A METHOD FOR MAPPING XML-BASED SPECIFICATIONS BETWEEN DEVELOPMENT METHODOLOGIES

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

BY
Fei Huang

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

S7N 5A9

ABSTRACT

The Unified Modeling Language (UML) is widely used by software engineers as the basis of analysis and design in software development. However, UML ignores human factors in the course of software development because of its strong emphasis on the internal structure and functionality of the application. This thesis presents a method of mapping human-computer interaction (HCI) requirement specifications generated by usability engineering (UE) methodologies (e.g. Putting Usability First (PUF)) into UML specifications. These two sets of requirement specification are specified, using Extensible Markup Language (XML) so that HCI requirement specifications can be integrated into UML ones. A Mapping Tool was developed to facilitate the creation of mappings between PUF XML tags and XMI tags. The Mapping Tool was used to create mappings between PUF and UML requirement specifications. This mapping process and its outputs were evaluated to demonstrate that the tool worked. The results of the evaluation show that the HCI requirement specification represented by the PUF XML tags can improve the UML specification by adding them into the XMI tags.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Unified Modeling Language (UML) [18] is widely used in the area of software engineering (SE) as the basis of analysis and design in software development. UML primarily focuses on technology issues. It does not contain sufficient human-computer interaction (HCI) requirements for software engineers who want to include HCI requirements in their own requirement specification. As a result, many usability problems may still exist in the resulting software systems. These systems may be too complex to learn, may not be comprehensible to users, or may not work properly. For example, when surfing web sites, the user may experience difficulties in:

(1) Navigation, because the navigation controls are hard to find or poorly labeled;

(2) Accessibility, because the site was not designed to be usable by users with disabilities;

(3) Ease of use, because the page design is counter-intuitive and the designer expects users to "learn" the site, rather than using standard conventions;

(4) Finding information, because the web content and its layout are poorly designed.

These and other problems result in systems that are not compatible with the end-user's needs and expectations. Usability Engineering (UE) can help software engineers to identify and to satisfy HCI-related concerns. This chapter serves to analyze the problems and present a solution.

1.1 Problems to Be Solved

A usable system is important to both end-users and developers. It makes the difference between user acceptance and user rejection. A usable system needs to be properly designed and many usability issues should be taken into consideration in the process of system development. However, many SE methodologies, for example UML, do not have a strong focus on usability issues. The usability-related requirements specified by UML are distributed in different UML models. UE methodologies provide additional usability-related requirements which can help software engineers in the area of UE.

1.1.1 Designing Usable Systems

Generally speaking, usability is a measure of how well an end user accomplishes a task while using a system and how satisfied he/she feels about the experience of interacting with the system.

ISO 9241-11 Guidance on Usability (1998) [1] defines usability as, “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”.

- Effectiveness is defined as “the accuracy and completeness with which users achieve specified goals.”
- Efficiency is defined as “the resources expended in relation to the accuracy and completeness with which users achieve goals.”
- Satisfaction is defined as “positive attitudes to the use of the product and freedom from discomfort in using it.” [1]

The ISO definition of effectiveness indicates that the system should be error-tolerant. A system should be able to prevent a user from making errors and should be easy to recover from errors or limit the effects of errors when they occur.

Efficiency addresses how much time and effort a user spends finding information or performing a task. A system should be easy to use. It should minimize the effort required for users to learn how to use it and should not make them struggle to find information in the system

Satisfaction addresses a user’s subjective impression about the system. It includes the following concerns:

- Does the user enjoy the process of accomplishing a task or does the user feel frustrated attempting it?
- Does the system seem attractive to the user?
- How much does the user like the system?

A usable system should ensure the user a pleasant experience of using the system.

Usability is important because a usable system would be of great benefit to both users and developers.

From users’ perspective, a usable system ensures that it takes less time and efforts to accomplish tasks with great accuracy and the experience of using the system should be pleasant rather than annoying. The benefits of a usable system include improved productivity and user satisfaction.

Usability is important to developers because it helps determines whether a system is a success or a failure. A system's lack of usability is risky because users often have choices of which system to use and

may choose a competing system that is more usable. If users encounter obstacles while interacting with the system, they might refuse to continue using the system. If a system is unattractive, the system might lose users. Lack of usability increases the chance of redesign and code revision and likely induces developers to spend extra time and efforts maintaining the system or fixing the errors. The opportunity cost here is significant. It takes away from other investments.

On the contrary, for developers, a usable system reduces cost of development because of the fewer late design changes and less need for technical support, and reduces cost of operation because of the decreased need for training.

Considering the importance of usability, many research groups have developed sets of principles for designing a usable system. Following are a couple of examples.

ISO 13407 Human-centered Design Process for Interactive Systems [2] identifies the human-centered design activities. This international standard complements existing design approaches and methods, aiming to improve usability of interactive systems. It points out that a human-centered design should have the following characteristics:

- "The active involvement of users and a clear understanding of user and task requirements";
- "An appropriate allocation of function between users and technology";
- "The iteration of design solutions";
- "Multi-disciplinary design" [2].

A set of more specific principles for designing interactive systems are established in *ISO 9241-110 Ergonomics of Human-system Interaction—Part 110 Dialogue principles* [3]. The seven dialogue principles can play a guiding role while user interface (UI) designers design and evaluate interactive systems. They are listed as follows:

- "Suitability for the task";
- "Self-descriptiveness";
- "Conformity with user expectations";
- "Suitability for learning";
- "Controllability";
- "Error tolerance";

- "Suitability for individualization" [3].

While we can recognize the need for usability and may be able to identify a variety of usability problems, it is better if these considerations are included in the design process in order to avoid such problems arising in the end product. IBM's User-Centered Design process [4] suggests six user-centered design principles and the process of the user-centered design.

