

Security Engineering for Ambient Intelligence: A Manifesto

AUTHORS:

Antonio Maña

Computer Science Department, University of Málaga,
Boulevard Louis Pasteur s/n, 29071 Málaga, Spain
Tel: +34952137142 Fax: +34952131397
amg@lcc.uma.es

Carsten Rudolph

Fraunhofer Institute for Secure Information Technology (SIT),
Rheinstrasse 75, D-64295 Darmstadt, Germany
Tel: +496151869344 Fax: +496151869224
Carsten.Rudolph@sit.fraunhofer.de

George Spanoudakis

Department of Computing, City University,
Northampton Square, London, EC1V 0HB, UK
Tel: + 44207 0408413 Fax: + 44207 0400244
gespan@soi.city.ac.uk

Volkmar Lotz

SAP Research,
805 Avenue du Dr. Maurice Donat, 06250 Mougins, France
Tel: +33492286444 Fax: +33492286201
volkmar.lotz@sap.com

Fabio Massacci

Dipartimento di Informatica e Telecom., University of Trento,
via Sommarive 14, 38050 Povo, Trento, Italy
Tel: +390461882086 Fax: +390461882093
Fabio.Massacci@unitn.it

Matteo Melideo

Engineering Ingegneria Informatica,
Via S.Martino della Battaglia 56, 00185 Rome, Italy
Tel: +390649201410 Fax: +390649201340
matteo.melideo@eng.it

Jose-Manuel López-Cobo

ATOS Origin,
Albarracin, 25, 28037, Madrid, Spain
Tel: +34912148610 Fax: +34917543252
jose.lopez@atosorigin.com

Security Engineering for Ambient Intelligence: A Manifesto

Abstract. The scenarios of Ambient Intelligence introduce a new computing paradigm and set new challenges for the design and engineering of secure and dependable systems. This chapter describes SERENITY, a comprehensive approach to overcome those problems. The key to success in this scenario is to capture security expertise in such a way that it can be supported by automated means. SERENITY's integral model of S&D considers both static and dynamic aspects by relying in two main innovations: (i) the enhanced notion of S&D Patterns and Integration Schemes; and (ii) the computer aided run-time monitoring of the implemented security solutions. The combination of these innovations lays the foundations of an integrated, solid, flexible and practical S&D framework for AmI ecosystems. The chapter aims at clarifying the challenges introduced in AmI ecosystems and pointing out directions for research in the different areas involved.

Keywords: Security Engineering, Software Engineering, Ambient Intelligence

INTRODUCTION

Well in the past millennium, the typical reaction to questions about “Information Systems Security” would have been another question “what is security?”. A decade ago the response to the same concern was “security is an added value service, a non-functional requirement”. Only in the last few years the response has become “security must be designed and built-in from the very start”. Although the latter response acknowledges the need for integrating security into the design and development phases of IT, this integration is still lagging behind in practice (Tryfonas, 2001). Security Engineering Capability Maturity Models, ISO 17799 and other standards detail a classification of properties that a secure system must have but do not actually provide any information on how such properties can be achieved. Even landmark textbooks on security engineering (Anderson, 2004) are often a collection of case studies and do not provide a structured methodology for actually engineering secure systems, no more than a dictionary can be called a grammar.

Today COTS (Components off-the-shelf) security services such as encryption, digital signatures, public key infrastructures, etc. and rather standard attack countermeasures such as firewalls, intrusion detection systems, etc. are selectively employed in the attempt to fulfil basic security requirements such as authentication or confidentiality. The usual deployment of those security mechanisms provides an “isolated” functionality, which is not always

appropriate for the specific system to be protected and is usually not effective for addressing the potential threats introduced by emerging environments like grid computing and wireless ad-hoc networks. Experience in the development of cryptographic protocols shows that even for relatively small systems, it is difficult and often error-prone to fulfil security requirements by simply combining existing security mechanisms. Even for such a brittle field, there are few and specialized solutions that supports in an automated way the precise identification of security requirements and the correct choice of security mechanisms. Their adoption by IETF protocol designers has still a long way to go (Interested readers might have a look at the AVISPA EU project effort into introducing the usage of formal methods into standard security protocols design at www.avispa-project.org).

Designers and users must also face other sources of complexity: the impact of *contextual information* (e.g. hardware and software configurations of the system), *external entities interoperating with the system*, *legal requirements*, *social behaviour*, etc. can be easily overlooked. As a consequence, expert knowledge in IT security is a prerequisite for applying existing security mechanisms. Unfortunately, such expertise is usually not available to average system developers and users and is often too expensive to acquire. Moreover, for large scale systems, the complexity of the current and emerging computing environments and their security requirements has increased to the point where experience is not enough to guarantee adequate security.

The new scenarios of Ambient Intelligence, their underlying pervasive technology, their notion of mobile services, where the IT environment moulds itself around the user's needs raise the bar for what is a satisfactory security solutions well beyond standard IT security technology. New challenges stemming from identity and privacy protection are appearing at the horizon. These emerging technologies need sophisticated approaches to determine the security requirements and then to provide adequate security solutions. In Ambient Intelligence

scenarios, where the interaction with the environment and the human stakeholders is predominant, the overall security depends on a variety of other factors not covered by conventional software engineering processes including social context and human behaviour, IT environments, and even protection of the physical environment of systems (e.g. buildings).

It is therefore evident that current 'ad-hoc' approaches cannot support a complete and rigorous treatment of security starting from the requirements elicitation process up to system implementation and, above all, system operation. Different factors result in this situation:

- (i) security is a non-functional requirement, thus it is hard to capture with standard software engineering techniques;
- (ii) security is partly a social and not only a technical problem, thus it is hard to capture in standard design languages;
- (iii) there is no homogeneous way to represent security concerns and security mechanisms at different levels of software description, thus it is hard to trace security issues; and
- (iv) the current practices of security engineering produce solutions that depend heavily on the details of the application context. These practices need to analyse the complete system, including all interacting entities, in order to provide some guarantees of the security of the solutions.

Therefore, in the emerging computing paradigms, where this context is highly dynamic, where systems are not completely under control of one part and where it is not possible to foresee all specific configurations that may arise, these design practices are bound to fail. Though the systems designed with current practices might not necessarily be non-secure, the process is likely to still have significant weaknesses:

- (a) the security requirements of the entire system might not all be satisfied by eliminating single, isolated weaknesses in the system and furthermore, their fulfilment can not be confirmed since the requirements have not been clearly specified in the first place;

- (b) different implementations of the same system cannot be easily compared with respect to their security properties nor the compositional effect of employing different security measures at different level of network and application development and
- (c) the security mechanisms introduced are not directly linked to the specific security requirements of the system. Linking between security mechanisms and requirements is particularly important in case of failures of security mechanisms, when the effects of attacks on the overall security of a system must be determined.

