

An Aspect of Application Security Management

Philip Robinson¹, Maarten Rits², Roger Kilian-Kehr¹
{philip.robinson; maarten.rits; roger.kilian-kehr}@sap.com
SAP Global Research & Innovation (GR&I)¹
Vincenz-Priessnitz-Str. 1, 76131, Karlsruhe Germany
SAP GR&I, University of Nice²
I3S, Route des Colles 930, BP145
06903 Sophia-Antipolis, France

Abstract. This paper first presents work in the area of application security management and then investigates relations to the application of Aspect Oriented Software Development (AOSD) to security, as they are both grounded on the systems engineering principle of “Separation of Concerns”.

1 Introduction


As pervasive computing becomes more of a reality, it is observed that the multiplicity and heterogeneity of computational devices and networks present new challenges for resource-friendly crypto protocols, authentication models, and the management of permissions, credentials and application security updates [4, 7]. The focus on developing crypto protocols that can be run on devices with little computational power seems to be lessening, except in the area of ad hoc sensor networks [4], as we are now seeing Smart-Phones, PDA’s and Pocket PC’s available on the market, with powerful microprocessors, reasonable memory, and networking capabilities. New models for authentication, based on attributes rather than naming, continue to emerge and also influence the way we think about managing credentials and permissions [7]. We therefore observe constantly changing security protocols, products and mechanisms, based on new research developments, vulnerability reports, and changing security requirements reflecting the dynamics of the deployment environment. Our focus is on managing the updates of application software security, such that the business logic (or functional requirements) need not be totally reworked with every security update. This is therefore an application of the Separation of Concerns principle, which is a foundation of software and systems engineering [5, 8]. We have found that Security is often extracted as a separable concern [3], due to its orthogonal nature with respect to the functional requirements of a system.

The first body of research that we looked into was the area of policy-based management, where security policies are often defined separately from those for resource management, configuration, quality of service, collaboration and other system wide concerns. Policies are generally defined as rules that govern the choices in behavior of a system [6]. *Security Policies* are typically concerned with defining access controls and how permission to services and resources are limited to authorized users. However, *Security Management Policies* may be considered as an integration of configuration management (“the relevant actions, based on the conditions resulting from an event”) and access controls (“authorized users must be correctly configured”). It is therefore possible to update the rules and propositions for correct security configuration and operation, based on the context or situation of the application, without modifying the base functionality of the application itself. Within the last few months, we have also taken more interest in the developments of Aspect Oriented Software Development (AOSD) and its utilization for separating security concerns from the application, and hence modularizing the tasks associated with specification and update of security policies. We have found that there is a significant relationship between the concepts and challenges of our work in policy-based separation of concerns and the objectives of AOSD. We are of the opinion that while policy-based techniques originated from the need for abstracting and reasoning about operational concerns [1], AOSD’s initiatives are in overcoming issues posed to software developers, that object-oriented programming do not completely resolve [1, 2]. Nevertheless, we have found some relationships between the two when applied to security management, which we present in this position paper.

The paper continues with section 2 by describing our current work in security management policies, including the background, architecture and operation, followed by a description of the two main management abstractions in section 3 – the Security Policy and the Security Context. Section 4 closes with the discussion about the relationship between our work and developments in AOSD.

2 An Application of the Separation of Concerns Principle

There is a common quote that software developers are not cryptographers [3], and the reverse can be applied. A software developer will tend to be well informed and motivated about what are known as the functional requirements of the application, and typically places little emphasis on an early comprehension of the application's constraint requirements, such as security and performance. A person with a systems security background and portfolio will be interested in performing a threat analysis on the application's functional specification, and to start contributing to the development of policies and procedures to protect the business' resources and likewise those of the user. It is at times impossible for these two development roles to work in parallel, and there may even be miscommunication as the concerns are not clearly defined. For example, an application developer thinks initially in terms of the correct queries for accessing a database and retrieving necessary data, while the security officer will be focusing on strategies for controlling access to the database.

