8-2012

An approach in defining Information Assurance Patterns based on security ontology and meta-modeling

Aniruddha Bhaumik

University of Nebraska at Omaha

Follow this and additional works at: https://digitalcommons.unomaha.edu/studentwork

Part of the Computer Sciences Commons

Recommended Citation

https://digitalcommons.unomaha.edu/studentwork/2871
An approach in defining Information Assurance Patterns
based on security ontology and meta-modeling

A Thesis

Presented to the
Department of Computer Science

and the
Faculty of the Graduate College
University of Nebraska

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

University of Nebraska at Omaha

by

Aniruddha Bhaumik

August, 2012

Supervisory Committee:

Dr. Mansour Zand

Dr. Harvey Siy

Dr. Robin Gandhi
An approach in defining Information Assurance Patterns based on security ontology and meta-modeling

Aniruddha Bhaumik, M.S.
University of Nebraska, 2012
Advisor: Dr. Mansour Zand

We have to realizing that information is the most important asset. We are at a stage where the information is not only becoming a valuable asset but also the volume of it is increasing at a rapid rate. Modern Enterprises are now working hard to ensure the proper storage, availability and integrity of this information. So we are having the information assurance discipline which is focusing on preservation of information confidentiality, integrity and availability in all of the informations’ various state. We have the information security patterns. But we have to realize that we have to design the Information assurance patterns which can be reusable solutions of addressing information assurance in enterprise-level information engineering. In this paper we are going to propose five Information Assurance patterns based on meta-modelling. And then we will be integrating the existing security ontology to the derived Information Assurance patterns. By this we can get the benefits of using patterns and at the same time we can use the concept of security ontology in our Information Assurance Patterns.
Acknowledgement

I am extremely grateful and indebted to my thesis advisor, Dr. Mansour Zand, for accepting me as his student. His constant guidance, encouragement, and support throughout the completion of my work were the key for my thesis. I owe a special thanks to Dr. Harvey Siy for his invaluable guidance and enlightening discussions on the topic which made this thesis possible. I am also grateful to Dr. Robin Gandhi for providing valuable feedback and comments on my thesis.

I would like to thank all the professors of the Computer Science department at University of Nebraska - Omaha, who taught great courses. I would also like to thank several students in our research group, and my friends for their support, motivation, and help during my stay here at Omaha.

Most importantly, I would like to thank my mother and father, Asha and Dr. G.C.Bhaumik, and my younger brother Anagh, who have cared for me continuously, in every possible way, from halfway around the world. I would also like to thank my sister Arundhati and my brother-in-law Malabendu for being my local guardian in United States and constantly supporting me. Last but not least I would like to thank my wonderful wife Sharmistha for standing beside me throughout my career. She has been my inspiration and motivation for continuing to improve my knowledge and move my career forward.
# Table of Contents

1. Introduction ........................................................................................................... 1

   1.1. Introductory Note .......................................................................................... 1

   1.2. An Overview of Security Patterns ............................................................... 2

   1.3. Our Approach ............................................................................................... 4

2. Information Assurance Lifecycle ........................................................................... 6

   2.1. Implementing IA lifecycle within System Lifecycle ....................................... 9

   2.2. Discussion ..................................................................................................... 10

3. IA MetaModel ........................................................................................................ 11

   3.1. Session Metamodel....................................................................................... 12

   3.2. Input-Specification Metamodel........................................................................ 13

   3.3. Risk Metamodel............................................................................................. 14

   3.3. Responsibility Metamodel.............................................................................. 15

   3.3. Environment Metamodel................................................................................ 16

   3.3. Discussion..................................................................................................... 17

4. Security Ontology.................................................................................................. 18

   4.1. Existing Security Ontologies.......................................................................... 19
4.2. Discussion ........................................................................................................ 25

5. IA patterns ............................................................................................................. 26

5.1. Deriving IA Patterns ......................................................................................... 27

5.2. IA Patterns ........................................................................................................ 31

5.2.1. Authentication Pattern ................................................................................. 33

5.2.2. Risk Pattern ................................................................................................. 37

5.2.3. Session Pattern ............................................................................................ 41

5.2.4. Check-Input Pattern ..................................................................................... 44

5.2.5. Environment Pattern .................................................................................... 47

5.3. Implementing Session Pattern for a scenerio ................................................... 50

5.4. Discussion ........................................................................................................ 52

6. Conclusion and Future Work ............................................................................... 53

6.1. Note to Conclude ............................................................................................. 53

6.2. Future Scope .................................................................................................... 54

REFERENCES .......................................................................................................... 55
List of Figures

Figure 2.1: Stages in IA lifecycle..............................................................7

Figure 2.2: IA within system lifecycle..................................................10

Figure 3.1: Session Meta-Model............................................................12

Figure 3.2: Input-Specification Meta-Model...........................................13

Figure 3.3: Risk Meta-Model.................................................................14

Figure 3.4: Responsibility Meta-Model................................................15

Figure 3.5: Environment Meta-Model...................................................16

Figure 4.1: IDS Ontology..................................................................19

Figure 4.3: Security and Dependability Taxonomy.............................22

Figure 4.4: Threat Section Model.......................................................23

Figure 4.5: User-Environment Ontology..........................................24

Figure 4.6: Session Ontology...............................................................25

Figure 5.1: Pattern Specification Using Metamodelling Architecture....27

Figure 5.2: UML Diagram for Authentication Pattern.......................35

Figure 5.3: Sequence Diagram for Authentication Pattern..................36
Figure 5.4: UML Diagram for Risk Pattern ................................................................. 39

Figure 5.5: Sequence Diagram for Risk Pattern ......................................................... 40

Figure 5.6: UML Diagram for Session Pattern .......................................................... 42

Figure 5.7: Sequence Diagram for Session Pattern ..................................................... 43

Figure 5.8: UML Diagram for Check-Input Pattern ...................................................... 45

Figure 5.9: Sequence Diagram for Check-Input Pattern .............................................. 46

Figure 5.10: UML Diagram for Environment Pattern .................................................. 48

Figure 5.11: Sequence Diagram for Environment Pattern ............................................ 49

Figure 5.12: Implementation of Session Pattern ......................................................... 51
CHAPTER 1

Introduction

1.2 Introductory Note

There are various security patterns in the sense originally defined by Christopher Alexander, applied to the domain of information security. A security pattern encapsulates security expertise in the form of worked solutions to the recurring security problems, presenting issues and trade-offs in the usage of the pattern. We want to use the existing security patterns ideas to design the Information Assurance (IA) patterns. But before getting the IA patterns we will have the IA metamodels. The IA metamodels will follow the same idea as we have in security metamodels. We will extend couple of security metamodels and will also be using the same security metamodels as IA metamodels. This document presents our original research into information assurance patterns derived from the IA metamodels and then integrating existing security ontology into IA patterns. We want to integrate the security ontology with IA patterns because the IA patterns will become more efficient by using the related security ontology. And as ontology is machine readable, so our IA pattern can be automated. We will also be reviewing existing security patterns to design the IA patterns.

