FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INSTITUTE OF INFORMATICS
BACHELOR OF COMPUTER SCIENCE

ANDRÉ RODRIGUES OLIVERA

Taim: A Safety Pattern Repository

Graduation Thesis.

Prof. Dr. Taisy da Silva Weber
Advisor

Prof. Dr. Elisa Yumi Nakagawa
Co advisor

M.Sc. Pablo Oliveira Antonino de Assis
Co advisor

Porto Alegre, December 2012

FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
Rector: Prof. Carlos Alexandre Netto
Vice-Rector: Prof. Rui Vicente Oppermann
Dean's office Coordinator: Profa. Valquiria Link Bassani
Institute of Informatics Director: Prof. Luís da Cunha Lamb
Computer Science Coordinator: Prof. Raul Fernando Weber
Librarian of the Institute of Informatics: Beatriz Regina Bastos Haro

ACKNOWLEDGMENTS

I would like to thank Pablo Oliveira Antonino de Assis and Elisa Yumi Nakagawa for all the support and devoted time for this work, first at Fraunhofer and, after that, with our countless remotely meetings through Skype during this year; I am also thankful to them for introduce me the passion about software architecture and design patterns.

I wish to thank my friend and advisor Taisy da Silva Weber who not only helped me to write this thesis but also gave me several personal, academic and professional advices during all my graduation.

My respect to all the employees and professors of Informatics Institute who helped to keep this university among the best in Brazil. I would start to cite name of excellent professors who I have the gratification to be one of their students, but the list would be too large (and probably I would forgot some important name).

Thanks to all the people involved with the international program within Kaiserslautern University, a special thanks to Arthur Harutyunyan that was a great friend for all the Brazilians in Kaiserslautern.

I wish to thank my friends from Santa Vitória do Palmar who despite the distance continued keeping in touch with me during these years and also my friends from Porto Alegre who provided me awesome moments in this city. A special acknowledge to the people who were with me in Germany for this awesome year and for the help with my thesis.

Finally, special thanks to my entire family that always prized for a good education and have trusted and supported me for many years that I lived far away from home and to my girlfriend to be with me all these not so easy years.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
CASE	Computer-Aided Software Engineering
DGML	Directed Graph Markup Language
DLL	Dynamic-link library
EA	Sparx Enterprise Architecture
ERD	Entity-Relationship Diagram
GoF	Gang of Four
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
IESE	Institute for Experimental Software Engineering
IIS	Internet Information Services
JSON	JavaScript Object Notation
LAN	Local Area Network
MEX	Metadata Exchange
ORD	Object-Relational Database
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPR	Safety Pattern Repository
SQL	Structured Query Language
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WSDL	Web Services Description Language
XML	Extensible Markup Language

XMI	XML Metadata Interchange

LIST OF FIGURES

ABSTRACT

Various important application domains, such as medical, avionic, and automotive, requires software based systems prepared for handling potential malfunctions, since these defects could cause severe harms to the environment as well as to humans. During the development of such systems, several techniques and tactics should be taken into account in order to avoid, remove and tolerate failures. Design patterns are proven solutions for recurring design problems. They represent experience and knowledge encapsulating best practices and techniques that can be shared among software developers. In the case of safety-critical systems, these patterns are called safety patterns.

This work presents the design and implementation of a centralized repository that stores safety patterns with the purpose of increase the use and sharing of patterns during the development of safety-critical systems. This work also describes the development of an extension for the UML CASE tool Sparx Enterprise Architect that accesses the repository through the network and allows the user to visualize, manage and store new patterns into the repository. The integration between the repository and a UML CASE tool permits a system architect to use this tool to create the pattern UML structure that will be stored in the repository. Another advantage of this approach is the possibility of reuse safety patterns from the repository directly into diagrams being modeled by the tool. Furthermore, the repository was created using open standards and web services in a way that third-party tools can access it independently from platform.

The resulting artifacts from the analysis and development, such as use cases, data model and class diagrams that shows the architecture of the developed system in different levels of abstraction, are used to describe the application. In addition to this documentation, some implementation concerns, such as details from web services and used protocols as well as concerns about Add-In development in Sparx Enterprise Architect are reported.

Finally, the solution was deployed in an academic environment and the tool was used to verify if the repository achieves its functional requirements. Some screens shots are depicted to show the system in use.

Keywords: Safety-critical system, safety pattern, design pattern, repository.

Taim: Um repositório de Padrões de Segurança

RESUMO

Diversos domínios de aplicações importantes, como médico, aviônico e automotivo, necessitam de sistemas preparados para lidar com falhas em seu funcionamento, pois caso essas falhas não forem tratadas adequadamente, elas podem causar prejuízos severos ao ambiente bem como às pessoas. Durante o desenvolvimento de tais sistemas, muitas técnicas e táticas devem ser levadas em consideração a fim de evitar, remover e tolerar falhas. Padrões de projetos são soluções efetivas para problemas de projeto recorrentes. Eles representam experiência e conhecimento encapsulando as melhores práticas e técnicas que podem ser compartilhadas entre desenvolvedores de software. No caso de sistemas críticos de segurança, esses padrões são chamados de padrões de segurança.

Este trabalho apresenta o projeto e implementação de um repositório centralizado que armazena padrões de segurança com o propósito de incrementar o uso e compartilhamento de padrões durante o desenvolvimento de sistemas críticos de segurança. Este trabalho descreve também o desenvolvimento de uma extensão para a ferramenta CASE UML Sparx Enterprise Architect que acessa o repositório através da rede e permite ao usuário visualizar, gerenciar e armazenar novos padrões no repositório. A integração entre o repositório e uma ferramenta CASE UML permite um arquiteto de sistemas usar essa ferramenta para criar a estrutura UML do pattern que será armazenado no repositório. Além disso, o repositório foi criado utilizando padrões (standards) abertos e serviços "*web*" de forma que ferramentas de outras companias possam acessar o repositório independentemente de plataforma.

Os artefatos resultantes da análise e desenvolvimento, tais como, casos de uso, modelo de dados e diagramas de classes que mostram a arquitetura do sistema desenvolvido em diferentes níveis de abstração, são utilizados para descrever a aplicação. Além dessa documentação, algumas questões de implementação, tais como detalhes dos serviços web e protocolos usados bem como preocupações relacionadas ao desenvolvimento de extensão para Sparx Enterprise Architect são relatadas.

Por último, a solução foi implantada em um ambiente acadêmico e a ferramenta foi utilizada para verificar se o repositório satisfaz seus requisitos funcionais. Algumas imagens das telas são retratadas para mostrar o sistema em uso.

Palavras-chave: Sistemas críticos de segurança, padrões de segurança, padrões de projeto, repositório.

1 INTRODUCTION

This chapter provides an introduction for this bachelor's thesis, which is the design and implementation of a safety pattern repository called Taim. It allows a system architect to store safety patterns in a database and use them directly into a model. Also the motivation behind the work and the project will be discussed. The objectives will be listed and defined. Finally, the organization of the text will be detailed.

1.1 Motivation

Various important application domains, such as medical, avionic, and automotive, requires software based systems prepared for handling potential malfunctions, since these defects could cause severe harms to the environment as well as to humans. Ideally, such systems, known as *safety-critical systems* (DUNN, 2003), should not fail; however, failures could happen, either by faults caused by programming or design errors, or even some defect at hardware level. Therefore, it is important to provide means with which such systems can recover from a failure, or, when a complete recover is not possible, there must be, at least, mechanisms that will bring the system to a *fail-safe state* (DOUGLASS, 1999).

Design patterns are general solutions for common occurring design problems (GAMMA et al., 1994). They have a structure (context, problem, solution) that makes it easy for an architect to understand and use them. In the field of safety-critical systems, these patterns, called safety patterns, include known fault-tolerant techniques, such as monitoring and redundancy, in order to improve the safety and reliability of such systems (DOUGLASS, 1999).

Thereby it is important to provide means to increase and facilitate the usage of patterns during the development of safety-critical systems. In this context, a tool that helps the system architect to store, share, visualize and apply safety patterns direct into models seems to be very useful.

The repository aims to capture and increase the knowledge about safety patterns get during the system architecture and provide it in an easy way.

1.2 Goals

The main objective of this work is to design and develop a tool to manage a safety pattern repository. Such repository may support safety-critical systems engineers to identify, select, apply and reuse patterns during the construction of such systems.

The repository will be provided as web service based system, enabling the storing, editing and retrieving of patterns from different systems. In addition to the basic and some flexible optional information, the description of the pattern will include a XMI file

with the UML description containing the model of the pattern making possible the manipulation of the pattern structure. With this, third-party UML CASE tools may be able to download this file and import the pattern in their workspace or create the UML structure and upload it in the repository.

Furthermore, an extension for an important UML CASE tool which will use the services provided by the repository will be developed in order to validate the overall solution. This Add-in will provide a user interface to access the services of the repository and also will serve as guidance to find out what are the services that need to be provided by the repository, thus enabling that other tools access and use the services in the same way.

1.3 Results

This work presents as main result the documentation of the implemented solution. This documentation includes: (i) an entity-relationship diagram that was used to generate the database structure; (ii) a conceptual view showing the overall architecture of the solution; (iii) a use case diagram showing the main cases of the tool; (iv) class diagrams to describe some relevant structures and how they were designed; (v) a dependency graph diagram generated from the code showing the architecture of the overall solution.

The implemented system was deployed: The server-side was deployed in this university (UFRGS), using a computer with Microsoft IIS web server to host the WCF services and PostgreSQL to store physically the data. The client-side, the Sparx Enterprise Architect (EA) extension that accesses the repository remotely, was installed in a computer with EA at Fraunhofer IESE, in Germany and accessed the repository from there over the internet.

Finally, I used the system to store a real safety pattern. Screen shots and observations about this experiment were taken and are presented as part of the results.

1.4 Organization

The rest of this work is organized as follows: Chapter 2 presents a study of the relevant concepts and terminologies important to the work, such as safety-critical system development, design patterns and existent design patterns catalogs and repositories. Chapter 3 describes the design and implementation of the repository depicting the architecture of the system, technologies used and implementation issues. In Chapter 4, the system in use is demonstrated through screen shots. Chapter 5 summarizes this work and outlines concerns for future works.

2 LITERATURE REVIEW

This chapter presents an overview of important concepts and terminologies adopted to create the repository as well as similar tools available nowadays.

Since this work is about patterns for safety-critical systems, it is important to introduce what are such systems and why their development must take into account several aspects that are not so important in regular systems. The section 2.1 presents the concepts about safety-critical systems.

Section 2.2 leads into the idea of design patterns. The section shows a resume of how and why these principles become important between the current software engineering techniques. It shows also central aspects of design patterns such as the representations used to describe them and the existent work of design patterns for safety-critical systems.

Finally, following the intent of this work, the section 2.3 exposes the existing catalogs and repositories of patterns. These catalogs can exist in the form of papers and books and also in digital form such as web sites or embedded into UML CASE tools.

2.1 Safety-Critical Systems

Safety can be defined as the "absence of catastrophic consequences on the user(s) and the environment" (AVIZIENIS et al., 2004). Following this concept, *safety-critical system*s are applications in which failures can cause severe harms to humans or to the environment (ARMOUSH, ASHRAF, 2010; DUNN, 2003). Typical applications include fire control, avionics, nuclear power, automotives, and medical systems.

Especially in this kind of systems, the occurrence of an accident or mishap is undesirable. *Mishap* is "an event or series of events resulting in unintentional death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. Including negative environmental impacts from planned events" (DEPARTMENT OF DEFENSE, 2012).

Since it is not possible to build an absolutely safe system, several measures must be taken in order to *mitigate the risk* of a mishap until an *acceptable (tolerable) risk level*. The *mishap risk* is measured as a combination of the likelihood of a mishap and its potential severity (LEVESON, 1995). The *acceptable risk level* is established by the public considering the willingness to tolerate the mishap as long as it occurs infrequently.

Risk mitigation can be achieved in three ways: (i) improving component reliability and quality; (ii) incorporating internal safety and warning devices; (iii) incorporating external safety devices (DUNN, 2003).

The designing of safe systems requires the knowledge of the potential conditions that can lead to a mishap, this real or potential conditions are called *hazards* (DEPARTMENT OF DEFENSE, 2012). *Hazard analyses* are performed to identify and define hazardous conditions in order to their elimination or control (FEDERAL AVIATION ADMINISTRATION, 2000). Hazard is the key element connecting a failure in the basic system to a subsequent mishap. After that, a *failure-mode analysis* is performed to discover all possible failure sources in each component of the system in order to determine how these components can fail and cause a mishap (DUNN, 2003).

All these terminologies and concepts outline a high level overview of the issues that must be considered when developing safe systems. However, safety practitioners also shall to understand in deep the various threats that may affect a system; this is known as the *fault-error-failure model*. In other words, it is necessary to comprehend how and the reasons why the *faults* are introduced and activated producing *errors* that can be propagated between components until cause service *failures* in the system enabling the "*chain of threats*". And especially to know what are the means to their achievement (*fault avoidance or prevention*, *fault tolerance*, *fault removal*, *fault forecasting*) (AVIZIENIS et al., 2004).

Three concepts are especially important for this work because they are used in the design pattern representation of the Taim repository: (i) *Fault Class*, (ii) *Service Failure Modes*, and (iii) *Safety Tactics*. The sub-sections below describe them.

2.1.1 Fault Classes

Faults are inevitable. They can be introduced in the system during the specification, design or implementation; faulty components; or by external factors such as radiation or electromagnetic interference, for example.

Azivienis (AVIZIENIS et al., 2004) categorizes the faults into 31 classes, some example includes: development faults vs. operational faults; internal faults vs. external faults; hardware faults vs. software faults; permanent faults vs. transient faults, etc. The combined fault classes belong to three major partially overlapping groupings: (i) *development faults* (all fault classes occurring during development); (ii) *physical faults* (faults that affect hardware) and; (iii) *interaction faults* (all external faults). Internal faults starts in a *dormant* state and become *active* due to some input to a component (the *activation pattern*) causing an *error*.

Knowledge of all possible fault classes allows the safety engineer to decide which classes should be included in the safety specification. This is useful to know what fault classes one needs to address when designing the system in order to choose the correct techniques.

2.1.2 Service Failure Modes

A *service failure* or just *failure* happens when the delivered service does not implement the system function, either due to fails in the functional specification achievement or due to faulty specifications. *Service failure modes* are the different ways that the deviation is manifested. Each mode can have more than one service failure severity.

The service failure modes characterize incorrect service according to four viewpoints: (i) failure domain (content and/or timing failures), (ii) detectability (signaled or non-signaled failures), (iii) consistency (consistent or inconsistent failures),

and, (iv) consequences (ranging from minor to catastrophic failures) (AVIZIENIS et al., 2004).

Several classifications of failure types exist in literature. In (WU; KELLY, 2004) they have adopted Pumfrey's failure classification (FENELON; MCDERMID; NICOLSON; PUMFREY, 1994) that classifies the failures in three categories: (i) *Service provision* (omission or commission); *Service timing* (early or late), and; (iii) *Service value* (coarse incorrect or subtle incorrect).

In the same manner as fault classes, service failure modes specification provides insights into the design of safety-critical systems.

2.1.3 Safety Tactics

"An *architectural tactic* is a means of satisfying a quality-attribute-response measure by manipulating some aspect of a quality attribute model through architectural design decisions." (BACHMAN; BASS; KLEIN, 2003) Tactics show how the quality attributes can be addressed through architectural design decisions, based on well-known patterns or reasoning frameworks.

The principle of architectural tactics is to identify and codify the underlying primitives of patterns in order to solve the problem of the intractable number of patterns existing. In this context, s*afety tactics* extends the notion of architectural tactics to include safety as a consideration trying to fill the lack of guidance in how to develop a basic safety strategy in software architecture design. (WU; KELLY, 2004)

The tactics are based on existing software safety architectural design techniques used in both research and practice. They are organized into three sets: tactics for *failure avoidance*, tactics for *failure detection*, and tactics for *failure containment*; and documented as a structured template with the fields: Aim, Description, Rationale, Applicability, Consequences, Side Effects, Practical Strategies, Patterns (patterns that implement the tactic) and Related Tactics.

