SECURE SERVICES: INTEGRATING SECURITY DIMENSION INTO THE SA&D

Mehdi Snene

Business School, University of Geneva, Geneva, Switzerland
mehdi.snene@unige.ch

ABSTRACT

Services security is often assimilated to a set of software solutions (Firewall, data encryption.) but rarely consider the organizational security rules as a fundamental part of the Services security policy. With the increasing use of new Services architectures (Open Services architecture, distributed database, multi web server, multi-tier application servers) security leaks become crucial and every security problem is harmful to the organization business continuity. To reduce and detect major security risks at an earlier step of the Services project, our approach is based on different knowledge exchange between end users, analyst, designers and developers collaborating at the Services project. The knowledge is mainly oriented to the detection of weak signals inside the organization. In this paper, we present the different knowledge surroundings an Services project and a knowledge pattern structure that can be used for the formalization aspects of the established exchange that should be established during the Services project between the different participants.

KEYWORDS

Information Systems, Services security, Project life cycle, Knowledge pattern.

INTRODUCTION

The fast evolution of Services architectures and of the corresponding technologies has not been followed by an adequate adaptation of SA&D (System Analysis & Design) methodologies for the new requirements of these new forms. The commonly used methodologies keep focused on representing the virtual world as close to the real world of the organization as possible. This is done without taking into account the resulting risks taken by using these new architectures. Indeed, the evolution of the Services architectures represents an informational danger: the data, process and results securities might be violated at any time. That violation can occur during any phase of informational production, information exchange, data collection phase, process execution or the results transmission and can be either internal or external. To cope with these risks, a in depth security SA&D approach is needed. This approach must cover the entire SA&D process and ensure the continuity of the security policy.

Basically, existing approaches such as intrusion detection, listening detection, exploits scan, etc. are commonly used to make the running Services architecture secure. These tools do not induce a coherent and continuous process that ensures the global potentiality of the Services security. Indeed, these tools are used subsequently to the deployment and are considered as a software package, which remains external to the architecture. Upstream of the deployment, these tools cannot be considered or integrated within the Services project life cycle because they protect the execution platform rather than the architecture. From this postulate, two major issues remain crucial for the survival of Services architecture. The first one is how to make sure that the tools

Sundarapandian et al. (Eds): ICAITA, SAI, SEAS, CDKP, CMCA, CS & IT 08,
pp. 259–269, 2012. © CS & IT-CSCP 2012 DOI : 10.5121/csit.2012.2523

guarantee the global security of the project without exposing their survival and restricting its functionalities. The second issue concerns the efficiency of these tools applied to answer the security needs of an Services architecture that was not conceived and developed while taking into account the secure dimension of data and process (Snene and al. 2004). Moreover, adding downstream security solutions to existing Services (information systems) is not an optimal method to limit vulnerabilities of the system. In fact, implementing a set of security process for an Services that has been designed and developed without taking into consideration the internal security procedures, which should be integrated in it, will lead to a patching maintenance policy. We must consider more efficient solutions in order to increase the security level and to define the different risks the system may encounter. Indeed the security process must be taken into account in every step of the Services project life cycle. We define the security process as the different measures applied during the entire Services project in order to build the Services based on security. For instance, during the users' needs analysis phase of common design methodologies, the security dimension of Services is still not well specified due to the fact that the organizational security is not considered as an autonomous procedure. Indeed, the security dimension is generally delegated to development and deployment steps where they consider the system security instead of the organizational security, which must take place at analysis and design steps. The rethinking of the different phases composing the Services project life cycle take into consideration the security dimension from the first phase to the last one in a continuous. The main purpose of such approach is to obtain an upstream security policy that allows us to prevent major vulnerabilities before Services deployment.