In this paper, first we are going to present the brief overview of the security patterns, including discussions of security ontology; patterns templates, IA lifecycle and IA metamodel. We outline our approach to the security patterns, and then introduce the IA lifecycle, along with the IA metamodel. We discuss some other related efforts in the IA patterns space and present our tentative conclusion.
Security is one of the non-functional goals with which developers must be concerned about while designing a system. And the security policies falls under the Information Assurance domain. There are various existing security patterns, but there are no such IA patterns. So this motivates us to propose the IA patterns based on metamodels and existing security ontology.

Like existing security patterns IA patterns will be proposed as a means of bridging the gap between developers and security experts. IA patterns are intended to be used and understood by developers who are not security professionals. IA patterns are meant to be constructive and educational. IA patterns will try to provide constructive assistance in the form of worked solutions and the guidance to apply them properly.

Before designing the IA patterns we will be going through various existing security patterns, so that we can get an idea of designing the patterns. In the next section we are going to discuss about some existing security patterns. We outline both the structural and procedural security patterns from various papers.

1.2 An Overview of Security Patterns

The overview of security patterns has been derived from “Final Technical Report: Security Patterns for Web Application Development” “[50] by Darrell M. Kienzle and Matthew C.Elder. The basic idea of security patterns evolved from system design patterns. We are now comfortable in having the patterns approach almost all the areas of software engineering. Alexander invented the concept in his writings on architecture and urban planning. He developed the approach in order to capture the essential knowledge of his field and provide a methodology of pattern approach. His patterns ranged from
high level to low level while he was considering the system architecture. In his words a pattern is “a solution to a problem which occurs over and over again in our environment and that describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”. Vlissides paraphrased Alexander’s definition of a pattern as “a solution to a problem in a context”. Vlissides added that in addition to the problem, context, and solution, a pattern must include recurrence, a teaching component, and a name. Alexandrian patterns generally include the following major elements:

• We need standard name for a pattern.

• We need the summary of problem which a pattern will be addressing.

• We need the solution to this problem.

• What will be the impact of the pattern when we are using it?

Security patterns in the sense originally defined by Christopher Alexander, applied to the domain of information security. The paper “Final Technical Report: Security Patterns for Web Application Development” by Darrell M. Kienzle and Matthew C.Elder identified two broad categories of security patterns:

Structural patterns. “These are patterns that can be implemented in the final product. They encompass design patterns, such as those used by the Gang of Four. They often include diagrams of structure and descriptions of interaction”.

Procedural patterns. “These are patterns that can be used to improve the process for development of security-critical software. They often impact the organization or
management of a development project. It is important to note that there are a number of different efforts bearing the name “security patterns”. There is no single, agreed-upon definition of a security pattern, though general consensus is emerging.”

1.3 Our Approach

Our information assurance patterns effort focused on three components: the IA lifecycle, IA metamodel and security ontology. After going through these three sections the IA pattern will be derived. In this section we outline our approach in each of these areas.

Our approach on the security issues and security patterns consisted of surveying related work. The Hillside Group maintains a resource for writing general patterns and pattern languages. We identified common fields and overlapping categories in existing patterns templates as the basis for our IA patterns template. We then augmented those categories with additional security-specific elements. Throughout the course of the project, we did extensive work documenting security patterns and revising those patterns based on review and feedback.

Our next step involved in designing the IA lifecycle. IA lifecycle relates to the security lifecycle in broader sense. We have improvised the system lifecycle by inserting the IA lifecycle within. The IA lifecycle will provide us with the stages on which we can generate the IA patterns. The seminal part of the IA lifecycle will have Assess, Design, Deploy, Manage and Educate at all phases.

The IA metamodel was designed on the basis of existing security metamodels. The IA metamodels provides the base from which the IA patterns are derived.
After establishing the IA metamodel, we will be discussing the existing security ontology. While deriving the IA patterns we will see that how we can use the existing security ontology.
Chapter 2

Information Assurance Lifecycle

We must understand a system accurately to know what security applications or services are needed for that system. The key of creating useful, transparent and enforceable network security comes from adopting a process that provides careful identification of network resource, access requirements, and data-driven implementation and management services. The process of having secured systems must follow a proper Information Assurance policy that significantly improves information availability, integrity and confidentiality. This chapter will discuss about the IA lifecycle; what pieces are needed to address all aspects of IA, and how often they should be addressed. According to the paper “Creating, Implementing and Managing Information Security Lifecycle” the fundamental security management life cycle will have five interrelated processes: Assess, Design, Deploy, Manage/Support and Educate.

These five elements which form the security management system can also be applied while we are looking at Information Assurance, because IA is the bigger domain where we have the security management system. Likewise we can have these five interrelated processes for our IA lifecycle.
Figure 2.1 describes the five stages in IA lifecycle which is the modified part of the security management lifecycle. In the next sections we are going to discuss briefly about the five stages.

**Assess:**

While designing a system following the IA policies we have to know all the dependency factors. The dependency Information is gathered through this assess process. We have to identify all the types of users who are going the interact with the system, the devices and the services that are related to the system. One of the most important steps in this process is to identify the risk associated with the system. We have to identify the possible attacks and the sub-systems which are prone to the attackers. There will be some proposed security measures; we have to check whether the security measures are in sync with the organization’s legal requirement. What is the legal liability if the system gets
compromised? For the assessment step many organizations use third party assessment services. The collection of data in this step illustrates the breadth of the entire security management environment.

**Design**

Design process starts at the completion of assessment step. In this design process the output data from assess process is very important. On the output data the organization needs to make a fundamental decision. The first step of the design stage is to create standardized levels of security service based on the data gathered in the assessment stage. The Access privileges for different users to the system will be given. The design process works on various interrelationships between information, systems, users and the proposed security needs. In this process we have to start designing a security architecture that will support the organization’s business strategy and IA policy.

