Towards a Security Meta-Model for Software Architectures

Mohammed Gasmi
Makhlouf Derdour
Nacera Ghoualmi Zine
Towards a Security Meta-Model for Software Architectures

Mohamed Gasmi, Makhlouf Derdour, Nacéra Ghoualmi Zine
LRS Laboratory, University of Badji Mokhtar – Annaba, Annaba, Algeria

Abstract:
Security is becoming a very important concern for distributed application architectures. Previous modeling approaches provided insufficient support for an in-depth treatment of security. There is currently no generic solution that can automatically deploy the security techniques at the creation of the software architecture. The identification of security requirements during the assembly of software components is necessary in such approaches. Indeed, software architectures validate the functional aspects, which are insufficient to ensure a realistic assembly to remedy the problem of security.

Facing the new challenges of security for distributed software application and giving the base that is provided by existence software architecture research, we propose a model based approach called Security Meta-model of Software Architecture (SMSA). Our model is focused on semantically rich software connectors that provide communication and secure the exchange of information between distributed components in the same configuration.

Keywords: Component, Connector, Security, Non-Functional Requirements, Software Architecture.

1. Introduction

The recent emergent applications are increasingly distributed, opened and have complex architectures. Such applications are composed of a large number of software entities, dispersed and cooperate to provide request services to their users. This complexity is related, for example to: the expansion of communication networks, the hardware and software heterogeneous, the inevitable need of security and both vital and permanent requirements for the evolutions.

Software architectures constitute a solution, as well for design as for maintenance of these complex computer systems. In fact, it gives an abstract view of a system in turn to allow the management of this dazzling growth. At this level, the architecture has known an increasing interest in those last years, so it became a sub-central domain of software engineering. This research has clarified the definition of architectures and architectural elements: components, connectors and configuration. Thus, the architecture allows an easy understanding and analysis of a system.

Among the software architecture for complex and distributed applications, there is component-based architectures, which have as objective to permit a better structuring of applications and to provide an increased opportunities for decomposition and reusability. The wishful aim is to be able to manipulate easily a group of components by connecting them to each other in order to obtain a solution to the problems at an abstract level. So the based component development facilitates the identification of business components that satisfies the functional concerns and allows the factorization of the components use and ensures the Non-functional concerns.

However, the detailed definition of key concepts such as configuration, interface, port, service provided/required, property, Non-functional services manager, differ from an ADL (Architecture Description Language) to another, which poses the problem of security management between components of the same application during communications and information exchange, as well as with their dispersion in a distributed environment. Facing this dilemma, the necessity of taking account of security communication and the exchange of information between the components at an abstract level is essential, so as to avoid ad hoc solutions not reusable and/or not generalized.

The analysis phase permits to separate the functional and non-functional concerns of an application. While the design phase gives us the possibility to have new non-functional concerns (communication, adaptability, security, etc....) Bind to components during the configuration. That’s why we need a modeling tool that is capable to detect the points that request the implementation mechanisms and security techniques between the components related in a configuration.

In this paper we propose Security Meta-model of Software Architecture(SMSA), a software architecture Meta-model that takes into consideration the concept of security separately from functional components by means of secure connectors, so as to contribute a more complete and deeper modeling of
software architecture, by integrating the security concept as a non-functional requirement and to facilitate the detection of points that request the security mechanisms implementation during the exchange of information and the communication of the various distributed application elements.

To reflect our goals, and after the exhibition of our motivation, this paper is organized as follows. Section 3 discusses related work. Section 4 presents our software architecture meta-model. Section 5 details the points of interaction that require a secure connector. Section 6 gives a global idea about the construction of our secure connector. Finally, the article ends with a conclusion and perspectives.

2. Motivation

Security becomes a critical parameter in the IT, particularly in the distributed applications, so it’s insufficient for a component to be secure. For the whole system to be secure, all relevant components must collaborate to ensure the system security. An architecture model guides the comprehensive development of security, such high-level modeling enables designers when it locates potential vulnerabilities and installs appropriate countermeasures. It facilitates checking that security is not compromised by individual components and it ensures also the secure interactions between components. In addition, an architecture model allows selecting the most secured alternatives based on existing components and supports continuous refinement for further development.

Our main motivation is to propose a meta-model that maintains the coherence of architectures by its consistency of dispersed components throughout a (re) assembly or a reconfiguration; by using a new architectural concepts (Domain, typing component, security services, etc) and connectors that have a richer semantics. Using those concepts allows the automatic detection of possible vulnerability points between components as the employ of secure connectors allows the resolution of those issues of insecurity. The systems are built by assembling components and connectors, where each element can be placed in the right place on the configuration of the architecture In most of the ADLs (Architecture Description Languages) we find the following[MED 00].

