SECURITY PATTERNS, TOWARDS A FURTHER LEVEL[*]

Beatriz Gallego-Nicasio
ATOS Origin

Antonio Muñoz, Antonio Maña, Daniel Serrano
Grupo GISUM, Universidad de Malaga, Malaga, Spain

Keywords: Security patterns, Ambient intelligence, Security and dependability services, Security measurements.

Abstract: Traditionally, security patterns have successfully been used to describe security and dependability (S&D) solutions, making them available to system engineers not being security experts. Recently, in the SERENITY research project, the notion of S&D pattern was extended to exact specifications of re-usable S&D mechanisms for Ambient Intelligence (AmI) systems. SERENITY is focused in AmI systems, but its results can be applied to other computer paradigms: grids, distributed computing, etc. SERENITY S&D patterns include information about both the S&D properties that the solution satisfies and the requirements on the context conditions. Along this paper, we describe how abstract S&D solutions can be implemented by means of functional S&D services. In order to do that, our approach is based on the use of SERENITY S&D patterns, and their implementations, called ExecutableComponents. Finally, we propose several examples and we prove their potential application to AmI scenarios.

1 INTRODUCTION

Traditionally, security engineers' tasks, in standard software and service development, have been the definition of security principles and policies, and the development and validation of solutions for achieving specific security properties. Due to the nature of the systems under development (usually, centralized or loosely connected systems) there was no need of a high integration of these tasks in the development process. Nowadays the provision of security and dependability (S&D) for information systems requires specialized expertise in the S&D engineering techniques. Indeed, the current S&D engineering techniques are able to warrant high levels of security and dependability, when they are rigorously and appropriately applied. Unfortunately, these techniques are based on a detailed analysis of every part of the system, but not as a standalone. Therefore, they do not work well when they are applied to highly dynamic systems, where it is impossible for S&D engineers to foresee all possible configurations of the system that will happen at run-time. In this paper, we present an overview of how security expertise is captured by means of the Serenity Project S&DPatterns. We also introduce the structure of these S&DPatterns at the implementation level: applications use S&DPatterns at run-time, consequently S&DPatterns are implemented by means of software and hardware components, called ExecutableComponents. Secondly, we introduce the Serenity Run-time Framework, which is a service providing support to secure applications at run-time by means of S&DPatterns. And to finish, we present how developers profit from all the advantages introduced by Serenity, by introducing the two Java packages developed to support the development of both ExecutableComponents and secure applications. The paper is structured as follows: section 2 compiles relevant related works, section 3 introduces Serenity artifacts used to capture security knowledge, section 4 presents the Serenity Run-time Framework and goes into detail describing the exploitation and deployment of S&D solutions at run-time. Section 5, presents the monitoring and reaction mechanisms to control the execution the S&D solutions. Section 6 shows how these concepts are implemented by means of ExecutableComponents and presents the supporting infrastructure created. An example scenario to demonstrate all these concepts is described in section 7. Finally, section 8 presents the current state and next

[*]Work partially supported by E.U. through project Serenity (IST-027587)

steps of the research, together with our conclusions.

2 RELATED WORK

Concerning the problem of providing adaptable security services at run-time, we found proposals based on Component-based Software development (CBSD for short) (Georgiadis et al., 2002), and others based on middleware and Frameworks (Schmidt and Buschmann, 2003). Some approaches based on the concept of Framework have been proved to be useful in the development of secure services, as defended in (Schmidt and Buschmann, 2003). Unfortunately, current frameworks provide static security properties. In other words, the security solutions offered can not be adapted to work when then number of context changes is high. Thus, they are selected and configured at application development time. This limitation makes the approach not applicable in heterogeneous and dynamic AmI environments.

Current work on CBSD is mainly focused on the dynamic analysis of component compatibility. Usually from a functional point of view, with the target of adapting components and synthesizing suitable software architectures (Becker et al., 2007). A remarkable research is presented in (Mei and Xu, 2003). This work is based on the use of Bayesian Belief Networks for an adaptive dependability model. Unfortunately, current approaches are not focused on the automated selection and setup of security solutions answering to different security requirements, which is one of the strongest points of our approach in Serenity. In (Kung, 2007), A. Kung highlights how the usage of patterns is widely used today in the specification of architectures and the design based on aspects. Concretely, these patterns refer to templates describing a solution to solve a commonly occurring problem. Work on security architectures for component systems has been focused in the security problems caused by the interaction of general components while we focus on the interaction of components providing security and dependability services. A security architecture for composing secure systems from components is presented in (Jaeger et al., 1998). This architecture is designed to support the dynamic composition of systems and applications from individual components. This approach differs from Serenity due to it only deals with access control. As this one, most of the current proposals are based on oversimplified views of security, like those based on security levels.