**Deploy:**

After the design process we have to follow the deploy process to implement the designs which have been created. The installation of the system, the testing and training all will fall under the deploy process. We have to check the desired processing of the system. The fail-safe must be implemented in the deploy stage. How a system will react when it is vulnerable to some defined attacks. There must be a reporting system where the system logs will get generated. The deployment process will lead to a list of detailed configuration guidelines. These guidelines also provide the baseline metrics by which security assessment and intrusion detection applications evaluate IA policy compliance once the system has been deployed.
Manage and Support:

After the deployment stage we will follow the manage and support process. In this stage we have to measure the performance data from the network security infrastructure against the goals stated in the IA policy.

Education:

This process can be viewed as the process of evaluation. This step will be executed throughout the above four steps. This step will ensure that the required IA policy is being followed in all the other four stages. The process educates the system designer with all the emerging threats to system and the countermeasures required for those.

2.1 Implementing IA lifecycle within System Lifecycle.

We have to integrate the IA lifecycle within system lifecycle to realize the importance of IA within a system. IA should be included in all the phases in system lifecycle. We have gone through few papers to see how we can have the security within a system’s lifecycle. We found a presentation “A Comprehensive View of Information Security” [51] by Hasan Sayani et.al. We have extended the model in the above presentation by impregnating IA in place of security. We have to realize that within the IA domain we have the security policies. The figure 2.2 shows how we can have IA implemented within our system lifecycle. IA is an integral part of the system management plan and gets triggered when data gets compromised in any phase of lifecycle. So we have included IA throughout system lifecycle to it is best to incorporate within the manage module. The manage module will interact with all the steps of a system lifecycle.
2.2 Discussion

As because we are going to propose the IA patterns so it is very important to know the IA lifecycle and how it affects in system design. This chapter focuses on IA lifecycle and how the lifecycle can be implemented within a system lifecycle. The IA Lifecycle stages work as a closed loop system, allowing the information cycle to grow and respond to changing network needs and conditions. When the IA lifecycle gets embedded within system lifecycle, it can manage or go through every step. This ensures the implementation of IA through every stages of system design. The next chapter will define the IA meta-models which will be one of the stepping stone in defining the IA pattern.

![Figure 2.2 – IA within system lifecycle](image-url)
Chapter 3
IA MetaModel

This chapter will go through the IA Metamodels which will be used in the process of designing the IA patterns. The IA meta-model is a conceptual model for Information Assurance engineering. The IA meta-model extends the existing usable secure requirements engineering Metamodels. The meta-modelling concepts are included which allow the usability of tasks, and the usability impact of IA pattern design decisions to be modelled. The meta-model will explicitly define concepts and associations which allow a ‘context of use’ to be modelled. The guiding principle which will be followed when devising the IA meta-model is, where possible, we will extend related usable secure requirements engineering meta-models in IA Engineering so as to reuse existing concepts; this involves simplifying relationships to make the conceptual associations clearer. So while defining the IA metamodels we will follow the paper “A meta-model for usable secure requirements engineering” by Shamal Faily and Ivan Fléchais.

We have sub-divided the IA meta-model analysis into five views as per the Security Metamodel. Four of these are; Session based on Task Metamodel for usable secure requirements engineering with the session class added, Input Specification Metamodel based on Task Metamodel for usable secure requirements engineering, Risk Metamodel which follows the same risk meta-model for usable secure requirements engineering, and Responsibility which follows the same responsibility model for usable secure requirements engineering corresponding to different perspectives of a secure system's context of use. The fifth centres on the concept of Environment. By introducing the
concept of environment as the fifth type, we want to view the environment within which the system will be working. The following sections will illustrate the different metamodel sections.

3.1 Session Meta-Model

Figure 3.1 - *Session Meta-Model*

The Session Meta-Model, illustrated in figure 3.1, has been derived from the task metamodel for usable secure requirements engineering. We have included the session object to track the session for a user (here it is Persona) while carrying out a task. It also captures elements describing how people carry out work in the system. In this metamodel people are characterised using personas. A persona is defined as how an indicative user behave, think, what they wish to accomplish, and why. Personas have the sense of all the participants to the context of use and, through the interaction between personas and their tasks; help identify assets, threats, and vulnerabilities which would otherwise be missed. A task is composed of a textual scenario which describes how the persona carries out some work associated with the system using the required asset. According to the diagram
each task will have an aggregation relationship with the personas. Again for each persona the usability attributes are specified. These usability attributes are defined from the perspective of the persona. These attributes are values relating to task duration, task frequency, task demand, and the task's support for different persona's interactions. The session object will track the session information for the user. The session object is associated with a task rather than a persona. So the session object will store the related session information for a task executed by persona. With this session metamodel we are going to define a IA pattern that will be discussed in chapter 5.

3.2 Input Specification Meta-Model

![Figure 3.2 – Input-Specificaton Meta-Model](image)

The IA meta-model are being proposed to help us in defining the IA patterns. So according to that we have to realize the use of IA within our system. When we have discussed about the IA concept we have emphasized on three important concepts, Confidentiality, Integrity and Authentication. So while defining the IA metamodels we
have to focus on these three concepts. We must have in our mind that we will need to check integrity within our system. For that we are proposing the Input-Specification Metamodel shown in Figure 3.2. The Input Specification meta-model will follow the concept of task meta-model for usable secure requirements engineering, but we are going to add the Input Specification with the task module. So for every task the inputs must be validated so that the Integrity of the system can be maintained. Again we will be using the Input Specification Metamodel to derive our IA pattern.

3.3 Risk Meta-model

![Risk Meta-Model](image-url)
The Risk Meta-Model, illustrated in figure 3.3[8], is the same as the risk model for usable secure requirements engineering. It models the elements contributing to the definition of risks, and will also incorporate concepts for responding and mitigating these risks. A single risk aggregates a single threat and vulnerability. When a threat is defined, security attributes are associated with it depending on the assets the attacker wants to exploit and how they are going to exploit them. For each defined risk, a Misuse Case has been specified to describe risk impact. The mitigation of a risk is preventing or avoiding it, detecting it, or reacting to it. We can associate goals with mitigating responses. Countermeasures are defined to mitigate the risk. We will see in chapter 5 how we can use this risk metamodel and also we will be showing the mitigation step in IA pattern.