Raïssa Katz-Haas illustrates the stages of the user-centered design process in "Ten Guidelines for User-Centered Web Design" [5]. They are:

- "Involve users from the beginning" by
    - discovering their conceptual models and requirements,
    - "observing them at their workplace; validating your assumptions about them; analyzing their tasks, workflow, and goals",
    - "eliciting feedback via walk-throughs, card sorting, paper prototypes, think-aloud sessions, and other methods".
- "Know your users" by asking questions about knowledge background, cultural background, etc., and use the answers to guide development and design decisions.
- "Analyze user tasks and goals" by observing and interacting with users "(preferably at their workspace) as you attempt to answer questions" about
    - what the tasks are,
    - how they are performed,
    - what their information needs are,
    - what their ultimate goals are.
- "Do not settle on a final direction too soon" by exploring different designs / approaches and getting user feedback "before making final direction, development, and design decisions".
- "Test for usability—repeatedly" by iterative usability testing throughout the development cycle.

1.1.2 Usability or User Interface Design in SE / UML

Methodology is defined as "a codified set of practices (sometimes accompanied by training materials, formal educational programs, worksheets, and diagramming tools) that may be repeatably carried out to produce software" [45]. The methodologies discussed in the thesis refer to the approaches employed and

the processes performed in one or more phases of system development under the guidance of the theories, principles or a systematic set of ideas. The details, instructions and results of the procedures carried out are explicitly documented with the benefit of tools, for instance, graphical notations and textual descriptions. In UML, for instance, use case diagrams are used to capture a system's behavioral requirements by graphically representing the interaction between the actors and the system. Use case templates are the written schemes for recording detailed requirements. These templates precisely describe the functional and non-functional requirements which the system should meet. The requirements captured both by use case diagrams and use case templates are used to drive the design or implementation of the system.

Software engineers use methodologies aiming at developing systems with fewer defects or deficiencies and ultimately providing better quality. However, because of the limitations or the drawbacks of the methodology, usability problems may be found in the resulting system.

UML is a methodology which is widely used by software engineers in the software development life cycle. UML provides a large set of diagrams (class diagrams, object diagrams, use cases diagrams, collaboration diagrams, state diagrams, etc.) which can be used for capturing requirements, design and implementation. UML is discussed in depth in section 2.1.

The UI is one of the crucial elements which determine user acceptance and satisfaction of the system [43]. From users' perspectives, the UI is regarded as the system which they interact with. The UI is the bridge for the communication and interaction between people and machines.

However, UML provides very limited support for usability or UI design. da Silva and Paton [23] point out that it is not easy to use UML to model "UI elements", such as "user tasks and presentations". Carter et al states that "the current structure and practice of use cases cannot meet the usability-related information needs, although many authors believe that a good use case should include a large amount of usability-related information" [6]. Moreover, the usability-related information specified by UML is scattered across different models in UML.

If SE methodologies overemphasize functionalities or technologies without performing relevant studies on these elements of the user's experience, the UI will often be unsuccessful. A bad UI costs money, and it could even defeat major system development projects. Although general

software-engineering development methodologies have helped developers create the UI on time and within budget, they often fail to deliver sufficient usability. Several examples in reality are given by Dray [41]:

- Users simply refused to use the application because the interface was "unlearnable and unusable."
- Developers did not find a serious problem in "their assumptions about how data would be entered until doing a User Acceptance test shortly" before deployment.
- "Extensive and expensive functionality in a Human Resources system was not used at all because users forgot how to access it a mere week after training".

In many cases, UI failures are due to the fact that software system development is often technology-driven. "Developing the actual software for computer systems is still a job for qualified personnel who are familiar with all the intricate details of programming languages, operating systems, and databases" [21]. UI requirement specifications are paid insufficient attention. The first law of UI design is:

"***Know your user*** *- your user is different from you!*" [6]

Failure to know the user can also result in not knowing what content the user needs, how well the functionality fits user needs, how well the flow through the application fits user tasks, and how well the response of the application fits user expectations. This information can be obtained only from end-users who actually use the system. If the software is developed without a full understanding of users, their tasks and needs, usability problems will not be detected as early as possible or resolved and may ultimately cause failure of the whole project.

1.1.3 HCI / UE Requirement Specifications as a Supplement to SE/UML Requirement Specifications

Usability does not just happen by itself. Some software developers try to balance a technology-centered perspective and a user-centered one. It is possible that usability problems are identified after design. Fixing the problems often requires tremendous effort and high costs to redesign the system or recode the application and imposes significantly elevated risk of additional error if it is a rushed redesign or recoding. Developers have to start learning more about what their users need and questioning the design. If the problems were left as is, user satisfaction could decrease after using the system. Users could choose an alternate product where they do not have this type of usability issues. To avoid this kind

of situation, software engineers should take usability issues into consideration as early as possible by employing the skills of usability engineers.

Usability engineers can bridge the communication gap between users and software developers. They adopt a number of communication methods and types of techniques – for example, questionnaires, prototyping, interviewing, etc. -- to capture both non-functional requirements and functional requirements. Usability engineers also employ informal methods to gather usability information, for instance, observation. They spend time at users' workplace to discover their work activities and procedure without being noticed. It is crucial for usability engineers to take an active part in the requirement, analysis, design and evaluation phases of a project, aiming at keeping each development phase user-centered. Doing so makes it possible for usability problems to be identified earlier in the life cycle. Therefore, costs can be minimized, delays can be reduced and product quality can be improved and risks lowered.

This thesis focuses on requirement specifications because requirements can provide guidance to the rest of system development. A poor-quality or incomplete requirement specification will lead analysis, design, and implementation to be built on a shaky foundation or in the wrong direction. Although an appropriate and complete requirement specification cannot promise a successful system, it at least makes it possible.