The management of the non-functional concerns of the components is ensured after the definition of architecture and the configuration of the components. The management of assembly does not take into account the vulnerability problems and the management risk caused by the distributed applications nature.

The choice of the available connectors in the environment is limited to the primitive connectors and non-compounds ones.

Few models are able to define new connectors with different treatments that ensure the non-functional concerns of the components (security, communication, conversion, etc.).

There is no direct or automatic correspondence between architectures (models) and the applications conceived following these architectures (instances).

In order to solve these problems, we propose an extension of architecture description language, which permits to introduce the concept of security into a Meta-model to allow the modeling of distributed applications and comfort the detection of points that requires the integration of security mechanisms. We use the concept of secure connectors to ensure the non-functional requirement of security in distributed applications. These connectors incorporate the required security services as well as qualitative extensions of those services to provide a measure of QoS reflecting the evolution security needs of data stream exchanged between components.

3. Related Work

Software components are the software entities that can be reused in order to minimize the cost of development, maintenance, and in software evolution. Currently, many proposals call for the development mode by assembling software components. Despite some common vocabulary (component, port, interface, service, Attachment, connector), those proposals differ in their origins, their objectives, their concepts and also their mechanisms.

ADLs (Architecture Description Language) [CLE 96] and [BAR 05] are used to specify software architectures. [MED 00] highlights the difference between an ADL and a formal specification to distinguish the other ADL modeling notations. Furthermore, several efforts are focused on understanding and developing connectors in the context of ADLs. Taxonomy of connectors is proposed in [MEH 00], where connectors are classified by services (communication, coordination, conversion, facilitation) and types (procedure call, event, data access, linkage, stream, adaptor, and distributor). The models cover all or part of the requirement in terms of language, semantics and tools. However in [MEH 00] the authors raise insufficiency for the specification of the non-functional properties of systems. In general, we noticed a semantic foundation lack to expressing constraints and refinement (component, connector and configuration) and a lack of tools for dynamic reconfiguration and evolution in real time.

Approaches like [ALL 97a] [BER 00] [MAX 05] [ATT 09] allow the separation of the functional concerns. They were proposed in order to capitalize the functional needs in modular entities. Several ideas.
were proposed within this perspective. We mainly distinguish two categories of approach for software architectures: those inspired on Component-Based Software Engineering (CBSE) and Service-Oriented Architecture (SOA). In the first case [SZY 97], [ALL 97a], [BER 00] the focus is on the static structure of the system; the software elements are components assembled by connectors in configurations. In the second case [PAP 03] [Max 05] [ATT 09] [ASR 09] the focus is on the functional structure of the system; software elements are functionalities (services) linked by relations of collaboration or combination.

Modern applications are more and more developed according to ADL-based development processes [AVG 05]. The ADLs allow analysis and verification of properties early in the development cycle that the future system will have to satisfy, in particular the properties availability, confidentiality and integrity in configurations of dispersed components. Indeed, the current applications consider embedded systems that include the flow of notion and security as an important feature of their behavior [AVI 04] [BAL 03]. Most of existing ADLs such as SPT-UML [GRA 04], MARTE [OMG 06], fractal [BRU 04], SCA [BAR 07], Kmelia [ATT 09] and AADL [SAE 08] do not take into account the adaptation and the properties related to multimedia flow during the software construction phase. Some of them, treat the problem of heterogeneity by modification of the configuration parameters (addition, withdrawal, or replacement of components) [MAR 04] or by a meta-model which verifies the adequacy of service regarding its context and research of the adaptation strategy [MAR 07].

C3 (Component Connector Configuration) [AMI 09] is an approach based on software architectures. It makes it possible to describe a view of logical architecture in order to automatically generate the physical architecture for all the application instances. The idea is based on the refinement and the traceability of the architectural elements. The software architecture is described in accordance with the first three levels of modeling defined by the OMG [OMG 06] [OMG 07]. Typing of connectors allows visualizing better the various connections between components. In particular, the connectors proposed do not ensure the connection of the heterogeneous components and do not take into account the semantics of configurations and that of the links between components.

ADL can be classified in three different categories [AMI 09]: ADL without connector, ADL with a preset set of connectors, and ADL with types of explicit connector. In the final case, the ADL provides connectors as first order elements of the language such as: Wright [ALL 97b], ACME C2 [GAR 00], XADL [DAS 05], AADL [ALL 02], and so forth. All these languages seek to improve the reusability of the components and the connectors by separating the calculation and the coordination. In our approach, we choose the explicit category of connector. So, in SMSA Meta-model, we present a generic and explicit type of connector that the system can specialize according to the architecture and the components needs.