The contribution of this paper

In this paper we present the SERENITY approach to the provision of security and dependability (S&D) in Ambient Intelligence (AmI) scenarios. This approach is based on the enhanced concepts of S&D Patterns and Integration Schemes and the integration of runtime monitoring techniques. The structure of the paper is as follows: Section 2 introduces the specific characteristics of the AmI scenarios and the challenges associated to the provision of security and dependability in these scenarios; Section 3 reviews relevant related work; Section 4 and 5 presents the SERENITY approach and describes the different elements that are part of the SERENITY Framework; Section 6 illustrates the use of SERENITY based on a specific application scenario; and finally, Section 7 presents the conclusions.

SECURITY IN AMBIENT INTELLIGENCE SCENARIOS

Defined by the EC Information Society Technologies Advisory Group (ISTAG), Ambient Intelligence emphasises on greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions. In this vision, people will be surrounded by intelligent and intuitive interfaces embedded in everyday objects around them and an environment recognising and responding to the presence of individuals in an invisible way by year 2010.

Ambient Intelligence (AmI for short) builds on three recent key technologies: Ubiquitous Computing, Ubiquitous Communication and Intelligent User Interfaces¹. Ubiquitous Computing means integration of microprocessors into everyday objects like furniture, clothing, white goods, toys, even paint. Ubiquitous Communication enables these objects to communicate with each other and the user by means of ad-hoc and wireless networking. An Intelligent User Interfaces enable the inhabitants of the AmI environments to control and interact with these environments in a natural (voice, gestures) and personalised way (preferences, context). Ubiquitous computers will seamlessly enhance the way we work and even the way we live thanks to the access to information technologies that will optimize the environment for people's needs in different physical spaces. Anytime/anywhere access to information and services will be provided.

The ISTAG vision is that AmI applications will be influenced by the computational, physical and behavioural contexts that surround the user (for instance, because of resource availability and security or privacy requirements). The concepts of system and application as we know them today 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 *AmI ecosystems* 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 placed on *security and dependability* (S&D), make application development more complex and the provision of security and dependability for applications increasingly difficult to achieve with existing security engineering mechanisms and tools.

¹ Some of these concepts are barely a decade old and this is reflected on the focus of current implementations of AmI

Challenge 1. Dealing with dynamism. *In the new AmI scenarios, not only systems as a whole but also individual applications running in or supported by those systems will have to adapt to dynamic changes to hardware and software, and even firmware configurations, to unpredicted and unpredictable appearance and disappearance of devices and software components. In other words applications must be able to adapt dynamically to new execution environments. As a consequence pre-defined trust relationships between components, applications and their system environments can no longer be taken for granted.*

The increased complexity and unbounded nature of AmI applications make it impossible, even for the most experienced and knowledgeable S&D engineers, to foresee all possible situations and interactions which may arise in AmI environments and therefore create suitable solutions to address the users' security and dependability requirements. Additionally S&D engineers will be faced with pieces of software, communication infrastructures and hardware devices not under their control. Thus, approaches based on the application-level security will not be sufficient to provide security and dependability to the AmI ecosystem as a whole.

Challenge 2. Dealing with heterogeneity. *AmI environments will contain a large number of heterogeneous computing and communication infrastructures and devices that will provide new functionalities, enhance user productivity, and ease everyday tasks. These devices will hold a variety of data with different security and privacy requirements. This information will be used in different ways in different applications and computing contexts and, therefore, different policies (possibly contradicting) will be applied.*

In such settings, securing the device or the information alone or even each individual application is not sufficient, and context information should be integrated in order to be able to choose appropriate security mechanism on-the-fly.

Challenge 3. Need for supervision. *Finally, because of their complexity, and because elements will be under the control of different owners, security mechanisms will need to be*

supervised (monitored) in order to identify potential threats and attacks and decide on recovery actions, if possible.

Some existing approaches can provide suitable solutions to support the dynamic evolution of security policies for specific security mechanisms —e.g. SAC model for access control (Lopez, 2003)— at particular system operation layers (application, networking). However, these approaches cannot be extended to support the dynamic evolution of general security mechanisms (as opposed to security policies for a single mechanism). Furthermore, their results are extremely complicated to integrate, monitor and dynamically evolve as would be required by AmI ecosystems. For the very same reasons, S&D approaches for AmI ecosystems cannot hope to synthesize new S&D mechanisms or new combinations of these mechanisms fully automatically and dynamically.

We can summarize the individual challenges that we have devised so far into a simpler and yet tougher grand challenge:

Challenge 4. Dynamic application of security expertise. *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.*

The intuitive solution would be 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. To meet this challenge in our time we need to look more closely to what technology is available for S&D mechanism in AmI ecosystems.

RELATED WORK

Even if the provision of appropriate S&D mechanisms for AmI ecosystems remains an unsolved issue, several approaches have been introduced with the goal of capturing the

specialized expertise of security engineers and making it available for automated processing, thus providing the basis for automated synthesis and analysis of the security and dependability solutions of systems (Schmidt, 2003):

Components capture expertise in the form of reusable software elements having a set of well-defined interfaces and an associated description of their behaviour (Llewellyn-Jones, 2004; Mantel, 2002; Shi, 1998). This concept is not appropriate to represent general security solutions because security mechanisms can not always be represented as units that can be connected to the rest of the system by well defined interfaces. Many security mechanisms are not about *what* but *how*. In fact, software components are good abstractions of functional elements, but security and dependability are non-functional aspects.

Frameworks capture expertise in the form of reusable algorithms, extensible architectures, and component implementations. Application frameworks (Fayad, 1999; Llewellyn-Jones, 2004; BEA, 2003) have emerged as a powerful technology for developing and reusing middleware and application software. In line with the general definition, the concept of security framework is frequently used to refer to system architectures, infrastructures, algorithms or even methodologies and processes that are used to provide security in certain scenarios. Also related to security, conceptual frameworks have been proposed as a means to describe security properties. Because frameworks are application templates, they are not well suited to cope with scenarios with high degrees of heterogeneity, dynamism and unpredictability. Likewise, this approach does not support secure interoperation with external (and not trusted) elements.

Middleware capture expertise in the form of standard interfaces & components that provide applications with a simpler facade to access the powerful and complex capabilities of frameworks. Some of the most successful techniques and tools devised to enhance the reuse of software focus on distributed computing middleware that helps manage the complexity and

heterogeneity of distributed applications. Although the flexibility, efficiency and interoperability with external elements of middleware-based systems are not optimal, some research has been carried out on the application of this approach to ubiquitous computing (Román, 2002; Banavar, 2002). An important problem with middleware-based approaches is that the computational cost of the middleware components is far too high for computing devices with limited capabilities. Finally, the security infrastructure of middleware systems is usually restricted to authorization and access control issues (OMG, 2004; BEA, 2003).

