENITY’s integration schemes specify ways for systematically combining S&D patterns into systems composed of dynamically collaborating elements that operate in mobile, heterogeneous, and highly dynamic ICT infrastructures.

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 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 scenario, the aforementioned 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, automated tools will help these engineers to analyse the resulting systems in order to check that the application of the components has been done in a proper way. For instance, an Integration Scheme can tell the developers that you can use a “digital signature pattern” and an “asymmetric encryption pattern” together as long as the keys used on each pattern are different.

In the second case (that is; 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 warehouse and the constant evolution of such patterns, taking into account the context in which they are applied. The SERENITY Framework runtime support 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, as shown in figure 1.

The global SERENITY framework provides a way to secure interconnected and heterogeneous systems. It is
composed of: (i) a set of patterns (for security and dependability), (ii) integration schemes (based on the previous set of patterns) describing how to combine patterns in order to achieve the security requirements of the application; and (iii) Deployment & Run-time monitoring mechanism able to: evolve according to new possible solutions and requirements (e.g., changes of legal regulations, privacy, new security breaches, new technologies), and validate in terms of dependencies, integration schemes, patterns, etc.

Figure 1. Global SERENITY framework

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. However, we have assumed that S&D experts will not be able to foresee all possible situations for the operation of the system. Therefore, we 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 will provide 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.

IV. CONCEPTUAL MODEL OF THE SERENITY FRAMEWORK

This section is based on a UML model that describes the essential elements of the SERENITY framework. The logical view of the system is introduced in figure 2, including the different conceptual elements as well as their relations.

We can observe that S&D Patterns and Integration Schemes (S&DPatterns) refer to solutions (S&DSolutions) and contain both: (i) the semantics (S&DSolutionSemantics) that describe the solution and (ii) the semantics that describe the hardware or software components (ComponentSemantics) that take part in the solution (Component). The solution semantics are described in terms of the semantics (S&DPropertySemantics) of the particular properties (S&DProperty) provided by the solution. Solutions can be monitored by using certain monitoring mechanisms (MonitoringMechanisms).

S&D Patterns and Integrations Schemes are certified by a special type of digital certificate (PatternCertificate). The libraries of patterns (S&DLibrary) are composed of S&D Patterns and Integration Schemes that have a certificate, denominated certified patterns (CertifiedS&DPattern).

Finally, users will define the security and dependability requirements (S&DConfiguration) for their systems, which will contain a set of specific requirements (S&DConfigurationElement). Each specific requirement will specify a set of properties (S&DProperty) that must be enforced for a particular element of the system (SystemElement).

SERENITY provides a series of tools for security engineers. These are tools for analysing security solutions, for creating S&D Patterns that represent those solutions, and for certifying the S&D patterns and their corresponding solutions.

The functionality offered by the system for S&D engineers are presented in figure 3. In the process of creating a new pattern, security engineers will probably use external tools (for designing, specifying, classifying, implementing ...) their solutions. SERENITY does not address these tools on purpose, in part because they are not directly related to the project goals, and in part because there are many different lines of research and we consider that all of them can provide some value in specific situations.

Figure 2: Logical model, conceptual elements.

Figure 3: Main functionalities for S&DEngineers.
On the other hand, SERENITY must provide tools for analysing security solutions. There are plenty of these tools available, but in contrast to the previous case, we need to include these tools in our work plan because we want them to produce results that can be later integrated in our framework. Consequently, we are currently in the process of creation and adaptation of tools to produce results that can be represented as S&D Patterns. These tools must address issues that are specific for each abstraction level. Although, each level requires different formalisms, tools and methodologies, they all have in common that the results of the analysis is captured in the form of S&D patterns and Integration Schemes.

With the support of the specification tools, security engineers will be able to capture their specific knowledge about a particular solution and to represent it in the form of S&D Patterns and Integration Schemes (see figure 4). Sometimes they will also need support for the semantic definition of security properties, which is the basis for the definition of the S&D Patterns and Integration Schemes. Finally, security engineers will use trust tools to certify their S&D Patterns and Integration Schemes.

Apart from S&DEngineer, two other actors will use the framework (figure 5). On the one hand, the users of the systems where the framework is running will be able to manage their S&D patterns and Integration schemes and to configure the framework. On the other hand, external S&D controllers (representing any external element, capable of managing the S&D configuration of a node) will be able to access the monitoring services offered by one SERENITY Framework and to negotiate the configuration in order to interact with it. Of course, from the point of view of a specific instance \( F_i \) of the framework, all other instances of the SERENITY framework play the role of external S&D controllers and, reciprocally, all other instances will consider \( F_i \) as an external S&D controller. It is not assumed that external S&D controllers must implement all the functionality contained in the SERENITYFrameworkInterface package in order interact with SERENITY frameworks.

Additionally, the SERENITY framework will be used by applications in order to manage the selection and configuration of their security and dependability mechanisms. These applications will use the SERENITY framework in order to enforce certain S&D properties. In some cases, the enforcement of a specific property may require the monitoring of the behaviour of the solution during runtime.

Every instance of SERENITY framework as system will provide interfaces in order to allow interaction with other systems or other SERENITY instances, the deployment model of figure 6 represents it.

In summary, the SERENITY framework will provide two main interfaces: On the one hand, it will provide a negotiation interface that will be used in order to establish the configuration of the interacting frameworks in a way that suits the requirements from all participants. On the other hand, SERENITY frameworks will offer a monitoring interface. External elements interacting with an instance of the SERENITY framework will be able to monitor that the behaviour of the framework is correct. Of course, the SERENITY framework will also feature the counterparts of these offered interfaces (required interfaces). It is foreseen that some external elements other than SERENITY frameworks will be able to interact with them by way of these interfaces.

Finally, figure 7 shows a simple life cycle for a pattern, including how a new S&D Solution is designed, analyzed, represented in the form of an S&D Pattern, and certified.

The S&D Pattern is then added to an S&D Library, and finally used by the SERENITY framework in order to enforce a specific set of S&D properties (that we will call S&DConfiguration) defined by the end users and their applications.
The conceptual elements in this model are those shown in the logical view. Security Engineer will complete the initial tasks, after of the analysis a new pattern is created, this pattern will be certify. The certified S&D Pattern will be added to a library by a SERENITY Framework User; this user will manage Framework instance patterns and will define a configuration. All this process will establish a SERENITY instance that satisfies security requirements.

V. CONCLUSIONS

The realization of the Ambient Intelligence concept entails many important challenges, but the most important barriers to this realization is the lack of adequate support for security. In this paper we have presented a new model for addressing the security issues in the development of distributed applications and systems for very dynamic and heterogeneous scenarios such as Ambient Intelligence.

Our model is based on the concept of Security and Dependability (S&D) Pattern. S&D Patterns are precise, machine-understandable and user-meaningful representations of validated security and dependability solutions and mechanisms. We have introduced the basic elements of the approach and have described the conceptual model of our solution.

We are currently in the process of implementing these concepts in the ongoing SERENITY project. Our interest focuses on the realization of the concept of S&D Pattern, the semantic definition of security and dependability properties in order to guarantee the interoperability of S&D Patterns from different sources, and the provision of a trust infrastructure for the patterns.

REFERENCES