4. SMSA Meta-Model

The architecture description languages (ADL) response partly to the assume of security issue at the architectural design by allowing the precise and common vocabulary definition for actors concerned by the specificaion related to the architecture and configuration (architects, designers, developers, integrators and testers). They specify the architecture components in the abstract without going into implementation details so as they explicitly define the interactions between system components and provide modeling support to help designers to structure and compose the different elements.

The ADLs are a support for describing the architectural structure of the application. They offer facilities about the reusing of components and with describing composition means by a description of dependencies between components and communication rules to follow [MED 00]. With an ADL, software architecture Impies three C: Components, Connectors and Configurations [MED00]. Components represent the computational units and data storage in a software system. The interaction between these components is encapsulated by the connectors.

The configuration represents an instantiate of a number of components and connectors. He binds them together in order to form the complete system. One part of the ADL answers this description. However, another part diverges. Some ADL as Rapide [LUK95] or Darwin [MAG96], for example, do not clarify the concept of connector. Others agree to hierarchical description of the components. The components can be seen as white boxes and might contain internal architecture structures. In certain ADL such as Rapide, the components are considered primarily as black boxes. In UniCon [SHA 95], Wright [ALL 97] Acme [GAR 00], and MMSA [DER 10] we can define composite connectors, when it’s impossible in others ADL.

SMSA is a software architecture approach, which enables the description of software architecture systems as a collection of components (homogeneous and heterogeneous) that interact through connectors. Components and connectors have the same abstract level and are defined explicitly by the separation of their interfaces and their internal configurations.

Our Meta-model is described in UML language under a class diagram format (Fig 1), it highlights the following concepts:
The component concept is used to represent any element that ensures functionality within an application. In other words, any feature from the logical application is explicitly supported by a component [ROS04]. A component can represent a complex application compound by other less complex applications. It may also represent a feature as a simple arithmetic operation.

SMSA component is typed in order to determine the security service required. The creating instance is controlled by typing (Presentation, Process, Data), which is adequate for our needs; in terms of definitions of a component model that allocates the items detection which requires a security check.

A component can be primitive or composite (configuration). The primitive component internal structure is inaccessible. A component is described by two interfaces (provided and required) and a set of services.

In addition, each component may have an interface with multiple ports and multiple services, including that the interface consists in a set of interactions points between the component and the external world that allow the invocation of the services. Also we distinguish between an “Output” interface exporting data of components, and the “Input” one importing data to components. Each interaction point of a component is called a port.

SMSA connector is the key element that ensures connection between the components. It ensures the nonfunctional concerns of components (quality of service, security, communication) but also to the assurance of security flow data exchanged during the execution.

There are two connector types: communication and security.

- **The communication connector** is established between two components that are encapsulated within a same space of abstraction.
- **A secure connector** connects components that are encapsulated in two different spaces of abstraction (Composites, Machines and Processes).

Ex: A component producer that provides a confidential data in a buffer and other component consumer which utilizes the contents of that buffer. Encryption and decryption of data is provided by a secure connector because of the two components are not in the same process.

The connector includes two parts: the first is the visible part: A required/provided interface of connector that is composed of a set of roles. Each role serves as a point through which the connector which is connected to the component, these roles define the modes (synchronous, asynchronous, continuous) and type (GPRS, WAP, MMS, etc...) of connection between components. The second part is a glue unit represented by three managers: communication, security and QoS. They manage the data transfer between components and allow securing it. The security aspects which are taken into account in our approach SMSA are: authentication, encryption and integrity.

- Configuration is a connected graph between components and connectors that describes architectural structure. The aim of configuration is to abstract the details of the various components and connectors. A configuration owns a name and can have an interface (represented by the interfaces of components that provide / require flows to / from the external environment) and a set of services (encapsulated in components).
- The interactions between components are defined between the ports of components (required or provided) and the roles of connectors (required or provided). It is clear that only ports provided (required) can be related to the roles required (included).
- An important element in our architecture is the domain. A domain defines the dispersion and distribution of components on different machines and processes. A domain can contain one or more composites each of which has components that run in on one or more processes on one or more machines (figure 2).
In Fig 3 we describe a domain that contains three composites and three machines. The composites, shown at the top of the figure, consist of five components which spreads over three processes on two different machines. The other two composites, show in the bottom of the figure, running all of their components on a single machine, dividing them into three separate processes. Communication differs between these components, either intra-process and an inter-process, or inter-machines [SEI09].