Patterns capture expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, components implementations, or frameworks cannot. The concept of security pattern as “a well-understood solution to a recurring information security problem” was introduced to support the system engineer in selecting appropriate security or dependability solutions. However, most security patterns are expressed in a textual form, as informal indications on how to solve some (usually organizational) security problem (Schumacher, 2003; Kienzle, 2003; IBM, 2004; Wimmel, 2001; Cheng, 2003; Yoder, 2000; Blakley, 2004; Romanosky, 2002). Some of them use more precise representations based on UML diagrams, but no rich semantic descriptions are included in order to automate their processing and to extend their use. Furthermore, there is no guarantee of the correct application of a pattern because the description does not consider the effects of interactions, adaptation and combination. This makes them not appropriate for automated processing. Finally, because this type of patterns is not designed to be integrated into the users’ systems but to be implemented manually, the problem of incorrect implementation (the most important source of security problems) remains unsolved.

Aspect Oriented Development is another paradigm investigated by researchers in an attempt to capture the specialized expertise of security engineers and to make it available for non-expert developers. The popularity of aspect-oriented approaches has originated research

on its application to the field of security. Unfortunately, aspects are mainly an implementation technique and not suitable to provide and manage security solutions as a whole.

The **dynamic validation** and adaptation of S&D mechanisms is another necessary activity when breaches or threats are detected. The need for run-time monitoring has been argued extensively in the scientific literature (Feather, 1998; Feather, 1995; Robinson, 2002) and there have been several strands of research focusing on different system aspects.

Research in run-time security monitoring has produced techniques focusing on monitoring security policies (Ko, 1996; Damianou, 2001; English, 2004; Serban, 1996). Typically, these policies refer to security conditions at an infrastructure layer (e.g. network connections) and fail to reflect application level and context specific security requirements. Furthermore, techniques in this area monitor security conditions in isolation and, therefore, are unable to detect security breaches and threats which arise due to interactions between different functional and dependability requirements and security requirements, or breaches and threats which arise due to violations of conditions that relate to non infrastructural system aspects (e.g. negligence at the level of system users).

Research in general requirements monitoring (Cohen, 1997; Feather, 1998; Feather, 1995; Robinson, 2002; Robinson, 2004) has been concerned with the specification of requirements for monitoring, the transformation of these requirements into events that can be monitored at run-time, and the development of mechanisms to support the generation and monitoring of these events. Most of the existing techniques express requirements in some high level formal specification language and assume the manual transformation of requirements onto monitorable events (Feather, 1995; Robinson, 2004). Due to the lack of systematic support for it, however, this manual transformation has a large cost that prohibits the wide use of the relevant techniques. Furthermore, existing techniques provide limited support for system adaptation following the identification of requirement breaches.

Research in dynamic program verification (Gruber, 2005) has focused on the development of generic components for program monitoring platforms including program instruments that can generate the events required for monitoring —e.g. jMonitor (Karaoman, 2004)— and general monitoring algorithms —e.g. algorithms for checking formulas expressed in temporal logics (Thati, 2004)— but does not directly support the monitoring of higher level system requirements such as security and dependability requirements.

Yet, existing monitoring techniques do not adequately support the diagnosis of the reasons underpinning run-time violations of S&D requirements nor inform system adaptation to ensure that violations will not re-occur. They also fail to support the specification of end-user personal and ephemeral S&D requirements, the automatic assessment of whether or not such requirements can be monitored at run-time, and the transformation of these requirements onto monitorable events. Furthermore, they fail to address the need for identification of scenarios of potential security threats and the translation of these scenarios into monitorable events that would allow the development of pro-active techniques for protecting security.

Challenge 5. Capturing the expertise of security engineers. *Can we take advantage of the recent developments in technologies of security engineering, run-time monitoring, semantic description and self-configuration that are able to capture some of the expertise of security engineers and make it available, supported by automated tools, to the AmI ecosystems?*

THINKING THE SERENITY: A MANIFESTO

The SERENITY approach aims at integrating the best of the aforementioned approaches in order to overcome the problems that have prevented them to succeed individually.

SERENITY Claim 1. *A S&D engineering framework will be based on the notion of **Security and Dependability Pattern**. A pattern describes a recurring problem that arises in a specific context and presents a well-proven generic scheme for its solution. S&D patterns will feature*

a precise functional description of the underlying security solution; a full semantic description of the security requirements they address; and descriptions of any preconditions or assumptions that govern the deployment of the pattern.

We have developed an enhanced concept of Security and Dependability (S&D) Pattern to represent existing security solutions. In fact, the term “security pattern” has been previously used in the literature to denote informal and ad-hoc security measures, mostly at a managerial level. Our semantic-based formal approach facilitates the definition of specific profiles and solutions for different environments.

SERENITY Claim 2. *The combination of patterns will not be let to users and designers alone: Security and Dependability Integration Schemes will be an extension of S&D Patterns designed to represent complex security solutions built by combining other solutions.*

The second key innovation factor of SERENITY is the support for the run-time pro-active and reactive identification of potential threats and attacks of implemented security solutions, the timely (where possible) adaptation of attacked or under-threat applications, and the amendment of S&D patterns and integration schemes to address weaknesses identified during their deployment through appropriate evolution mechanisms.

One time too often designers have focused on one of their stakeholder: either system developers or end-users. The complexity of AmI scenarios needs an approach that can talk to both of them rather than either of them. We need a dual-interface approach.

SERENITY Claim. 3. *The S&D Engineering methodologies will support security engineers in the development of formally-proven security solutions, and the specification of these solutions as S&D Patterns. They will also support system engineers and end-users in the specification of their security requirements, the selection of the most appropriate combination of the available security solutions regarding the security requirements and the context, and the integration of these security solutions in their systems.*

This dual-interface approach is based on the belief that it is not realistic to expect that system engineers and end users will ever become security experts.

Automating the processing of semantic information represents a big challenge and a promising line of work for the resolution of many relevant problems. Existing expertise in the application of semantic information, i.e., formal specifications and reasoning methods, to the field of information security ensures the suitability of this approach, which will be the basis of many of the SERENITY methods and tools.

SERENITY Claim 4. *A sound solution to S&D engineering will develop semantic models to specify the semantics of security requirements, patterns and properties. Automated tools for classification, selection, and composition of security patterns will take advantage of this semantic information allowing the automated integration of security patterns implementing the specified security and reliability requirements.*

Semantic descriptions will not only help in the selection of the right pattern, but will be also essential for pattern composition and system analysis. Furthermore, semantic descriptions will provide the foundation for reuse and secure interoperability of these patterns.

The SERENITY approach will materialised our claims in an integrated framework to enable end-users, system engineers and security engineers collaborate seamlessly to build and operate dependable software-based systems for AmI environments with inherent and adaptable security capabilities. This framework will be based on an integral model of security, therefore considering not only static issues (related to secure systems development) but also dynamic aspects (related to monitoring and controlling the software at runtime). The different elements composing this framework will foster the development of new applications, computing paradigms supporting services, in current and future complex, heterogeneous and dynamic computing and communication infrastructures by eliminating the security and dependability problems they bring about.