In commonly used methodologies, the differentiation between procedure and process remains implicit. Both are generally used together; however the distinction is essential in order to define the scope of security. Indeed, the security approach is different whether the procedure or the process is considered. A procedure is defined as a formalism of data treatment. It is in fact a set of steps, means and methods used in the execution of a task in order to achieve a predefined result. Hence, the security policy must be applied at the different levels composing the procedure. We define a process as an arranged succession of operations performed in order to realize a procedure in an automated manner. Existing security solutions provide answers to process security through key exchange, data encryption, etc. (Snene 2004). Different studies underline the fact that security problems are generally due to a misunderstanding of the organizational security needs or an inadequate system solution. In fact, implementing a set of security process for an Services architecture that has been designed and developed without taking into consideration the internal security procedures, which should be integrated in it, will leads to a patching maintenance policy. Such security policy is considered as a downstream solution that reacts after vulnerabilities detection or attacks. The presented approach is based on the early detection of security leak at the different Services project life cycle and on the analysis and the integration of user informal security procedures. The main purpose of such approach is to obtain an upstream security policy that enables us to detect major vulnerabilities before Services project deployment (Snene 2004). Some existing standards aim to develop an Services security roadmap by considering the global Services project lifecycle. These standards establish a minimum Services standards and guidelines, while ensuring flexible implementation based on diverse missions and business functions (Ross 2007). The NServicesT standards (National Institute of Standards and Technology) developed a generalized framework for managing enterprise risk for information systems that support organisational missions and business functions. They developed a Risk Management Framework (RMF) that represents the security related activities that occur within an enterprise's system-development life cycle (SDLC). Our approach is similar to the NServicesT standards since both of them support a bottom-up approach to the Services project life cycle security.

Next section presents the Weak Signal concept that we consider as the cornerstone of our approach. Then we present the different security aspects that must be taken into account while the Services system is designed and developed. Finally we adapt the different aspects to the new Services system architectures and we define a knowledge pattern structure that will be used for the Services project partner's communication establishment.

1. Rethought Services Project Life Cycle

1.1 Weak Signal

The Services focus on representing the virtual world as close to the real world as possible. Such realization is difficult and complex to handle. Every decision made during the Services project life cycle can have repercussions on the next Services project phases or in the resulting Services. For this reason, it is primordial to focus on security from the beginning of the Services project. Postponing this to the late phases of the Services project life cycle endangers the required global security. As explained before, organizational security is often neglected because of the focus on external security. This negligence is reinforced by the difficulty to detect the subsequent vulnerabilities in the resulting Services because of their ambiguity. Indeed, organizational security addresses threats resulting from the environment of the organization. Moreover, process inside an organization can lead to critical vulnerabilities when transposed to the virtual world of the Services.

The proposed approach is based on weak signal detection. We define a weak signal inside an organization as information or knowledge, which cannot be explicitly expressed during the requirements analysis by actors involved in the creation, diffusion and the use of such information or knowledge. A security weak signal is an implicit information or knowledge existing inside the organization, which can be harmful to the system security if not detected. In fact, such security vulnerability cannot be resolved during the final system life cycle steps because it requires a complete rethinking of the system.

A common case of security weak signals appears during confidential documents exchange automation. Indeed, such exchange in the real world involves different security rules based on human responsibility and vigilance. For example, in the case of a document exchange, actors ensure that copies of the document do not exist and guarantee by their presence that the right person receives the document. The exchange life cycle is closed and any leak is easily identifiable. The adaptation of this human document exchange process to an electronic process (E-mail, ftp, etc.), only the common security rules applying to the data flow are implemented if no more requirements are specified by actors. However these security rules basically address process issues but do not consider procedure issues.

These procedure issues are constituted by the different elements surrounding the exchange, which we identify as security weak signals. For example, existing security frameworks do not guarantee that there is no other copy of the document. In the case of an E-mail exchange, the sender and the recipient keep by default a copy of the document in their E-mail application. This multiplication of copies greatly increases the risk of security break. This is the kind of security weak signals the proposed approach tried to identify and to circumscribe.

Due to its implicit nature, weak signals are considered as fuzzy knowledge, which is difficult to detect and to express. The users commonly undergo the weak signals without being aware of the risks engendered, while the project life cycle actors do not have the necessary means and knowledge to address these weak signals. To ensure the continuity of the Services project, this knowledge has to be shared between the different actors contributing to the project.