It is considerable to differentiate techniques from tactics; usually some technique may implement multiple tactics of a quality attribute and also a technique may enclose mechanisms that are not related to quality attributes.

Since one of the characteristics of architectural tactics is that they can be combined into patterns, the design pattern repository present in this work uses the safety tactics implemented by the patterns to their description.

The next section presents the concepts related with design patterns.

2.2 Design Patterns

In a common sense, design patterns are generalized solutions to commonly occurring problems. They were firstly introduced by the architect *Christopher Alexander* in his book called "*A Pattern Language: Towns, Buildings, Construction*". He says, "Each pattern describes a problem which occurs over and over again in our environment, and then 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" (ALEXANDER; ISHIKAWA; SILVERSTEIN, 1977). He was talking about buildings but the definition fits nicely for software as well.

Thus, a pattern expresses a relation between three parts: (i) the *context*, that is the (recurring) situation in which the pattern applies; (ii) the *problem* refers to the constraints that occur and the goal you are trying to achieve in this context, and; (iii) the *solution*, a general design that anyone can apply which resolves the goal and set of constraints.

Based on Alexander's book, many other domains have been using the idea of design patterns. In software domain, the book "*Design Patterns: Elements of Reusable Object-Oriented Software*" written by *Gamma et. al. (*group known as GoF) is considered the most popular work about this topic. The design patterns in this book are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context". The book describes, through a structured template, 23 design patterns classified by two criteria: purpose (creational, structural, or behavioral) and scope (classes or objects) (GAMMA et al., 1994).

There are many reasons to use design patterns in software development. In summary, design patterns make it easier to reuse successful designs and architectures that have worked in the past to solve some class of problem. They represent experience and knowledge encapsulating best practices and techniques that can be shared among software developers. Moreover, the pattern names provide a common vocabulary that allows designers to communicate more effectively and provide a good way for documenting proven design techniques.

The design patterns described in GoF book were catalogued to be used in no specific application domain. However many other researches has focused in design patterns for a specific domain, such as, Service Oriented Architecture (ERL, 2009) or for specific programming languages, such as, Java (BIEN, 2012).

2.2.1 Safety Patterns

Safety patterns are design patterns that include known techniques for fault tolerance, such as channel monitoring and redundancy, in order to improve the safety and reliability of safety-critical systems.

The most popular work about safety patterns is in (DOUGLASS, 1999, 2002). It describes several patterns for real-time development, including patterns that deal with safe and reliable architectures based on well known fault-tolerant design methods. He defines a design pattern as means to optimize some aspect of the system in some way while deoptimize it in some other way. Thus, one important aspect of these patterns (that is explicit in the pattern representation) is the consequences of applying the pattern, thus, the architect must understand the negative as well as the positive aspects, in other worlds he must deal with the trade-offs of the pattern application.

2.2.2 Pattern Representation

An important aspect of design patterns is how they are represented or, in other words, which template is used to describe the pattern. There is no agreement between representation forms on a single template. Some authors use precise and short structure and others more comprehensive and expressive. In some cases, the author uses different field names to represent the same aspect of the design pattern. However, they often use the basic structure suggested in GoF's book with four important aspects: *name, purpose, solution* and *consequences* (ARMOUSH, ASHRAF, 2010).

The pattern representation is intended to make design patterns easier to learn, compare, and use. Below three well known examples of design pattern representations are described.

1 - The GoF Patterns uses the following template:

- **Pattern Name and Classification;**
- **Intent**: What does the pattern do? What particular design issue or problem the pattern addresses?
- **Also Known As**: Other well-known names for pattern;
- **Motivation**: A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem;
- **Applicability**: The situations in which the design pattern can be applied;
- **Structure**: Graphical representation of the pattern, usually using UML;
- **Participants**: The classes and/or objects and their responsibilities;
- **Collaborations**: How the participants collaborate to carry out their responsibilities;
- **Consequences**: The trade-offs and results of using the pattern;
- **Implementation**: The pitfalls, hints, or techniques that one should be aware of when implementing the pattern;
- **Sample Code;**
- **Known Uses**: Examples of the pattern found in real systems;
- **Related Patterns**.

2 - Martin Fowler in (FOWLER, 2002) proposes the following fields to represent the design pattern (obs.: not all sections appear in all patterns):

- **Pattern Name**;
- **Intent and Sketch**: The intent sums up the pattern in a sentence or two; the sketch is a visual representation of the pattern, often but not always a UML diagram;
- **How It Works**: Describes the solution as independent as possible of any particular platform;
- **When to Use It**: Discussion of the trade-offs that make you select this solution compared to others;
- **Further Reading**: References that point to other discussions of this pattern;
- **Examples**: Simples examples of the pattern in use with some code in Java or C#.

3 - Bruce Douglass in (DOUGLASS, 2002) write the patterns with these fields:

- **Abstract**: Brief description of the pattern use or justification. An overview of the problem, solution, and consequences;
- **Problem**: Statement of the problem context and the qualities of service addressed by the pattern;

- **Pattern Structure**: Structural UML diagram, showing the important elements of the pattern;
- **Collaboration Roles**: Properties of the individual elements in the pattern collaboration;
- **Consequences**: Tradeoffs made when the pattern is used;
- **Implementation Strategies**: Issues around implementation of the pattern on different computing platforms or in different source-level languages;
- **Related Patterns**;
- **Example Model**: Example that illustrates how the pattern is applied in some particular case.

2.2.3 Safety Pattern Representation

Many researches target in finding a design pattern representation that fits better with classes of problem from a specific domain. *Armoush* in (ARMOUSH, A.; SALEWSKI; KOWALEWSKI, 2008) argues that the conventional design pattern structure lacks a consideration of potential consequences and side effects on non-functional requirements (more precisely safety and reliability) and proposes a new pattern representation for safety-critical applications that includes fields for the implications and side effects of the represented design pattern on the non-functional requirements of the overall systems.

Antonino, in (ANTONINO et al., 2012), presents an approach for extending the safety patterns representation in a way that allows them to be properly modeled and also offers means to support their application in architectural models. To this, it proposes the joint use of a UML profile and rules as descriptive structures stating safety patterns application constraints. The graphical representation of these safety patterns with stereotypes instances is an appropriate front-end that allows engineers to reason on the safety pattern structure (roles and connections) in terms of abstract functional entities. Moreover, it provides means to state safety specificities (*fault classes*, *services failure modes* and *safety tactics*) that are singular to each safety pattern.

The Antonino's pattern representation is used as basis to the development of the Taim's safety pattern repository; in different words, the repository aims to store safety patterns as described in this paper. Thus, the representation includes the basic fields proposed by Douglass and also the extension ideas described by Antonino.

2.3 Related Work

The application of a pattern in system architecture, called *Pattern Hatching* in (VLISSIDES, 1998), requires a good familiarity with a specific pattern catalog in order to choose the pattern which best fits to achieve the current problem/issue that is being addressed.

As stated in section 2.1, several kinds of patterns were and continue being published, either focusing in general or specific domain areas, such as enterprise software, business system or real-time systems; focusing in different level of abstraction or to be used in different development phases. All these discovered patterns are catalogued in books and papers following a specific pattern representation discussed in previous section. Thus,

the traditional way to study design patterns is to read and get familiarized with these books and papers.

In addition to this, the internet becomes an important way to disseminate the design pattern knowledge. Many web sites are focused in cataloguing patterns, much of these web sites are based on some book or they are even maintained by the author of the book.

Ward Cunningham together with other important people from the software engineering area, such as *Kent Beck* and *Grady Booch* created the *Hillside Group* ("The Hillside Group - A group dedicated to design patterns. Home of the patterns library") which provides a central resource for patterns work. The site provides information on many patterns-related resources such as articles, books, mailing lists, tools and links pointing to other important resources. One of them is the well known *Portland Pattern Repository* ("Portland Pattern Repository") created by *Cunningham*. This repository is devoted to all things related to patterns. It presents many patterns and good practices in software development. The inclusion of patterns in the site is made sending an e-mail to the author with a document that must follow some stipulated conventions. Some other web sites examples are described below:

In ("Patterns · intuio") Zahler, based in his Ph.D. thesis, lists many pattern intended for touch-screen systems for safety-related environments. The company Yahoo presents many web user interface patterns in ("Yahoo! Design Pattern Library"), the catalog is improved through user feedback. User Interface patterns are present also in ("Design Patterns Library | Fellowship Technologies", "UI-Patterns.com"). The Open Security Architecture Company ("11.02 Security Architecture Patterns") maintains a list with several security-related patterns; the user can contribute sending patterns by e-mail. In (".NET Design Patterns in C# and VB.NET - Gang of Four (GOF) - DoFactory") it is possible to find a catalog with the GoF Patterns "optimized" to be used in .Net Platform.

Several pattern book authors also maintain web catalogs where is possible to access a brief description of the patterns contained in the book ("Catalog of Patterns of Enterprise Application Architecture"), or read the detailed description of the patterns ("Elements of Parametric Design"), or even repositories which allow the user to add patterns to the existent catalog ("Net Objectives Design Patterns Repository", "Wikipatterns - Wiki Patterns").

Either in physical or online catalogs, the patterns are represented in a textual way, in other words, they do not provide mechanisms to apply the pattern directly into the model that is being designed, making the pattern applying process harder. Furthermore, they do not provide means (or it is hard to understand and use) to include or to modify patterns from the catalogue.

An open-source project called *Open Pattern Repository* started by Uwe Van Heesch in 2009 ("openpatternrepository - Publicly available Online Pattern Repository - Google Project Hosting") aims the creation of a patterns repository that allows the user the pattern managing, browsing (according to categories) and searching following some quality criteria.

However these online repositories lack when it is necessary to direct apply the pattern into the model.

In addition to these catalogues and repositories, we can find design patterns also embedded in UML CASE tools. Some well known tools like *Sparx Enterprise Architect*

("Enterprise Architect - UML Design Tools and UML CASE tools for software development"), *IBM's Rational Software Architect* ("IBM Software - Rational Software Architect Family") and the open-source tool *Star UML* ("StarUML - The Open Source UML/MDA Platform") have a built-in catalog or an extension that enables the user to apply design patterns direct into the diagrams being modeled, usually this tools also provide instrumentation to create or extend a design pattern library. Nevertheless the design pattern collaborative sharing of these tools are not intuitive, besides, very often these tools doesn't not work well with data exchange between other companies tools, in other words, they are not interoperable.

Aside from *Douglass* book (DOUGLASS, 2002), safety patterns catalogs are not so frequent. In addition to the pattern representation proposed by Armoush, mentioned in the previously section; in (ARMOUSH, ASHRAF, 2010) he also presented a tool to serve as pattern repository for the patterns presented in his thesis, the tool offers safety and reliability assessment, firstly proposed in (ARMOUSH, A.; BECKSCHULZE; KOWALEWSKI, 2009), along with a decision support mechanism that help the user to find a suitable pattern given the non-functional requirements needed.

In this context, the repository presented in this work aims to manage the safety patterns as presented in (ANTONINO et al., 2012), discussed earlier; and, at same time, it tries to overcome the limitations of the existent repositories stated in this section using the following features:

- Allowing the catalog content management;
- Providing an interoperable interface to manage the repository, and;
- Permitting the direct application of the pattern into the system design.

2.4 Summary

This chapter presented the fundamental terminologies and concepts related to this work as well as the important research existent in the area. At first, it was discussed about the important aspects of safety-critical systems explaining some central attributes that are used in the pattern representation of the Taim repository.

After, the fundamental concepts of design patterns were presented, together with the safety pattern representation upon which this work is based. And finally, a study of existent design pattern catalogs and related tools was introduced. With this study, it was possible to state many limitations of currently available catalogs and repositories, outlining a set of desired characteristics to be addressed by the repository being developed.

The next chapter describes the design and implementation issues of the Taim repository.

3 DESIGN AND IMPLEMENTATION

This chapter details the development of the safety pattern repository. The section 3.1 outlines some important aspects of the methodology used; section 3.2 presents the resulting artifacts from the analysis and design of the system; section 3.3 describes some important implementation issues and technologies that were applied during the development; section 3.4 discusses the obstacles and limitations faced during the process, and; the section 3.5 concludes this chapter.

3.1 Methodology

This work is part of a broader project, more specifically, a Ph.D. project being conducted at *Kaiserslautern University of Technology* together with *Fraunhofer Institute for Experimental Software Engineering* (IESE), in Germany. Since the people from this project are the direct and indirect users of the repository, they acted as system stakeholders for the work. The main purpose of this work is to develop a safety pattern repository. The overall process can be summarized as follows.

First an extensive study was performed in order to understand the fundamental concepts and the existing related works in the field. The result of this study was summarized in chapter 2.

After, within the software was firstly sketched and discussed, the needed technologies available to implement the system were studied and the solution was designed and implemented. All these activities were developed in parallel, always with feedback from the stakeholders. One of these stakeholders, the owner of the project, took the final decisions about the development.

I designed and implemented the software concurrently with the holding of weekly meetings of approximately 30 minutes with the stakeholders. In each meeting, many aspects, ideas and technologies for the system were discussed; some diagrams, such as conceptual view, use cases and architecture, were sketched or reviewed; many prototypes or even working versions of the system were presented, reviewed and validated by the stakeholders.

The requirements were discussed, defined and improved during the development of the repository. The final decisions were made by the owner of the project. The models and the implementation of the system were made by me and are described in sections 3.2 and 3.3.

3.2 Analysis and Project

The analysis and project of the system are described below.

Almost all the models used for the documentation are UML diagrams created in *Sparx Enterprise Architect* with exception of the *Directed Graph Markup Language* (DGML) which was created directly from the code with *Visual Studio 2010.*

3.2.1 Requirements

As stated in chapter 2, the safety pattern repository developed in this works tries to achieve the following characteristics:

- Allowing the catalog content management;
- Providing an interoperable interface to manage the repository, and;
- Permitting the direct application of the pattern into a model.

The first characteristic describes what is expected from any repository. In opposite to simple catalogs, a pattern repository must provide to the user a way to create, read, update and delete patterns. The second characteristic says that the repository needs to be available through different platforms and systems. And finally, the repository must provide means to use the patterns directly into the models that are being designed by the system architect.

Together with these characteristics, other aspects of the system were defined:

- Each pattern will have many pattern versions and each pattern version belongs to exactly one pattern. The pattern has as attribute only a name and a description and the pattern is used to group pattern versions. Each pattern version represents the same pattern, but is slightly different from the others pattern versions that also belongs to the same "parent" pattern. This difference can arise from a part or a connection of the pattern that exists in one version and doesn't exist in other, for example;
- The pattern version represents completely the safety pattern and has all the necessary information to represent the pattern (see section 2.2 for more information about pattern representation);
- One pattern can be related with other pattern. This means that all the versions of some pattern are also related with all the versions of the related pattern;
- The pattern is composed from many parts. These parts represent the roles of the pattern, for example, actuator, data transformation, input processing, etc. Each version of the pattern can use or not any part of the pattern. In other words, the pattern owns the parts of all its versions and the part is identified by its name;
- The pattern version should be flexible within its information. Eventually the system architect wants to insert additional information that is not present in the representation into the pattern;
- The pattern version should keep an UML structure representing the pattern in order to the user to be able to apply this structure directly into some model being developed;
- The same repository should be accessed from different computers in the network (intranet or internet). This enables the pattern exchange (increasing reusability) by different people involved in the project or in other projects. Furthermore, this means the pattern catalog is build in a collaborative way.

3.2.2 Conceptual Model

The basic entities and their relationships are depicted at high level in Figure 3.1: Conceptual View.