A further discussion is found in (Nobukazu et al., 2004), where security patterns are applied to construct secure and efficient intercompany coordination systems. Authors show an application of their method to the Environmentally Conscious Product design support system. As a result, they provide guidelines to developers for modeling the performance of data associated with each pattern. Unfortunately, these guidelines are not expressive enough for the security patterns developed in the Serenity project and this approach have no mean for automating neither the discovery, nor selection and nor deployment of the candidate patterns.

3 CAPTURING KNOWLEDGE

This section gives an overview of the Serenity Project. The main objective of Serenity is to provide a framework for the automated treatment of security and dependability issues in AmI scenarios. For this purpose, the project is two-folded: (i) capturing the specific expertise of the security engineers in order to make it available for automated processing, and (ii) providing run-time support for the use and the monitorization of these security and dependability mechanisms. These two cornerstones have been deployed by means of: A set of *S&D modeling artefacts* (S&D artifacts, for short), used to model security and dependability solutions (S&D solutions) at different levels of abstraction. S&D solutions are isolated components that provide security and/or dependability services to applications. The use of different levels of abstraction responds to the need of different phases of the software development process. These artefacts are supported by an infrastructure created for the development and the validation of S&D solutions. This infrastructure includes concepts, processes and tools used by security experts for the creation of new S&D solutions ready for automatic processing.

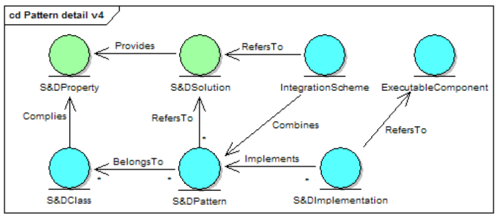

Figure 1: Simplified model.

A *development framework*. Under the name of Serenity Development-time Framework (SDF) there is an infrastructure that supports the development of secure applications. These secure applications, called Serenity-aware applications, are supported by S&D solutions, consequently, they include references to the aforementioned S&D artefacts. A *run-time frame-*