1.2 Services project Life Cycle

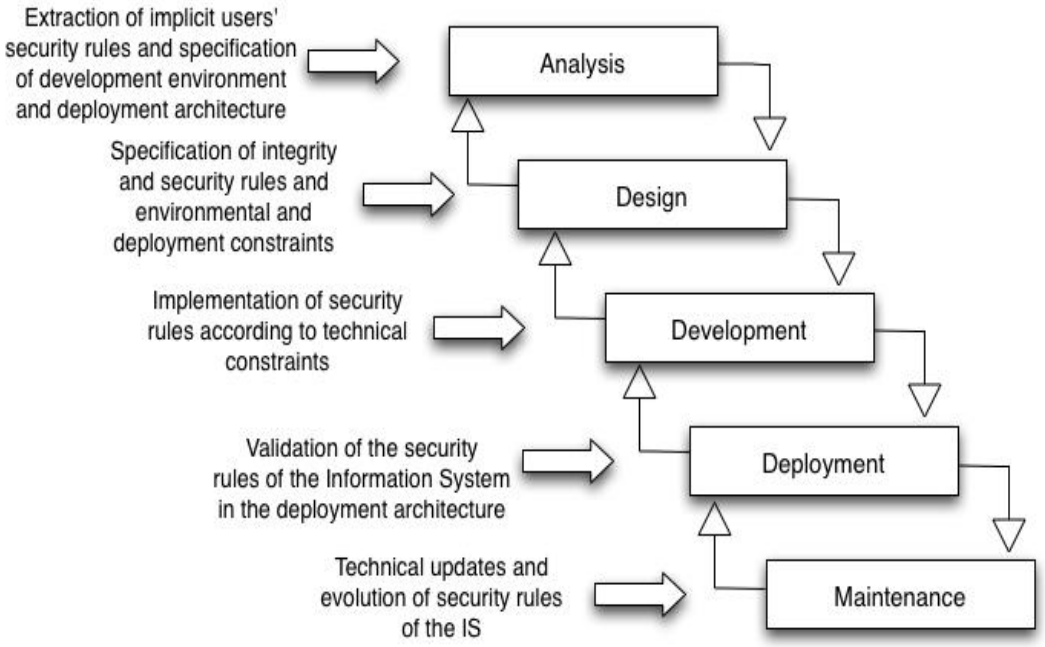

Figure 1. Security dimension into Services Project Life Cycle

During the users' needs analysis phase of common design methodologies, the security dimension of Services system is still not well specified due to the fact that the organizational security is not considered as an autonomous procedure. Indeed, the security dimension is generally delegated to development and deployment steps where they consider the system security instead of the organizational security, which must take place at analysis and design steps. The rethinking of the different phases composing the life cycle take into consideration the security dimension from the first phase to the last one in a continuous way (see Figure 1). The NServicesT standards give a first approach for the establishment of the security continuity across the Services project life cycle. In fact, the first recommendation of these standards is to establish a well-defined information system boundary and to understand the mission or business-case impact resulting from a breach or compromise to the system's confidentiality, integrity or availability (Ross 2007). This comprehension should be achieved through a well-conducted user analysis step.

The analysis of users' needs must emphasize the risk level linked to each category of data and process. It is also primordial that the users' requirements in term of confidentiality and security and the constraints determine the system runtime. Indeed, user actions implicitly trigger security policy decisions (Snene 05), thus the activities of a user and the weak signals surrounding these activities have to be clearly defined. The design phase must take into account the technical platform and the development environment to obtain an adaptable schema that preserves the security specificities relative to the latter. It must also express the users' needs gathered during analysis phase. Finally, it has to express the different security components relative to data process and to the data itself. The development of the Services systems adapts the security requirements of the platform while respecting the schema specifications. In the deployment level, it's imperative to validate the technical platform along with the system constraints and the processes in order to ensure that the system answers correctly to the security criteria. The maintenance

phase guarantees the global security of the system through the platform updates and the adaptation of the architecture and of its security rules to modifications and new needs.