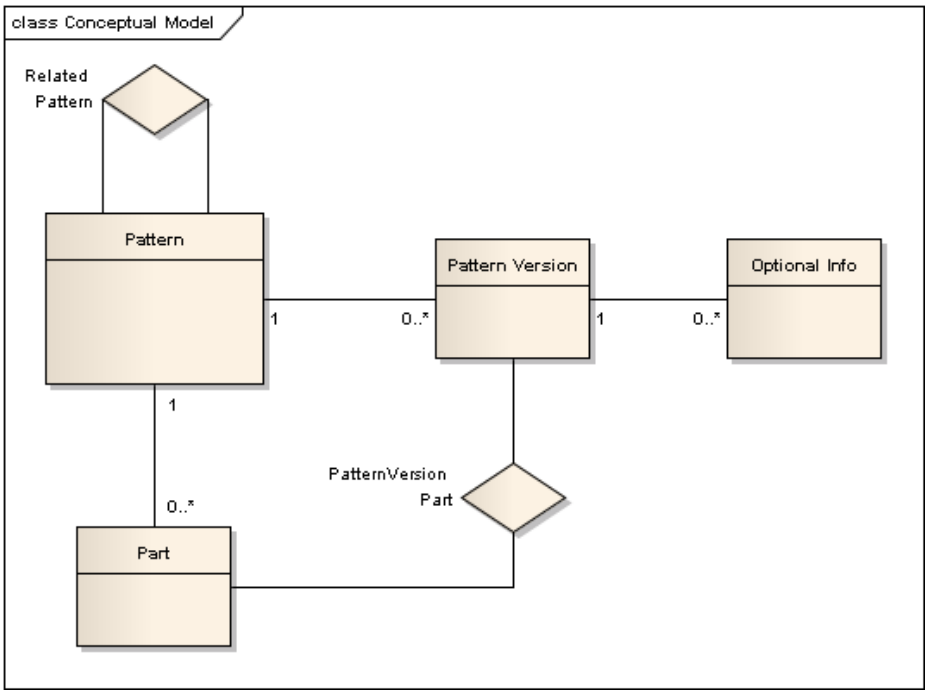

Figure 3.1: Conceptual View

This model helps to understand how the entities are interrelated and serves as basis for the construction of both the database structure and the fundamental classes of the system.

Each entity represents the concepts that were addressed earlier. Some relationships have a direct connection with the definitions made before, during the requirements elicitation. For example: a *Pattern* owns many *Pattern Versions*; the *Pattern* can have several *Parts*, each one can be within some *Pattern Version* through a *PatternVersion Part*; a *Pattern* can be related with another Pattern through *Related Pattern*, and; the *Pattern Version* can have several *Optional Infos*. The last one tries to achieve the requirement that says "*the pattern version should be flexible within its information*". Thus, each of this flexible information can be inserted together with the pattern as an Optional Information.

With this in mind, the data structure showed in Figure 3.2: Data Model was created.

To create this diagram, it was used a built-in UML Profile that maps classes to tables, attributes to columns and association to relationships in a way that is possible to generate data definition code from the model.

In Taim repository, the patterns were stored physically within an object-relational database (ORD). Specifically, the PostgreSQL was used to this end and the script in SQL produced from the data model depicted in "Figure 3.2: *Data Model*" was used to build the data structures in the database.

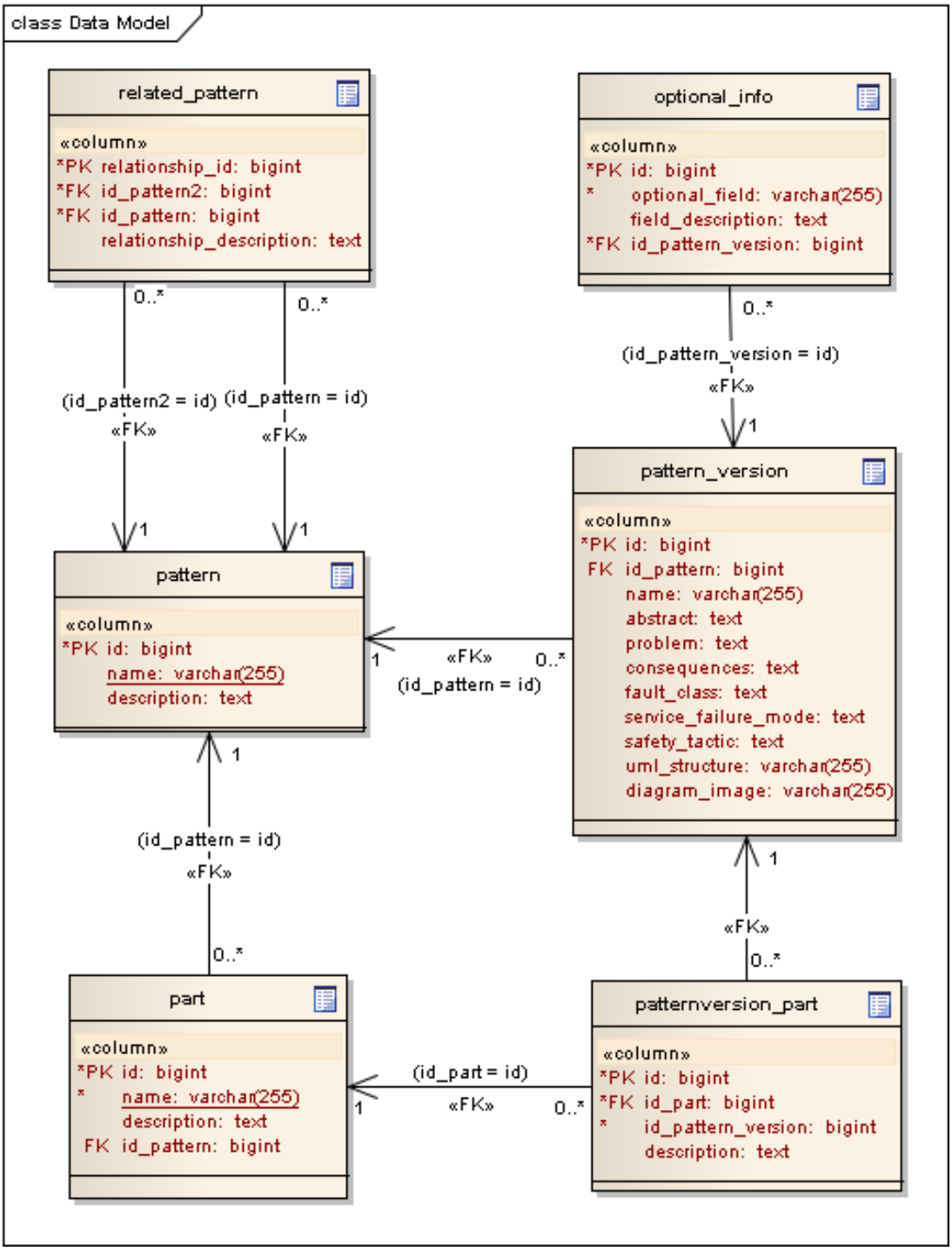

Figure 3.2: Data Model

As said in chapter 2, the pattern template used to represent a safety pattern in the repository is based mainly in Douglass's work (DOUGLASS, 2002) together with Antonino's work (ANTONINO et al., 2012). The columns *name*, *abstract*, *problem*, *consequences* from the table *pattern_version* and the table *related_pattern* map directly into the fields with the same name in Douglass work. The table *parts* and the pattern_version's column *diagram_image* together with *uml_structure* are similar semantically to respectively the *Collaboration Roles* and the *Pattern structure* described

by Douglass. The fields: *fault_class*, *service_failure_mode* and *safety_tactics* were taken from Antonino's research.

Other important observations from the model are:

- The relationship between patterns, given by the table *related_pattern*, contains, in addition to the identification of the two patterns that are related, a description that says how the patterns are related.
- Each part belongs to a pattern and it has a description that explains the part in a common meaning. Additionally, the pattern version part, which is a part that belongs to a specific pattern version, also has a description. This description, however, is specific for the pattern version owner of that part.
- Each pattern version can have many optional information, these information are described by a pair: optional field and field description. This construct gives flexibility for the pattern to keep additional information that is not in its description *a priori*.

3.2.3 Use Cases

The Figure 3.3: Use Cases Diagramshows the primary use cases diagram where the main functionalities of the system are depicted.

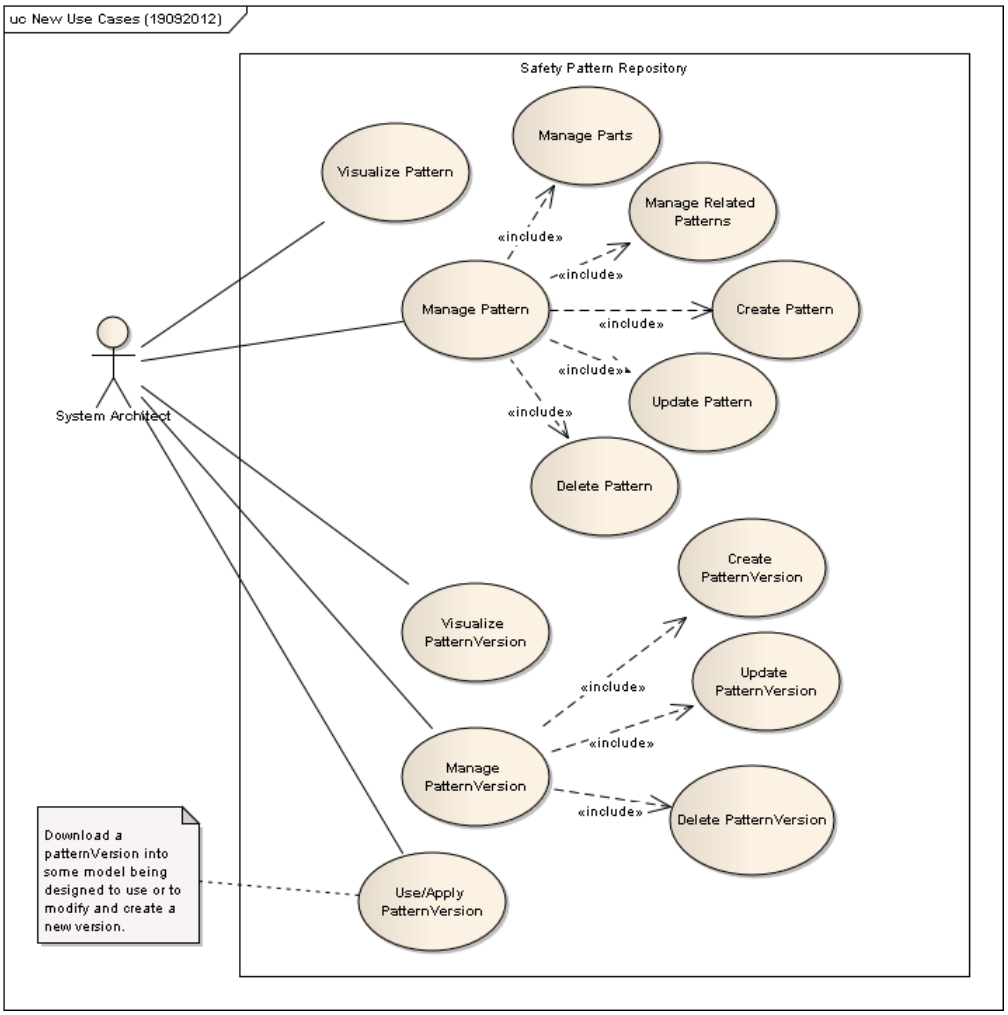

Figure 3.3: Use Cases Diagram

For simplicity purpose, there is only one actor in the diagram that represents the direct user of the system. In summary, the system architect may create new patterns (and pattern versions) to share with other architects; manage them; and he also may see and use the patterns into some model (*Use/Apply PatternVersion*).

3.2.4 System Architecture

Given some concerns that must be addressed by the system such as interoperability, platform independency and network support; the Taim repository was designed as a client-server application. In this context, the main architecture characteristic for the solution design is the provision of a centralized interface through web services using open protocols with the purpose of enabling third-party systems to access and use the repository.

Thereby, in addition to the development of the server-side repository (the services implementation together with data persistence into a physical database), this work also includes the implementation of an application that consumes the web-services provided by the repository and offers a user interface for the repository.

This application was developed independently from the repository. It is an extension (Add-in) for the UML CASE tool *Sparx Enterprise Architect*. Thus, this tool can be used to create the pattern's UML structure in order to save the pattern into the repository as well as to retrieve it from the repository and use it into some model that is being designed in the tool. In addition to this, it encourages the development of similar applications or third-party extensions that access and use the services provided by the repository

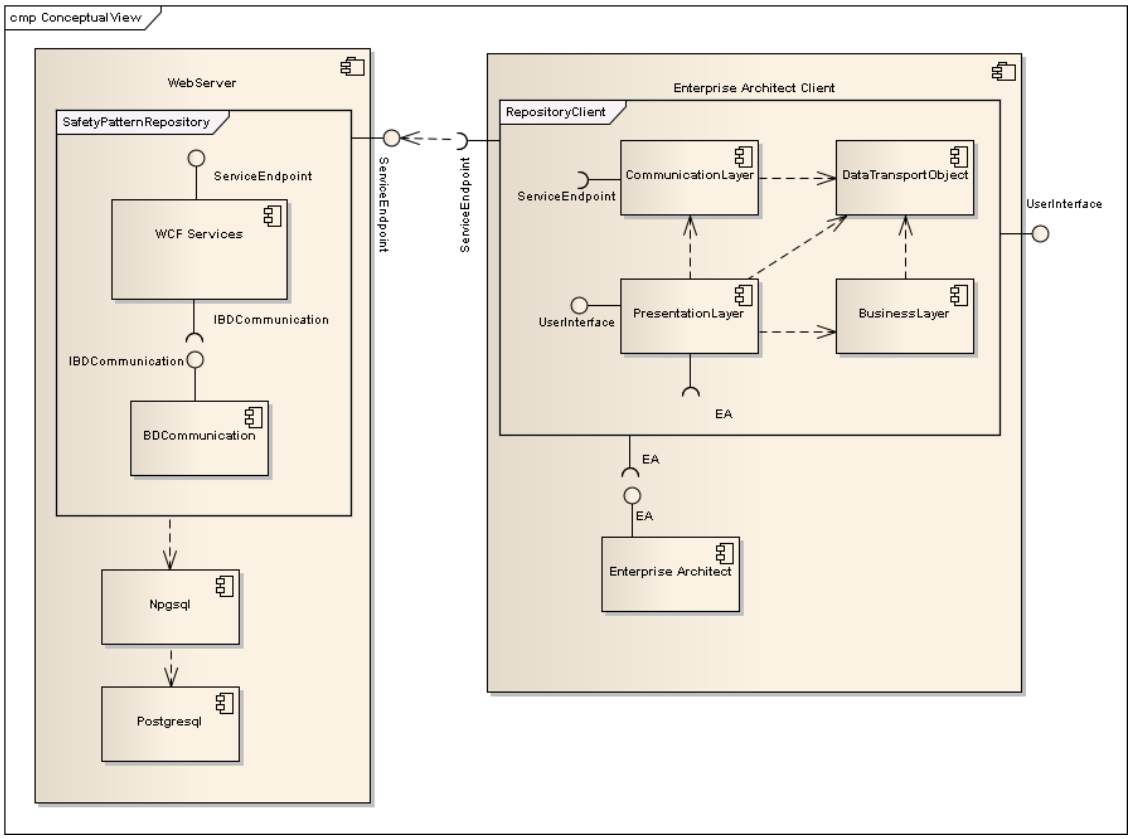

Figure 3.4: Conceptual View

An overall view of the whole solution developed in this work is presented in Figure 3.4: Conceptual View.

The image “Figure 3.4: *Conceptual View*” shows clearly the two parts of the client-server solution. In the left side is the Web Server which is composed by the Repository that access a PostgreSQL database through NpgSQL component and offers *WCF services* through an interface (the Service Endpoint). On the other side, the EA extension uses these services and presents a user interface through by means of EA.