3.4 Responsibility Metamodel
The responsibility metamodel that is illustrated in figure 3.4 has also been copied as it as from the responsibility metamodel for usable secure requirements engineering. Roles are an important concept within IA metamodel. Roles define the various users who are going to use the system. The metamodel shows the various roles used when accessing the asset. So an attacker can be tracked by this responsibility metamodel. Again the role is connected to dependency which will assign the task and the asset respect to roles. We are going to use the responsibility metamodel to derive the Authentication Pattern.

3.5 Environment Meta-Model

![Environment Meta-Model Diagram]

Figure 3.5 - Environment Meta-Model
The environment metamodel that is illustrated in figure 3.5 has also been copied as it as from the environment metamodel for usable secure requirements engineering. The environment metamodel is designed to track all the related attributes of the environment in which the system is designed. We have to know the users, the attackers, the threats etc which will be interacting with the system. Using this environment metamodel we will be designing the Environment IA pattern. We have to wait for sometime to get to the chapter 5.

3.6 Discussion

Whenever we will be deriving the IA patterns there will be a need for the IA metamodels. Because based on IA metamodel we are going to design the IA patterns. We have modified and used the existing usable secure requirements engineering meta-models by introducing concepts which facilitate the modelling of a system's IA needs. The IA metamodels that have been discussed in this chapter will be used in defining the IA patterns. We will see next what are the existing security ontologies we can use in defining the IA patterns. Because we are going to use both IA metamodels and the security ontology to define our IA pattern.
Chapter 4

Security Ontology

In defining the IA patterns we have walked through the chapters on IA lifecycle and meta-model. But without the use of ontology we will not be able to define the patterns. Ontology, in the field of knowledge representation, is most often defined as “a representation of a conceptualization” [35]. A more detailed description of ontology is that it is a formal representation of the entities and relationships which exist in some domain. It should also represent a shared conceptualization in order to meet any useful purpose [36]. Ontologies are useful for representing and interrelating many types of knowledge. Marc Donner was the first to urge the necessity of having good security ontologies. He argued that too much security terminology is vaguely defined, thus it becomes difficult to communicate between colleagues and, worse, and confusing to deal with the people we try to serve: “What the field needs is ontology – a set of descriptions of the most important concepts and the relationships among them. A great ontology will help us report incidents more effectively, share data and information across organizations, and discuss issues among ourselves” [37]. The need for security ontology has been also recognized by the research community in [38].

Since the awareness about information security has grown while designing systems, many security ontologies have been proposed during the last decade. Though there are various security ontologies, but there are various questions. Do they meet all the security requirements? Do they cover all the security aspects? Which ontology a security analyst will choose seeking for security knowledge for the definition of IS requirements? We
faced these questions, and conclude that we will definitely need a general survey of existing security ontologies to find out the use of them in defining our IA patterns. We have studied some papers on security ontology.

4.1 Existing Security Ontologies

We studied various papers on security ontology for defining IA patterns. This section will give a brief overview of the security ontologies that we will be using with our IA patterns.

In the paper *Modeling computer attacks: an ontology for intrusion detection* [22], the authors propose a security ontology names IDS ontology specifying a model of computer attack. The figure 4.1 shows the IDS ontology diagram.

![Figure 4.1: IDS Ontology](image-url)
They have defined the Input-Validation-Error class which is comprised of the subclasses (BufferOverflow, BoundaryCondition and MalformedInput). In the next section where we are going to derive the IA patterns we will show how we can use the IDS ontology within our patterns.

In the paper on NRL Security Ontology [27] has seven separate sub-ontologies Main Security Ontology, Credential Ontology, Security Algorithms Ontology, Security Assurance Ontology, Service Security Ontology, Agent Security Ontology, and Information Object Ontology. Main Security ontology is the ontology to describe security concepts; Credentials ontology is the ontology to specify authentication credentials; Security Algorithms ontology is the ontology to describe various security algorithms; Security Assurance ontology is the ontology to specify different assurance standards; Service Security ontology is the ontology to facilitate security annotation of semantic Web services; Agent Security ontology is the ontology to enable querying of security information; Information Object ontology is the ontology to describe security of input and output parameters of Web services. For our purpose we are more interested in the Credential Ontology. Figure 4.2 gives us the model diagram of the Credential ontology.
Ekelhart a et al. [29] proposed a security ontology framework to provide a solid base for an applicable and holistic IT-security approach for small and medium sized enterprises, enabling low-cost risk management and threat analysis. The ontology is based on four parts: the first part is the security and dependability taxonomy, the second part presents the risk analysis methodology, the third part describes concepts of the IT infrastructure domain and the fourth part provides a simulation enabling enterprises to analyze various policy scenarios. The figure 4.3 shows the Security and Dependability Taxonomy.
The risk analysis methodology follows the basic idea to first enumerate all assets including their values and to determine threats to them then in second step to estimate the probability that a threat will occur and the damage it will cause. We will use this methodology of the Ontology in our derived IA pattern. Figure 4.4 shows the Threat Section Model.

**Figure 4.3: Security and Dependability Taxonomy**
Figure 4.4: Threat Section Model

In the paper “Towards an Ontology-based Organizational Risk Assessment in Collaborative Environments Using the SemanticLIFE” [23] the authors have defined the user environment ontology which captures the concepts of an environment in which users are working. So when we will be designing the IA pattern related to environment within which a system will be working, we will be integrating the user environment ontology to our IA pattern. Figure 4.5 shows the User Environment Ontology.
In the paper “Ontology-Based Access Control Model for Semantic Web Services” [52], the authors have described the session ontology which represents the profile of a particular session. There is the session profile which records all events that happened in each session. The ontology provides a variety of security conditions for supporting dynamic role assignment. We are going to use this session ontology within our session IA pattern. Figure 4.6 shows the session ontology model.
There are various security ontologies defined. We have discussed only few of them. It is really tough to determine which ontology to use. While defining various IA patterns we have used the ontology because, if the design of the IA patterns is based on ontology techniques then it can provide structured information that can be reused and combined. The concept of a IA Pattern is a representation of the defined security patterns and is connected with the concept of security ontology with a provide relationship. In practice, each security pattern is matched with a set of countermeasures during the ontology instantiation. In this way, one can start from the generic security objectives, find the Security Pattern Contexts that match them and, thus, choose specific IA Patterns. In this way, the high level IA requirements and objectives can be fulfilled by implementing the respective countermeasures.
Chapter 5

IA patterns

In the previous chapters we have discussed about the IA lifecycle. The lifecycle helped us to determine the IA metamodel. Then we planned our way to security ontology. The ontology and the metamodel will help us to derive the various IA patterns. While defining the IA patterns we have also read various papers related to security patterns. So while defining the IA patterns we have considered some security patterns. The security patterns encapsulate security expertise in the form of worked solutions to the security recurring problems, presenting issues and trade-offs in the usage of the pattern. The security patterns are broadly classified into two:

Structural patterns. “These are patterns that can be implemented in the final product. They encompass design patterns, such as those used by the Gang of Four. They often include diagrams of structure and descriptions of interaction” [40].