If the requirement specification contains more user-centered requirements, there is a much greater chance that the information system will support users and enhance their tasks adequately. It is necessary to focus on system's usability from the requirement specification stage to ensure the design will work. To make full use of UE's advantages developers should combine UE requirement specifications with those of SE. Some key principles to aid in this are listed below:

- The analysis stage in UE stresses the understanding of users, the conceptualization of their work, the data and processes needed for their work by means of modeling the users. Therefore the analysis is human-centered. The requirement specifications obtained help or allow for the generation of computing technology that augments and supports human performance.
- As far as design is concerned, UE employs iterative design, which progressively refines the design through evaluation from the early stages of design. The evaluation steps enable the

designers and developers to incorporate user and client feedback until the system reaches an acceptable level of usability.

It is important for HCI/UE to find a way to supplement SE's requirement specifications to make them more complete. By doing so, the combined specifications will include usability-involved properties as well as the functional ones of a system. This thesis will suggest a method for integrating HCI requirement specifications developed by UE methodologies into SE/UML requirement specifications by means of Extensible Markup Language (XML) [10].

1.1.4 A Variety of HCI / UE Methodologies

There is no agreed upon methodology within the areas of HCI and/or UE. Various different methodologies addressing usability issues have been created, such as: Put Usability First (PUF) [7], "The Usability Engineering Lifecycle" [8] and "User Centered Design (UCD)" [9] which is a widely accepted methodology for designing usable applications.

There are many similarities between these methodologies. Developers consider the user needs from the very beginning at the requirement analysis stage; developers make full use of the available information to model users; developers follow guidance on how to identify, analyze, and describe different user groups; developers analyze tasks, users, and concern about tools' constraints. Developers employ a wide variety of approaches to conduct user and task analysis.

Many HCI/UE methodologies focus either on processes or on requirement specification structures. Only PUF combines both processes and requirement specification structures into a robust methodology. As far as requirement specifications are concerned, PUF emphasizes not only the usability requirement specifications posed by the tasks, tools, users, and content, but also those that arise in the interactions between them so that PUF provides more complete and more detailed usability information for developers. Furthermore, PUF goes beyond a mere concern for the user and concerns itself with all facets of usability, both for the user and for developers. Concepts and details in terms of PUF will be further described and illustrated in Section 2.2.

1.2 Discussion of Solution in General

There is a need to provide a general means of integrating different HCI/UE methodologies within a SE led development. My solution is to map and integrate a set of usability requirement metadata developed by

UE methodologies (such as PUF) to a set of functionality-oriented SE requirement metadata (such as UML). The method suggested in this thesis is generic so that it can be adapted to other UE methodologies. Compared with the current UML diagrams, the resulting UML requirement specifications that capture "non-functional" information as well as its counterpart, the "functional" information, can help software engineers look at a system from multiple perspectives rather than only from a technical perspective. The solution ensures that human-computer interaction requirements can be captured at the very beginning of the software development life cycle. The relatively complete and proper requirement specification plays the role of a foundation upon which UML diagrams are constructed. The resulting UML diagrams will assist software engineers in knowing their users better and being aware of usability issues in the course of development. When they design a UI or a system, its functionalities will be supplemented and improved because our solution will make sure that the UI or the system is developed "in a context-rich information environment" [6]. Hopefully, this kind of requirement specification will avoid the situation that the problems are not found until the later processes where they are then more costly to fix. Reducing this danger can make a successful system more likely.

A requirement specification is composed of requirement metadata and project requirements. The solution in this thesis focuses on requirement metadata. To make the focus clear, it is necessary to define project requirements and requirement metadata and differentiate between these two concepts.

In this thesis, project requirements are the detailed and specific requirements managed within a specific project. These requirements are usable only in their particular project and cannot be applied to other projects. Therefore, project requirements vary from one project to another. Metadata is normally known as "data about data". In the scope of this thesis, requirement metadata provides information about aspects of requirements, including their attributes (name, type, etc.) or descriptive information about the context, or characteristics of the requirements. Requirement metadata provide a general framework for identifying, recording and dealing with project specific requirements. The generality of requirement metadata allows developers to apply it to various projects. According to the definition of the terms methodology and requirements specification, each methodology should have a set of requirement metadata to direct developers to collect project requirements and document them. The requirement metadata generated by HCI/UE is beyond that found in the UML. This thesis focuses on mapping between

two sets of requirement metadata generated by two methodologies, and using this mapping to integrate HCI/UE requirement metadata into SE metadata. The "mapping" in this thesis is aimed at requirement metadata rather than project requirements. The generality of metadata determines that there should not be any variation between mappings across projects. Specifically, once the mapping is done, it can be used for all future translation of project requirements for that specific project.

The process of integrating project requirements from one methodology into those from another methodology is called "translation". It is beyond the scope of this thesis, but would rely on the mapping produced in this thesis.

The precondition of the solution proposed in this thesis is to express two different methodology requirement metadata and information in the same language. I chose XML [42] to undertake the task. My method is to map XML representations of PUF requirement metadata into XML representations of UML diagrams. The general procedure for this solution is:

- Identify the mappings between a PUF XML file and XML Metadata Interchange (XMI) [14] file.
- Determine the mapping relations for each mapping.
- Integrate the PUF requirement metadata within the UML specification via XML. According to the mapping relation, carry out the transformation operations on the XMI tags so that the resulting XMI file will contain PUF XML tag information.

Computer Aided Software Engineering (CASE) tools are needed to perform the above activities. I will depict the existing CASE support for HCI/UE and SE/UML respectively.