The concept of domain brought much for SMSA, especially when it gives us an assembly structure in wider meaning than the composite, which provides the scope of cooperation between components and allows the detection of points that require the secure connectors.

5. Secure Connector Structure

Connectors play a key role in our approach; they have the same abstract level as the component, and also have potentials to provide secure interaction between insecure components.

There are not many works that treat the security issue in ADL, among those that exist there is the architecture description language proposed by Ren [REN05] who defined a connector for access control, so as expansion for xADL language [DAS05] with necessary elements for architectural modeling (subject, principal, security resource, privilege, Safeguard and policy).

The SMSA connector is a composite connector, it constitutes the entity of communication and security in our approach, so it's able to transfer the data between the various components while ensuring the security of the latter, we can then proceed to model its architecture by proposing a Meta-model. The Fig 4 shows the Meta-model of our SMSA connector that is defined by three managers and two interfaces.

SMSA connector is defined by two interfaces ("Input" and "Output") and "glue" which imply three managers: communications, security and quality of service. They manage the data transfer between components and allow the realization of security. Connector interface required/provided consists of a set of roles. Each role serves as a point that connects the connector to the component. Consequently two components can be linked by a connector, so that two connectors can be related together to create complex adaptations.

We have also extended the glue by a security manager which cooperates with a QoS manager to ensure the security task. This security manager is a set of security services that cooperates to achieve security. Three types of adaptation can be realized in software architectures: authentication, confidentiality and integrity.

6. Connector Taxonomy

Compared with those of the languages of description of architectures [ALL 97a] [MEH 00], the connectors that we propose can be simple or composite as well as ensuring services. These connectors do not only guarantee the communications links but also the security of the data exchanged between components. From SMSA approach we can detect the points that require security and integrate necessary secure connectors. We distinguish two types of connectors; the communication connector and the security connector, the first is a simple connector intended for a link role and used generally for binding two component that are encapsulated within a same process, for security connector we distinguish three sub-types: Authentication, Confidentiality and Integrity.
• **Authentication Connector**: Composed of three managers (Fig 5), the communication manager, the QoS manager and that of the security service manager which assures the authentication to prove the identity of communicating components. This connector is typically installed between two components which are in the same composite but running on two different machines.

![Figure 5. Authentication Connector Structure](image)

• **Confidentiality Connector**: Composed also of three managers (Fig 6), the communication manager, manager of the QoS and security service that ensures the data exchange in an understandable way for the two components concerned. This component involves the implementation of encryption / decryption service of data exchanged flow. This connector is usually installed between two components which are located in two different processes or between a component of type DATA and a component of type PRESENTATION / PROCESS.

![Figure 6. Confidentiality Connector Structure](image)

• **Integrity Connector**: The same as the two previous connectors (Fig 7), except that the security manager here provides protection against modification of information by the malicious stakeholders using the fingerprint functions. This connector is usually installed between two components which are in two different composites.

![Figure 7. Integrity Connector Structure](image)

• **Simple Connector**: This connector has a liaison role and a simple communication; it consists of a communication manager, usually installed between two components which are in the same processes at the same machine.

![Figure 8. Simple Connector Structure](image)

7. **Conclusion**

Contrary to the ADL commonly presented, we consider connectors as interaction units endowed with dynamic behaviors that allow them to take into account non-functional concerns.

In this paper, we have presented a generic Meta-model to describe the software architecture by integrating the security concepts and distribution components. This allowed us to present separately the distribution parameters (Domain) because of the fact that they represent a very important component assemblies and configurations.

Our proposition can be used as a support to develop the management applications of the numerical resources (DAM: Digital Asset Management), for example. Such applications handle a wide variety of media, and communicate with the users through various platforms (Mobile phones, PDAs, Desktops, laptops, etc...). SMSA can bring an effective solution to the DAM development. Especially in parts of: acquisition, processing, distribution and content use. It allows to resolve the integrity and confidentiality problems of content and also the problem of user authentication at the architectural level by injecting the adaptation connectors at the execution level by the management of QoS and the reconfiguration of these connectors.

The prospects of this work are twice. The first is to finalize the SMSAPlugIn (for Rational Software Modeler) that permits to create /modify quickly software architecture, to easily build architectural models in UML 2.0 with the SMSA approach and also to verify the security assurance of product architectures. The second which is at longer term, is to propose a description of security policies and QoS management, to automate as much as possible the integration of secure connectors and the connections through the implementation of different managers (security, QoS, communication).

**References:**


Towards a Security Meta-model for Software Architectures