 In our opinion, all software programs may be logically abstracted into a set of *states* (the collective values of a programs variables at a certain time), *transition functions* (functions that operate on a subset of the variables of the current state to advance to the next state) and *controls* on these transitions (rules that constrain the possibilities of the next state). **We also refer to controls as policies for the remainder of the paper.** Figure 1 depicts the relationship between these abstract application logic parts:

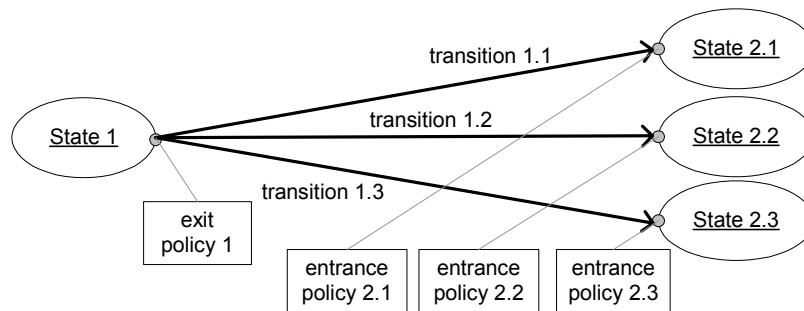


Figure 1. States, Transitions and Controls (referred to as Policies)

The states of an application capture its objectives and therefore can be referred to as the application logic, or the set of propositions that represent the achievement of a particular objective. Transitions are transformations of a defined state towards meeting other objectives of the application, in conformance with the controls imposed by policies that define if a target state is acceptable. We therefore see security logic as applicable to the control of transitions between states. An *application developer* will therefore start by understanding the objective states of the application and then implement the transitions, while the *security developer* will start from the transitions and define the acceptable target states, based on exit policies from the current state (e.g. State 1) and entrance policies of the target (e.g. State 2.1, 2.2, 2.3). We further reason that the functional requirements of any application can be separated into a set of “universal” states and related transitions. This model is depicted in figure 2. We propose that any application is essentially an instance of this model, which defines the properties of each of these states and the functional logic for each transition for a particular set of application requirements. Consider a workflow client application that receives authorized workflow items from a server and allows the user to process the item and then return it to the server. Firstly, the workflow application (software, data, and resources) must be “Available” to the user based on an installation. The application can either change state to “Running” by the user invoking the application to check and retrieve all outstanding items, or by the server pushing a critical workflow item to the user and alerting him. The latter would entail a “Connect” state, which is a sub-state of “Running” and thus conforms to the same controls that apply to the transition to the “Running” state. In both events the application's data and required resources will be loaded into the memory of the device in order to advance from the “Available” state. A policy on the “load” transition states the conditions that must prevail before the state of “Running” or its sub-state “Connected” can be accepted. There can also be transitions between the Running-Connected state and the Running state, as they exchange instance data between each other.

Therefore, when an application sends data to another process, service or network interface, we represent this as an “out” transition to the “Connected” state. Subsequently, when the application reads data from a process, service or network interface this defined as an “in” transition from the “Connected” state. From the “Running” state an application then transitions to the “Terminated” state when its resources are unloaded from the memory of the host. The application can then be reset in order to place it back in the state of “Available”. We have done work on describing and analyzing this formally, but it is not presented in this paper.