1.2.1 CASE Support for HCI / UE

When taking a look at CASE support for usability, there are many tools especially devoted to usability oriented web design. Chak lists a number of technologies for designing a usable web site, such as, Web Static Analyzer Tool (WebSAT), Lift Online and Lift Onsite, NetRaker Suite [11]. There exist many tools to support certain activities of a UE methodology. EZSort [12] is a tool that supporting UCD that is used to organize the web content to meet users' requirements.

CASE support is needed to increase the impact of HCI/UE requirement specification on SE requirement specification. This requires that the CASE tools should support any UE methodologies rather

than only a particular methodology. To do so, the University of Saskatchewan USERLab is conducting a UE project. The overall structure of this effort is shown in Figure 1.1.

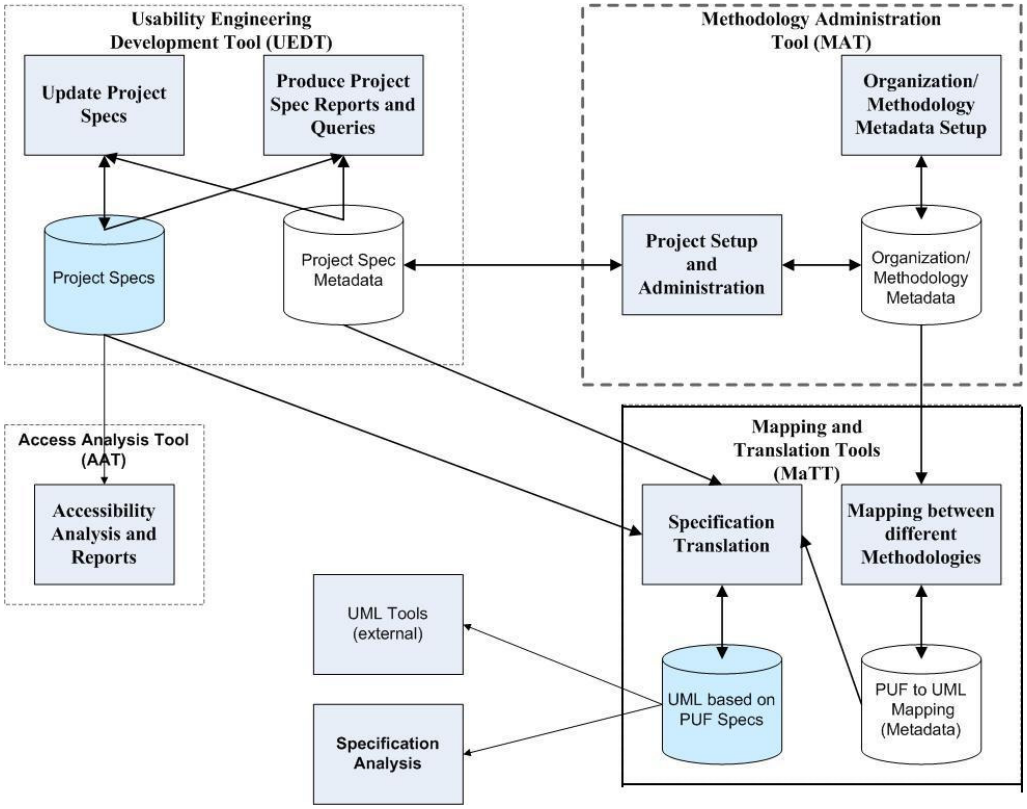

Figure 1.1: UE Project Diagram

Work is now underway to define generic usability documents as XML files. In Figure 1.1, The Methodology Administration Tool (MAT) and the Usability Engineering Development Tool (UEDT) in the dotted boxes in Figure 1.1 will be constructed by other members of USERLab. The descriptions of the tools in the solution are as follows:

Methodology Administration Tool

The MAT is used to specify HCI/UE requirement metadata in XML format. The Organization/Methodology Metadata database stores the following files:

- a general XML schema definition (XSD) [13] describing metadata of HCI/UE methodologies, known as a Methodology Metadata XSD file.

- an XSD describing a generic record of UE requirement metadata that has record metadata, section headers and row items (headers, questions and linkages), known as a record template XSD file.

The XML for a particular PUF record is generated to meet the specifications laid out in a record template XSD file.

Usability Engineering Development Tool

The UEDT is used to record and work with project specific usability requirement specifications. It generates a XML instance document for usability-related project requirements which are gathered from end-users based on requirement metadata provided by MAT. The resulting XML instance document is one of the inputs of the Translation Tool. Using CASE tools to store project requirements in a database facilitates reusing and updating of these data.

1.2.2 CASE Tools for Integrating the Results of HCI / UE CASE within SE CASE

Because both types of methodologies, SE and HCI/UE, describe the functionalities of a system, the interactions between users and the system, and the context of these interactions, there are likely to be overlaps between the requirements identified by HCI/UE methodologies and those by SE methodologies. Some requirements will be identical. Since some focus on different aspects of the system, their overlaps will be only partial. However, a large amount of HCI/UE requirements do not exist in SE specifications. Due to the complexity of both types of methodology requirements and different types of relationships between methodology requirements, mapping between two sets of requirements and transforming one set of requirements to another one are especially complex and difficult. CASE tools are one of the assistant technologies to which we can turn.

The Methodology Administration Tool (MAT) in the USERLab project is able to represent PUF requirement metadata via XML. With regards to UML, XMI is an Object Modeling Group (OMG) standard which is commonly used for interchanging UML diagrams via XML. Currently, CASE support for UML is developed very well. It is easy to obtain XMI encodings of UML. A large number of UML tools create XMI files for UML diagrams, such as Poseidon for UML [15], ArgoUML [16], and EclipseUML [17].