In addition to the elements *SafetyPatternRepository* and *RepositoryClient* which were actually developed in this work, the diagram shows external components that collaborate in some way to the overall solution. In the server-side, the element *PostgreSQL* represents the database used to store physically the data and the element *NpgSQL* is a .Net data provider for PostgreSQL, it allows .Net application to access a PostgreSQL database. And, in the client-side, *Enterprise Architect* component provides an API to Add-In development making possible the integration between the UML tool EA and the repository.

The diagram in Figure 3.5: Server-side repository presents in a more detailed level the architecture of the server-side.

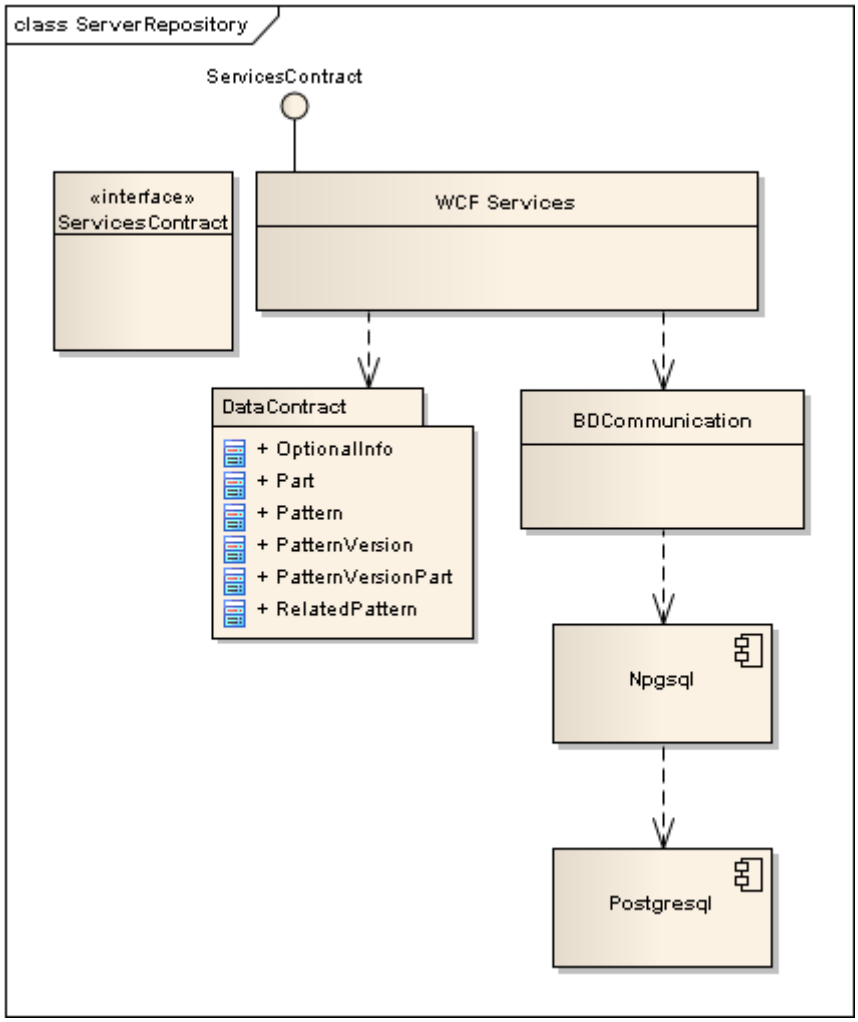

Figure 3.5: Server-side repository

All the communication between the services implementation (element *WCF Services*) and the database is managed by *BDCommunication*, which uses the *NpgSQL* component to connect and access the PostgreSQL database.

The *ServicesContract* element is the interface that specifies what services are provided by the repository, these services are implemented by the element *WCF Services*. The *DataContract* package contains the classes that represent the model. Together with the ServicesContract, the DataContract is used to create the *Web Services Description Language* (WSDL) file, which is a *W3C* standard to describe services. The Data Contract UML diagram is depicted in Figure 3.6: Data Contract.

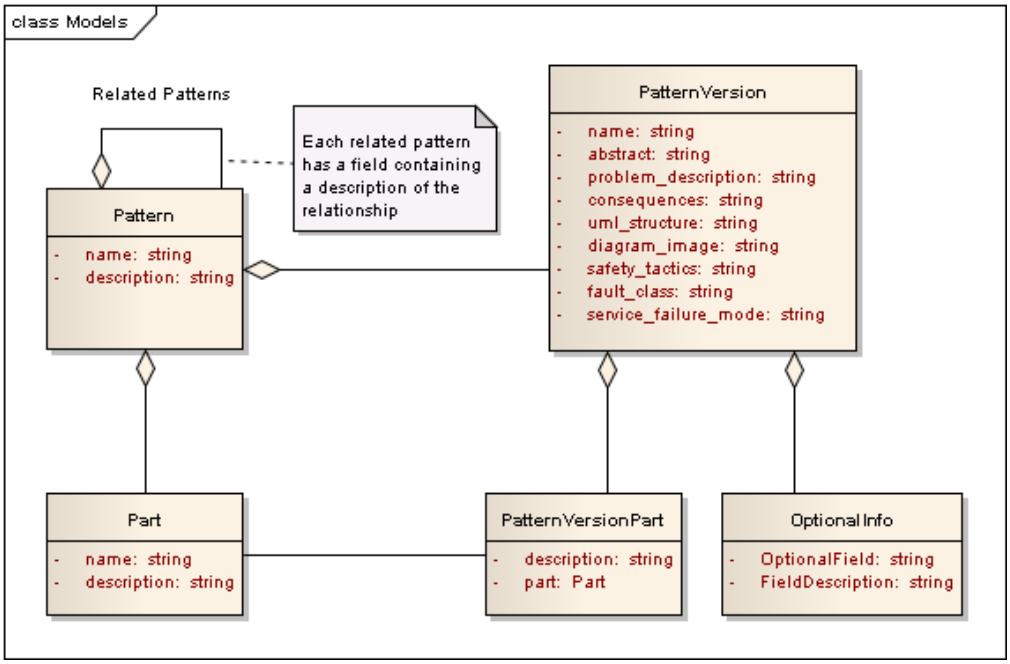

Figure 3.6: Data Contract

As expected, the data contract is similar with the database structure described previously.

The client-side application was designed using a multi-layer approach as depicted in Figure 3.7: Multi-Layer Client Architecture.

EAAddIn is the starting point for the application, it has the main function which is invoked by Enterprise Architect (EA) when it is launched and when the user accesses the Extensions (Add-In) Menu. Such function creates the central window *FormRepository* (more details about EA AddIn implementation will be described in the next section).

The *Presentation Layer* contains the *Windows Forms* that are displayed to the user. In a general sense, each form is responsible to present to the user the actual state of some resource, such as pattern, or PatternVersion. For Example, *FormPattern* is used when it is necessary to visualize an existent pattern, create a new one (the form will be created in blank) or edit an existent one. The form receives an object *Pattern*, shows it and permit (or not) its edition.

To retrieve an object from, or to store it in the repository (actually, this means send a message with the object to the remote repository through the network) the forms use the

services provided by the *Business Logic Layer*. This layer is responsible for the client-side validation of the object before sending him through the network (by the Communication Layer); also, it is responsible to ask to the Communication Layer for an arbitrary object that is being required in the Presentation Layer.

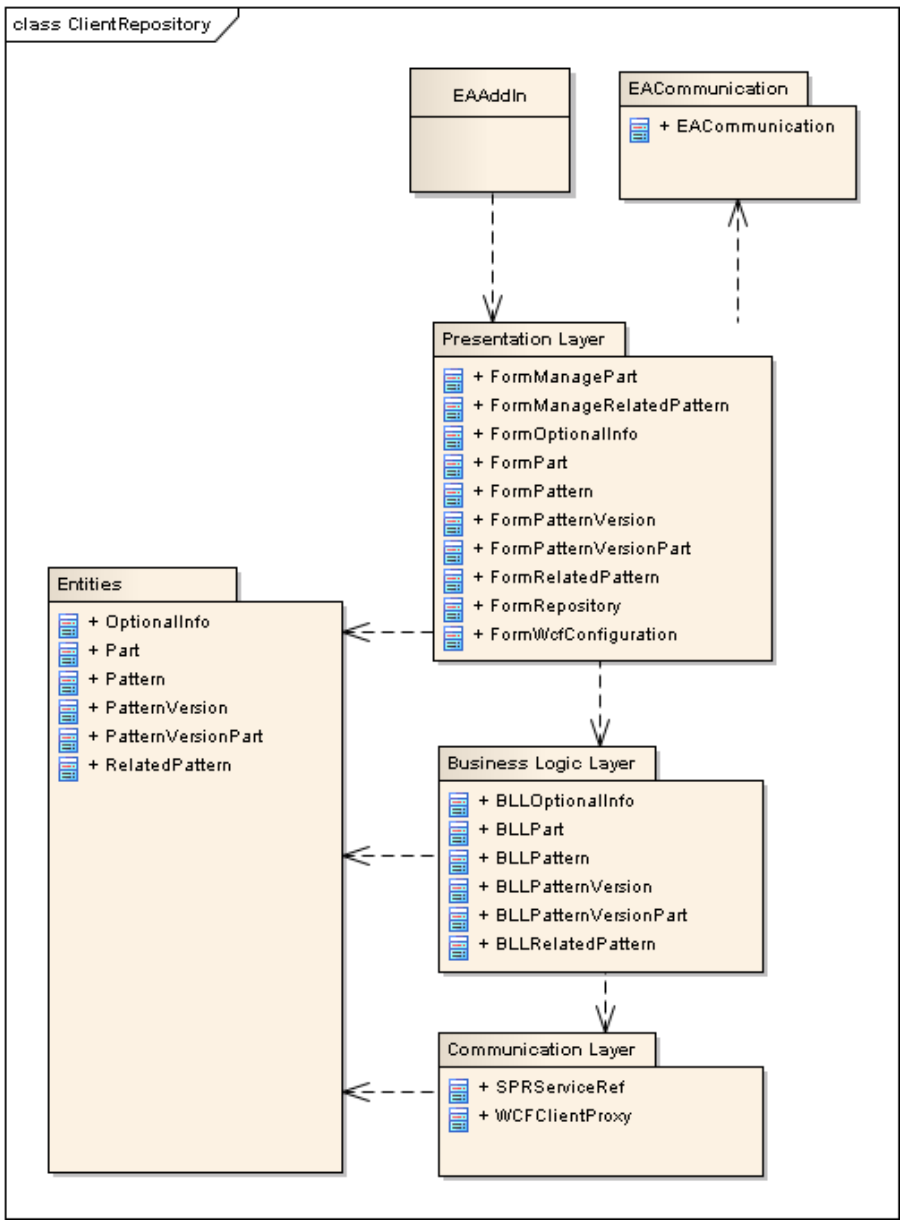

Figure 3.7: Multi-Layer Client Architecture

Closing the three main layers, the *Communication Layer* is responsible for the creation of the channel that communicates with the server through web services. It is also responsible for authentication issues and marshalling/unmarshalling of the data that is being transmitted and received through the communication channel.

In addition to these layers, *EACommunication* is used by the *Presentation Layer* to get access to the *Enterprise Architect API*; it is used, for example, to get information about an UML element from the model being designed in EA or to include a pattern that

is in the repository into the model. The class *SPRServiceRef* is generated automatically in the client by the *.Net Framework* tool *SvcUtil* from the WSDL file retrieved from the web server.

The classes in *Entities* package are referenced in all layers. It contains classes representing the domain model. Such classes are used as container to pass data between the layers, thus they doesn't have any special behavior (methods). These classes follow the same structure as the aforementioned *DataContract* in the server-side and the *Conceptual Model.*

The Figure 3.8: Deployment View shows how the artifacts (software components, often DLLs) are physically deployed in the nodes (hardware components).

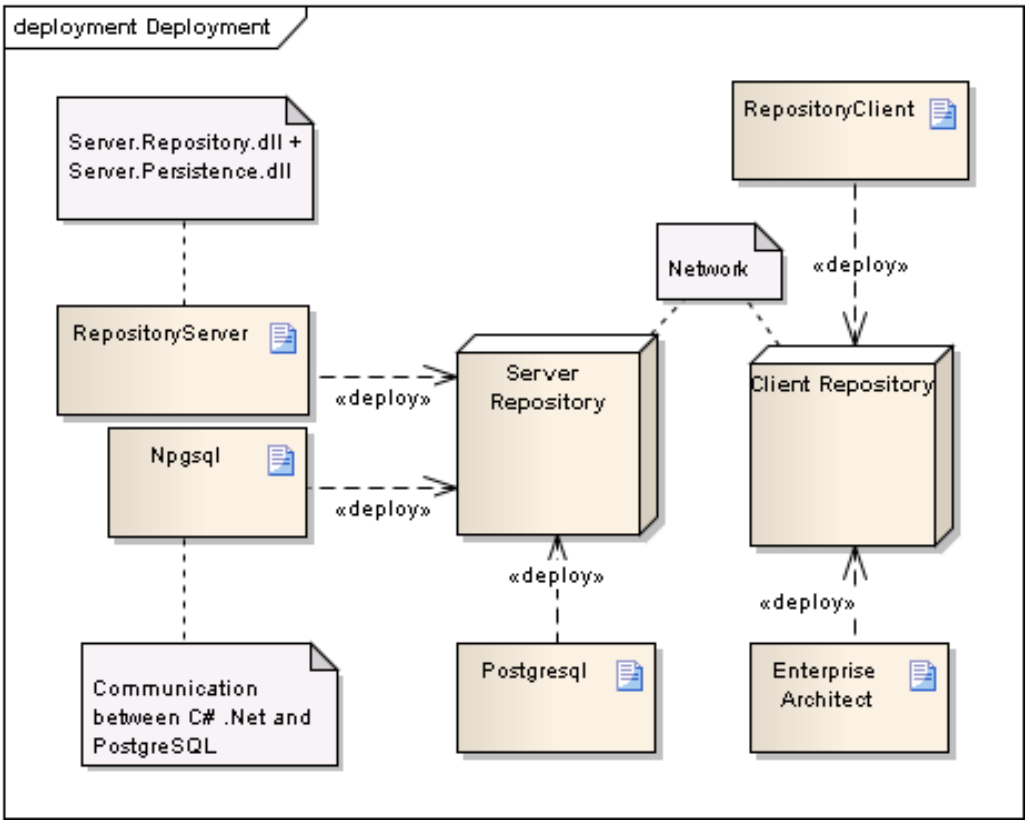

Figure 3.8: Deployment View

These diagrams used to represent the architecture of the system try to reflect what is actually implemented, however the diagrams and the implemented code were created independently, and, thus, their synchronization was did manually.

In order to see the implementation in a higher level and also be able to keep the code the most close possible with the designed architecture, it is interesting to generate models from the code. In this context, it was used *Visual Studio 2010* to generates a *Dependency Graph* in DGML and visualize it.

The "Figure 3.9: *DGML generated from the code*" shows the generated dependency graph displayed in Visual Studio.