SERENITY Claim 5. *Static aspects of the S&D engineering framework deal with the provision of appropriate abstractions of S&D solutions with the objective of capturing the specialized expertise of security engineers and making it available for non-experts, while at the same time providing the basis for automated synthesis and analysis of the security and dependability of systems.*

An important issue is the ability to include abstractions of the different collaboration schemes and software and hardware architectures that can occur at runtime in very dynamic and heterogeneous scenarios. In this area, SERENITY's S&D patterns provide the basic building blocks of security and dependability solutions along with a formal characterisation of their behaviour and semantics that will enable the use of these solutions over a wide range of heterogeneous computing and communication infrastructures and devices. On the other hand, SERENITY's integration schemes provide a framework for systematically instantiating and integrating these building blocks in applications composed of statically or dynamically collaborating components that operate in mobile and highly dynamic ICT infrastructures.

Whereas dynamic composition is an important problem, it is only one side of the coin. The need for run-time requirements monitoring has been argued extensively in the scientific literature (Feather, 1998; Feather, 1995; Robinson, 2002) and there have been numerous strands of research on broader run-time monitoring of applications without, however, focusing on security issues, with a few exceptions (Ko, 1996; Serban, 1996). The need for runtime monitoring and verification becomes even more important, when it comes to systems that incorporate autonomous collaborating components such as systems that incorporate web-services or components which collaborate via peer-to-peer architectures (Robinson, 2004; Mahbub; 2004; Mahbub; 2005).

SERENITY Claim 6. *A complete S&D Engineering framework will be able to cope with dynamic aspects and to provide mechanisms for monitoring of security requirements at run-*

time and detecting definite or potential deviations (a.k.a. threats) from such requirements. The framework will support the engineers in the process of amending S&D patterns and integration schemes to address weaknesses identified during their deployment through appropriate evolution mechanisms.

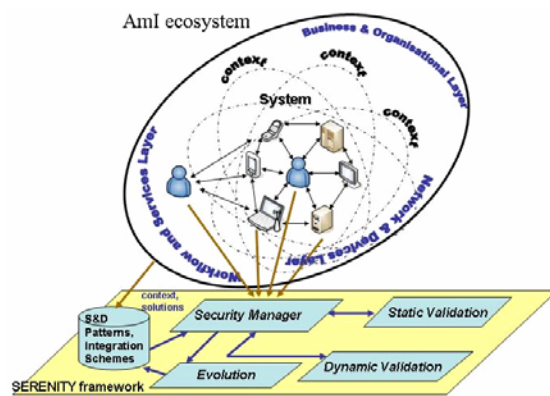
IMPLEMENTING SERENITY

As shown in Figure 1, the SERENITY framework that we envisage is a suite of integrated **software tools** including:

- A *Security Manager* supporting the dynamic selection, adaptation, integration and monitoring of S&D mechanisms that are available in AmI ecosystems in order to address the security requirements which arise in specific operational settings. Selection, adaptation and integration are performed as specified by the S&D patterns and integration schemes that are applicable in specific operational settings. The security manager is supported by:
 - o *Static validation tools* supporting the analysis of compliance of S&D solutions offered by the different components of an AmI ecosystem to inform their selection and integration.
 - o *Dynamic validation tools* supporting the monitoring of AmI systems and their active S&D solutions to detect breaches and potential threats of security and inform the reconfiguration of S&D solutions to recover from such breaches or prevent them in cases where this is possible.
- *Evolution tools* recording and analysing the operational aspects of S&D solutions as identified by run-time monitoring to identify gaps and ways of improving these solutions and support the amendment of the S&D patterns and integration schemes that underpin them.

Every instance of the SERENITY framework should provide interfaces to allow the use of some of its functionalities by external elements. These external elements might be other instances of the SERENITY framework running in other systems or other types of SERENITY-aware elements. In particular, the framework should support the publication of the security mechanisms used in the specific instance and also the publication of some of the runtime monitoring results.

Next subsections show how the claims in section 4 can be materialized. The complete Section 5.1 covers claims 3 and 5. Subsections 5.1.1 and 5.1.2 cover claims 1 and 2 respectively. Subsection 5.1.3 relates to claim 4. Finally, Section 5.2 deals with claim 6.



Formatted: Indent: First line: 0 pt

Figure 1: The SERENITY framework.

Static Aspects: Abstraction of S&D Solutions

The enhanced notion of Security and Dependability (S&D) Pattern, together with the new concept of Integration Schemes provide important advantages over existing solutions, especially in complex, heterogeneous and dynamic scenarios, in the goal of capturing the specialized expertise of security engineers and making it available for non experts, while at the same time providing the basis for automated management and analysis of security aspects. Moreover, these concepts are very appropriate in order to deal with applications where parts

of the software, communication infrastructures and hardware devices are not under the control of one part and all particular combinations of these can not be foreseen by the S&D engineers.

S&D Patterns

S&D Patterns represent security and dependability solutions. They are materialised as files that contain models that could be described using formal and non-formal languages (e.g. XML and logic-based language) to capture the expertise of security engineers with the final objective of being processed by automated means. These models provide semantics and behavioural indications that are necessary to describe particular security or dependability contexts that could occur in an AmI Environment (i.e. public administration, nomadic context, business, etc.) using well defined rules, methods and specification.

It is important to emphasize that the idea behind the SERENITY approach is that systems will have security requirements and that there will be ways to satisfy these requirements automatically by using the appropriate combination of the available solutions. With this respect Figure 2 shows a partial conceptual diagram of the SERENITY framework operation. Instead of developing a sophisticated “intelligent” mechanism that can “synthesize” a specific solution for the problem in the given context, the SERENITY approach is to find a solution within a set of predefined ones. In order to automatically select the most appropriate solution and also to be able to integrate that solution in our system, we need to have on the one hand, very precise descriptions of the solutions, and on the other hand very flexible descriptions, which can be accommodated to our specific context. The Security Requirements Analysis and the Pattern Integration modules shown in the figure are used for this purpose.

In order to represent the solutions, S&D Patterns (but also Integration Schemes described in the next section) contain precise specifications, which include:

- The representation of the solution itself (this is the S&D Pattern in Figure 2) corresponding to the behavioural description.

- The pattern semantics, which are related to the semantics of the security properties provided and the restrictions imposed by the solution, including:
 - o The representation of what is achieved by using the pattern (properties provided);
 - o The proofs done to validate that the description of the pattern is correct. This can be seen as “guarantees” from the provider;
 - o The representation of the mechanisms that are used to provide trust in the pattern and the mechanism. At least, we will for sure want to know who developed the mechanism and that the mechanism has not been altered.
 - o The representation of the restrictions imposed on the context. That is, in which situations can we use this pattern? ;
 - o The representation of the possible variants or ways to adapt the mechanism. Usually, these will be parameters. For instance, the length of the key used in an encryption mechanism, the sender and recipient of a message in a protocol, etc.;
 - o The representation of the ways to monitor the behaviour (this is specific to the AmI scenarios);
 - o Any other context-specific information (efficiency, legal compliance, etc.).