Although both HCI/UE and SE have CASE tools available for requirements gathering, there is no CASE support to fulfill the integration task. CASE support for mapping and transformation is needed to satisfy the need for automating the integration process. The Mapping and Translation Tools (MaTT) shown in the solid box in Figure 1.1 are a proposed means to integrate the results of HCI/UE CASE within SE CASE.

In order to perform the tasks described at the beginning of this section, the Mapping Tool, the Translation Tool and a UML CASE Tool are required. Following is the brief description of the role of these tools:

Mapping Tool

The role of the Mapping Tool is to map HCI/UE requirement metadata to SE/UML requirement metadata via XML, specifically PUF requirement metadata and UML requirement metadata, and to identify the relationship between them, called the mapping relation. For each mapping, according to its mapping relation, a corresponding transformation operation is specified. The resulting mapping information including the identified mappings and their mapping relations will guide developers to translate various project requirements using the Translation Tool.

Translation Tool

The Translation Tool is then needed to integrate actual usability-related project requirements into UML requirement specifications. Based on the requirement metadata from MAT, specific project requirements from UEDT and mapping information provided by the Mapping Tool, the Translation Tool integrates specific XML-based HCI/UE project requirements into XML-based SE/UML requirement specifications.

UML Tool

An open source UML modeling tool is needed to output UML diagrams from XML representation of UML diagrams. It needs to be flexible enough to work with the additional XML tags resulting from the mapping and translation. Open source can make it possible to include specific ways of representing this additional data.

1.3 Contributions of This Thesis

Because we acknowledge the importance of a usable system to both developers and end-users and are aware of the issue of the lack of usability in current applications, this research is motivated to provide aids in the development of usable systems. Specifically, it places a focus on generating relatively comprehensive requirement specifications by integrating HCI/UE requirement metadata into SE requirement metadata.

The main objectives of this thesis are outlined as follows:

- Develop a general method for integrating HCI/UE requirement metadata into SE specifications. Although the requirement metadata of the existing HCI/UE methodologies may be various, as long as the HCI/UE requirement metadata are represented in XML format, this method should work for any HCI/UE methodology, rather than a particular methodology. UML, a widely accepted SE specification methodology, and PUF, a relatively robust UE methodology, are adopted as a pair to demonstrate the feasibility of this general method.
- Explore UML diagrams to identify the preliminary mappings between PUF requirement metadata and UML metadata and investigate how to integrate the former into the latter.
- Determine how much overlap there is between the UE requirement metadata and SE requirement metadata by analyzing the preliminary mappings. The analysis also intends to show how UML could be improved by integrating with PUF requirement metadata.
- Develop the Mapping Tool in Figure 1.1 to demonstrate this solution using PUF as an example of UE methodologies and UML as an example of SE methodologies, supporting and automating the process of creating mappings between XML-based PUF requirement metadata and UML requirement metadata and integrating the former ones into latter ones.

1.4 Chapters in This Thesis

The thesis is structured as follows:

Chapter 2 summarizes the technical background to this thesis. It covers the concepts of UML, PUF, XML and how XML works for PUF and UML.

Chapter 3 establishes requirements for mapping PUF to UML. These requirements will be analyzed from three perspectives, user, task and content.

Chapter 4 offers a detailed depiction and demonstration about the design of the Mapping Tool. It includes database design, function design, screen design and program design. An example is performed to demonstrate how to use the Mapping Tool.

Chapter 5 discusses the results of the evaluation of the possible mappings produced by the Mapping Tool.

Chapter 6 provides a summary of this thesis, lists a number of recommendations and discusses appropriate future work following from the efforts made for this thesis.

CHAPTER 2

BACKGROUND

This chapter delves into the details of the technologies used in our approach. Figure 2.1 is the detail of Figure 1.1, showing the document flows between various methodology CASE tools. XML and the related technologies constitute the major part of the infrastructure on which we perform the translation of two methodologies. The ovals represent the tools used in this project. The rectangle represents requirement specifications generated by these tools. The representatives of SE methodology (such as UML) and UE methodology (such as PUF) are shown on the very right and very left.

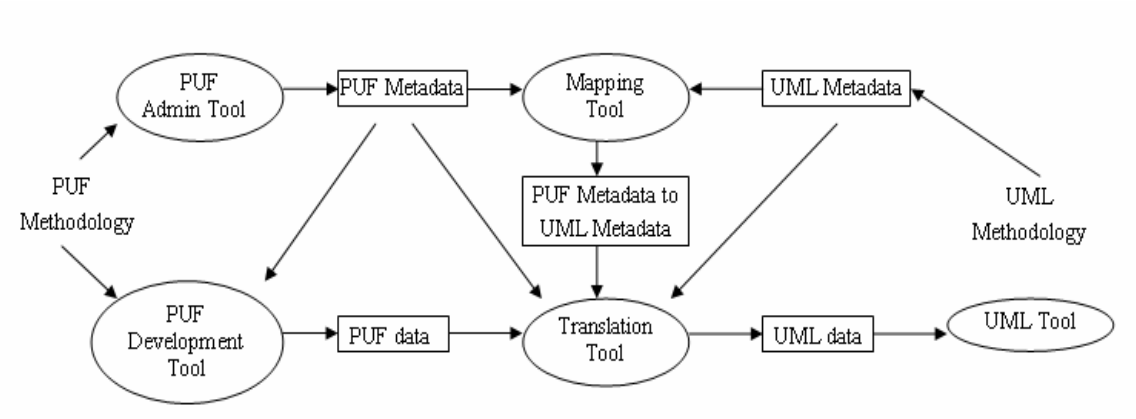

Figure 2.1: Details of UE project

The introduction will start from two ends and move towards the middle. The elements involved in the integration process will be illustrated in the following logical order: The first three sections discuss about PUF, UML and the mappings between their components at the high level. Then the next three sections discuss how PUF and UML requirement specification can be represented in XML and how to do XML tag mapping between two heterogeneous XML documents. The following list shows the outline of this chapter.