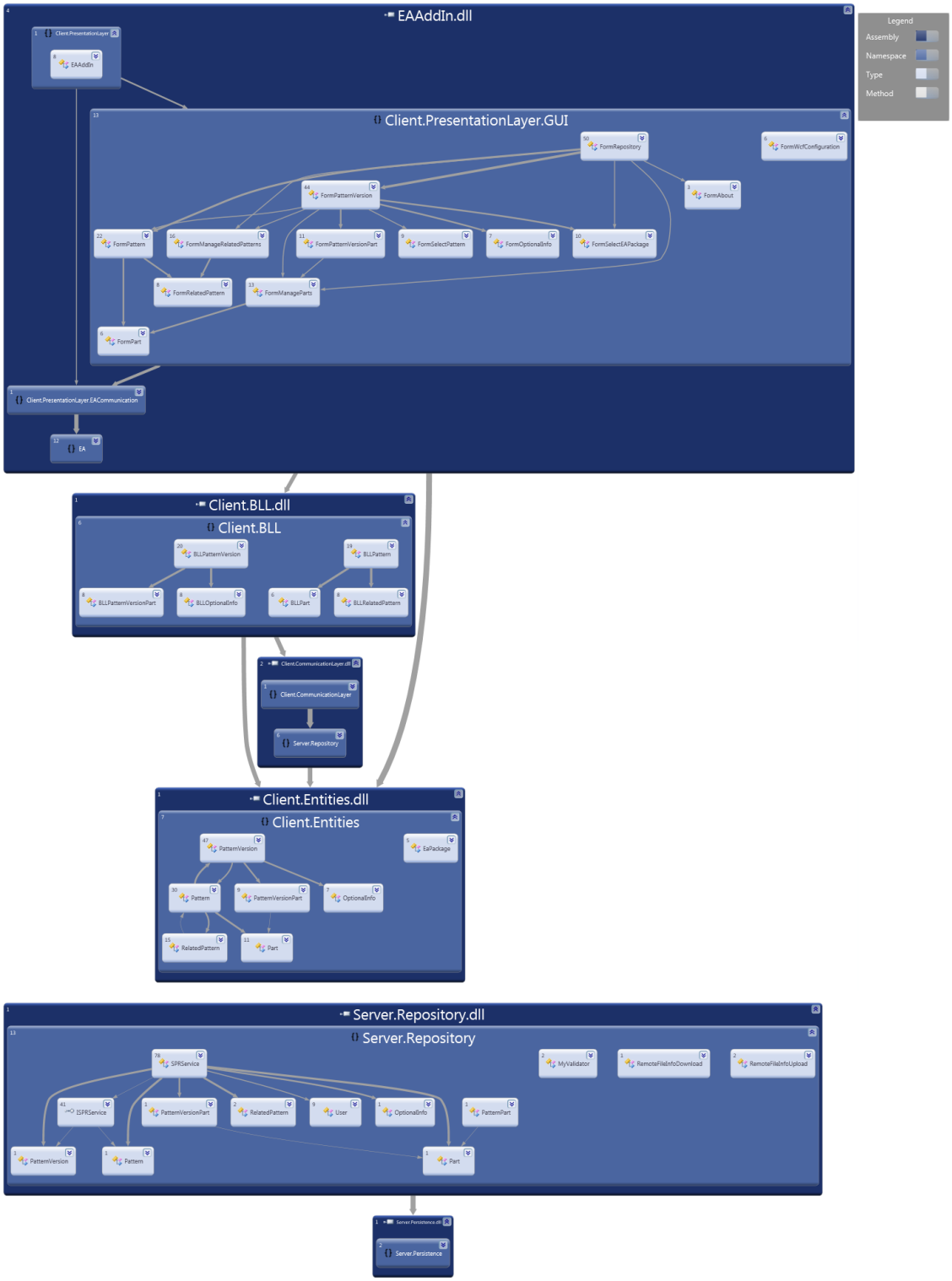

Figure 3.9: DGML generated from the code

The image shows in different level of abstraction (assemblies, namespaces and classes) the dependencies between each component. Each layer was implemented as a separated DLL file in order to provide more independency between the layers.

3.3 Implementation Issues

This section describes some concepts, technologies and protocols that were relevant for the Taim repository development. Some of them were already mentioned when explaining the architecture of the system, in this section they will be described in a more detailed level.

Two topics that are most relevant for this work are the issues about Add-In development in EA and the technologies and protocols used to implement the web services.

All the system was implemented in Microsoft Windows platform using the Microsoft Visual Studio 2010 IDE and .Net Framework 4.0. The programming language used in both the server and the client side was Microsoft C# and to store the data physically the PostgreSQL database was used. The WCF framework was used to create the web services as described in 3.3.2.

3.3.1 Sparx Enterprise Architect AddIn Development

Such as other UML CASE tools, *Sparx Enterprise Architect* (EA) offers means to develop new functionalities in the form of extensions. In summary, the "*Automation Interface*" provides the following options to extend EA:

- The definition of menus and submenus;
- Notifications receipt from events occurred inside EA;
- Access to the opened project, including the packages, elements, diagrams and connectors.

Basically for this work, it was necessary to import and export XMI files containing an UML Package into some UML Package of an opened model on EA and also to browse between the packages of the actual opened project in EA.

The essential steps to create an EA Add-In are described below.

First, it is necessary to create the AddIn as a Class Library project (DLL). The project must reference the file *Interop.EA.dll* that can be found in the EA installation folder. The main class of the AddIn should implement some key functions to communicate with Enterprise Architect, such as *EA_Connect*, *EA_GetMenuItems*, *EA_GetMenuState* and *EA_MenuClick* (the parameters were omitted). With this, it is possible to create a sub-menu in EA that is controlled by the Add-In. As said before, the *Interop.EA.dll* provides also in the namespace "*EA*" all the elements necessary to access the project browser of EA, the class name of the Project Browser is "*EA.Repository*". The API reference can be found in Sparx web site ("Reference [Enterprise Architect User Guide]").

After that, the created assembly, a DLL file, must be registered in the .Net Framework, to accomplish this, it was used the *Assembly Registration Tool* (RegAsm.exe) available with the .Net Framework. This allows the Enterprise Architect to use the AddIn.

Finally, it is required to tell EA what is the main class of the created AddIn. When EA starts up it will read the registry key "*HKEY_CURRENT_USER \ Software \ Sparx Systems \ EAAddins"*. Each key in there represents one AddIn; the (default) value of the

key says the main class. The name includes the namespace (or assembly name) and the name of the class, for example, *Client.PresentationLayer.EAAddIn*. Thereby, it is necessary to add this to the register; the *Registry Tool* (*regedit*) of Windows can be used to accomplish this.

3.3.2 Web Services

According to *World Wide Web Consortium*, "Web services provide a standard mean of interoperating between different software applications, running on a variety of platforms and/or frameworks." ("Web Services Architecture"). Despite both the server and the client implemented in this work were built upon Windows and .Net, the web services implementation using open standards in the server-side allows them to be accessed and used by different clients in different platforms.

The first choice when developing web services is what message protocol will be used. For the Taim repository, it was used the SOAP protocol over HTTP, utilizing XML as message format. Other alternative that could be used is the REST architecture together with JSON for message formatting. The choice for SOAP takes into consideration, between other reasons, the platform that the client pretend to run upon, in my opinion, SOAP is more suitable to desktop applications.

To support the development of the web services, it was used the *Windows Communication Foundation (WCF)* which is a framework for building service-oriented applications. The following tasks are required to build a WCF application, the text below describes the tasks and how they were achieved in the context of this work.

- Define the service contract which specifies the signature of a service, the data it exchanges, and other contractually required data. Looking back to Figure 3.5: *Server-side repository*, the interface "*ServicesContract*" and the classes inside "*DataContract*" define, respectively, the service operations and its data that are exposed through the network.
- Implement the contract that is a class that implements the service contract and specify custom behaviors that the runtime should have. The class *WCF Services* implements the *ServicesContract* interface. This is the core of the server-side solution.
- Configure the service by specifying endpoints and other behavior information. The configuration of the repository is made in the file *web.config* that is hosted together with the repository in the web server.
- Host the service. During the development, it was used the *ASP.NET Development Server* of Visual Studio 2010. To deploy, the application was hosted in an IIS web server.
- Build a client application.

In order to expose the characteristics that an external entity needs to understand to communicate with the service, the service metadata is published through *WS-MetaDataExchange* (MEX) protocol. This metadata includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service. The metadata can be used to generate automatically client code necessary to access the services. This code generation is one advantage of the SOAP/XML architecture.

In this work, the *ServiceModel Metadata Utility Tool* (Svcutil.exe) is used to generate the file *SPRServiceRef* (presented in *Figure 3.7: Multi-Layer Client Architecture* inside the *CommunicationLayer*). This file assists the programming of the client exposing locally the contracts provided by the server.

In addition to the service contract, the client and the server must agree also with the endpoint configuration in order to create the communication channel between them. As said before, in the server side these configurations are established in web.config file. The client should assent with the server also what protocols that will be used. One key issue to taken into account is the binding configuration. It is used to specify the transport, encoding, and protocol details required for clients and services to communicate with each other. In this work the *WSHttpBinding* is used; according to the *msdn* web site it described as "a secure and interoperable binding that is suitable for non-duplex service contracts" ("Bindings").

3.4 Difficulties and Limitations

The first obstacle faced during the development of this work was the inexperience with web services development. Adding to this, the lack of practical knowledge coding layered software architecture demonstrates a challenge to the development of this application. Specifically, the configurations related with the web service, mainly the binding configuration, were one of the hardest issues in this work.

Other difficulties arise from the changes in the requirements specification during the development. To minimize the loss of work, many prototypes were presented and validated during the development process.

3.5 Final Considerations

This chapter tried to show in details all the issues related with the development of the Taim safety pattern repository. It includes the analysis, project and implementation of the server-side and the client-side of the application.

The next chapter presents the repository in production environment through screen shots taken during its use and describes some concerns that were not achieved in this version of the software.

4 DEMONSTRATION

I deployed the Taim repository in a server at UFRGS in Brazil. Then, I installed the AddIn in a computer with Sparx Enterprise Architect (EA) and accessed the repository remotely over the internet in order to test the basic functionalities and to take screen shots to demonstrate the system. At the same time, several notes about the appliance were taken describing desirable changes for a next version of the tool.

As said before, the Taim repository is intended to store safety patterns as proposed by Antonino in (ANTONINO et al., 2012). However, this pattern representation was not finished during the tool implementation and, thus, some aspects were not foreseen during the analysis and the design of the system. Despite that, I think that is important to use the system with this representation so the limitations of the tool can be documented to be addressed in a posterior version of the tool.

Therefore, the safety pattern *Homogeneous Redundancy Pattern* was created in EA using the guidelines described by Antonino's research. After that, the implemented AddIn was used to access and to store the created pattern into the repository. The section 4.1 describes this process step by step with screen shots taken during the test. Another important use case for this application is to retrieve a previously pattern from the repository and put it into an EA's project. This mechanism is depicted in section 4.2. Section 4.3 points out the notes that were taken during the steps below. And finally the section 4.4 concludes this chapter.

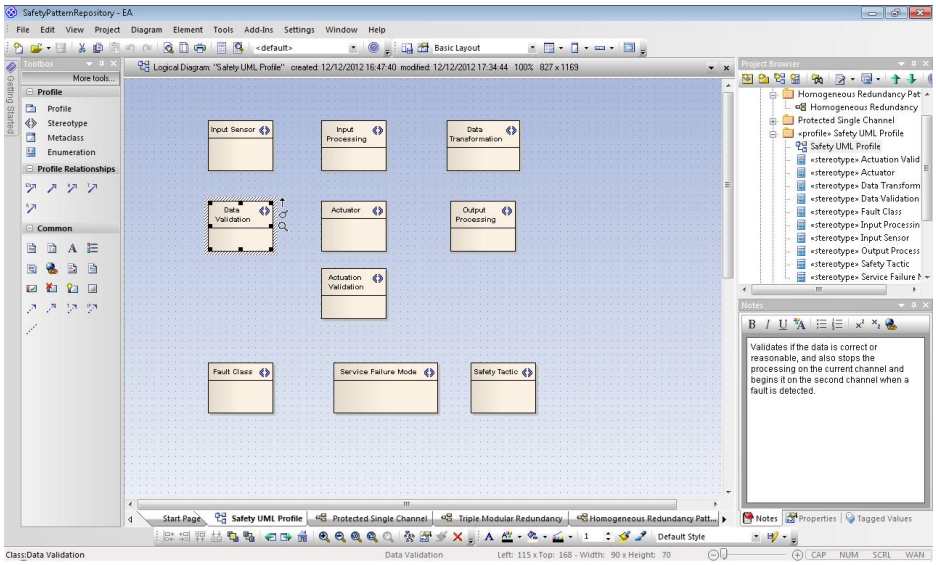

Figure 4.1: Safety UML Profile diagram in EA

4.1 Storing a Safety Pattern in the Repository

Following the steps to create a safety pattern, firstly, a *Safety UML Profile* was created in EA as depicted in Figure 4.1: Safety UML Profile diagram in EA.

For brevity, no special concerns were taken to create the profile. Each stereotype represents one role of the pattern, for the purpose of this work; each role is a part, as described before, of the pattern. The pattern was forged with instances of these stereotypes.

The Homogeneous Redundancy Pattern built in EA is showed in Figure 4.2: Homogeneous Redundancy Pattern in EA.

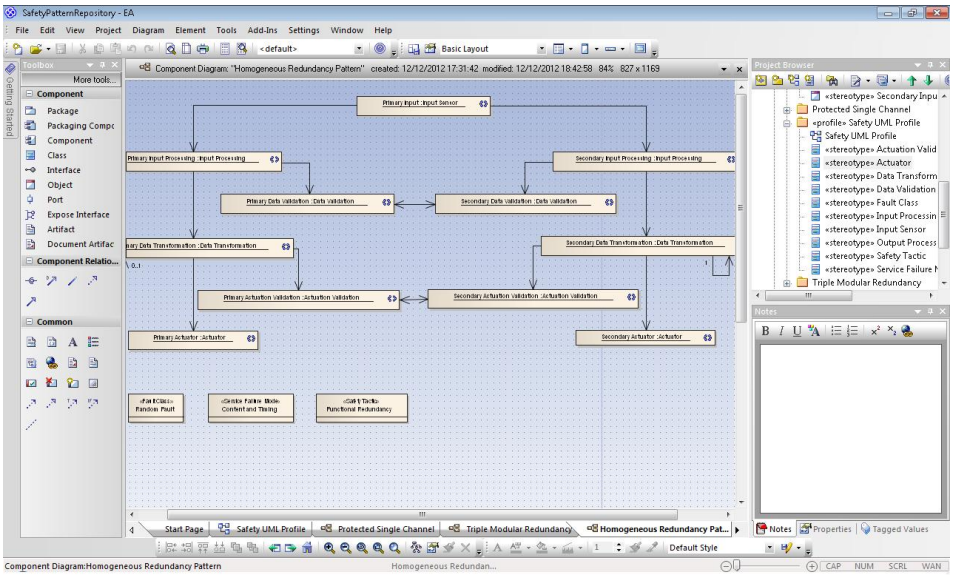

Figure 4.2: Homogeneous Redundancy Pattern in EA

The elements of the pattern are instances of the stereotypes from the *Safety UML Profile*. In addition to the parts of the pattern, the diagram shows elements to describe the *FaultClass*, the *Service Failure Mode* and the *Safety Tactic* that this pattern is related to.

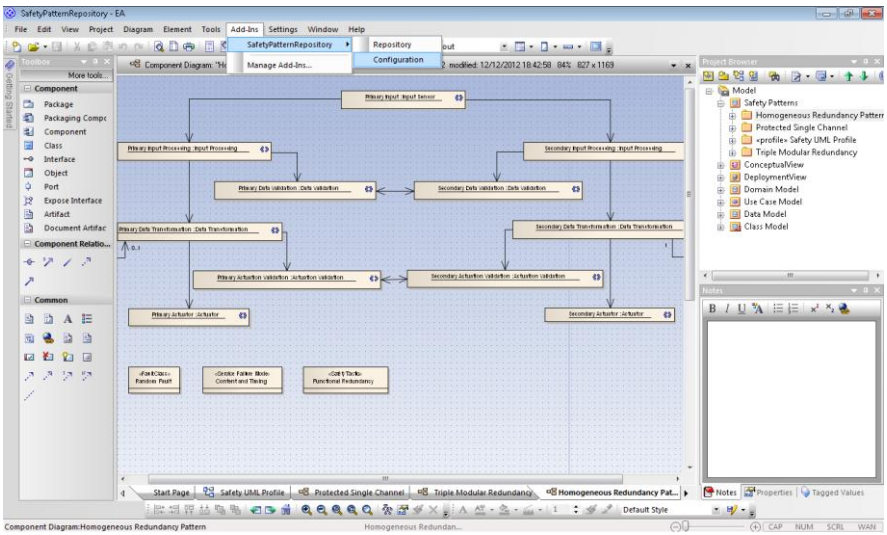

Figure 4.3: Repository Access from EA

After creating the pattern in EA, the AddIn Menu in EA was used to access the repository as shown in Figure 4.3: Repository Access from EA.