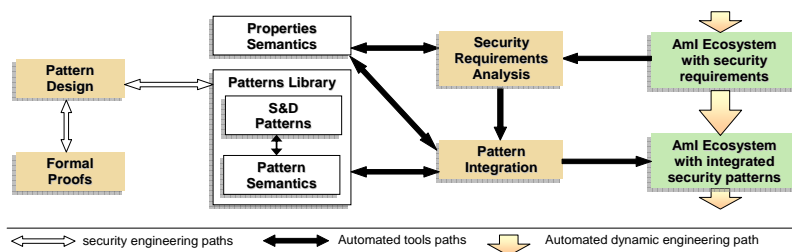


Figure 2: SERENITY framework operation (partial).

We recognize the complexity of providing complete and adequate semantic descriptions for software elements. However, it is our view, validated by our early experiments (Maña, 2003), that while in the general case it seems unfeasible to define mechanisms that are able to capture the semantics of a software element (at least with the current state of the art), in the

case of security semantics it will be possible to develop a suitable solution. The reason is that security is a much more restricted field, where we can rely on a limited number of precise descriptions.

Integration Schemes

S&D patterns, as described above, are assumed to cover only one specific security or dependability problem (e.g. specific security issue concerning the network level), but it is quite usual that more than one solution (and therefore S&D patterns) must be combined in order to fulfil a specific set of requirements. Integration Schemes are the means to overcome and to figure out this problem that is not easy by itself as it is not always possible to foresee the result of merging different security solutions (whatever they are).

The approach followed is again a practical one as in the case of S&D Patterns: Integration Schemes will capture proven known ways to securely integrate different solutions. The added value is that they represent a practical and easy approach to facilitate the task of finding the best and more secure way to integrate several solutions (patterns) in order to create complex S&D solutions.

Expressing semantics about security properties

The most important component of the semantic description of a security solution as an S&D Pattern is the reference to the security properties provided by this solution. Therefore, the mechanism used to describe such semantics has to be carefully designed.

We start from the assumption that there will be different definitions for the basic security properties. There are many reasons for this: for instance, legislation, cultural etc. Let's illustrate this with an example. Assume Willie wants to use a specific confidentiality solution $S_{Conf_{ACME}}$ from provider ACME. In order to correctly use this solution, Willie needs to be able to (i) univocally identify the properties (i.e. "confidentiality") declared by ACME; and

(ii) to “understand” the meaning of “confidentiality” as defined by ACME. In particular, Willie needs to know if the meaning of “confidentiality” as defined by ACME is enough to fulfil the confidentiality requirement as defined by him.

The first point can be easily solved by applying general naming scheme, following the common practice of defining namespaces. For instance, in such scheme properties could be named “Confidentiality.GSM.Mobile.ETSI”, “Confidentiality.Crypto.Java.Sun” “Confidentiality.ISO”, or “Confidentiality.ACME”.

The second part is a bit more difficult, but basically, we can use three approaches:

- **Standardization**, that is, establishing a standard classification of properties with fixed and predefined semantics;
- **Implicit semantics**, which implicitly describe the properties as changes in the state of the system, usually as a set of pre and post conditions.
- **Explicit semantics**, which provide means for the description of the semantics of the properties, therefore representing an open and interoperable solution.

The description of security properties in SERENITY is based on the third approach. S&D pattern creators can define their own properties or refer to properties that have been semantically described by third parties. In fact, we foresee that standardization bodies will semantically define sets of properties related to their areas of interest. Therefore, when describing their solutions as S&D patterns, security engineers will have to model the semantics of their solutions. It's at this point when the use of the Ontological Engineering applied to the patterns will make them more reusable and machine understandable. The use of ontologies can help in the description of a “formal, explicit specification of a shared conceptualization” (Gruber, 1993), and reasoning mechanism can be applied to this model, inferring new knowledge from it. Patterns, and in particular the properties they refer to, will be described using formal ontology languages (like OWL or WSML) to ensure maximizing

the automation of the knowledge lifecycle and to achieve semantic interoperability between different patterns as heterogeneous information resources.

The most basic form of the explicit semantics approach consists on the definition of mechanisms to relate the different properties. For instance, in the previous example two properties are involved: on one side Confidentiality.ACME (the one provided by the solution) and on the other side Confidentiality.Willie (the one required by Willie's system). The relation between properties is harder to express but we can start by using "implication".

Consider the following rules:

(1) Confidentiality.ACME \rightarrow Confidentiality.ISO

(2) Confidentiality.ISO \rightarrow Confidentiality.Willie

Rule 1 means that if a solution provides confidentiality according to ACME, then it also provides confidentiality according to ISO. Informally, we could say that the definition of confidentiality by ACME is more restrictive than the one by ISO. Rule 2 states that solutions that provide confidentiality according to ISO provide also confidentiality according to Willie. Then, from the analysis of the previous rules we can determine that Willie's confidentiality requirement can be fulfilled using the $SConf_{ACME}$ solution from ACME.

Trust mechanisms

We envisage a scenario where a library composed of several security solutions from different sources, is available to a system. In this regard, and considering the previous example, the owner of the system (ACME) would like to have the following guarantees:

- that the $SConf_{ACME}$ S&D pattern has been created by ACME;
- that the semantic description of $SConf_{ACME}$ has been created by ACME;
- that the semantic description of $SConf_{ACME}$ corresponds exactly to this solution (version, build, etc.); and

- that Willie can understand the properties provided by $SConf_{ACME}$ as described in the previous section.

Additionally, Willie needs to know how ACME has analysed and tested the pattern and which proofs can ACME provide to support the semantic descriptions of $SConf_{ACME}$.

To address these requirements, a trust infrastructure must be in place. This infrastructure will be based on digital certificates, digital signatures, etc. but also on semantic descriptions, in a similar way to the one used to describe the semantics of attribute certificates (Yagüe, 2004). This approach facilitates interoperability and supports the commercialization of the security solutions described as S&D patterns.

Dynamic Aspects: Run-time Monitoring

Despite analysis which might have been taken to ensure that the specification of a system includes a precise model of the security requirements that need to be realised, and that the design of a system includes components that can realise these requirements, it is in principle difficult to guarantee that such an analysis will provide the full proof that is required. This is because requirements and design models cannot be guaranteed to be complete. When present, incompleteness in requirements specification and design models makes it impossible to prove with certainty that security requirements will be addressed during the operation of a system. Such proofs are also difficult due to the fact that assumptions about the behaviour of actors in system environments cannot be verified, including assumptions related to security. Furthermore, static verification of the satisfiability of security requirements may be intractable due to either infinite-state system models or the presence of a prohibitively large system state space that must be explored (Bensalem, 2004). These limitations are exacerbated by the unbound and dynamic nature of AmI environments and applications.

In SERENITY, we envisage that it will be possible to develop a framework for monitoring security requirements building upon existing approaches and expertise on runtime requirements monitoring for highly distributed service centric systems that is available in the consortium (Mahbub; 2004; Mahbub; 2005). The extensions and customisations of these approaches will be aimed at introducing capabilities for: (a) specifying monitorable security requirements, (b) identifying conditions that may indicate potential threats to security requirements, (c) monitoring not only deviations from security requirements but also potential threats to them, and (d) recovering from detected deviations or taking dynamically additional protective measures against threats. On the one hand, the development of runtime monitoring will have to address limitations of existing techniques, notably the lack of automatic support for the generation of the events which are used in monitoring, transformation of security requirements into patterns of such events and identification of conditions that may indicate potential threats. On the other hand, the development of support for recovery will have to address the absence of reasoning mechanisms allowing the identification of the source of detected security deviations (diagnosis) and the selection of appropriate recovery actions.