work, called Serenity Run-time Framework (SRF). The SRF provides support to applications at run-time, by managing S&D solutions and by monitoring the systems' context. The SRF is briefly described in Section 4. Once infrastructural pieces have been described, the rest of this section explains how to use S&D modelling artefacts to bridge the gap between abstract S&D solutions and actual implementations of these S&D solutions. Interested readers could refer to Section 3 in order to find information on how the SRF supports applications at run-time. Back to the abstractions, five main artefacts are provided to achieve a logical way to represent S&D solutions in the Serenity project: S&DClasses, S&DPatterns, IntegrationSchemes, S&DImplementations and ExecutableComponents. These artefacts, depicted in figure 1, represent S&DSolutions using semantic descriptions at different levels of abstraction. The main reason for using different artefacts, each one addressing an abstraction level, is that, by doing this, it is possible to cover the complete life cycle of secure applications, especially at development and run-time phases. *S&DClasses* represent abstractions of a set of S&DPatterns, 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 this artefact is to facilitate the dynamic substitution of S&D solutions at run-time, while facilitating the development process. Applications request S&D Solutions to the SRF to fulfill a set of S&D requirements. Usually, these requirements are hard coded by means of calls to S&DClasses or S&DPatterns interfaces. At run-time all S&DPatterns (and their respective S&DImplementations, described below) belonging to the same S&DClass, will be selectable by the SRF automatically. *S&DPatterns* are precise descriptions of abstract S&D solutions. These descriptions contain all the information necessary for the selection, instantiation, adaptation, and dynamic application of the solution represented by the S&DPattern. S&DPatterns describe the security pattern's functionalities and how to use them in a structured way. The most interesting elements of the S&DPattern structure are: (i) The pattern interface, describing the functionalities provided and how to use them; (ii) references to the S&DClasses the S&DPattern belongs to; and (iii) the ClassAdaptor, describing how to adapt the S&DPattern interface to the S&DClass interface. S&DPatterns represents monolithic isolated S&D solutions, but a special type of S&D artefact called *Integration Scheme (IS)* also exists, which consists on an S&D solution at the same level than S&DPatterns. They represent S&D solutions that are built by means of combining other S&DPatterns. At Serenity-aware application development time, Integration Schemes are used similarly as S&DPatterns are. However, they differ in their development process, presented in (Mana et al., 2006). All along this paper we use the notion of S&DPatterns to refer to S&DPatterns and Integration Schemes indistinctly. *S&DImplementations* are specification of the components that realize the S&D solutions. S&DImplementations are not real implementations but their representation/description. An S&DImplementation describes an implementation of an S&DPattern and, thus, a S&DPattern may have more than one S&DImplementation. Finally, *ExecutableComponents* are real implementations of the S&DImplementations. These elements are not used at development time, but they are the realization of the selected S&D solution at run-time. An ExecutableComponent works as a stand alone executable S&D solution ready to provide its services to applications. They are software, and sometimes hardware, components. As shown in figure 1, every S&D solution provides at least one security property. Every S&DPattern (and every Integration Scheme) refer to an S&D solution. On the contrary, every S&D solution can be represented by one or more S&DPattern and/or Integration Scheme. Each S&DPattern is implemented by means of at least one S&DImplementation. Finally, there is an ExecutableComponent entity for each S&DImplementation. The entities in the top of the figure 1 conform a hierarchy. While, S&DClasses are the most abstract level entities to represent S&D solutions, ExecutableComponents, being software components, are the lowest abstraction level way to represent an S&D solution. For the representation of S&D solutions, following the Serenity approach, developers need to count on, at least, one artefact for every level of the hierarchy. To sum up, S&DClasses, S&DPatterns and S&DImplementations are development-time oriented artefacts, while ExecutableComponents are specially suitable for run-time. Serenity-aware applications include references to development-time artefacts. Depending on the artefact level of abstraction, at run-time, the SRF has more/less flexibility to select S&D solutions. In other words, this approach enables the creation of open architectures where, at run-time, the SRF completes by applying the ExecutableComponents that implements the S&D solutions fixed at development-time. The main purpose of introducing this approach is to facilitate the dynamic substitution of S&D solutions at run-time while facilitating the development process.

4 EXPLOITING S&D SOLUTIONS

Security experts capture their expertise into S&D Artefacts, but in order to go a step forward, these artefacts should be exploited by applications at runtime. In other words, applications with security and dependability requirements should be able to fulfill them at runtime, by means of deployed S&D Solutions, automatically provided and controlled by a trustworthy third-party. This approach also considers that application developers do not necessary know anything about particular security solutions but about certain generic S&D requirements (such as confidentiality or reliability), probably identified by a functional analyst with a basic security expertise knowledge. Moreover, these applications should rely on the provided S&D solutions to be able to automatically be reconfigured and adapted in the event of a typical AmI environment condition change. Serenity Runtime framework (SRF) provides this support.

4.1 Serenity Run-time Framework

Bellow, Figure 2 presents a simplified view of the SRF and the most relevant components that interact with it at runtime, in the process of exploitation of S&D Solutions. The figure shows two components inside the SRF: the S&DLibrary and the ContextManager. The S&DLibrary component is a local

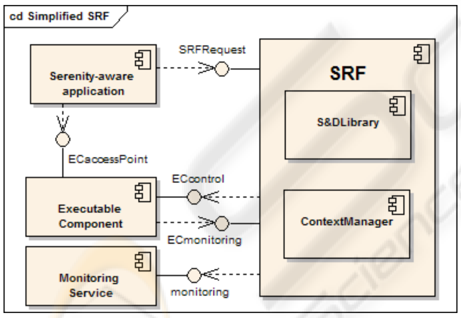

Figure 2: Simplified architecture.

S&D solution repository that stores the S&DClasses, S&DPatterns and S&DImplementations, specific to the platform and that might, potentially, be used in the device. The ContextManager component keeps device's context related data (SRF configuration, active patterns and monitoring information). Context information is used by the SRF to select the most appropriate S&D solution for a given scenario under a set of particular conditions. Outside the SRF, figure 2 depicts all the components and interfaces that are involved in the run-time deployment, following further detailed. A *Serenity-aware application* is the application that uses the security services provided by S&DPatterns, by means of the ExecutableComponents. The SRF provides a *SRFRequest* interface, to be used by applications in order to request S&D artefacts. Usually, applications request S&DClasses, S&DPatterns and S&DImplementations, but these requests result in the activation of an ExecutableComponent implementing the S&D solution requested.