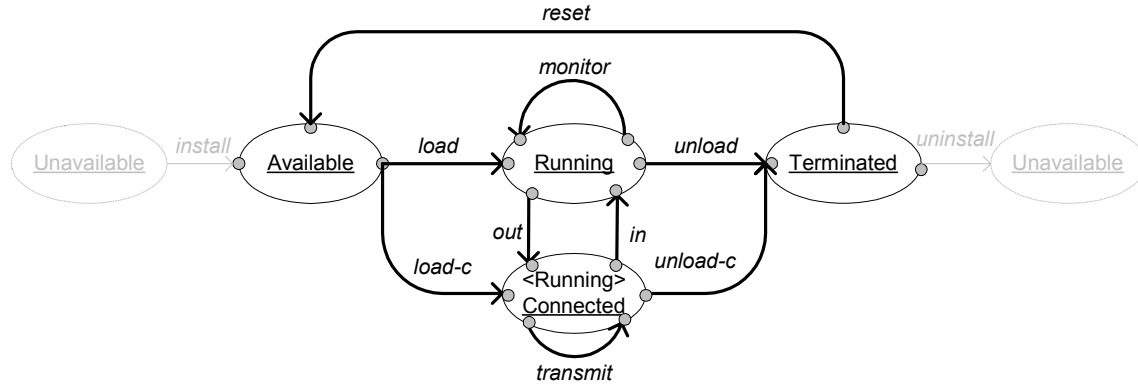


Figure 2. General State Model for an Application

Based on this state model, we defined a policy language that facilitates the declaration of controls for particular transitions, and an architecture for associating, monitoring and managing these policies with their respective applications.

3 The Application Security Policy Management Architecture

The central component of the architecture is referred to as the Application-Manager, which manages the interaction between the application logic components. We identified four aspects of application logic, which may be implemented and maintained by separate software development expertise:

- (1) *Business Logic*: The software that implements the functional requirements of the application, by considering the objective states independently of the security concerns for the transitions. The instances of these logic components are referred to as Partlets in the architecture. However, the Partlet class is a container for the logic, and the software developer implements an interface called “Parttable”. The Application Manager instantiates a “Local Partlet” for each Parttable business logic implementation installed. A “Remote Partlet” is created as a proxy to another partlet or service on a networked host (this is further clarified in the Communications Logic part).
- (2) *Security Management Logic*: Each business logic software bundle includes a Security [Management] Policy, which contains a set of Controls that correspond to declared transitions (see figure 2) of the application. A Security Management Policy therefore verbally states: “Before or after a transition T in a program x occurs, then the policy P (x, T, {C₁, C₂, ..., C_n}) must be applied before a new universal state S is accepted, where all the controls C₁...C_n are correctly executed.”

$$T \in \{\text{load, load-c, monitor, transmit, in, out, unload, unload-c, reset}\}$$

$$C_n = \{\text{requirement, utility, instance, transition, utility-parameters, post control directive}\}$$

The terms that compose the control tuple are defined in table 1.

- (3) *Security Enforcement Logic*: The Security Enforcement Logic is the software that provides the utility for executing the stated control. The result of installing an application (Parttable) with its Security

Policy is an instance of the application (Partlet) with a Security Context. The Security Context is a tree structure, where each node is referred to as a “Security Context Element”. The elements are either:

- *Active Element*: Execution of this type of node invokes a particular utility/ Security Provider
- *Input Element*: Execution of this type of node prompts the user of the application with a user interface to provide data or make a decision (e.g. username and password)
- *Passive Element*: Execution of this type of node just returns a static, constant value from the application host profile/ registry
- *Session Element*: Execution of this type of node reads or writes data to the relevant communications channel or session (i.e. an instance of the Connector component of the communications logic).

The root of the tree structure is always an Active Element, and its parameters may require execution of any other Security Context Element, including another Active Element.

- (4) *Communications Logic*: The Communications Logic is a software abstraction of the available network interfaces on the application host. Whenever an application requires a connection to another service, process, resource or device, it creates a Session and hence a Connector, transferring its Security Context to the Connector. Controls of policies are therefore relevant to the “in”, “out” and “transmit” transitions of the application.