The proposed approach aims to elaborate architecture with a global security context that takes place at the first steps of the design phase and continuously during the next phases. Indeed, it is primordial to determine the different characteristics of every phase to obtain a global knowledge of the system.

At every level of the life cycle, the responsible actor extracts the inherent local constraints. These constraints represent critical information that must be diffused to the other levels. Indeed, they represent guidelines for the elaboration of each step (Snene and Leonard03). This emphasizes the necessity for other actors to be aware of the global system constraints to adapt to the other phases' needs thus maintaining the good continuity of the project. In order to gather all the required characteristics at the different phases, a strict collaboration between the different actors of the project is needed. Therefore technical and conceptual constraints have to be communicated both upstream and downstream of the Services project life cycle (see Figure 2).

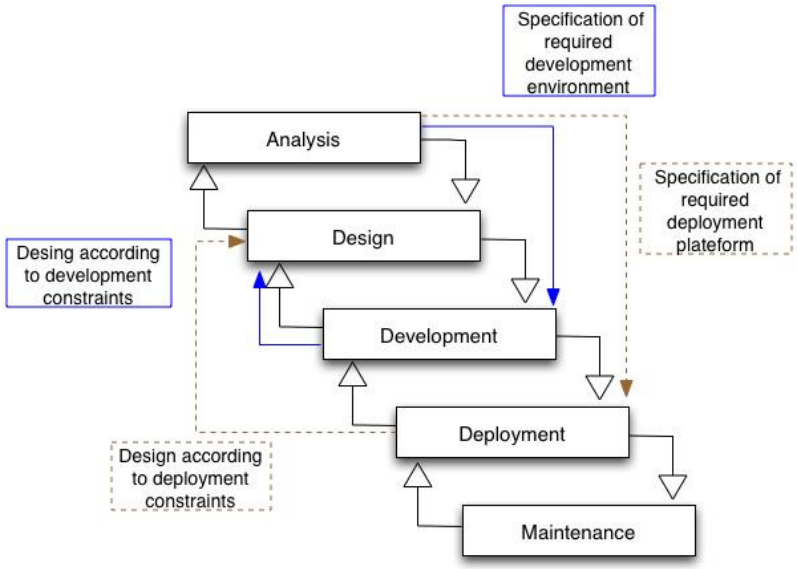

Figure 2. Services Project Life Cycle exchange

2. Open and Distributed Services Project Life Cycle

Among the different emerging architectures, special attention is brought on the open and distributed architectures. Because of its collaborative nature, this type of systems is potentially more sensitive to threats and requires therefore a particular attention. Due to its physical and logical structure, distributed Services architecture needs to reach a high security level to ensure the confidentiality of the transiting information. Indeed, the inherent communication between the different sites, due to the data exchange, increases the risk to encounter information leaks or to undergo an attack. This underlines the need to take into consideration the security along with the distribution.

The Distributed architecture is a collection of distributed data and process over multiple sites that are connected with some kind of distributed system architecture commonly known as middleware. It exposes a common set of services across platforms and provides a homogenous computing environment in which distributed information applications can be built. Today several middleware are available. The most used ones are CORBA (Common Object Request Broker Architecture) from OMG and EJB (Enterprise Java Beans) from Sun (Ekberg 1999).

The existing gap between current Services system design methodologies and the distributed system design models, forced developers to operate modifications on the design models to adapt them to the technology specification. Every middleware specification has its own implementation constraints and generally, designers do not have enough knowledge about the implementation process to realize a well-designed system that corresponds to the implementation model without major modifications (Hirschfeld 1996).

In fact, methodologies for the design of Services architecture have usually paid modest consideration to distribution and communication characteristic of systems. However, this situation has improved: architectures have noticeably grown in dimension and range, organizations want many of their separate systems to be integrated and consequently the number and the variety of geographically distributed users of these systems have become wide. The distributed systems community has produced methodologies for the design of distributed systems (Casal, and al 2002). However, these methodologies pay little attention to the information aspects of distributed Services architecture; instead their strength is in the distribution and communication aspects of these systems. The information aspect still not well specified in these methodologies despite its importance for the Services system survival (Sol and Grosslin 1992).