SERENITY's runtime monitoring mechanisms will be complemented by reasoning mechanisms that will analyse information about detected deviations from security requirements at run-time in order to identify gaps in the static requirement and design specifications of a system that did not allow the detection of the potential for these deviations by static analysis. The analysis of detected runtime deviations will inform the process of S&D patterns and integration schemes evolution.

AN APPLICATION SCENARIO

This section describes one of the envisaged application scenarios and shows how the services of the SERENITY framework can be used to support the provision of security at both

development time and at runtime. The scenario focuses on Business Information Systems based on Web Services.

Modern enterprise information infrastructures are characterised by evolving from stand-alone systems supporting a well-defined and static set of business processes and interacting in a controlled way to open system landscapes providing and integrating services that can be flexibly composed to adapt to rapidly changing business needs. The infrastructure is driven by the evolving business processes and tightly integrates a set of services tailored to the situation at hand. Such an Enterprise Service Architecture integrates services and components from different owners, is open to new services, and optimises investment needs by being able to integrate legacy services as well. In such settings, the owner of the business has to rely on the functionality and properties of the components that are not completely under his control. This raises particular challenges when it comes to meeting requirements on auditing, monitoring, and reporting as given by recent regulations, e.g., Sarbanes-Oxley or Basel II: The owner of the business is responsible for compliance to the regulations, but has to enforce the requirements even for components and services not under his control.

This application scenario consists on an enterprise software application built through orchestration of set of services that are made available through a service-oriented architecture. The service architecture is spanning over several organisations (companies), thus the set of services is not centrally controlled. A typical example is a supply chain application, consisting, from a simplified point of view, of tasks quotation (including subtasks such as request-for-quote, submit-quote, place-order etc.), order processing (including subtasks such as check-order, schedule-manufacturing, purchase-components etc.), and order fulfilment (including subtasks such as manufacturing, shipping, invoicing etc.). Each of the subtasks is provided by a service (that might be structured itself, since each subtask represents a complex process, e.g., accessing several databases and performing several actions), potentially owned

by different entities. Distributed ownership occurs, for instance, if subtasks are offered by specialist providers (e.g., invoicing or procurement is outsourced by a company).

Let's assume that the services offering the subtask functionality (functional services) only satisfy basic security properties, like local (from a service point of view) authentication mechanisms – the invoicing service includes a login function – and basic authorisation – a user logging in to the invoicing service has to be assigned to an appropriate role. Let's further assume that the application owner has advanced security requirements, since, for example, the application may be subject to auditing and reporting requirements. These requirements may include, for instance, a non-repudiable event log containing each invoice issued by the invoicing service, advanced authentication mechanisms for triggering the manufacturing service, or integrity protection for the communication between services.

SERENITY at Work 1. *The SERENITY framework can be used to implement the advanced security requirements at the time of the design of the application by taking advantage of available advanced patterns and integration schemes or by representing the specific solutions required by the system as new S&D Patterns and Integration Schemes. The framework will be then used at runtime providing dynamic reaction capabilities and monitoring support.*

At design time, we assume that the application developer is aware of the functional services he is about to orchestrate into his application, and the basic security mechanisms and properties they offer. This is a valid assumption for common development environments. Note that we do not require the security properties being specified compliant to the SERENITY framework: requirements imposed by integration schemes may later be verified informally. However, the approach provides stronger results if the security properties of the functional services can be mapped to SERENITY pattern descriptions, since this provides a stronger evidence for the claimed properties. Such a mapping can be achieved through usage of the SERENITY framework in the development of the functional services.

SERENITY at Work 2. *Application developer specifies the advanced requirements through the SERENITY framework. The framework offers the languages and tools necessary to do so, on different levels of abstraction matching different types of properties. In our example, we have the integrity requirement referring to the communication, i.e., network and devices level, and the event log and non-repudiation requirement referring to a workflow and service level. The SERENITY framework then assists the application developer in selecting those patterns out of the available library that are likely to satisfy the requirements. For each requirement, the framework may suggest several patterns: for integrity protection, it might suggest using an integrity-preserving communication protocol or applying digital signatures to the messages that are communicated. Formal specifications attached to the patterns allow analysing their properties (e.g., environment assumptions) and matching them with the application specific requirements.*

For instance, a pattern for event logging might assume a reliable clock. Patterns also indicate dependencies: a digital signature pattern requires a PKI pattern and the properties of the digital signature pattern depend on properties of the PKI pattern selected.

The patterns selected through the framework result in the design of a security architecture (i.e., a set of security services and their properties). The example shows that typically patterns are selected for each property in isolation.

SERENITY at Work 3. *In order to avoid undesired interference between the patterns, the resulting architecture has to satisfy restrictions imposed by integration schemes. An application can only be considered secure if all applying integration schemes are satisfied. Indeed integration schemes may also impose assumptions on trustworthiness.*

For example, if we choose an “SSL pattern” for integrity protection and a “signature pattern” for non-repudiation, an integration scheme will require the use of different cryptographic keys for the two patterns. If we choose to implement the event log pattern

locally attached to the invoicing service, this will only be allowed if the invoicing service is owned by the application owner (because the application owner is responsible for reporting, and obviously trusts himself). If the invoicing service is run by a different owner (in case of outsourcing), the integration scheme either requires a trust establishment mechanism (again represented by different patterns) or a non-repudiation mechanism (another pattern).

SERENITY at Work 4. *If the solutions suggested by the system are not considered appropriate by the application designer, there are several actions that can be taken:*

- *enter an iteration loop in the framework, by selecting a different architecture (e.g., by choosing a “trusted third party pattern”) subject to different restrictions, dependencies, and integration schemes;*
- *acquire new solutions (patterns and/or integration schemes) from third parties; or*
- *the developers can even create new security solutions and then use the “security engineering” interface of the SERENITY framework to validate their solutions and to specify them as S&D Patterns and Integration Schemes. After these are incorporated into the library of available solutions, they can go back to the “system engineering” interface in order to analyse their system in the light of the new solutions developed.*

The example above illustrates the use of the SERENITY framework in the design of a secure service-oriented business application. An advanced instantiation of the framework will provide service implementations realising the patterns, enabling the dynamic configuration of application-specific security architectures. The process at design time remains the same, with the framework additionally offering the security services described by the patterns used and integrating them through the service infrastructure. Through such architecture, the framework additionally offers the necessary runtime support.

A variant of the previous scenario consisting on the establishment and execution of virtual organisations illustrates the runtime support aspects of the framework. A virtual organisation

consists of entities dynamically setting up business relationships to follow a common goal that the individual entities cannot achieve by themselves. In order to set up the organisation, the entities negotiate properties, including security functionalities, policies, and configurations. The SERENITY framework supports virtual organisations by establishing an appropriate security solution according to the security capabilities of the different entities involved and the environment of the virtual organisation, and by adapting it, if necessary, to changes within the lifetime of the virtual organisation.