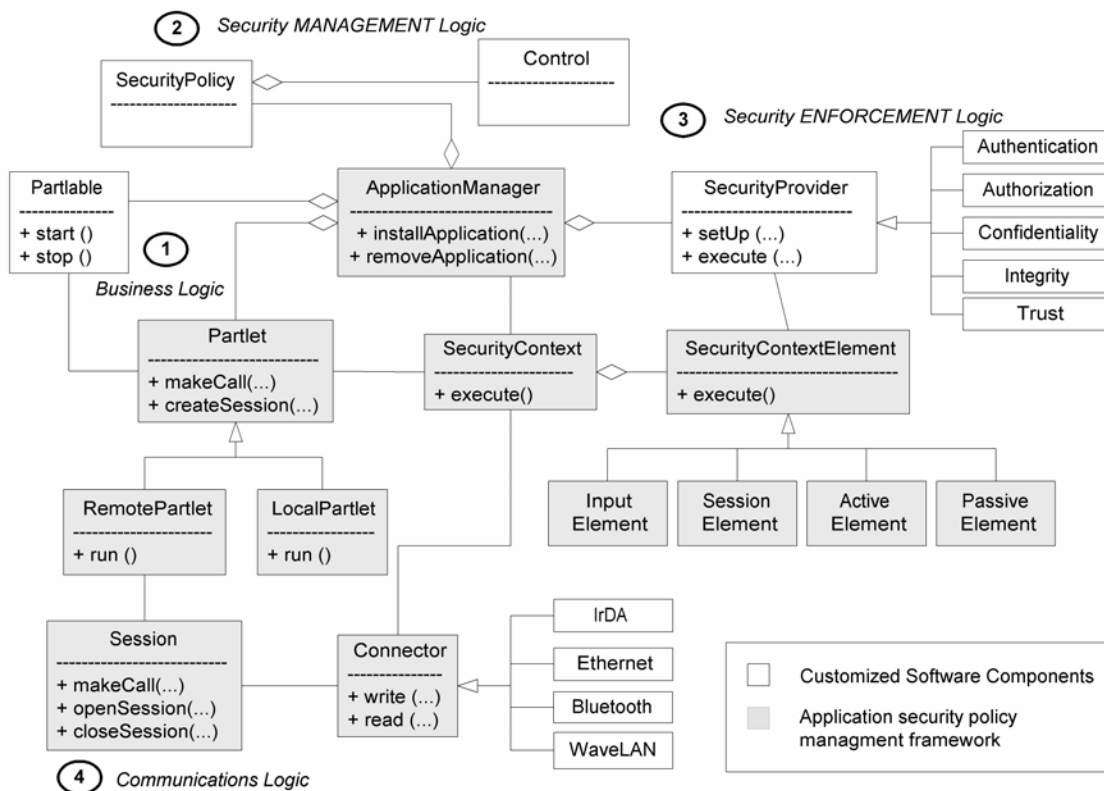


Figure 3. UML of Application and Security Policy Management Architecture

When Security Policies are updated, the Application Manager first cryptographically verifies the origin, as they are either signed (with the private key of the administrator) Java objects or documents, and then generates and replaces the modified Security Context of the running Partlet. The next time the Partlet carries out a security-relevant transition (as defined by our universal state and transition model), it executes within its new Security Context. The policy is a set of propositions and assertions that define what resource is being protected, how it is to be protected, and when it is to be protected. The most complex part of the

policy is defining the Controls. The fields of a security policy control are explained in table 1 with examples.

Element	Description	Example
REQUIREMENT	This is a high level classification of the requirement handled by the control or assertion	“Authentication”
UTILITY	This specifies the resource identifier of the utility that provides the functionality of the requirement. This would have been developed by a security utility developer, and may be reused for different applications	“AuthenticationProvider”
INSTANCE	The utility will typically have different modes in which it can be instantiated, and this allows the policy to specify this particular mode.	“RSAPubKey”
<u>TRANSITION</u>	This is the explicit connection to the separation of concerns principle discussed in section 2. The policy actually specifies the transitions for which the control or assertion needs to be executed.	“IN, OUT”
[PARAM = VALUE]	This is an optional field, but specifies parameters that may be required by the utility being used	“KeyLength = 1024”
DIRECTIVE “;” or “>”	This is either a semi-colon “;”, which specifies that the control is complete, or it may be a “>”, which specifies that there is a subsequent control that operates on the output of this one.	“> {Control 2 that specifies a hash function as the utility}”