Firstly, in the analysis level, we extract the users' needs that the open system has to answer. These different needs are categorized into different functional parts that will be distributed among the distribution sites. The subdivision of the system into sub-systems inherently creates data exchange inter or extra-sites. Thus, the open system must be secured from internal and external threats. Indeed, the subdivision into subsystems generates the need to focus into confidentiality and access restrictions to data flow. The separation into functional parts has to be followed by security rules that ensure confidentiality between these parts.

In the analysis level, the deployment platforms must be specified, because every site may have its own architecture. These platform architectures have their own security framework, which are influential for the global security design of the open system done at the following level. At the design level, the distribution decisions and the system schema are conceived taking into account the development environment and the distribution specifications made by the developer, the required deployment platforms and their security constraints. Indeed, the developer has to specify the distribution possibilities in order to optimise data flow between the different sites. In this level, critical decisions related to performances, security and distribution are made. These decisions establish the complexity of the system. At the development step, the design schema is applied implementing the security constraints, using the platforms security frameworks and creating the different sites for distribution. At the deployment level, it is imperative to verify that the communication between the sites respects both design and deployment platform specifications and that the security constraints apply correctly to the distribution. These conditions ensure the Distributed system integrity, correctness and both local and global security. At the maintenance step, any modification in the distributed system has to be followed carefully to ensure the needed continuity of the system hence avoiding any data or security violation. That's why the schema must be adapted and rethought through changes. Moreover, the deployment platform has to be up-to-date in order to prevent any attack through exploits or known bugs (see Figure.3).