The ExecutableComponents are activated by the SRF and they implement S&DPatterns functionalities. ExecutableComponents provide two interfaces: the *ECcontrol* interface, to be used by the SRF; and the *ECaccessPoint*, to be used by applications. The SRF controls the proper execution of ExecutableComponents by using the *ECcontrol* interface. Serenity-aware applications access to the security services provided by ExecutableComponents through the *ECaccessPoint* interface. The *ECmonitoring* interface (provided by the SRF) is used by ECs to keep the SRF informed about some particular aspects of its execution. The SRF uses this information to control the status of the execution of the EC and its impact into the environment, with the support of the Monitoring Services.

This process of control is described ahead in this document. Finally, *Monitoring Services* provide an interface, called *monitoring*, that allows the SRF make use of a set of services provided to support the process of control of the execution of ECs, and to perform recovery and reconfiguration actions. Now we have a more clear idea of the architecture and components involved at run-time, we are going to describe the functionalities provided by the SRF in more detail.

4.2 Selection of the S&D Solution

In the context of the Serenity project, the SRF is implemented as a service of the operating system that listens to application S&D requests. By means of these requests, the applications specify their S&D requirements, and the SRF uses those specifications to search the repository of S&D Artefacts, here referred as S&D Library, and selects the most appropriate S&DPattern that fulfills the application request. This selection consists of a pattern-driven algorithm based on some particular aspects of the S&DPattern specification, as well as on the S&D requirements specified by the application developer. This means that the SRF looks into the descriptor of each S&DPattern stored in the repository, and checks if its characteristics, such as security properties or features, match the application's particular S&D requirements. In addition to this, the SRF needs to check if the S&DPattern is appropriate for the particular context environmental situation, looking at the context preconditions specified by the security expert within the S&DPattern descriptor. Once the SRF has selected an S&DPattern, it is time to start exploiting the S&D Solutions that represents.

4.3 Activation of S&D Solutions

ExecutableComponents (ECs) are realizations of S&D Solutions. Each S&D pattern may have several implementations, described as S&DImplementations, and, each S&DImplementation has its corresponding EC. ECs are software components that the SRF is able to activate and execute, and have some particular characteristics: (i) provide applications with the interface defined in the S&DPattern, (ii) implement the functionality of an S&D Solution, described by the S&DPattern, and (iii) provide a monitoring interface to enable the SRF to control its execution. Therefore, to activate the selected S&D Solution that will provide the application with the S&D properties requested, the SRF creates a new instance of the EC that runs in the operating system, as an independent software piece, within the same S&D realm as the application and the SRF are running. To make it accessible to the application, the SRF uses handler, ECHandler in the context of Serenity, which is configured with all the information the application need to communicate with the EC. This ECHandler is brought back to the application as a result of its request. By doing this, the EC can be accessed by the application directly, and provide the specific S&D Solution functionality without any more intervention of the SRF. Up to here, the initially mentioned approach could be seen, basically, as repository of S&DPatterns, plus a software component that searches and instantiates the corresponding S&D Solutions. For AmI environments, we must go further. We must react to continuous changes of the context conditions and be able to reconfigure the running ECs in order to keep fulfilling the security and dependability requirements of the applications despite the changes.

5 MONITORIZATION

The Monitoring Services, briefly introduced above, are actually a framework of components that support the SRF to monitor, diagnose and detect threats related with the execution of the ECs.

In figure 2, there is a component labeled Monitoring Services. Security experts define a set of monitoring rules in the S&DPattern description that need to be satisfied during the whole lifetime of a deployed S&D Solution. These rules are basically assumptions about the behaviour of a system, and how runtime events may affect its state, making sure the security and dependability properties the S&D Solution is providing remains no matter the conditions of the context are,. The SRF, by means of the Monitoring Services, checks every certain period of time that those rules keep holding. In the event of a monitoring rule violation, the SRF makes use of the diagnosis and threat detection mechanisms of the Monitoring Services to evaluate if any reaction should be perform. How to react to a monitoring rule violation is defined by the security experts within the S&DPattern descriptor. For each monitoring rule there is a reaction behaviour associated, that basically are intended to take corrective actions to avoid any potential system failure derived from the malfunction of the EC, and to adapt the system security and dependability mechanisms at runtime. Some examples of reactions could be to stop the execution of a particular EC, to restart the EC with other initial parameters or to stop/start monitoring certain set of rules, amongst others. The Monitoring Rules of an S&DPattern rely on the ability of the ECs to generate traces during its execution. Those traces are encapsulated in the form of events that are captured by some components, named Event Capturers in the context of Serenity project, attached to the ECs. Those events captured feed the Monitoring Services allowing it to deduce information about the state of the system being monitored. The SRF polls the Monitoring Services frequently about the status of each active Monitoring Rule and if there is a violation detected, the SRF triggers the corresponding reaction.