-- UML

- Brief description of UML diagrams
- UML models used in analysis and design activities
- Related work to expand UML to include user interface requirements

-- PUF

- Major processes within PUF
- Possibility types within PUF
- The PUF requirement metadata structure

-- The comparison of UML and PUF concepts

-- XML representation of UML specifications

-- XML representation of PUF requirement metadata

-- XML is used for mapping PUF requirement metadata to UML metadata

2.1 UML

UML is described as a "general-purpose visual modeling language that is designed to specify, visualize, construct, and document the artifacts of a software system" [19]. A set of diagrams are employed to depict the function, structure and internal behavior of software systems reliably through the use of the Object-Oriented approach.

2.1.1 Brief Description of UML Diagrams

In UML 2.0 there are 13 types of diagrams that are used to view a system from different perspectives. To understand them, it is sometimes useful to categorize them hierarchically, as shown in the chart in Figure 2.2 [19]. These diagram types are briefly described below. The concepts used in these diagrams, such as classes, objects, states and activities, are the foundation of UML. UML users can customize these concepts according to their needs. Specifically, UML can be extended by applying three mechanisms: stereotypes, tagged values, and constraints.

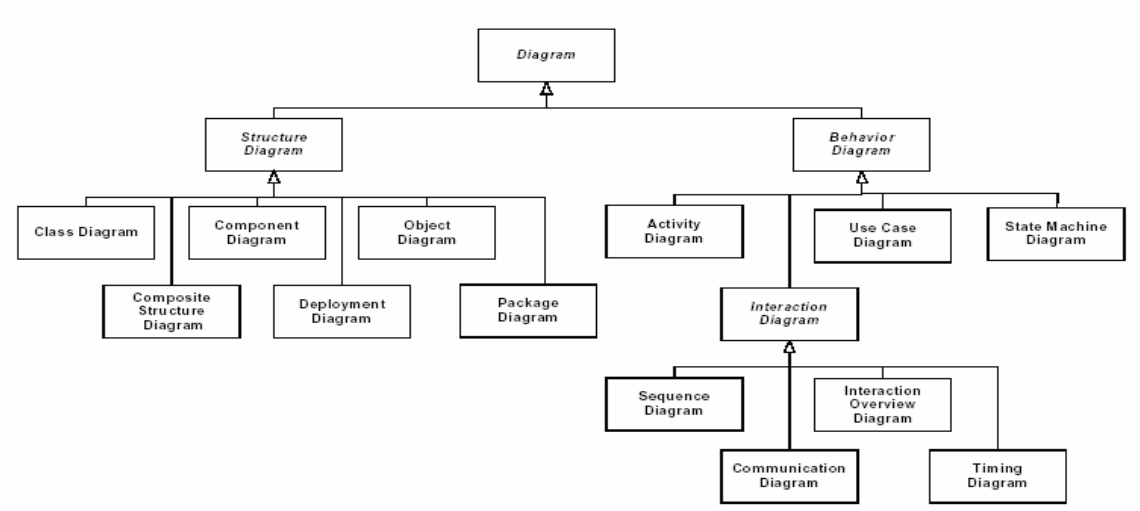

Figure 2.2: Hierarchy of UML 2.0 Diagrams [19]

Structure Diagrams emphasize "what" must be in the system being modeled:

- Class diagrams

  Class diagrams describe properties of objects, such as attributes, associations and the various static relationships among them. They represent a group of objects sharing the same look and behavior. Classes could be software things, hardware things, problem domain objects, etc., when used to document usability engineering requirements. Therefore, the concepts in class diagrams are more general and abstract, compared with object diagrams. The way to represent a property of a class is as an attribute. The operations notation describes the actions which a class carries out.

- Component diagrams

  "Component diagrams aim to show components' interrelationships through interface or the breakdown of system components into a lower-level structure." [22]

- Object diagrams

  Object diagrams depict instances of classes. They are useful for showing the interrelationship between instances of objects. Thus a couple of object diagrams are adopted to make an abstract structure defined by a class diagram easy to understand and to describe the specifications of the run-time interactions. The class diagrams are more general.

- Composite structure diagrams

  Composite structure diagrams break down an object into parts hierarchically. The required and provided interfaces are shown in these diagrams. Ports can be added to interact with external structure.

- Deployment diagrams

  "Deployment diagrams show a system's physical layout, revealing which pieces of software run on what piece of hardware" [22].

- Package diagrams

  Package diagrams are normally used to describe a whole picture of the relationships between major model elements, for instance classes, states and activities. They describe the high level structure of the system.

Behavior Diagrams emphasize the "how" by modelling interactions within a system at different levels of abstraction:

- Activity diagrams

  Activity diagrams are concerned with interactions at the lowest level of abstraction, that is, those within an operation. They are used to explore and describe "a work flow, the actions carried out in the operations of a class" [21]. In other words, an activity is directly related to the operations in a corresponding class diagram.

- State Machine diagrams

  State machine diagrams are used to model the interactions within a class or its associated objects. They specify the states a class or an object can have and how these states are changed by different events over time.

- Use case diagrams

  Use case diagrams are used to model the context of a system within which the interactions between an actor and a system take place. The context of a system includes the system context and the business context. Use case diagrams capture three types of requirements of the system: "business", "functional" and "non-functional".