Table 1. Specification of Controls and Assertions

Our experience with using this separation of business application logic, security management logic, security enforcement logic and communications logic has proven useful from the point of system requirements specification until the software is deployed and maintained. We were able to delegate the tasks of developing certain software components to different development partners with different expertise, and supported modular updates without changing the functional focus of the application. Additionally, we were able to define security requirements at the application layer independently of the network available. Therefore we did not have to dedicate time to dealing with the security issues of Bluetooth versus those of WaveLaN. Furthermore, incorporating new security enforcement techniques such as Smart Cards proved to be a seamless upgrade for meeting the protection goals of the application.

4 The Relation to AOSD

Although we were satisfied with the flexibility and extensibility of our policy-based architecture, we thought that selling the concept to developers, with established development practices, would be more challenging. We reasoned that the methodology might seem proprietary to the architecture presented, and that developers might sense that the balance of trust would have to be shifted away from the application towards the central Application Manager. One of the authors however had the notion that there might be some interesting connection to the developments in AOSD (Aspect Oriented Software Development), as it has the same underlying principle of "separation of concerns" yet has emerged from the broader field of software engineering and not application security in particular. These are explored in this section.

The advantages of AOP for implementing non-functional aspects of an application like persistence and performance have been discussed much more in-depth however. Security researchers are currently experimenting with the idea to use AOSD to help software development groups secure their applications [1, 2]. An important reason why well-known security vulnerabilities keep on reappearing is that they can appear anywhere in a program. Therefore, if one desires that an application attains a high security level, the developers are expected to all have good security expertise. However, because security implementation can be complex and error-prone, this is mostly not the case. This is a reason why AOSD is generating interest, in that it allows a separation of security implementation from the application logic, with motives comparable to those of our policy architecture: the (few) security experts can concentrate on implementing the security aspects, while the other developers can concentrate on the application logic. Another reason why AOSD can be useful is that it allows an easier adaptation of security aspects, even during the execution of the application, so that unanticipated flaws or environment changes can be resolved. Both of these advantages are also present in our Application Security Policy Management Architecture. This is not entirely unexpected, as the two approaches are based on the separation-of-concerns principle. To properly investigate the relationship of our Application Security Policy Management Architecture to AOSD, we have selected the Interaction Specification Language (ISL) [5] to provide concrete examples. In this language, interactions are used to allow dynamic adaptation of software components, with the following properties:

- Interaction schemes specify behavioral dependencies between components; instances of these schemes are called shortly interactions and are used to impose this behavior on specific objects in the application.
- The definition of an interaction scheme is independent of particular implementations; an interaction can link components, which are defined on different platforms.
- The interaction scheme is based on the interface of the components.
- An interaction scheme and an interaction can be created and removed during execution.

A scheme defines different interaction Rules, which express the operations that have to be made on the linked objects. The left part of a Rule defines the method that will be adapted and the right part the Reaction that will lead to a rewriting of the method's code at runtime. Consider code fragment 1, where we implement the authentication example given in table 1. The security scheme can link any *Object* in an application to a module *AuthenticationProvider*, with authentication implementation from a provider, and to the access control policy *AccessControl*. The Rule expresses that every time a method from the object is invoked (obj.*->) the caller will have to authenticate himself and the policy will be checked to see if he is authorized to call the method. The AOP platform (Noah) [5], performs a check before every actual method invocation, to see if there are new interactions imposed and merges multiple interactions if necessary. The role of Noah, is similar to the Application-Manager role in our Policy Management Architecture, which manages interaction between application logic components. However, in the AOP approach the application logic has a more central or dominant role, security is implemented as an aspect. In the Policy Management Architecture, the different modules (Business logic, Security Management Logic, Security Enforcement Logic and Communication Logic) are considered more equal. This is because AOP is more a practical approach, while Application Security Policy Management is more focused on reasoning about the interaction between the modules. ISL is limited in the sense that there exists no higher-level policy management, security aspects can be hard coded, but in a real-life application there should also be a framework that allows one to easily (automatically) choose between different specific security aspects depending on the context. **In other words, a convergence between AOP architectures and policy management is desirable.** We could use the Application Policy Management approach for this purpose. The different elements in the Controls specification of table 1, are specified on a higher level than the ISL aspects implementation. This allows an easier policy management. They could however be mapped and compiled to ISL aspects, as the example clearly states.

```
interaction security(Object obj,
                    AuthenticationProvider prov,
                    AccessControl policy)
```

```

{
    obj.* -> id := prov.authenticate(RSAPubKey, 1024);
           if policy.authorized(_call, id) then
               obj._call;
           else
               exception 'unauthorized user'
           endif
}