6 DEVELOPMENT SUPPORT

Serenity project aims to provide end-users not only with a set of methodologies and abstractions, but also with some means to benefit of them and to apply to within their particular scenarios. To support this, here we introduce two infrastructures provided by Serenity. One supports the development of the ExecutableComponents, and the other supports the development of Serenity-aware applications. As aforementioned, ExecutableComponents have a one-to-one correspondence with S&DImplementation abstractions, fully implementing the security solutions described in the referenced S&DPatterns. ExecutableComponents are software/hardware components runnable independently in a system. They provide interfaces to be used by either the Serenity-aware applications or the SRF itself.

These two infrastructures actually consist on two programming libraries, one for each purpose, and either EC programmers or Serenity-aware application programmers may integrate them by simply import them within their specific software distribution. The upper package represents the library that developers may use to create Serenity-aware applications. It includes classes for accessing the SRF (SRF_AccessPoint) and for managing the ExecutableComponents (SerenityExecutableComponent_AP.). The other package is the library developed to assist developers creating ExecutableComponents. In order to create an ExecutableComponent, programmers must extend the class SerenityExecutableComponent_AP. Following the structure of the presented infrastructure, a Serenity-aware application uses instances of the SerenityExecutableComponent_AP class, to establish communication with the ExecutableComponents. This class is an interface between the application and one ExecutableComponent, providing standard mechanisms no matter how the implementation of the ExecutableComponent is.

Thus, applications only need to create an instance of the SerenityExecutableComponent_AP class and then, the application uses the ECHandler information provided by the SRF to access ExecutableComponent interface. Figure 3 depicts the S&D artefacts developed for this scenario.

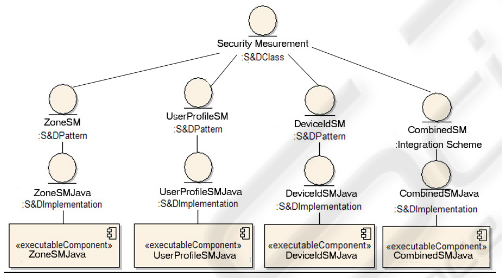

Figure 3: Hierarchy of S&D Artefacts.

By doing this, applications access the services that S&DPatterns described. We clarify this process a bit more next. At development-time, application developers learn the behaviour expected from the security solution, such as the set of available calls and the appropriate call sequence to use the security services, by looking at the description of the development-time S&D artefacts (S&DClasses, S&DPatterns, and S&DImplementations). These artefact descriptors contain this information in the form of, what we call in Serenity, interface calls, sequences and constraints. S&DPatterns' specifications also contain information about how to adapt the interface offered by S&DClasses to the one offered by S&DPatterns, we call this adaptation part Class Adaptors. ExecutableComponent developers strictly follow the guidelines described in the corresponding S&DClass and S&DPattern descriptors, so ExecutableComponents are *Serenity compliant* and are expected to respond to those queries as described (part of the Serenity Project is devoted to guarantee that ExecutableComponents fulfill the specifications of the S&DPattern they implement). It is important to take into account that all Serenity S&D Artefacts are certified by its creators, opening a new business model based on the provision of security solutions by means of S&DPatterns. Continuing with the process description, the SRF manages all ExecutableComponents instantiated as result of Serenity-aware applications requests. It keeps a record of important information such as status, location or assigned monitoring services. It is important to highlight that the instantiation of ExecutableComponents generates new context information for the SRF, and this influences the results of the S&D solution selection algorithm. Moreover, the events generated by ExecutableComponents feed the Monitoring Service infrastructure and they could result in the violation of Monitoring Rules. Monitoring rules are associated to reactions, and for instance, one common reaction is to de-activate the S&DPattern whose Monitoring Rule was violated.