Procedural patterns. “These are patterns that can be used to improve the process for development of security-critical software. They often impact the organization or management of a development project” [40].

So we have combined the ideas of the above two classified patterns while deriving the IA patterns. The following section will describe how we can use the concept of metamodeling and the security ontology to derive the IA patterns. Then we will discuss about the five IA patterns following a pattern template.
5.1 Deriving IA Patterns

While designing the IA patterns we will follow the Pattern Modelling Framework [53]. The Pattern Modeling Framework (PMF) will offer an approach to pattern specification and detection using existing metamodels. We have modified the meta-modelling architecture in Figure 5.1 to derive our IA patterns from existing metamodels.

![Diagram of Pattern Specification using meta-modelling architecture]

**Figure 5.1:** Pattern Specification using meta-modelling architecture

In the figure 5.1 we can see that the patterns can be derived from metamodels. The User Objects are the instances of Instance models. Using Instance Model we can detect the pattern instances. And again the Instance Models are the instance of Metamodel. This Metamodel will help in deriving the IA patterns as IA patterns are defined for the metamodels. We will follow this architecture to get our IA patterns.

The previous chapter gave us the idea on existing security ontologies. We are going to use the security ontology while designing the IA patterns. We are going to integrate the
required security ontology within one of our steps in IA patterns. This way we are going to take the advantage of existing security ontology while using the IA patterns. Each IA pattern will be connected with the concept of existing security ontologies: each IA pattern provides a working pattern integrated with the security ontology. Ontology-integrated IA pattern descriptions will be computer readable and will therefore be suitable for automated processing. We will implement one of our patterns for a scenario.

**Authentication Pattern:** This IA pattern is designed to focus on various roles accessing the defined system. We have used the concept of the **protected system pattern** from Security Design Patterns. This will authenticate the user to use the required asset. This pattern is based on Responsibility Metamodel [Figure 3.4]. The Metamodel shows the relationships between the roles, personas, attackers, etc. According to the login authentication the request is forwarded to the necessary classes. So we can have a pattern based on this metamodel where the IA pattern will ensure that the roles are defined properly. This pattern will integrate the sub-ontology Credential Ontology of the NRL Security Ontology [27]. The Credentials ontology allows for specification of credentials used for authentication purposes [27]. In credential ontology we have different classes for authentication purposes like PhysicalToken, ElectronicToken, Password, RBACCertificate etc. So we can integrate the Credential Ontology within one of the Authentication Pattern steps.

**Risk Pattern:** This IA pattern will reveal the threats for the defined system. It will list the typical threat sources, actions and consequences. In addition, it will also evaluate the risk for each asset within the system. This pattern is based on Risk Metamodel. In Risk
MetaModel [Figure 3.3]; we have the classes for Threat and Vulnerability which are acting upon the class Asset. It shows how we can mitigate the risk so that we can have a countermeasure for the asset. In the next section we are going to derive the risk pattern using this risk metamodel. We will be integrating our pattern with Ekelhart A et al.’s [29] security ontology framework. This security framework ‘knows’ which threats endanger which assets and which countermeasures could lower the probability of occurrence, the potential loss or the speed of propagation for cascading failures.

Session Pattern: The intention of the Session Pattern is to provide different parts of a system with global information about a user. This may be used to facilitate accountability and to enforce privilege violations. This pattern is in-line with the concept of check-pattern in Security Patterns by Ronald Wassermann and Betty H.C. Cheng. This pattern is based on session metamodel [Figure 3.1]. The session metamodel has the session object which is connected to the Task class. So when a user (here it is persona) is accessing the asset, the instance will be stored in a session for the particular task. This session object will enforce the privilege violations. So we will have the session pattern where a checkpoint will be there to check the session of the particular user while accessing the asset. Again the session pattern will integrate the session ontology presented in Ontology-Based Access Control Model for Semantic Web Services [50] paper. There is session ontology will take into account the session variables for a particular user. So if we integrate the ontology with the checkpoint step in our IA pattern, it will use the class User Access to detect users’ session.
**Check-Input Pattern:** The *Check-Input pattern* intends to prevent users from performing illegal operations by offering only valid operations to them. This pattern is based on Input-Specification metamodel [Figure 3.2]. The Input-Specification metamodel has the Input-Specification class which is connected to the Task class. So when a user (here it is persona) is accessing the asset, the inputs are checked for the particular task. So we will have the check-input pattern where an input-validator will be there to check the inputs for a particular task while accessing the asset. Again the check-input pattern will integrate the security ontology presented in “*Modeling computer attacks: an ontology for intrusion detection*” [22] paper the **IDS ontology**. There is the class Input-Validation-Error which is the subclass of Means. The Input-Validation-Error subclass has its own subclasses Buffer-Overflow, Boundary-Condition, Malformed-Input. So if we integrate the IDS ontology with the input-validator step in our IA pattern, it will use the class Input-Validation-Error to check the desired input.

**Environment Pattern:** This pattern will take into account the network or the operating environment on which the system will be operated. This pattern is based on Environment metamodel [Figure 3.5]. The Environment metamodel has the Environment class which is connected to various classes like personas, attackers, risk, threat, task, goal etc. So when a System Administrator is building the server by copying files and packages, he will have the first hand knowledge of the environment on which the server will work. This will also ensure that only the absolutely necessary components are installed on the server. It will also ensure that the administrator knows the function of every component installed on the server and will be able to detect tampering with the server. Again the
environment pattern will integrate the User Environment ontology presented in “Towards an Ontology-based Organizational Risk Assessment in Collaborative Environments Using the SemanticLIFE” [23] paper. There are the classes like Software, User-Environment, and Actor etc which will determine the environment for the server on which a system admin will be working on.