3. Knowledge pattern

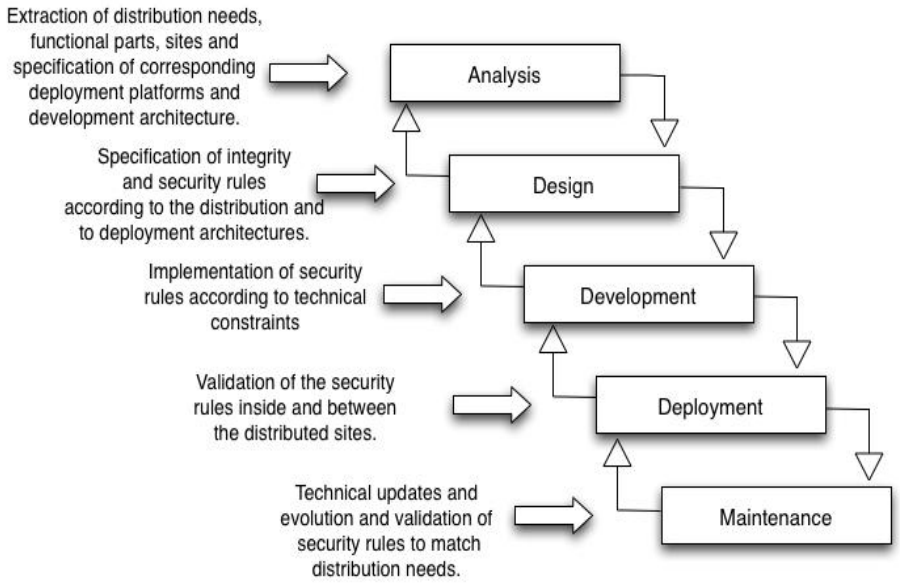

Figure 3. Security requirements of an open and distributed Services system

We define a set of interaction pattern between the various participants in the life cycle of an Services project. These patterns initially treat the exploited knowledge by each one of these participants. Then, these patterns will be the intermediary of communication to establish a form of formal communication between them to allow a better comprehension of the system and thus to ensure a better continuity between the various levels (Hui 2000). These patterns are divided into three categories corresponding to three knowledge interaction levels. The first is the designers' knowledge needed by developers to capture the design environment and to well understand the purposed conceptual diagram. For example, in some special case designers need to fix some critical data on some specific sites to ensure its availabilities in case of network shutdown. Developers usually adapt the purposed conceptual diagram to the technical environment to increase the distributed Services system performance and can, if not noticed, move these data from the specified site to another one without informing designers. The second category is the developers' knowledge pattern that regroups the different implementation parameters and the technical environment specificities. This pattern is useful for designers and assists them in adapting their purposed conceptual diagram to the technical environment. For example, some application servers used to develop Services system (Tomcat, J2EE, …) have their own security framework. Any security approach used to design the Services system must fit into this framework. Finally we purpose a knowledge overlap pattern that encapsulates the common knowledge between the designers and the developers. An example of this knowledge is to exception treatment. The technical behaviour of each used technical environment is different from others and its integration in the conceptual diagram must be discussed and approved by both developers and designers.

As explained above, three knowledge categories are distinguishable in a Services system project. These zones are called Knowledge zone. In fact, every zone is characterized by its own knowledge that is needed by other partner to successfully achieve the project. To put this knowledge under a formalized form comprehensible by different project partners who do not

share the same technical and knowledge language, we define it as knowledge pattern form. These patterns will contain and express the different needed knowledge contributing to ensure a normal continuity for the project. Three different pattern categories are identified.

The first pattern category is the Services system design knowledge pattern. This pattern conceptualises the diverse knowledge used to design the system and to produce it under the form of schema. As this schema is the result of a transformation of different information collected from end users, this transformation, as any other transformation, causes some data loss. Two domains are concerned by this data loss: the conceptual domain and the process domain. The first domain represents the different data that make up the system. They are represented under the form of a design schema. But, some of the collected data cannot be represented under this form due to their particularity. Nevertheless, these data can be important for developers for a better understanding of the given schema. For example, on a database table, designers can specify a user-address attribute under many fields, which are street-number, street, and town. This specification can result from a user need, such statistical use or for a further evolution. At the development step, as these needs are not clearly expressed in the design schema, developer for better performance and to decrease the answering time can unify the different fields in a unique one that represents the same data. In this case, the obtained Services system will provide users with the same data then the wanted Services system. But at the time when the end users will try to get the statistical information which concern only a part of the global address such as the street, this information will be unavailable and the Services system will be unable to satisfy the users' requirements. The second domain is the process domain that represents the different Services system functionalities. These functionalities are represented under the form of functions and transactions. At the design step, designers usually do not specify the runtime manner and site needed for each transaction. However, this information can be vital for some critical functions. The term critical does not indicate their system aspect but the organizational one. In fact, if we suppose that the statistical functionalities mentioned in the example above will be used for each new data entry, and if we take the case of an international interim worker company, the location of this functionality runtime is crucial to determine the system answering time and its quality of services.

The second pattern category is the Services system development knowledge pattern. This pattern regroups structural and technical information related to the development languages and the runtime platforms. Such information is generally extracted from the different specifications of the used technologies. This information is expressed through the developed system but still hard to get at the design step. In fact, designers usually ignore the technical constraint that can be transgressed by their produced design schema. This obliges the developers to adapt the design schema to their used technologies and by the way to carry out some modifications that can be harmful to the system objectives. The different knowledge that must be considered by these patterns concern: security, performance and availability.

The security knowledge is of two kinds. The first is the data security framework defined and requested by the technical platform. Indeed, according to the used platform different protocols exist and must be respected at the design step. This special requirement must be expressed at the beginning of the design step by a clear and formal specification. Different forms can be used to express this specification, but the most comprehensible one is to provide the designer with a conceptual schema that must be integrated in the system schema. For example, the Tomcat server obliges the designer to use a unique security framework defined by the Tomcat specification. The security data that are: User, Login, Password and Session must be defined as fields of a unique table called security. Otherwise, even the data are protected by other control checks and constraints; the transition from the login step to the session step cannot be done. In fact, if the requested data are expressed under another form than the one that the platform specifies, they cannot be treated.