The first time that the repository is used, it is useful to set the network address where the repository is deployed as well as the address to access the service.

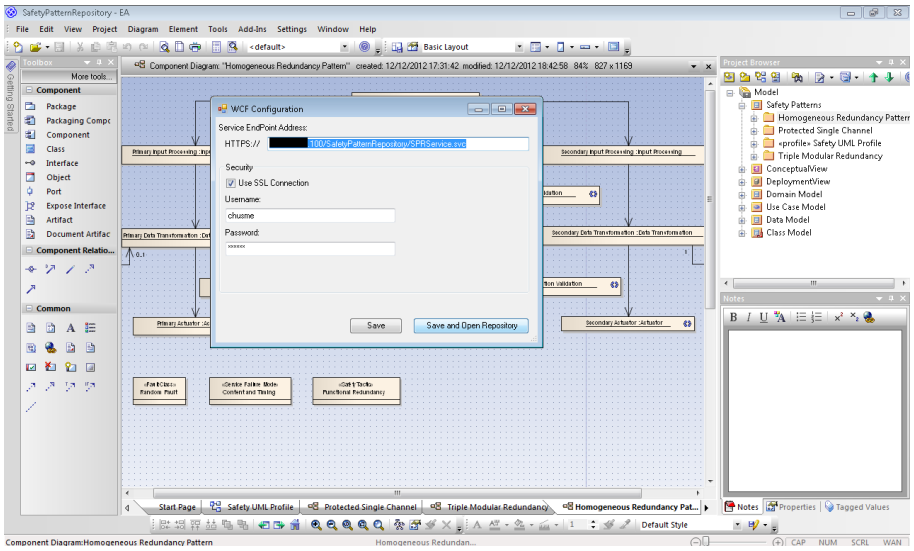

Figure 4.4: Repository access configuration

The deployed WCF services offer two service endpoints to access them. In this example it is used a security connection over HTTPS (the other service endpoint doesn't use any security to communicate with the repository). This kind of connection is recommended when the repository is deployed in internet. Another issue that is possible to see in the figure is the use of a username and password validation (that is used only in secure communication). In the actual version of the repository, only one user exists and it is hardcoded. This feature was implemented aiming a future expansion of the repository to a multi-user role-based system.

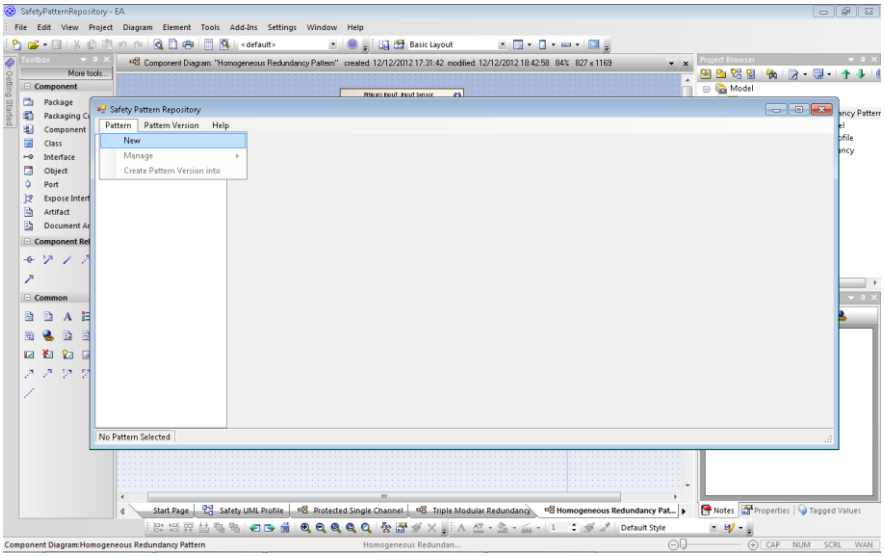

Figure 4.5: Repository main window and Pattern menu

After connect with the WCF service, the repository is shown to the user as in "*Figure 4.5: Repository main window and Pattern menu*". In this case, the repository is empty.

The menu *Pattern->New* presents to the user a form to create a new pattern. As said before, the pattern will serve as container to many pattern versions. The "*Figure 4.6: Pattern Creation*" shows this form already filled to create the new pattern.

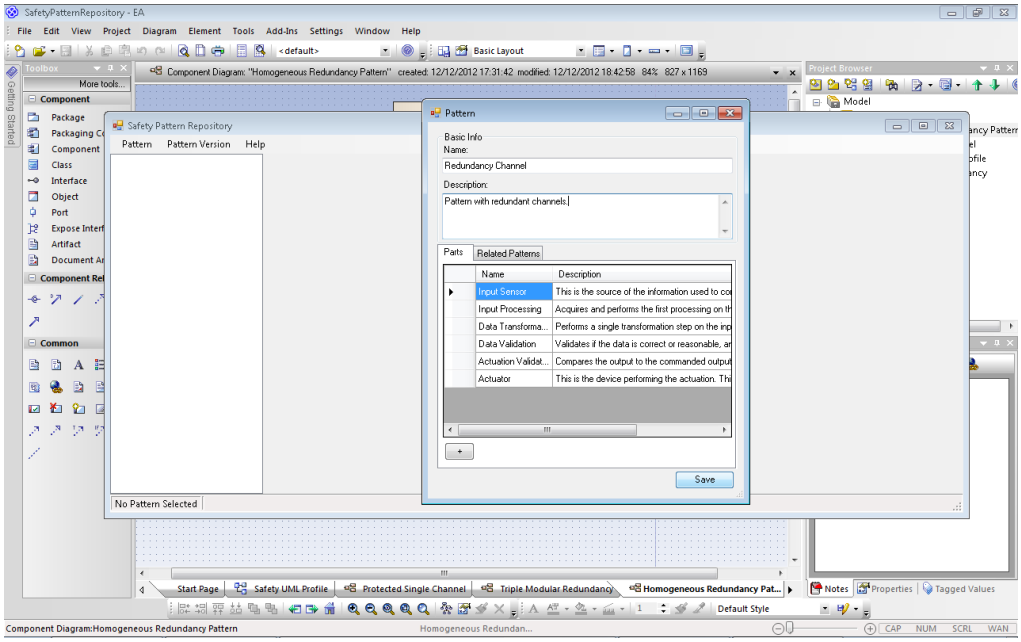

Figure 4.6: Pattern Creation

This form allows the user to set the pattern name and description, which is optional, as well as the Parts (Roles) that compose the pattern. It is also possible to define the related patterns, but they have to be stored in the repository first. The parts depicted in the figure represent the stereotypes defined before in the *Safety UML Profile*. When the user clicks the button "*Save*" the pattern is stored in the repository.

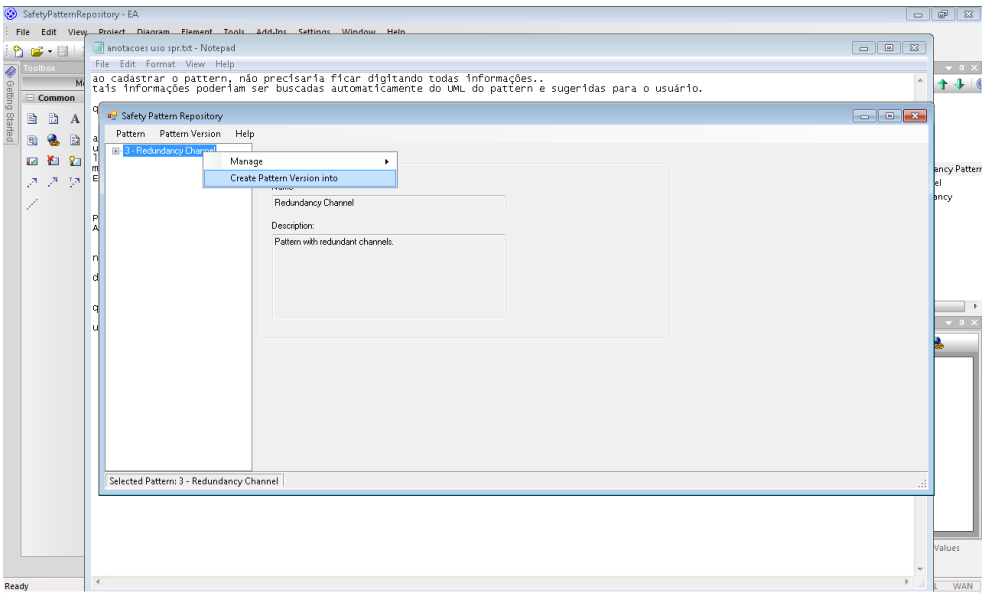

Figure 4.7: Menu to create a PatternVersion into an existent Pattern

After that, it is possible to create the PatternVersion into this pattern. One of the ways to create a new PatternVersion in the repository is depicted in Figure 4.7: Menu to create a PatternVersion into an existent Pattern.

The menu "Pattern -> *Create Pattern Version into*" opens a form to fill the information about the PatternVersion that will be stored into the selected Pattern. This form is shown in Figure 4.8: General Description from Pattern Version.

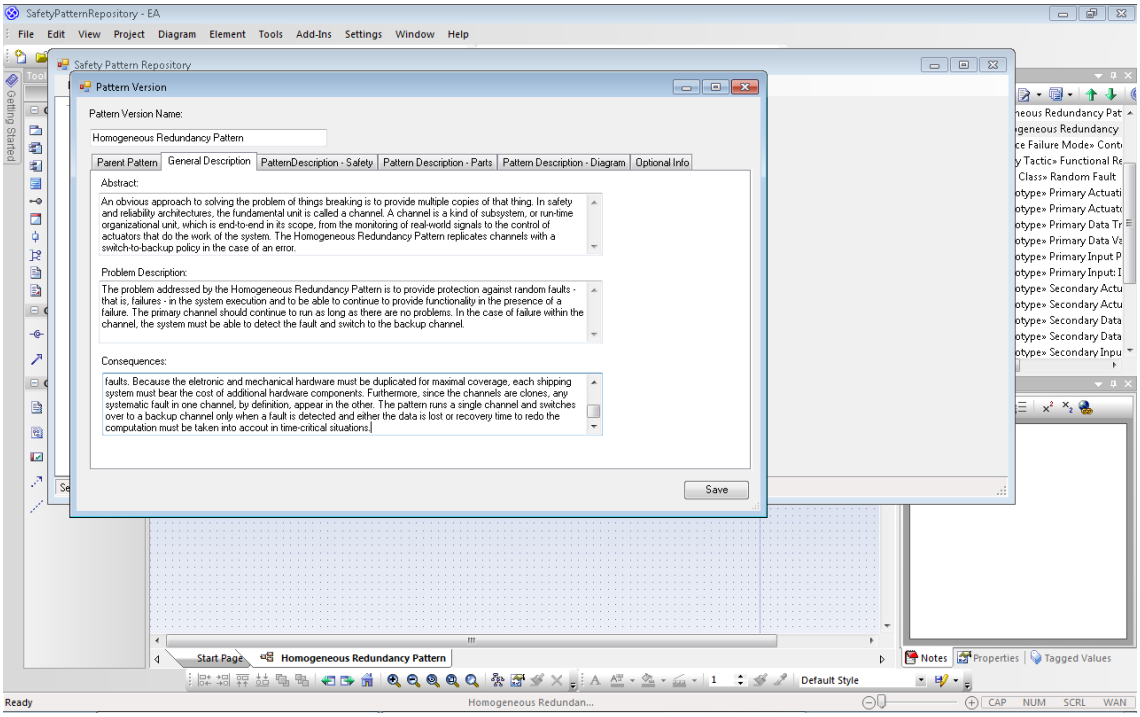

Figure 4.8: General Description from Pattern Version

Since a Pattern Version contains several information fields to store, for visual purposes, the form was subdivided in many tabs, each tab related with some characteristic of the pattern. In the tab "*Parent Pattern"* it is possible to set the pattern in which the PatternVersion is being allocated as well as create new parts and set the patterns related with it.

The "*Figure 4.8: General Description from Pattern Version*" shows the "*General Description*" tab that contains the *Abstract*, the *Problem* and the *Consequences*. The next tab "*Safety*" is very similar and shows text boxes to fill information about *Fault Class*, *Service Failure Mode* and *Safety Tactics*.

The *Parts* tab "*Figure 4.9: PatternVersion Parts*" shows the parts that compose this specific PatternVersion.

Each part from a PatternVersion is a part from the Pattern (parent of the pattern version) plus a description specific of it in the context of the PatternVersion.

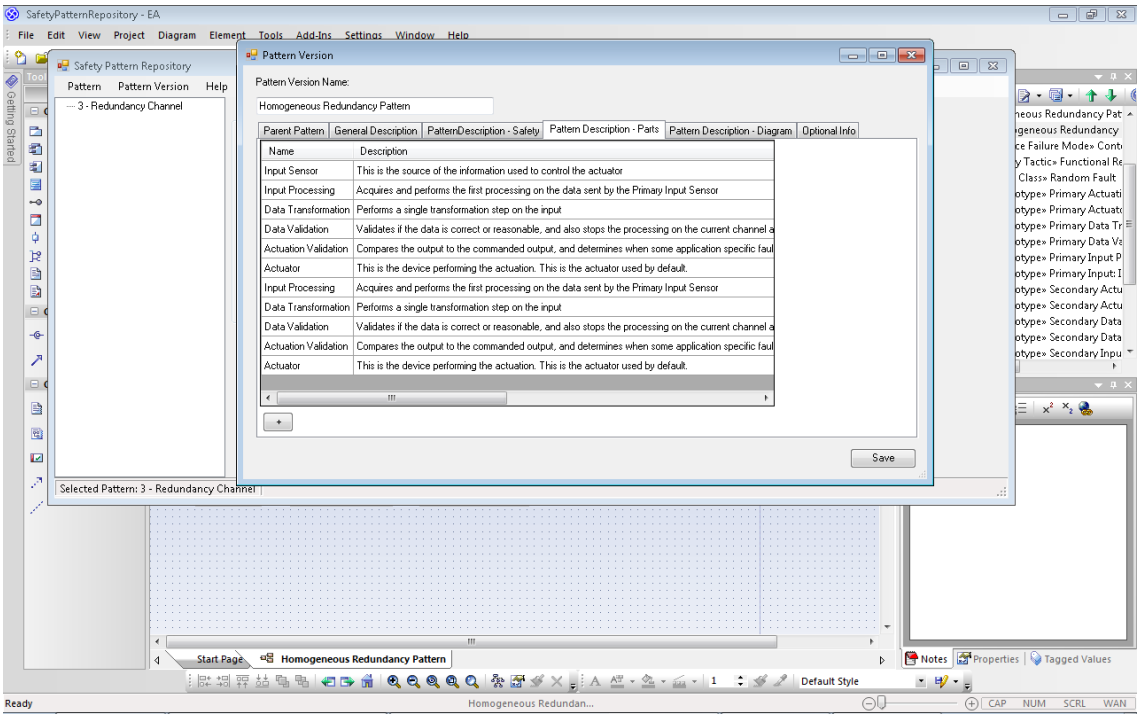

Figure 4.9: PatternVersion Parts

The next tab is used to define the UML Structure of the PatternVersion.