Interaction Diagrams, "a subset of behavior diagrams, emphasize the interactions between objects or between subsystems" [22]. Each of the four types of interaction diagrams realizes a particular example of the execution of a system, which is known as a "scenario".

- Sequence diagrams

  Sequence diagrams depict "the sequence of messages between a set of objects, including the order and the timing of the messages" [22].

- Collaboration (UML 1.x)/Communication diagrams (UML 2.0)

  Same as sequence diagrams, communication diagrams explore and visualize how objects communicate with each other. Normally, Communication diagrams are used to model scenarios.

- Interaction overview diagrams (UML 2.0)

  Interaction overview diagrams are "a grafting together of activity diagrams and sequence diagrams" [22].

- Timing diagrams (UML 2.0)

  Timing diagrams are adopted to "represent time constraints either for a single object or several objects" [22].

Since the detailed requirements of the system are specified by modeling use cases, use case diagrams will be discussed more deeply. Based on the function of each UML diagram, Figure 2.3 [20] shows a meta-model describing the relationships between use case and the other types of diagrams. An interaction diagram is used to describe a scenario which is regarded as an instance of a use case. Thus a use case corresponds to one or more interaction diagrams. The interaction diagrams also depict the interaction between the objects at an abstract level. State machine diagrams describe the behaviour of one or more objects or actors. Figure 2.3 helps identify the possible detailed mappings between PUF requirement metadata and UML requirement metadata.

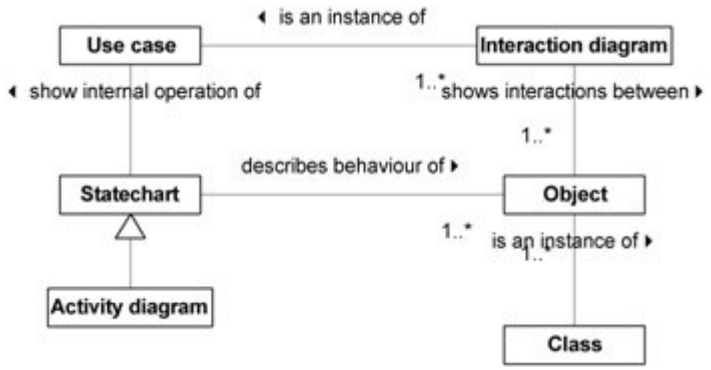

Figure 2.3: Meta-model showing relationships between use cases and types of diagram [20]

UML is an expressive modeling language. The expressiveness of UML is realized by the extensibility mechanisms it supports. UML has three extensibility mechanisms. They allow developers to depict and enrich the semantics of possible situations. Following is a brief description of these extensibility mechanisms:

- *Tagged values* allow developers to add new element properties for any model element. The information is attached to the corresponding elements.
- *Stereotypes* allow developers to produce a specific model element as a subtype of an arbitrary one. This extensibility mechanism can be considered as an equivalent to subtyping.
- *Constraints* allow developers to specify additional semantic constraints for a model element.

2.1.2 UML models Used in Analysis and Design Activities

SE has largely adopted UML as the basis of analysis and design activities. In order to get a better idea about how the UML models apply to real life experience, the requirement, analysis and design activities during a typical iteration of the development process along with the diagrams applied in these activities will be discussed.

- Requirement Workflow

  The purpose of this activity is to find out what system functionalities need to be realized according to users' expectations and requirements. Using understandable and clear diagrams can facilitate both non-technical people and developers interpreting the system. There are a number of UML techniques adopted in this development activity, such as use cases, class, and interaction diagrams including sequence diagram and collaboration diagrams. A use cases diagram is used to

gather functional requirements. A sequence diagram is used to depict a use case's main flow and alternative flows. A state machine diagram specifies the internal operations of a use case [22].

- Analysis Workflow

  After setting up the initial requirements, developers begin analyzing the architecture and behavior for the system in detail. They polish up the use cases, model the architecture of the system, and demonstrate the communication among classes. A couple of UML models get involved in this development activity. For example, activity diagrams are used to describe the workflows within the use case. The collaboration diagrams are used to specify a series of messages passed between objects [22].

- Design Workflow

  When software engineers are doing design, more details and precision are required. A number of UML techniques can come in handy, for example, class diagrams, sequence diagrams, object diagrams, component diagrams [22].

2.1.3 Related Work to Expand UML to include Usability Requirements

As mentioned in Section 1.1.2, although UML has proven to be effective in both academic and industrial areas, UML provides limited support for UI design. The survey [24] shows that software engineers tend to use Class Diagrams to gather requirements and "many projects are not use case driven" [24]. Although some projects are driven by use cases, usability requirements recorded in the use case specifications focus on the value provided by the system to external entities such as human users or other systems. If developers want to capture requirements that lie outside the scope of use case descriptions, they have to add supplementary specifications to use case or other specifications.

Various researchers exerted their efforts to expand UML to support UI design. The typical examples follow:

Carter et al [6] suggest a method by which the HCI requirements identified by Usability Engineering methodology (e.g. PUF, CAP) is integrated into the UML specification used to develop the end product. The high-level mappings between the PUF requirement concepts and the UML concepts are identified in the paper [6].

Nunes (2001) [27] proposed the Whitewater Interactive System Development with Object Models (Wisdom) method. The Wisdom method is composed of Wisdom process, architecture and notation. The user-centered development process and architecture suggested in the paper ensure that usability concepts are included in UML.

The method proposed by Kruchten et al [28] is to extend use cases to support user interface design. He brought in the concept of "use case storyboard" into UML. The use case storyboard describes the interaction between an actor and a system in text and related usability requirements in a use case. Sequence and collaboration diagrams in the use case storyboard illustrate how to visualize the collaboration between objects and actors in the use case through the user interface.