SERENITY at Work 5. *The runtime validation mechanisms of SERENITY can monitor a range of security issues dynamically. SERENITY monitors can, for instance, be used to monitor the satisfiability of the assumptions made by specific security patterns about the behaviour of the parties involved in them.*

Consider, for instance, a pattern realising an interaction of fair-exchange where a service A places an order for digital goods that can be delivered electronically (e.g. a business report) to a supplier service B with a partial signature of it as proof and, following the receipt of the goods from B, it is expected to deliver its full signature to B that can be used to obtain payment. In this case, SERENITY monitors can check the behaviour of the involved parties that guarantee fairness. They can, for instance, check that B does not produce a valid signature of A without having received it from A earlier and that A does send the expected signature to B following the dispatch of the goods. Run-time monitoring can be also deployed to detect attacks that can lead to denial of service by the parties involved (e.g. crashing suppliers by sending them extremely large inputs on purpose, flooding suppliers with a huge number of fake requests from a specific "manufacturer" etc).

The supply chain use case above can be extended to virtual organisations as follows. Consider the different roles taking part in the scenario, including supplier, manufacturing, invoicing, administration, notary etc., being virtualised through services within the

SERENITY framework embedded in a service architecture. The manufacturer service, as the coordinator of the virtual organisation to be established, issues a request for quotes that can be answered by all supplier services available. The incoming quotes are evaluated by the manufacturer according to several criteria, including the security properties offered by the respective supplier. The manufacturer selects the best offer and establishes the organisation with its providing service. During lifetime of the virtual organisation the manufacturer might decide to add an additional supplier, giving rise to additional or modified security requirements, e.g., keeping the confidentiality of the quotes of the individual suppliers.

The SERENITY framework supports the setup of the initial security architecture and its adaptation to evolving requirements. In its initial phase, with only one supplier given, the main security requirement of the manufacturer is the provision of a non-repudiable event log from the supplier (cf. design time use case above). If the services are equipped with SERENITY compatible specifications of their security abilities (e.g., through references to security patterns applying), the framework can decide (through the use of integration schemes) whether the required security properties can be met with the given service orchestration. In our example, the following cases may happen (we only consider security and assume that all other criteria applying to the supplier service are met):

1. The supplier provides a non-repudiable event log on its own, according to the appropriate S&D pattern of the library. Through the framework, the manufacturer detects that the pattern is sufficient to meet his requirement, and engages with the supplier in the virtual organisation.
2. The supplier provides an event log according to an event log pattern, but no non-repudiation mechanism. The framework offers an integration pattern expressing that a non-repudiable event log can be achieved through the integration of a trusted third party service. If such a service is part of the service universe, it can be integrated, configured according to the integration scheme, and virtual organisation establishment continued. The SERENITY

framework can support in detecting and integrating such a service by providing service references, or even offering such a service on its own.

3. The supplier provides no logging mechanism at all. Then, another integration scheme may apply, e.g., requesting an advanced trusted third party mechanism.
4. If no patterns and integration schemes offered by the SERENITY framework apply, the manufacturer service can conclude that his security requirements cannot be met (at least not through the given framework instantiation) and reject the quote.

The runtime monitoring mechanisms of SERENITY can further address non repudiation in cases (2)-(4). The basic capability offered by monitors in these cases would be the recording and provision of an event log, e.g. by intercepting and logging inter-service communication events (order placing and order acceptance events). This task may be carried out by monitors acting on behalf of the manufacturer in case (2) above in order to provide information that can be cross-checked against the event log provided by a supplier, or by monitors acting a trusted third parties in case (3) above. The provision of this capability can be deemed sufficient for engaging in interactions that would have normally been avoided, as in case (4) above.

If a second supplier enters the virtual organisation, additional security requirements apply: the individual quotes issued by the suppliers should be kept confidential. The virtual organisation host is aware of the additional requirement either through this being encoded as part of the manufacturer service, or through the framework. In the latter case, the manufacturer signals the change in the organisational structure to the framework, which checks for integration schemes applying. Such integration scheme is likely to suggest a modification of the security architecture (e.g., by adding additional services, or changing configurations of existing services). The modification is executed as indicated above, either through the virtual organisation itself, or through the SERENITY framework.

In order to show the applicability of the SERENITY approach and framework to the broad spectrum of systems appearing in the AmI ecosystem realm, the eBusiness application scenario is complemented by three additional scenarios showing different characteristics: (1) mobile communications, with absence of central control and emphasising on the need to adapt security policies with respect to location and communication infrastructure context, (2) sensor networks, having to deal with limited capabilities of devices and their public exposure, (3) eGovernment, with applications subject to laws and regulations, and the demand to integrate in existing platforms showing the scalability of the SERENITY approach.

CONCLUSIONS

In the next future and especially in a society where information and other digital assets are seen as a high-value commodities, the security and dependability challenges will arise from complexity, ubiquity and autonomy of computing and communications as well as from the need for mobility, dynamic content and volatile environments. In the next 10 to 15 years, the communication frameworks will have to support operating scenarios where it will not be possible to rely on a-priori well defined systems having a pre-established security manager.

We have given the challenges ahead to the development of a S&D Engineering framework that will release expertise and solutions to allow people to build secure and dependable applications for AmI environments, and to monitor the security and dependability of these applications in different operational contexts. The different methods and tools constituting that we have advocated will be consistently incorporated into an integrated framework that considers real-world requirements as the drivers for more secure and dependable systems.

Surprisingly enough this is not just a research manifesto. Our ideas are backed up by a concrete R&D project proposed by a number of leading academic partners (City University of London, Fraunhofer Institute for Secure Information Technology, Katholieke Universiteit

Leuven, Università di Trento, University of Aegean and University of Málaga) and key industry players (Engineering Ingegneria Informatica S.p.A, Athens Technology Center, ATOS Origin, Deep Blue, NOKIA, SAP AG, Security Technology Competence Centre, Stratégies Telecoms & Multimedia and Thales) and supported by the European Commission (Unit: ICT Trust and Security, Area: Towards a global dependability and security framework) under the grant IST-27587. Time will tell whether such efforts and expertise will be well invested. We are confident it will be so.

SPECIAL THANKS

The authors wish to express their gratitude to all members of the SERENITY Consortium and the European Commission for the support and the contributions provided for the definition of this chapter.