Deriving Check-Input Pattern from Input Specification Metamodel

In this section we are going to show how we can derive one of our IA patterns from metamodels. We have selected Check-Input Pattern as the one to work with. Following these derivation steps we have defined the other four IA patterns. The steps that we followed are:

1: *In Input Specification metamodel the Objects are persona, asset, task and the InputSpecificator.*

2: *We have the instance models like interaction of persona with task.*

3: *So we get the pattern instance for this pattern as User has an interaction with asset through Input Handler for a specific task.*

4: *Hence this pattern instances will form the pattern where User is interacting with InputHandler which in turn interacts with the InputValidator to validate inputs.*

5.2 IA Patterns

According to Chad et.al [52] we have used the concept of “capture a number of demonstrably security-effective techniques from existing designs that can and should be replicated in other systems” while designing the IA pattern. A pattern can be characterized as “a solution to a problem that arises within a specific context” [42]. In addition, a pattern includes not only the solution but also the context and problem for which the
solution is used. Hilside [43] defines a pattern as “Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.” We have also introduced IA Patterns based on the security pattern presented in a survey paper; [46].

While defining the IA patterns we must have a general template. We will follow the template structure from the paper “RePa Requirements Pattern Template” [54].

**Name**: The name of the pattern.

**Context**: The situation in which the pattern may be applied.

**Problem**: The problem the pattern addresses.

**Forces**: The conflicting concerns that need to be addressed.

**Solution**: A detailed specification of the structural aspects of the pattern and how it addresses the problem.

**Application**: How to implement the pattern.

**Known Uses**: Where the pattern has been used.

While defining the patterns we will also follow the following two concepts:

1. A pattern describes both a process and a thing: the ‘thing’ is created by the ‘process’ [44].
2. A pattern language is a network of tightly interwoven patterns that defines a process for resolving a set of related, interdependent software development problems systematically [45].

### 5.2.1 Authentication Pattern

This IA pattern will focus on various roles accessing the defined system. So if the defined system is a role sensitive one, this IA pattern will ensure that the roles are defined properly.

**Name:** Authentication Pattern

**Context:** We will be using this pattern whenever access to resources must be granted selectively based on roles and policies. When designing secure systems, Protected System should be considered based on the access to the system.

**Problem:** It is often desirable or imperative to protect system resources against unauthorized access. In order to do this it is necessary to evaluate requests to determine whether or not they are permitted by policies which define the various roles based access. All requests must be evaluated against the policies; otherwise, an unauthenticated user might abuse the system.

To assure that all access requests are evaluated against the system’s policy, a role based policy enforcement mechanism with the following properties must exist:

- The mechanism must be invoked on every access request.
- The mechanism must not be bypassable.
- The mechanism must correctly evaluate the policy.
- The mechanism’s correct functioning must not be corruptible.

This pattern includes three elements:

1. A “client”, from which all access requests originate

2. A “resource-module”, in which all resources are located

3. A correct, verifiable, incorruptible, non-bypassable “policy-handler”, which enforces policy on all requests from “client” for resources in “resource-module”

**Forces:** *Not Applicable.*

**Solution:** The pattern has a single centralized policy-handler which will handle the requests from clients for all resources in the system. We have integrated the Credential Ontology to the Policy Role. This will make the pattern more robust while accessing the authentication of the client.

**Participants**

- Client — Submits access requests to policy-handler.

- Policy-handler — Mediates requests to access protected resources. Evaluates each access request against a policy; grants requests which are permitted by the role based policy, and denies requests which are forbidden by the role based policy. This cannot be bypassed (no direct access by Clients to Resources is possible).

- Role based policy — Determines whether access to the resource should be granted. Encapsulates the rules defining which Clients may access which Resources.

- Resource-Module — Services requests for access to protected resources.
- **Resources** — The services which the client is requesting for.

**Figure 5.2:** UML Diagram for Authentication Pattern showing the Integration of Security Ontology

**Application:** Clients submit access requests to the Protected System’s Policy-Handler. The Policy-Handler determines whether access requests should be granted or denied by consulting the Role based Policy. If the Policy-Handler determines that an access request should be denied, it discards the request and returns. If the Policy-Handler determines that the access request should be granted, it passes the request on to the Resource-Module; the Resource-Module will give the access to the particular Resource
and returns the response to the Client. Interactions among modules are presented in the following sequence diagram 5.3.

**Figure 5.3 - Sequence Diagram for Authentication Pattern**

Use of the **Authentication Pattern**:

- **Isolates resources** - The system’s resources are isolated by the Policy-Handler from any accesses which do not conform to the security policy within Role Based Policy.

- **Loosens coupling** between security policy and Resource implementation. Resource implementations do not need to be aware of security policy and do not have to be modified when security policy changes, since the policy is enforced by the Policy-Handler.
• Improves system assurability - Only the Policy-Handler and the Role Based Policy implementation need to be evaluated for correctness in order to ensure that the system correctly enforces its security policy.

• Degrades performance - Interposing a Policy-Handler between the Client and the Resource-Module imposes a performance penalty; this penalty may be significant. In network configurations, Policy-Handler are usually the Firewall and their use requires an extra network message for each resource access.

**Known uses:** *Not Applicable.*

### 5.2.2 Risk Pattern

This IA pattern will reveal the threats for the defined system. It will list the typical threat sources, actions and consequences. In addition, it will also evaluate the risk for each asset within the system.

**Name:** Risk Pattern

**Context:** We will be using this pattern whenever we have to formulate the various risks across the System. When designing secure systems, it should be evaluated based on various risk factors.

**Problem:** It is often desirable or imperative to recognize the various risk factors. In order to do this it is necessary to evaluate various levels of risk. Esri GIS Security Patterns [50] are based on three increasing levels of security. Basic Security Risk Implementations, Standard Security Risk Implementations, Advanced Security Risk Implementations. We are following the Risk level structure as in Esri GIS Security Patterns [50].
• **Basic Security Risk Level**
  o The sensitivity of the data will be less like the data that can be shared publicly.
  o The server will have all the architecture tiers, means due to security issues data will not be accessed on different tier with respect to presentation layer.

• **Standard Security Risk Level**
  o Data loss will pose to moderate effect.
  o For a system there will be different architecture tiers.
  o Federated service is needed.

• **Advanced Security Risk Level**
  o The data will be sensitive one which cannot be publicly shared.
  o Components must be redundant and of high availability.
  o System must use 3rd party security components.

**Forces:** *Not Applicable.*

**Solution:** Risk pattern includes three elements:

1. A “risk-module”, in which all levels of risks are identified

2. A “risk-handler”, which will recognize the level of risk and accordingly pass the information to “risk-modulator”

3. A “risk-modulator”, which will mitigate the risk factors.

There are two main sections of this pattern. The first sections has a single centralized risk-handler which will recognise the levels of risk in the system. The second section risk-modulator will mitigate the risk factor. We will also integrate the security ontology with
the risk handler to detect the risk levels and also provides the risk mitigation countermeasures.