```

Code Sample 1. ISL syntax example applied to example of Table 1

There are also other methodologies, architectures and technologies in the area of software engineering that provide a similar approach to the separation of concerns principle and its application to security. These include CORBASec (Common Object Request Broker Architecture Security) [10], where the security logic is captured within a particular service of the ORB (Object Request Broker), the Java Security Manager [11], as well as other research projects where application security requirements are explicitly declared as executable content separate from the core application logic [9].

Acknowledgements. We acknowledge the contributions of our colleagues Laurent Gomez, Jochen Haller and Cedric Hebert towards defining the overall application architecture, as well as Emin Islam-Tatli for contributing to refinement of the policy language and implementing the first version of the policy components in Java during his Master's Thesis. The architecture emerged from the WiTness project IST-2001-32275 www.wireless-trust.org. We would also like to thank Karima Boudaoud and Michel Riveill from the Laboratoire I3S, rainbow.essi.fr, for their valuable input about the applicability of AOSD for security.

References

- [1] B. De Win, B. Vanhaute, and B. De Decker. Security Through Aspect-Oriented Programming. *Advances in Network and Distributed Systems Security*, p.125-138,2001.
- [2] B. De Win, F. Piessens, W. Joosen and T. Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering. *WAEPPSSD*, Boston, USA, November 6-8, 2002.
- [3] J. Viega and D. Evans. "Separation of concerns for security". *ICSE 2000 Workshop on Multi-Dimensional Separation of Concerns in Software Engineering*, pages 126--129, June 2000.
- [4] J.-P. Hubaux, L. Buttyan, and S. Capkun. "The quest for security in mobile ad hoc networks". In *The 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing*, October 2001
- [5] M. Blay-Fornarino, A-M. Dery, and M. Riveill. Towards Configuring Distributed Applications. In *2nd IEEE International Workshop on Aspect Oriented Programming for Distributed Computing Systems*, Vienna, Austria, pages 487-492, July 2-5 2002.
- [6] M. S. Sloman. "Policy driven management for distributed systems". *Journal of Network and Systems Management*, 2(4):333--360, 1994.
- [7] Sadie Creese, Michael Goldsmith, Bill Roscoe, Irfan Zakiuddin, "Authentication for Pervasive Computing". *Security in Pervasive Computing*, First International Conference, Boppard, Germany, March 12-14, 2003
- [8] Stephan Herrmann, Mira Mezini, PIROL, "A case study for multidimensional separation of concerns in software engineering environments". *Proceedings of the conference on Object-oriented programming, systems, languages, and applications*, p.188-207, October 2000
- [9] Wayne Jansen, Tom Karygiannis, Michaela Iorga, Serban Gravila, and Vlad Korolev, "Security Policy Management for Handheld Devices", *The 2003 International Conference on Security and Management (SAM'03)*, June 2003.
- [10] Blakley, B., Blakley, R., Soley, R.M., *CORBA Security: An Introduction to Safe Computing with Objects*, Addison-Wesley, 2000.
- [11] Gong, L., M. Mueller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2," in *Proc. of Usenix Symp. on Internet Technologies and Systems*, 1997.