The second security knowledge form is the system security Knowledge. In fact, any Services system needs to be protected from external or internal attacks. For this purpose, developers use different technologies and define different lockers on the system. This set of technical information must be transmitted to designers by indicating the security constraints that will be applied to the design schema at the development step. For example, using distributed databases must be accompanied by a data flow restrictions. This means that the critical data will be used just on a defined site and cannot be called by non-authorized sites. If designers distribute the different process without taking in account this fact, the security can be transgressed and the critical data are doomed to be in danger. Moreover, due to the dynamic technical and organisational environments that surround organisations, continuous monitoring of a Services security controls is becoming increasingly important. Changes to hardware, software, operations and implementing technologies have the potential to perturb the system's security state at any time (NServicesT 2006). The proposed pattern will work as a repository of common technologies issues and known security's leaks cases in order to avoid security discontinuity inside the system that can be occurred during a technical evolution or a platform change.

The third pattern category is the Services system knowledge overlap pattern. By overlap we designate the overlapping zone between the design and the development step. This knowledge results from the mixture of different kinds of information kinds related to both domains. It is composed mainly of three different components that are: data location, site relation and integrity constraint.

The data location concerns the distribution manner of the data. Indeed, a part of the data cannot be distributed in a classical way (fragmentation and allocation (Özsu and Valduriez 1999)). Some data is needed on specific site because of its importance to the good running of this site, or due to the end-user request. This information results both, from designers and developers. It implies that the design and implementation constraints have to be resolved jointly. The first pattern component's aim is to regroup these different locations' information in a formal way to facilitate the resolution of their related different restrictions.

The site relation component is the set of different information concerning the different existing relations between the sites. These relations can be of different kinds. The first relation category is the organizational relation. In fact, some needs require special connection between the different organization sites. For example, some Services systems with critical sites are completely replicated due to their information importance. In such case, each site must inform the others of its correct management. It has also to maintain its backup site correctly and continuously. The second relation category is technical. Indeed, some technical constraints need some specific relations between the different existing sites. For example, using the J2EE platform requires the establishment of a special permanent connection between the different application containers in order to guarantee their coherences.

The integrity constraint component represents the set of the different sorts of information needed to design and implement these constraints correctly between the different sites of the distributed Services system. In this approach, we have a special focus on this component due to its major importance and impact on the system good running. The different data existing around the integrity constraints are necessary for a good implementation of these constraints. In fact, as these different constraints regroup and result from different system environments such as the organizational, the security and the technical, it is necessary to classify these data in order to obtain the best possible level of system integrity.

4. Conclusion

The set of Knowledge Patterns extracted from a Services system project are a group of proven reusable assets that can be used to increase the speed of developing and deploying distributed applications. These patterns have to help and to identify the interaction and processes of selecting and runtime topology. They will provide enterprise developers with a set of guidelines for building information application, including performance, technology options, application design, development and security. These patterns will aim to reduce the existing gap between information designers and the developers by providing them with a unified interaction language. This language will enrich the design step by new concepts, which help developers manage the distribution step while respecting the project goals. They will also provide designers with different information summarizing the technical environment with its constraints. Such information is important due to the modification that has to be done on the design schema to be adapted it to the technical platform. Finally these patterns will define a formal way of communication between the different participants of the project. It will be useful specifically in the overlap domain case.

The knowledge patterns facilitate the communication between the different actors of the Services project. In our approach, we use these communication forms to share the different information every actor possesses and to contribute to the continuity the Services project. The set of actors are able to define their different security needs. Our approach does not have a static view limiting security to the deployment phase but considers it in a continuous way and tries to provide the needed information for everyone to work in a complementary and homogeneous manner. Organizational security rules, considered as a fundamental part of the Services security policy, and weak signals can therefore be clearly defined and taken into account at every level of the project.