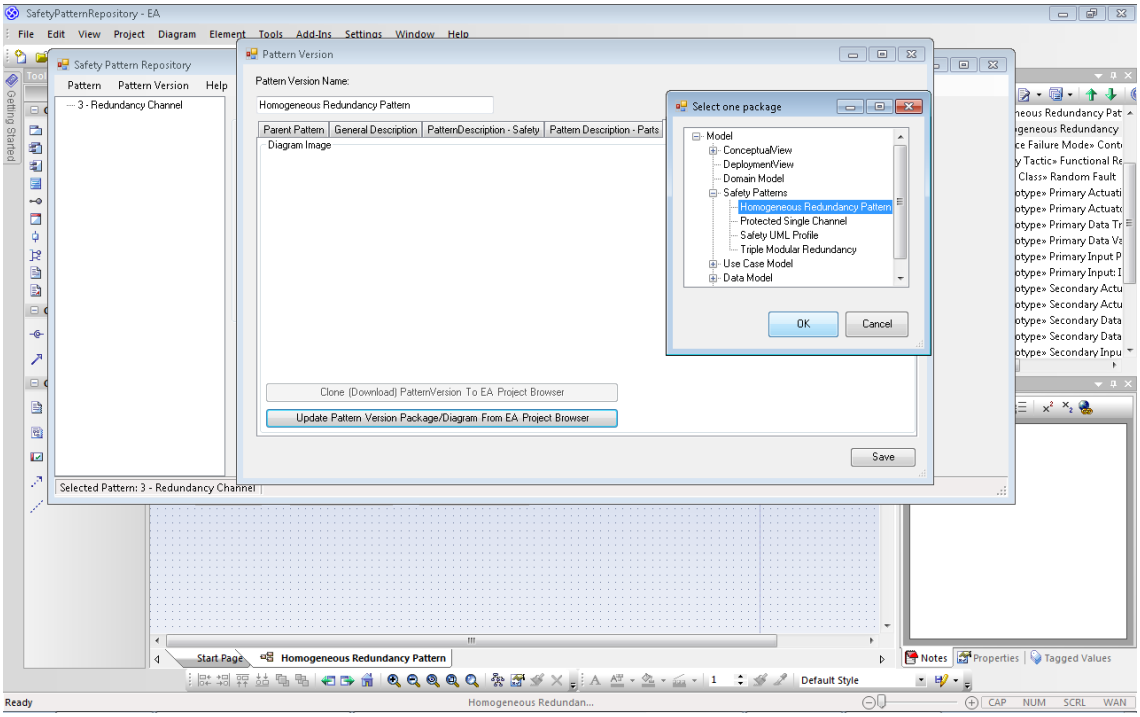

Figure 4.10: Choosing a UML Package from EA

The figure shows the Add-In accessing information about the opened project in EA. More specifically, it shows all the UML Packages from the current EA project so that the user can select the pattern that was previously designed in EA (the pattern in EA was showed in *Figure 4.2: Homogeneous Redundancy Pattern in EA*).

With this information, the Add-In uses a method provided from the EA API that exports a XMI and an image file from the selected UML Package in EA. These files are stored in the repository together with the textual information.

The last tab (Figure 4.11: Inserting an Optional Info into the PatternVersion) allows the user to fill the optional information about the PatternVersion.

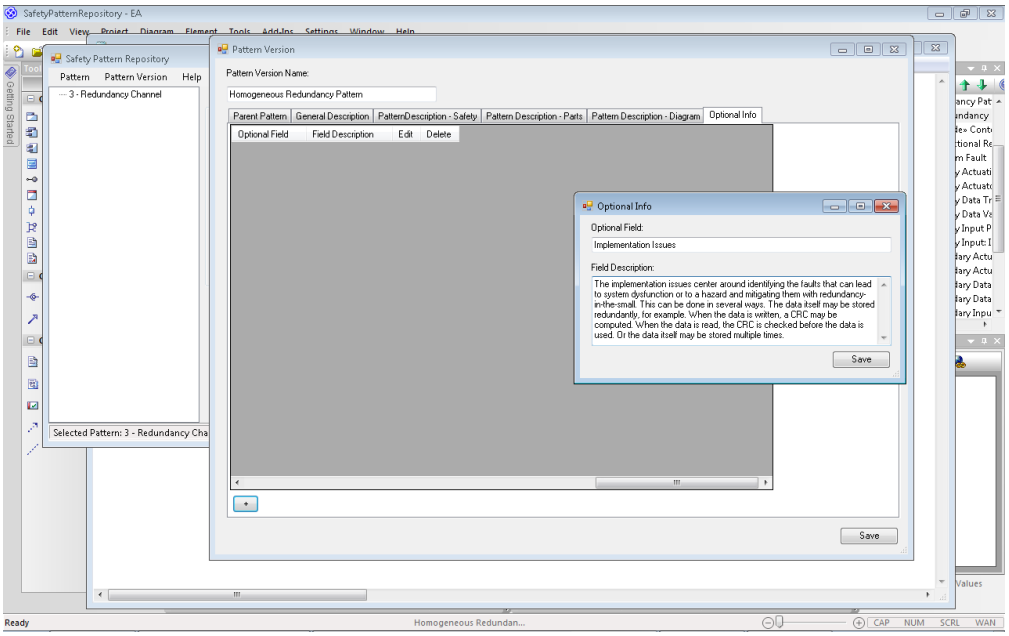

Figure 4.11: Inserting an Optional Info into the PatternVersion

4.2 Retrieving a Safety Pattern from the Repository

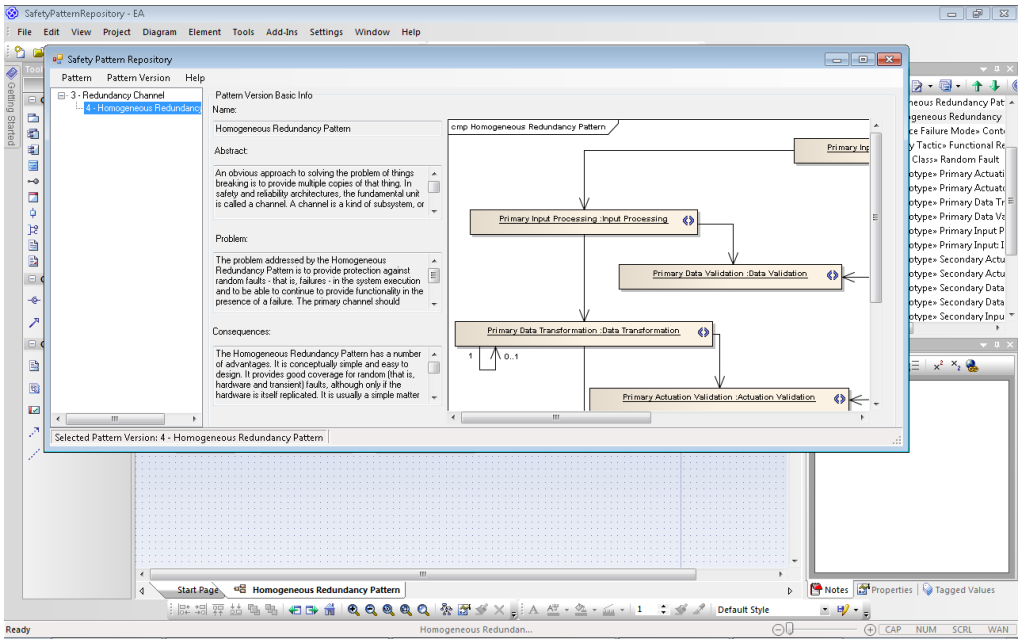

Figure 4.12: Browsing in the Repository

The process of retrieving a pattern (in the repository, this action is called *Cloning*) from the repository is quite simple. The main repository window (*Figure 4.12:*

Browsing in the Repository) presents all the patterns and pattern versions actually stored in the repository. It is possible to see some basic information of them and also the UML diagram of the Pattern Version.

After select the pattern that is desired to clone into EA, the menu "*Pattern Version -> Clone (Download into EA Project Browser*" (*Figure 4.13: Cloning a pattern from the repository into EA*) is used to imports the XMI package stored in the repository into the opened project in EA. It is necessary to choose what UML Package from the current Project Browser will be the container for the cloned pattern.

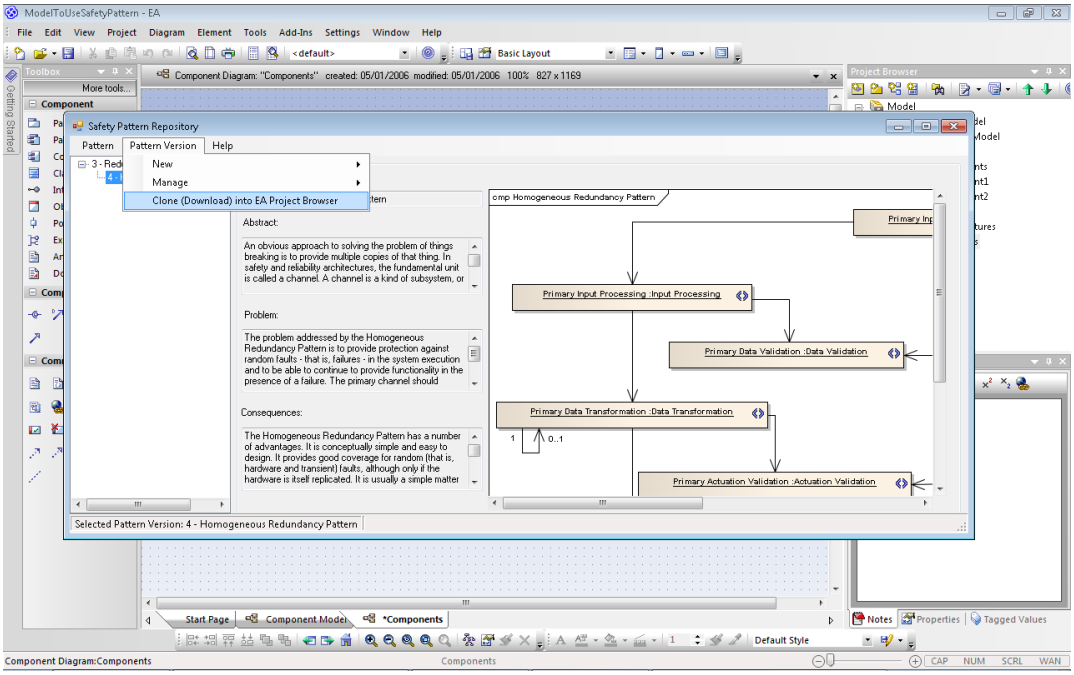

Figure 4.13: Cloning a pattern from the repository into EA

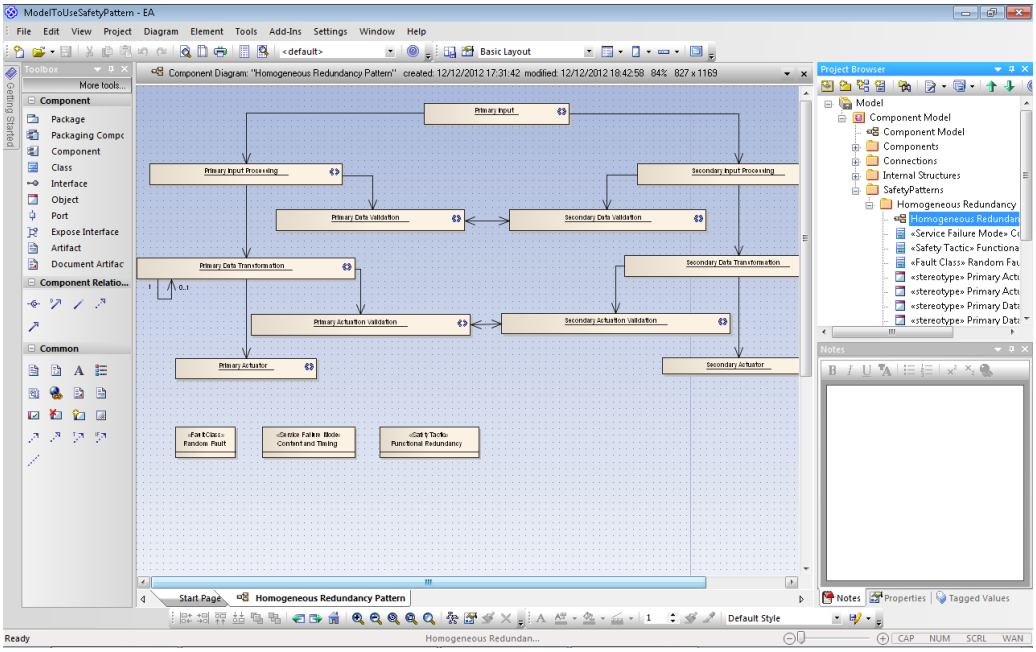

Figure 4.14: Cloned Pattern in EA

In this case, it was created a new package called "*SafetyPatterns*" for this purpose. The resulting of this action is presented in Figure 4.14: Cloned Pattern in EA.

4.2.1 Creating a new version from a previously cloned

One of the characteristics of the AddIn is that it keeps in memory the last pattern that was cloned. Therefore, it is possible to create quickly a new version of some pattern in the repository. To show this feature, the pattern cloned before was slightly modified (It was created a secondary input sensor) as shown in Figure 4.15: Cloned Pattern modified.

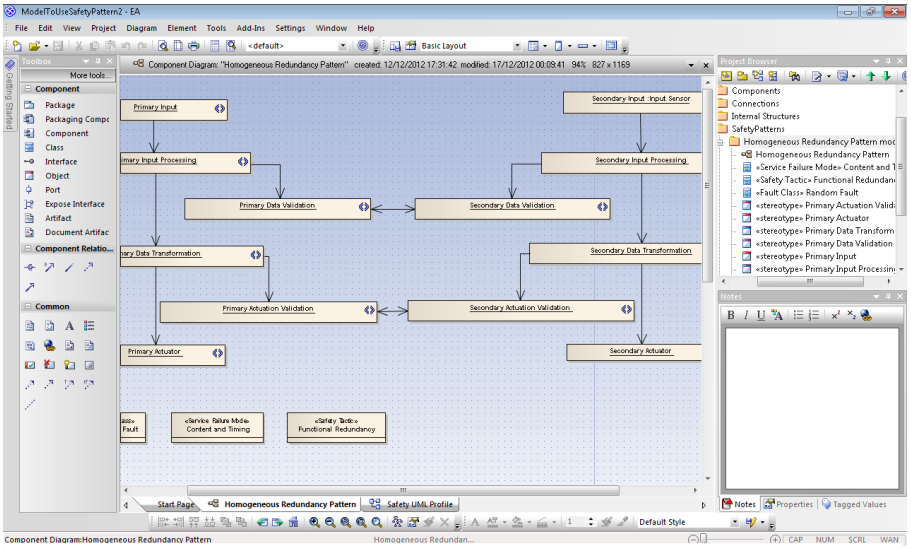

Figure 4.15: Cloned Pattern modified

After modify the pattern in EA, the repository should be accessed and then the menu "*Pattern Version -> New -> PatternVersion filled with last cloned one info*" (*Figure 4.16: Creating a new pattern from the previously cloned*) is used to create a new Pattern Version with the fields already filled with the information of the last Pattern Version cloned and with the modified UML structure.