REFERENCES

- Anderson, R. Security Engineering: a guide to build dependable systems. Wiley and Sons. 2004.
- Banavar, G., & Bernstein, A. Software infrastructure and design challenges for ubiquitous computing applications, Communications of the ACM, vol. 45(12), pp. 92-6, Dec 2002.
- BEA. BEA WebLogic Security Framework: Working with Your Security Eco-System. Retrieved May 2003 from <http://www.bea.com>
- Bensalem S, et al., 2004. Testing conformance of real-time software by automatic generation of observers, 2004. Proceedings of 4th Workshop on Runtime Verification (RV '04). Retrieved January 2005 from <http://ase.arc.nasa.gov/rv2004>
- Blakley, B., Heath, C., & members of The Open Group Security Forum. Security Design Patterns (SDP) technical guide. <http://www.opengroup.org/security/gsp.htm>. The Open Group. 2004.
- Cheng, B.H.C., Konrad, S., Campbell, L. A. & Wassermann, R. Using Security Patterns to Model and Analyze Security Requirements. High Assurance Systems Workshop (RHAS03), Monterey Bay, CA, USA, September 2003.

- Cohen D. et al., Automatic Monitoring of Software Requirements. Proc. of 19th Int. Conf. on Softw. Engineering, 1997
- Damianou, N., Dulay, N., Lupu, E., & Sloman, M., The Ponder Policy Specification Language, POLICY 2001, LNCS, pp. 18-38, Springer-Verlag Berlin Heidelberg 2001.
- English C., Terzis S., & Nixon P. Towards Self-Protecting Ubiquitous Systems Monitoring Trust-based Interactions, Proceedings of UbiSys '04, 2004
- Fayad, M., Johnson, R., & Schmidt, D. C., eds., Building Application Frameworks: Object-Oriented Foundations of Framework Design. Wiley & Sons, 1999.
- Feather M. et al. Reconciling System Requirements and Runtime Behaviour, Proc. of 9th Int. Work. on Software Specification & Design, 1998.
- Feather M., & Fickas S. Requirements Monitoring in Dynamic Environments, Proc. of Int. Conf. on Requirements Engineering, 1995.
- Focardi, R. & Rossi, S. Information flow security in dynamic contexts, in Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW 15, 2002.
- Gruber, T. R., Toward principles for the design of ontologies used for knowledge sharing, 1993. In Formal Ontology in Conceptual Analysis and Knowledge Representation, Nicola Guarino and Roberto Poli, editors, Kluwer Academic Publishers.
- IBM's Security Strategy team. Introduction to Business Security Patterns. An IBM White Paper. Available at <http://www-3.ibm.com/security/patterns/intro.pdf>. 2004.
- Karaorman M., & Freeman J., jMonitor: Java runtime event specification and monitoring library, 2004. Proceedings of 4th Workshop on Runtime Verification (RV '04). Retrieved January 2005 from <http://ase.arc.nasa.gov/rv2004>
- Kienzle, D. M., & Elder. M. C. Final Technical Report: Security Patterns for Web Application Development. 2003. Retrieved March 2005 from <http://www.scrypt.net/~celer/securitypatterns/final%20report.pdf>
- Ko C. Execution Monitoring of security-critical programs in a distributed system: a specification-based approach, PhD Thesis, University of California at Davis, 1996.
- Llewellyn-Jones, D., Merabti, M., Shi, Q., & Askwith, B. An Extensible Framework for Practical Secure Component Composition in a Ubiquitous Computing Environment. Proceedings of International Conference on Information Technology, Las Vegas, USA, April 2004.

- López, J., Maña, A., Pimentel, E., Troya, J.M., & Yagüe, M.I. Access Control Infrastructure for Digital Objects. Int. Conference On Information and Communications Security 2002. Springer-Verlag. LNCS. 2513. Singapore. 2002
- Mahbub K., & Spanoudakis G. 2004. A framework for Requirements Monitoring of Service Based Systems, Proc. of the 2nd International Conference on Service Oriented Computing - ICSOC 2004, New York.
- Mahbub K., & Spanoudakis G. 2005. Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience, 3rd International IEEE Conference on Web Services (ICWS 2005).
- Maña, A., Montenegro, J.A., Ray, D., Sánchez, F., & Yagüe, M.I. Integrating & Automating Security Engineering In UML. IASTED International Conference on Communication, Network and Information Security (CNIS'03). Acta Press. New York (USA). 2003
- Mantel, H. On the composition of secure systems. Proc. of 2002 IEEE Symposium on Security and Privacy, 2002.
- Object Management Group (OMG), Common Object Request Broker Architecture (CORBA) Core Specification. Retrieved July 2004 from <http://www.omg.org>
- Robinson, W. Monitoring Software Requirements using Instrumented Code. In Proc. of the Hawaii Int. Conf. on Systems Sciences. 2002.
- Robinson, W. Monitoring Web Service Requirements, 2004. Proceedings of 4th Workshop on Runtime Verification (RV '04). Retrieved January 2005 from <http://ase.arc.nasa.gov/rv2004/>
- Román, M., Hess, C. K., Cerqueira, R., Ranganathan, A., Campbell, R.H., & Nahrstedt, K. Gaia: A Middleware Infrastructure to Enable Active Spaces. Proceedings of IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.
- Romanosky, S. Enterprise Security Patterns. Proceedings of the Seventh European Conference on Pattern Languages of Programs. 2002.
- Schmidt, D. C. Patterns, Frameworks and Middleware: Their Synergistic Relationships. Invited talk at IEEE/ACM International Conference on Software Engineering, Portland, Oregon, May 3--10, 2003.
- Schumacher, M. Security Engineering with Patterns: Origins, Theoretical Model and New Applications. Springer, 2003
- Serban C, & McMillin B. Run-Time Security Evaluation (RTSE) for Distributed Applications, 1996 IEEE Symposium on Security and Privacy, 1996.

- Shi, Q., & Zhang, N. An effective model for composition of secure systems, *Journal-of-Systems-and-Software*, vol. 43(3), pp. 233-44, Nov. 1998.
- Thati P., & Rosu G., Monitoring algorithms for metric temporal logic specifications, *Proceedings of 4th Workshop on Runtime Verification (RV '04)*. Retrieved January 2005 from <http://ase.arc.nasa.gov/rv2004/>
- Tryfonas, T; Kiountouzis, E. & A. Poulymenakou. Embedding security practices in contemporary information systems development approaches, *Information Management & Computer Security*, Volume 9, Number 4, pp. 183-197. 2001.
- Wimmel G. & Wisspeintner, A. Extended Description Techniques for Security Engineering. *Trusted Information, The New Decade Challenge*, IFIP TC11 16th "International Conference on Information Security (IFIP/Sec'01)", Paris, Michel Dupuy, Pierre Paradinas (ed.), pp. 470 - 485, Kluwer Academic Publishers, 2001.
- Yagüe, M, Maña, A. & Sanchez, F., Semantic Interoperability of Authorizations. In *Security In Information Systems*, *Proceedings of the 2nd International Workshop on Security In Information Systems, WOSIS 2004*.
- Yang, H., Luo, H., Ye, F., Lu, S. & Zhang, L. Security in Mobile Ad Hoc Networks: Challenges and Solutions, *IEEE Wireless Communications*, February 2004, Vol.11, No. 1
- Yoder, J. & Barcalow, J. Architectural Patterns for Enabling Application Security. *Pattern Languages of Program Design 4*, pp. 301-336. Reading, MA: Addison Wesley Publishing Company. (2000).