The proposed approach aims to elaborate Services with a global security context that takes place at the first steps of the design phase and continuously during the next phases. Indeed, it is primordial to determine the different characteristics of every phase to obtain a global knowledge of the system. At every level of the life cycle, the responsible actor extracts the inherent local constraints. These constraints represent critical information that must be diffused to the other levels. Indeed, they represent guidelines for the elaboration of each step. This emphasizes the necessity for other actors to be aware of the global system constraints to adapt to the other phases' needs thus maintaining the good continuity of the project. In order to gather all the required characteristics at the different phases, a strict collaboration between the different actors of the project is needed. Therefore technical and conceptual constraints have to be communicated both upstream and downstream of the Services life cycle.

The organisational security level is not addressed using technical means. Contrary to the system security, the organisational security takes place in the early phases of the Services project life cycle. For this purpose, a global security taking into account the described levels of security has to be thought in a continuous way during every step of Services project life cycle: analysis, design, development, deployment and maintenance. A continuous and enforced global security is needed in order to avoid any security breach in the system. To ensure the system survivability, security has to be a key component at every step of the life cycle. Discontinuity of security represents a major issue and can lead to vulnerabilities in the Services and thus endangering the Services integrity. Indeed, skipping security at one phase of the project exposes the Services to major risks. Beside their primary task, it is imperative that every step in the Services project life cycle remains focus on security.

This approach tends to demystify the existing gap between the organisational environment and the technical one. In fact, due to the lack of communication between project actors, the compromise between technologies usability and users requirements is hardly achieved and often

on security depends. Making the security the cornerstone of any Services project based on distributed and open architecture will be helpful for the system continuity. In fact with a well-established requirement analysis and considered technical constraints, the obtained system can be considered as a sustainable system. This includes the fact that the system will be easily modified and any evolution can be included without leading to complete rethinking of the system or the system security. More over, the business continuity plan of the system will be embedded in the Services, since that the SA&D are thought basically while thinking into account the security notion and integrating answers to security risks will be much easier than in a classic project lifecycle. All of these facts will leads to make closer the security aspects and the use of new architectures and by the way enabling the production of more efficient and more secure Services.

References

[1] A. Ekberg, Enabling technologies for web centric applications, PhD thesis, Lund institute of technology, November 1999.

[2] H.G. Sol, R.L. Crosslin, Dynamic modelling of information systems II, 1992.

[3] J.A. Casal, J.A. Garda, R.G Vazquez, S.R. Yarrez, A practical experience in analysis and design of distributed information systems, I+D Computation, Vol. 1, No.1, July 2002.

[4] K. Hui, Knowledge Fusion and Constraint Solving in a Distributed Environment, PhD Thesis, University of Aberdeen, 2000.

[5] M. Snene, J. Pardellas, M. Leonard, Information system architectures: where are we?, ICTTA Conference, IEEE Press, 2004.

[6] M. Snene, Knowledge patterns of distributed information systems- the case of distribution design and implementation based on integrity constraints optimisation, Ph.D thesis, N576, Geneva University, 2004.

[7] M. Snene, M. Leonard, Distributed Framework for real time web based collaboration: M7TOOL CASE, AICCSA Conference, IEEE Press, 2003.

[8] M. Snene, Secure design and implementation of distributed and interoperable Services based on overlap knowledge pattern, IBEC conference, 2005.

[9] M. Swanson, J. Hash, P. Bowen, Guide for Developing Security Plans for Federal Information Systems, NServicesT Special Publication 800-18, 2006.

[10] M.T. Özsu, P. Valduriez, Principles of distributed database systems, Prentice Hall Edt, 1999.

[11] R. Hirschfeld, Three tiers distributed architecture, Proceedings PloP 96, 1996.

[12] R. Ross, Managing Enterprise security risk with NServicesT standards, Computer Journal, IEEE Press, Volume 40 number 8, 2007.

[13] T. Grance, J. Hash, M. Stevens, Security Considerations in the Information System Development Life Cycle, NServicesT special publication 800-64, 2004.

[14] R. Gonggrijp, W.J. Hengeveld, Nedap/ Groenendaal ES3B VOTING COMPUTER: a security analysis, We do not trust voting computers Foundation, Netherlands, 2006.