Figure 5.4: UML diagram for Risk Pattern showing the integration of *Ekelhart Security Ontology*

*Participants*

• Risk-Module — Contains the various levels of risk.

• Risk-handler — Evaluates the risk levels.

• Risk-Modulator — Mitigates the risk factors.

*Application:*

• Modules of system’s information are passed to the Risk-Handler.

• The Risk-Handler invokes the Risk-Module.

• The Risk-Handler determines the level of the risk. Accordigly it will pass that information to the appropriate Risk-Modulator.
• The Risk-Modulator will then mitigate the risk factors.

Interactions among modules are presented in the following sequence diagram 5.5.

**Figure 5.5 - Sequence Diagram for Risk Pattern**

Use of the **Risk Pattern**:

• Recognizing the Risk Factor.

The system’s informations are examined by the Risk-Handler which in turn determines the levels of Risk.

• Improves system efficiency.
The Policy-Modulator mitigates the risk factors and in turn increases the system efficiency.

**Known Uses:** *Not Applicable.*

### 5.2.3 Session Pattern

**Name:** Session Pattern

**Context:** The intention of the *Session Pattern* is to provide different parts of a system with global information about a user. This may be used to facilitate accountability and to enforce privilege violations.

**Problem:** *Session* objects are very useful to keep security-relevant information like roles, privileges or authentication data. This information enables components inside the system to conduct further checks based on the provided information. A *Session* object can be created after a user has successfully logged into the system. In this way, the system can identify a specific user to be authorized for certain operations by checking the respective *Session* object.

**Forces:** *Not Applicable.*

**Solution:** A *Session* object is created during runtime and shares global data with other system components. In order to facilitate access to a *Session* object from various positions in the system, a mechanism for distributing its instance needs to be implemented. We have integrated the Session Ontology with the checkpoint module; this will ensure that user’s session can be tracked.
Figure 5.6: UML diagram for Session Pattern showing the integration of *Session Ontology*

*Participants*

*Session*

- Stores information that will be provided at the time of a user’s *Session*.
- The stored information is initialized on creation.
- Provides methods for sharing the information.

*System components that use Session*

- Need to know the instance of the *Session* object they use.
- Call methods of session to retrieve information.
**Application:** System internal components that require global information need to have a pointer to the instance of their *Session* object. Depending on the design of the system, they might be given the respective *Session* instance with a user's request to perform some action. Once they have access to the instance, it can be used to query the desired information.

Figure 5.7 depicts the message flow upon a user's access request. After the *Check Point* receives a message from a user it demands the corresponding privilege information from a *Session* object. In the example we outlined here, the access request is permissable according to the privilege structure and the message is forwarded to its intended recipient.
**Figure 5.7** - Sequence Diagram for Session Pattern.

*Use of the Session Pattern*

- Only authorized users may access the system according to the information stored in the Session.
- The Session Information related to any user will be kept secured.
- User’s actions are traced and logged for a particular session.
- By providing role and privilege information in a user's Session object, unauthorized access can be recognized and stopped by internal components.
- Integrity of a user's operations can be checked against his privilege structure in order to identify actions that compromise integrity.

**Known Uses:** *Not Applicable.*

### 5.2.4 Check-Input Pattern

**Name:** Check-Input Pattern

**Context:** The Check-Input pattern intends to prevent users from performing illegal operations by offering only valid operations to them.

**Problem:** Users should be deterred from performing restricted operations. The Check-Input pattern prevents a user to choose illegal actions. In order to achieve this goal, there must be a check-point to check all the valid inputs for a particular operation. This implies that a user's inputs are checked in advance. Thus, a check is performed in advance to the perform a particular operation. Hence, any operation that the user may choose will depend upon valid inputs.

**Forces:** *Not Applicable.*
**Solution:** Check-Input pattern includes three elements:

1. A “input-validator”, in which all the inputs are validated.

2. A “input-handler”, which will recognize the input and accordingly pass the information to “input-validator”

3. A “check-point”, which will check all the necessary required inputs for an operation.

**Figure 5.8:** UML diagram for Check-Input Pattern showing the integration of *IDS Ontology*

*Participants*

- Input-handler — Recognize various inputs.

- Input-validator — Evaluates input validation.
• Check-point — Checks all the necessary inputs.

**Application:**

• User Input is passed to the Input-Handler.

• The Input-Handler invokes the Input-Validator.

• The Input-Validator validates the input and will pass that information to the Check-point.

• The check-point will recognize all the valid inputs for a particular operation.

Interactions among modules are presented in the following sequence diagram 5.9.

**Figure 5.9 - Sequence Diagram for Check-Input Pattern**
Use of the **Check-Input Pattern**:  

- Recognizing the Valid Inputs.

The user inputs are checked and hence this will prevent from being attacked by some invalid inputs.

- Improves system security.

The system security will not get compromised by any hacker.

**Known Uses:** *Not Applicable.*

### 5.2.5 Environment Pattern

**Name:** *Environment pattern*

**Context:** This will take into account the operating environment on which the system will be operated. This pattern will help the system admin in building the server from the ground up; he can understand the default installation of the operating system and applications.

**Problem:** Most system administrators generally do not understand the server complexity. They depend on pre-existing installation programs to configure the system correctly. But there are many things to consider while working with the system server. The types of user interacting with the systems, the vulnerable sub systems and there are lot more to take into account.

**Forces:** *Not Applicable.*

**Solution:** Environment pattern includes two elements:

1. A “environment-detector”, in which the environment for the system is detected.
2. A “installation-detector”, which will recognize the installation needs.

**Figure 5.10:** UML diagram for Environment Pattern showing the integration of *User Environment Ontology*

**Participants**

- Environment-Detector — Recognize the environment for the system.
- Installation-Detector — Recognize the installation needs.

**Application:** The system administrator should attempt to build the server by taking into account of the environment in which the system will work. This will ensure that only the required aspects will be taken into account. It will also ensure that the administrator knows the function of every module in the system and also why it has been implemented. When any modules will get compromised the system administrator will know that what sub-module will be hard-hit. Accordingly he can mitigate the risk manually whenever necessary. Interactions among modules are presented in the following sequence diagram 5.11.
Use of the **Environment Pattern**:

- This pattern will help system administrator to reinstall the system if there are any attacks.
- As this pattern will also have the information related to attackers, it will prevent any attacks.
- This pattern will reduce the complexity of the system and will enhance manageability.
This pattern will improve performance by eliminating unnecessary checks which will otherwise slowdown the working rate of the system.