7 SECURITY MEASUREMENT

This section introduces an example which highlights all the concepts presented along this paper. This example consists in a prototype implementing a Control Access System for networks. Briefly, the goal of this system is allowing or denying access to network resources attending to several parameters what can be considered security measurements related to sensible security data like user profile, device identification and location (this last element provides a small component of AmI). The prototype has been developed following the methodology proposed by the Serenity project, as explained in previous sections. First step was to extract S&D requirements from the goals, just like in any other development process. Full list of requirements is out of the scope of this paper, but once obtained, we design a set of Serenity S&D Artefacts, modelling S&D Solutions to cover all those requirements. For the sake of simplicity, in this paper we focus on solutions for managing the aforementioned security measurements.

7.1 S&D Artefacts

The first artefact on top is an S&DClass *Security Measurement*, representing an abstract S&D Solution for a generic security assessment providing a quantitative value, let's say a number. Below the S&DClass, there are four entities: three S&DPatterns and an Integration Scheme. The S&DPatterns represent particular implementations of the S&D solution represented by the *Security Measurement* S&DClass. The difference among them is the parameter they are based on, to perform the security assessment. The *UserProfileSM* S&DPattern performs a measurement based on the profile attached to a connected user (please, assume every user is authenticated when connecting to the network). The *ZoneSM* and *DeviceIdSM* S&DPatterns do the same, taking into account the location and the digital identification (by cryptographic means) of the connecting devices, respectively. The *CombinedSM* Integration Scheme combines the three S&DPatterns and provides a final security measurement working as a unique component. The rest of the figure 3 presents an S&DImplementation for every S&DPattern, and an ExecutableComponent for every S&DImplementation.

7.2 Development Phase

After building the abstract elements in the form of S&D Artefacts we proceed to develop both the ExecutableComponents and the main access Control Application, this time with the help of the SRF's specialized APIs (one for Excutable Components and another one for applications), presented in previous section. The Control Application was developed as a Serenity-aware application. It opens a communication channel with the SRF and requests an S&D solution, represented by the *CombinedSM* Integration Scheme. As a result of this request, the SRF instantiates the corresponding ExecutableComponent and provides the application with the configured ECHandler. This particular EC works internally as a new Serenity-aware application which requests the three S&DPatterns mentioned before, and combine their results. This way, the core Control Application is quite simplified with just one request to perform all the measurement. In the context of the proposed scenario, the application would expect a solution for a very simple problem: authorize a resource request. It is as simple as getting a *ok* or a *ko* at certain point of the execution flow of the application, no matter how, the great advantage of using Serenity is, in fact, to make it that easy. The application developer, at design-time, may assume that a properly configured SRF, when requested for a certain S&D solution, will provide a mechanism to use a running instance of that S&D solution for its own purposes. Developers do not need to take into account how the S&D solution is implemented.

7.3 Monitoring and Reactions

Last but not least, the prototype exploits the Monitoring Services features described in section 5. During development time, we created for each S&DPattern a set of monitoring rules, and we also added to each ExecutableComponent the necessary Event Captors to feed that rules. For instance, the implementation of the location solution is based on a client-server model, and we considered important to monitor the server availability during the lifetime of the solution. Therefore, we added a specific rule within the S&DPattern descriptor. The rule, roughly, describes the following: *the server must respond always before n seconds*. If the response takes longer than n seconds, or it never responds then, the rule will be considered violated. Some Events Captors were developed and attached to the ExecutableComponents, in order to provide custom events for our rule. In this case, we have two types of events: one signaling the *ping* to the server and another marking the *ping response*. With all these elements on the table, a Monitoring Service will be able to check the time interval between both events happens, and to determine whether the rule is violated or not. Other rules (and its corresponding events) could be added to control the other ExecutableComponents as well. Regarding the ability of the SRF to perform corrective actions on the event of a monitoring rule violation, we use as an example a reaction mechanism called *notify application*. This one is used by the SRF to notify the application a message whenever a rule is violated. We need to mark the rule in the S&DPattern specifying the reaction we would like to have performed if it is violated, and write down the custom message we want to notify. In our scenario, it would be very useful if, in the case the rule is violated, the SRF sends a particular warning message, so the application could display it. It is a very simple reaction but demonstrates the functionality. It would be possible to design more complex reactions to improve the adaptability of the system. For instance, in our scenario, imagine the S&D solution has been instantiated with an initial parameter (specified by the application) which is actually the server url. Pretend, the rule that monitors whether the server is alive is violated. Here, the reaction could be restarting the solution with a different server url value, such as a backup server one specified in the S&DPattern descriptor. Then, the application will be able to continue accessing the services provided by the S&D Solution with no damage.