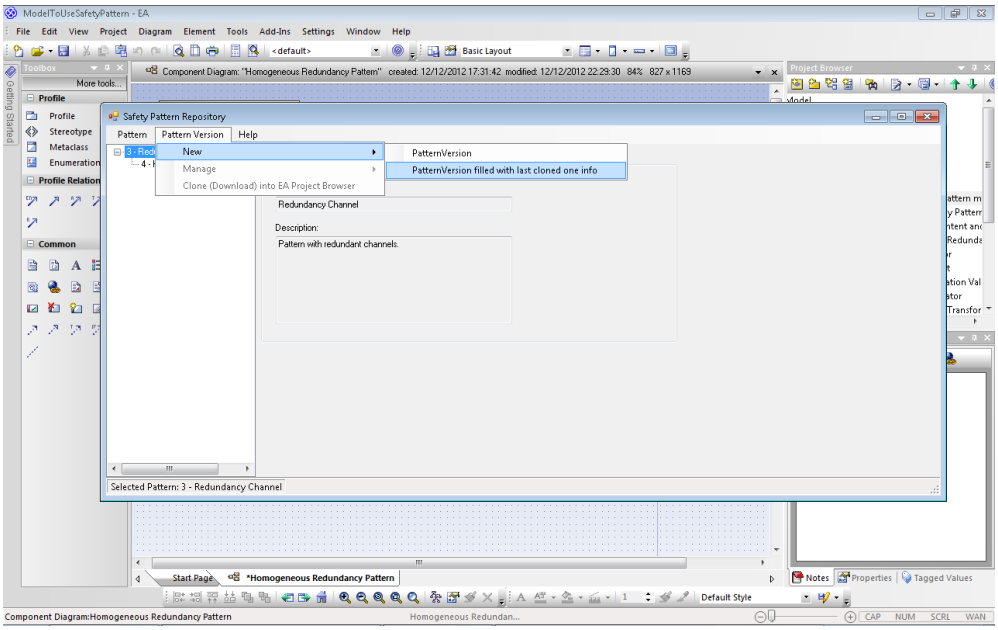

Figure 4.16: Creating a new pattern from the previously cloned

All the form fields were already filled as shown in Figure 4.17: PatternVersion information filled automatically.

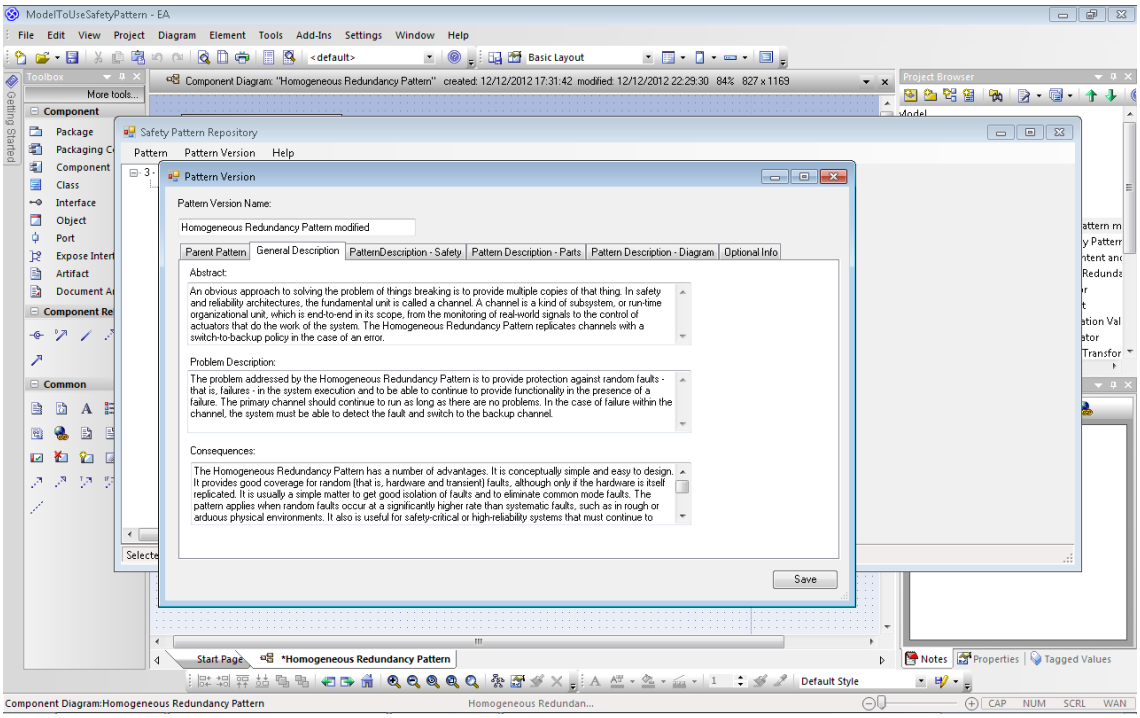

Figure 4.17: PatternVersion information filled automatically

With this option, the user can easily change only what effectively changed from one version to the other. The UML structure of the modified pattern was exported to the repository as shown in Figure 4.18: Modified UML diagram.

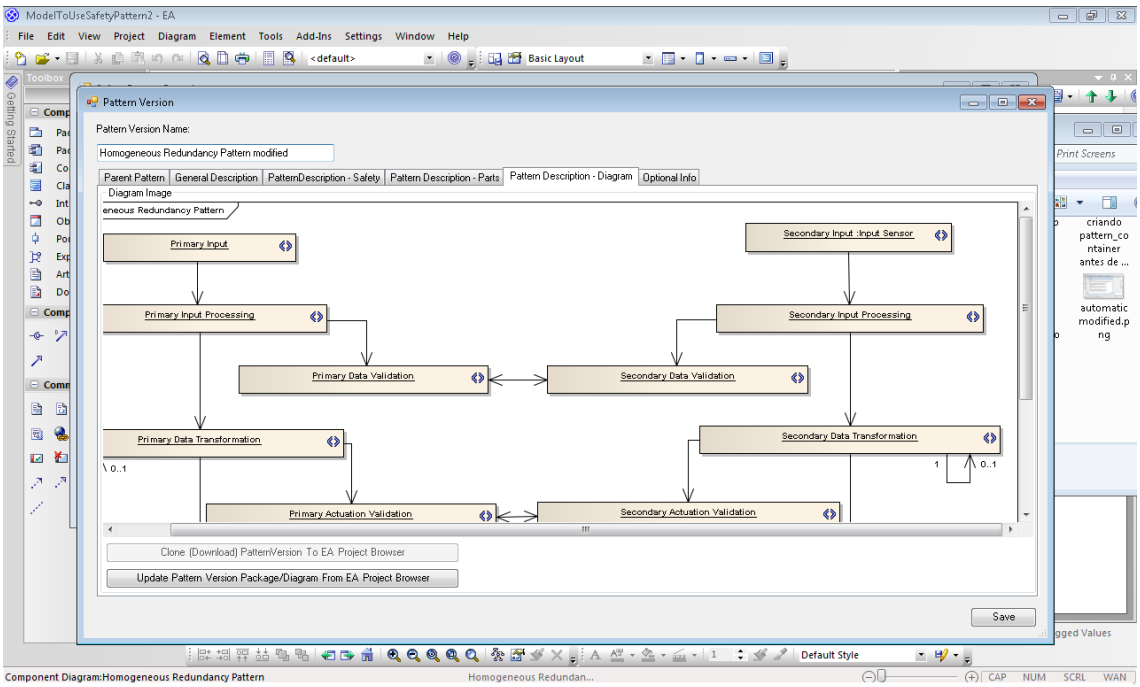

Figure 4.18: Modified UML diagram

It is important to note that the modified pattern doesn't override the cloned one, but it creates a new version that may be slightly different from the previous.

4.3 Observations about the Usage

Several observations were taken and it will serve to improve the next version of the repository.

The most relevant problem faced was that the UML structure loses the stereotypes when stored into the repository (this can be viewed in *Figure 4.14: Cloned Pattern in EA*). The reason for this is that the safety pattern is constructed using instances of UML stereotypes from a *Safety UML Profile*. When the UML package is exported from EA to the repository, as the UML Profile was not inside this package, these data were lost. One way to avoid this is to create the profile inside the UML Package of the pattern, but this is not desirable because the purpose of creating stereotypes in the profile which would be reused among several patterns would be lost.

Another characteristic noted was that when choosing the parts of a specific pattern version, there was no space to define a name for that part, the user interface provides components just to choose one part from the pattern parts and to write an observation of that part in the context of the pattern version. Since the Part in the Pattern represents a Stereotype and the Part in the PatternVersion represent an instance of this stereotype, should be possible to define a name for the part in the same manner than set a name for the instance of the stereotype.

One issue observed during the usage was that, to store the Pattern/PatternVersion in the repository, the user needed to create or write manually many things that had already been described during the construction of the UML structure in the CASE tool. It is desired the automation of this process in order to facilitate the storing of the patterns in the repository. For example, the Add-In could parse the UML structure to create all the Parts and PatternVersion Parts ensuring that the UML structure really represents the pattern description that is actually being stored.

4.4 Final Considerations

This chapter demonstrated the tool in use and described some considerations made during this usage. These considerations are very important to review some concepts that were previously specified during the requirements elicitation. Since the application was developed during the creation of the pattern representation, some characteristics such were not addressed in this version of the tool; however it is important to define them to create a next version consistent with the pattern representation that must be addressed.

The next chapter concludes the work and outlines it in the context with the project being developed and the future works.

5 CONCLUSION

This work described the development of the Taim Safety Pattern Repository. The system can be divided into two parts, the server-side is a remote repository that provides SOAP/XML web services and stores physically the information about the safety patterns in a PostgreSQL database; the client-side was built as an extension for EA that accesses remotely the repository and shows to the user its content, allowing him to use the patterns directly into EA or manage the content of the repository.

This work satisfies its original goals of designing and implementing a complete solution to store and retrieve safety patterns. It provides an interoperable interface through web services using standardized protocols in a way that third-party systems be able to access these services. In addition to this, the Sparx Enterprise Architect (EA) AddIn was created to prove this concept; the AddIn offers a way to create the UML structure of the safety patterns through EA and enables a system architect to store or to retrieve them from the repository.

The artifacts produced during the design of the system such as use cases, data model and system architecture diagrams together with relevant details of the implementation were presented as part of the results in section 3.3. Additionally, a demonstration of the most important cases regarding with the solution was carried out. The screen shots together with the notes taking during this use were showed in chapter 4.

5.1 Future Work

Some characteristics that must be addressed in a future version of the repository have already been discussed in Chapter 4. The most important of them is to create means to avoid losing the stereotype information when storing a pattern in the repository and try to automate the process of storing the pattern into the repository using information taken directly from the UML structure of the pattern.

To increase interoperability, in addition to what is actually implemented, the repository can be extended to communicate over different protocols, and the web services can be provided also following REST architecture with JSON. This would facilitate the access to the repository from web apps or applications in other platforms but .Net.

In addition to these major concerns some features are highly desired for a next version. Some of them include a pattern search mechanism and a user authentication engine that enables the pattern rating in order to improve the quality of the repository content.

The continuity of this product will be done by the research group in Germany.

REFERENCES

.NET Design Patterns in C# and VB.NET - Gang of Four (GOF) - DoFactory. Retrieved December 5, 2012, from http://www.dofactory.com/Patterns/Patterns.aspx.

11.02 Security Architecture Patterns. Retrieved December 5, 2012, from http://www.opensecurityarchitecture.org/cms/library/patternlandscape.

ALEXANDER, C.; ISHIKAWA, S.; SILVERSTEIN, M. **A Pattern Language: Towns, Buildings, Construction**. Oxford University Press, 1977.

ARMOUSH, A.; BECKSCHULZE, E.; KOWALEWSKI, S. **Safety Assessment of Design Patterns for Safety-Critical Embedded Systems**. 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009. SEAA '09. p.523 –527. doi: 10.1109/SEAA.2009.12, 2009.

ARMOUSH, A.; SALEWSKI, F.; KOWALEWSKI, S. **Effective Pattern Representation for Safety Critical Embedded Systems**. 2008 International Conference on Computer Science and Software Engineering. v. 4, p.91 –97. doi: 10.1109/CSSE.2008.739, 2008.

ARMOUSH, ASHRAF. **Design patterns for safety-critical embedded systems.** RWTH Aachen University. Retrieved from http://darwin.bth.rwth-aachen.de/opus3/volltexte/2010/3273/, 2010.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11 – 33. doi: 10.1109/TDSC.2004.2, 2004.

BACHMAN, F.; BASS, L.; KLEIN, M. H. **Deriving architectural tactics: A step toward methodical architectural design**. Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/03tr004.cfm, 2003.

BIEN, A. **Real World Java EE Patterns-Rethinking Best Practices**. 2012.

Bindings. Retrieved December 6, 2012, from http://msdn.microsoft.com/en-us/library/ff649327.aspx.

Catalog of Patterns of Enterprise Application Architecture. Retrieved December 5, 2012, from http://martinfowler.com/eaaCatalog/.

DEPARTMENT OF DEFENSE. **MIL-STD-882E Standard Practice for System Safety**, 2012.

Design Patterns Library | Fellowship Technologies. Retrieved December 5, 2012, from http://developer.fellowshipone.com/patterns/.

DOUGLASS, B. P. **Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns**. Addison-Wesley Professional, 1999.

DOUGLASS, B. P. **Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems**. Addison-Wesley Professional, 2002.

DUNN, W. R. Designing safety-critical computer systems. **Computer**, v. 36, n. 11, p. 40 – 46. doi: 10.1109/MC.2003.1244533, 2003.

Elements of Parametric Design. Retrieved December 5, 2012, from http://www.designpatterns.ca/.

Enterprise Architect - UML Design Tools and UML CASE tools for software development. Retrieved December 5, 2012, from

http://www.sparxsystems.com/products/ea/index.html.

ERL, T. **SOA Design Patterns**. 1st ed. Prentice Hall PTR, 2009.

FEDERAL AVIATION ADMINISTRATION. **FAA System Safety Handbook**, 2000.

FENELON, P.; MCDERMID, J. A.; NICOLSON, M.; PUMFREY, D. J. Towards integrated safety analysis and design. **SIGAPP Appl. Comput. Rev.**, v. 2, n. 1, p. 21–32. doi: 10.1145/381766.381770, 1994.

FOWLER, M. **Patterns of Enterprise Application Architecture**. 1st ed. Addison-Wesley Professional, 2002.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1st ed. Addison-Wesley Professional, 1994.

IBM Software - Rational Software Architect Family. Retrieved December 5, 2012, from http://www-01.ibm.com/software/awdtools/swarchitect/.

LEVESON, N. G. **Safeware: system safety and computers**. New York, NY, USA: ACM, 1995.

Net Objectives Design Patterns Repository. Retrieved December 5, 2012, from http://www.netobjectivestest.com/PatternRepository/index.php?title=Main_Page.

ANTONINO, P. O.; NAKAGAWA, E. Y.; KEULER, T.; TRAPP, M.; **Towards an Approach to Represent Safety Patterns**, 7th International Conference on Software Engineering Advances (ICSEA'2012), Lisboa, Portugal, 18 a 23 de Novembro de 2012, p. 1-6.

openpatternrepository - Publicly available Online Pattern Repository - Google Project Hosting. Retrieved December 5, 2012, from https://code.google.com/p/openpatternrepository/.

Patterns · intuio. Retrieved December 5, 2012, fromhttp://intuio.at/en/blog/?cat=patterns.

Portland Pattern Repository. . Retrieved December 5, 2012, from http://c2.com/ppr/.

Reference [Enterprise Architect User Guide]. Retrieved December 6, 2012, from http://www.sparxsystems.com/uml_tool_guide/sdk_for_enterprise_architect/reference.htm.

StarUML - The Open Source UML/MDA Platform. Retrieved December 5, 2012, from http://staruml.sourceforge.net.

The Hillside Group - A group dedicated to design patterns. Home of the patterns library. Retrieved December 5, 2012, from http://hillside.net/.

UI-Patterns.com. Retrieved December 5, 2012, from http://ui-patterns.com/.

VLISSIDES, J. **Pattern Hatching: Design Patterns Applied**. 1st ed. Addison-Wesley Professional, 1998.

Web Services Architecture. Retrieved December 6, 2012, from http://www.w3.org/TR/ws-arch/.

Wikipatterns - Wiki Patterns. Retrieved December 5, 2012, from http://www.wikipatterns.com/display/wikipatterns/Wikipatterns.

WU, W.; KELLY, T. **Safety Tactics for Software Architecture Design**. Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01. , COMPSAC '04.. p.368–375. Washington, DC, USA: IEEE Computer Society. 2004.

Yahoo! Design Pattern Library. . Retrieved December 5, 2012, from http://developer.yahoo.com/ypatterns/.