**Known Uses:** *Not Applicable.*

### 5.3. Implementing Session Pattern for a scenario

In this section we are going to show how we can implement Session Pattern for a particular scenario.

**Scenario:** Bob is having a bank account. He logs in to his account from Chicago. At the same time the account got compromised by a Hacker. Now the hacker wants to access the account from LA.

**Problem:** Bob’s account has been compromised.

**Solution:** Let us use our session IA pattern to prevent the hacker from accessing Bob’s Account. The intention of the *Session Pattern* is to provide different parts of a system with global information about an user. So Session Pattern will track the information of Bob whenever he is using his account. Whenever the hacker is going to access his account from LA, the IA pattern has the SessionHandler class; the SessionHandler will track the session information for that particular hacker. The Session Handler is connected with the Session Ontology where the session informations are stored. So at the same time the account is being accessed from two locations having different IP address. The SessionHandler will throw an error response to the checkpoint class. Checkpoint class is there to determine the authorized access to the account. So the checkpoint class will prevent any interactions with the bank information. And hence the account will not get compromised by an unauthorized user. Figure 5.12 shows the scenerios for both Bob and hacker accessing the bank account.
Figure 5.12 – Implementation of Session Pattern
5.4 Discussion

This chapter has presented the five information assurance patterns which can be adopted to fit the domain of security. We proposed the five patterns that addresses the needs inherent to the development of security-critical systems thereby contributing a number of new aspects to information assurance. Our patterns identify information assurance specific consequences of applying a pattern and formally specifies security-oriented constraints. When applying a pattern related security-specific development principles have been identified to help developers new to addressing information assurance issues. Using our patterns, insight into how IA concerns can be modelled and analyzed is conveyed. In our IA patterns we have integrated the relevant security ontologies. This integration provides us with a more robust IA pattern. Not only we will be using the pattern concept will designing system but also will be getting the help of security ontologies.

Using patterns, IA aspects can be addressed during early stages of software development, expert knowledge can be conveyed and reuse facilitated. Thus, error-prone post-design security modifications can be reduced contributing to the quality of software. Further research may explore how we can incorporate constraints that capture various beliefs into the IA pattern template. Especially interesting would be a way of expressing and analyzing properties of various patterns modules. Finally, we need to combine the various IA patterns to suit the needs in various domains. We need to formalize the domain specific IA patterns.
Chapter 6

Conclusion and Future Work

6.1 Note to conclude

The main objective of our paper was to define some IA patterns based on metamodels and security ontology. The metamodels which we used to derive IA patterns gave us the broad overview of the security issues that are addressed and how they are handled. The security ontology worked on more finer side on us. It gave the granular structure of the security issues. So when we have integrated the security ontology with our IA patterns we have been benefited as solving the security issues more efficiently. In our process we have gone through the existing security patterns, how the security ontology has been used to design the related security patterns. We have identified common fields and overlapping categories in existing security patterns. On the basis of the overlapping usage functions we have derived the five IA patterns which will be provisioning the security services; Availability, Integrity, Authentication, Authorization, Confidentiality, and Non-Repudiation. Before deriving the patterns we have proposed the IA lifecycle based on existing security lifecycle. We have embedded the IA lifecycle within system lifecycle to show how we can realize IA while designing a secured system. We have presented the meta-models for IA engineering. This facilitates the modelling of a system's different contexts of use. The IA patterns have been derived from the metamodels using pattern modelling framework. The security ontology has been integrated to the IA patterns because, if the design of the IA patterns is based on ontology techniques then it can provide structured information that can be reused and combined. One can start from the
generic security objectives, find the Security Pattern Contexts that match them and, can choose specific IA Patterns. We have presented the five information assurance patterns which can be adopted to fit the domain of security. Our patterns identify information assurance specific consequences of applying a pattern and formally specify security-oriented constraints. When applying a pattern related security-specific development principles have been identified to help developers new to addressing information assurance issues. Using our patterns, insight into how IA concerns can be modelled and analyzed is conveyed. The approach described in this paper aims to exploit accumulated knowledge and expertise in the field of security requirements.

6.2 Future Scope

Throughout our paper, we have tried to define some IA patterns based on the knowledge in the field of information security. We have identified five patterns. But there are lot more IA patterns that can be derived. An extensive study to have more IA patterns based on various domains can be done. As we know security ontologies can vary from one domain to another where as the metamodels are domain independent. So we can derive a specific pattern based on metamodel and can have various variations of it dependent on domain when we integrate with domain specific security ontology. Again the proposed patterns can be fine-tuned. Like existing security patterns the “related patterns” for a defined IA pattern can be specified. The IA patterns can be used solving the real-life instances. This paper is just a stepping stone in realizing the need for Information Assurance.
REFERENCES


Graduate School of Mgmt & Technology University of Maryland University College


[3] Building Security into the Software Life Cycle, Marco M. Morana, Senior Consultant, and Foundstone Professional Services a Division of McAfee


[6] Information Lifecycle Management: An Analysis of End User Perspectives; James E. Short; 06-01 January, 2006


[19] Axel van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models", in ICSE '04: Proceedings of the 26th International Conference on


[40] Darrell M. Kienzle, Ph.D.Matthew C. Elder, Ph.D.David Tyree, James Edwards-Hewitt----“Security_Patterns_Repository_Version1.0”
http://www.scrypt.net/~celer/securypatterns/repository.pdf

[41] Chad Dougherty, Kirk Sayre, Robert C. Seacord, David Svoboda, Kazuya Togashi
“Secure Design Patterns” TECHNICAL REPORT CMU/SEI-2009


[43] http://hillside.net/component/content/article/50-patterns/222-design-pattern-definition

(1979)


DARRELL M. KIENZLE, MATTHEW C. ELDER, DARPA Contract # F30602-01-C-0164
Effective Date of Contract: June 7, 2001, Contract Expiration Date: May 30, 2002

[52] Ontology-Based Access Control Model for Semantic Web Services

[53] A Metamodeling Approach to Pattern Specification and Detection; Maged Elaasar, Lionel C. Briand, and Yvan Labiche; Carleton University, TR SCE-06-08, March 2006

[54] RePa Requirements Pattern Template; Lawrence Chung, Barbara Paech, Liping Zhao, Lin Liu, and Sam Supakkul