8 CONCLUSIONS

This work presents an approach for producing secure applications by means of a provision of S&D solutions. This approach, called Serenity, is composed by two frameworks. On the one hand, the Serenity Development-time Framework (SDF) includes concepts, processes and tools supporting the development of S&D solutions, and secure applications. These S&D solutions are implemented by means of four S&D artefacts: (i) S&DClasses, (ii) S&DPatterns, (iii) S&DImplementations, and (iv) S&D ExecutableComponents. This hierarchy enables the use of security and dependability (S&D) patterns implementations to develop secure applications. These S&DPatterns are not a simple set of *best practices* or recommendations, like those proposed in the literature as *security patterns*, but precise, well-defined and automated-processing-enabled implementations of security mechanisms. The extensive use of semantic descriptions, enable the use of automated reasoning mechanisms capable of solving problems such as pattern composition and adaptation. Thus, we have presented an infrastructure for supporting both the development of ExecutableComponents and Serenity-aware applications. On the other hand, the Serenity Run-time Framework (SRF) completes the open software architectures designed using the SDF, by supporting applications when they requires the use of security solutions at run-time. It is important to highlight that the SRF includes monitoring mechanisms that guarantee that selected S&D solutions are running properly. In this paper we have presented both the SDF main concepts and the SRF architecture. Furthermore, we have provided the description of full developed demo scenario. To deploy this scenario a set of S&D artefacts has been developed. These artefacts are used to provide several secure measurement and authentication security patterns, all of them represented at different abstractions levels, ranging from the most abstract representation of the solution (S&DClass), to the real implementation of the pattern (ExecutableComponent). Among the more relevant features of this approach is the selection of patterns at runtime, as well as the configuration of them. Indeed, this allows context awareness while running obtaining exceptional results. Currently, we count on a fully operational prototype of the SRF and on XML based languages for the creation of all S&D artefacts. Besides, we have developed two APIs, one for the implementation of ExecutableComponents, and the other one oriented to the implementation of serenity supported applications (Serenity-aware applications). Finally, we count on an on-line repository and a useful tool to search for S&D artefacts. These elements have been the basis to develop a first set of S&D solutions. Next steps are (i) to improve the capacities of the SRF prototype with more reaction capabilities, (ii) to create a plugging for include design concepts in a Java IDE, and (iii) to provide a new version of the APIs for taking the new advantages that will be introduced in the next SRF prototype.

REFERENCES

Becker, S., Canal, C., Diakov, N., Murillo, J., Poizat, P., and Tivoli, M. (2007). Coordination and adaptation techniques: Bridging the gap between design and implementation. In Springer, L., editor, *Report on the ECOOP Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'06).*

Georgiadis, I., Magee, J., and Kramer, J. (2002). Self-organising software architectures for distributed systems. In *WOSS '02: Proc. workshop on Self-healing systems*, pages 33–38, New York, NY, USA. ACM.

Jaeger, T., Liedtke, J., Pantellenko, V., Park, Y., and Islam, N. (1998). Security architecture for component-based operating system. In ACM, editor, *In ACM Special Interest Group in Operating Systems (SIGOPS) European Workshop.*, page 118.

Kung, A. (2007). Architecture and design patterns for ambient intelligence: an industry perspective. In *Proc. of AmID 2007*, volume ISBN: 978-2-287-78543-6, pages 55–67, Sophia-antipolis (France). Springer-Verlag.

Mana, A., Sánchez, F., Serrano, D., and Munoz, A. (2006). Building secure ambient intelligence scenarios. In *18th conf. on Software Engineering and Knowledge Engineering (SEKE'06).*

Mei, L. and Xu, Y. (2003). An adaptive dependability model of component-based software. In ACM, editor, *ACM SIGSOFT Software Engineering Notes*, volume 28.

Nobukazu, Y., Shinichi, H., and Anthony, F. (2004). Security patterns: A method for constructing secure and efficient inter-company coordination systems. In International, E. I., editor, *Enterprise Distributed Object Computing Conference*, pages 84–97. IEEE Computer Society Press.

Schmidt, D. C. and Buschmann, F. (2003). Patterns, frameworks, and middleware: their synergistic relationships. In *ICSE '03.*, pages 694–704, Washington, DC, USA. IEEE Computer Society.