The above related work has a few problems. First, some of them suggested the development of a user interface during the design stage. They lack recognition of the importance of requirement specification. Second, HCI specialists have to learn UML modeling language, which they may not know very well. Otherwise, they cannot make full use of UML models to exhibit HCI concerns in UML-HCI models. Third, from the resulting UML-HCI models, software engineers utilizing UML leave a false impression that they have usability engineering experience. So the importance of HCI professionals in the development team is not underlined in these methods. In fact, software engineers must cooperate with HCI professionals to capture HCI modeling.

Besides "extending UML to include user interface modeling", De Paula et al [25] point out that there are other ways by which to integrate HCI aspects with Object-oriented (OO) modeling, for example, "using models that represent HCI aspects to guide OO modeling and creating OO design method to specify the concrete user interface".

2.2 PUF

Putting Usability First (PUF) is a usability engineering methodology which takes usability issues into consideration in the course of software development to ensure the resulting system is developed in an environment of rich user-centered information. PUF is a usable methodology that assists developers in developing a usable system for end users.

As for the "effectiveness" in the definition of usability [1], the structure of requirement metadata is able to help developers accurately and completely identify user groups, tasks, scenarios, content and tools that provide detailed context of use.

As for the "efficiency" in the definition of usability [1], PUF guides developers and provides the flexibility to choose the most efficient usability methods out of those ones identified in ISO 16982 Usability Methods Supporting Human Centered Design [14] concerning a specific context of use.

As for the "satisfaction" in the definition of usability [1], PUF supports supplying usability-related requirement metadata to software engineering specifications. PUF also supports integrating UE activities into SE activities. Consequently, developers will feel comfortable using this methodology.

In contrast to other UE methodologies, the PUF methodology is a relatively robust methodology because it highlights not only processes but also requirement metadata structure. Many UE methodologies focus on either processes or requirement metadata structures. For example, the UE method suggested in Nunes's (2001) paper [27] stresses the Wisdom process and the Wisdom architecture. He did not indicate the structure of the requirement specification. Most UE approaches use informal methods, with a lot of freedom and little structure to aid the designers in gathering requirements. As a result, they start and end with informal sets of requirements [7]. Informal requirements are likely incomplete because they are not organized strictly and are not precise enough because they are often written in natural language. The incompleteness and the imprecision of requirement specifications cause difficulties in analyzing the requirements. There are great needs for a requirement metadata structure to collect requirements and for a series of processes which make full use of this structure and its content in the course of system development.

The four cooperative processes within PUF and the resulting usability requirements produced using the requirement structure provide sufficient usability-related information for system developers. In the following sections, we will describe the major processes of PUF and the PUF requirement structure in detail respectively.

2.2.1 Possibility Types within PUF

For better understanding of the PUF life cycle processes and the PUF requirement structure, PUF's basic elements are introduced first. In PUF, a possibility is defined as "an opportunity for improving the way we

serve one or more users" [7]. PUF identifies five types of possibilities, namely, tasks, users, content, tools and scenarios, which are five basic elements of PUF. They are interconnected as illustrated in Figure 2.4. For each type of possibility, PUF uses a common record format to gather its information and requirement metadata.

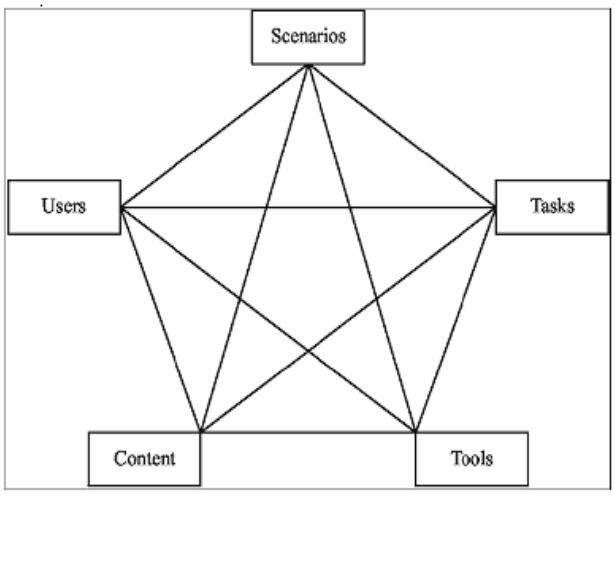

Figure 2.4: The Five foci of PUF specification

Users are a specific user or user group rather than a "generic" user. We analyze their characteristics to help us acknowledge their special needs and then meet them.

Tasks are "specific accomplishments fulfilled by an individual or more individuals" [6]. In the case of analyzing the tasks, we should answer questions such as what is currently done, what could be done, why it should be done, in order to find the most appropriate way to perform the task

Content is "the material processed by computer systems" [6]. Content provides data, information, or knowledge to users to perform their real tasks. Furthermore, how to use this content should be specified in its corresponding content record.

Tools are "any computerized or non-computerized procedures and/or artifacts", which help a person (end-user or developer) carry out some tasks [6].

Scenarios are "specific instantiations of specific combinations of {users, tasks, content, tools}" [6]. PUF is concerned with the specific interactions between them.

2.2.2 Major Processes within PUF

There are four major phases involved in the PUF life cycle, including possibilities analysis, requirements analysis, design and implementation of the releases. We should particularly point out that evaluation is an integral part of each life cycle process that cannot be omitted. Evaluation in each process performs particular tasks and offers qualitative and quantitative information to drive the next development process. PUF supports release-based development. It is practical and feasible to gradually develop a usable system from a simple one. Iteration is crucial in developing potential releases by applying the results of evaluation to the previous release. Therefore, the collaboration of iteration and evaluation could make it possible to yield a useable system.

- Possibilities analysis

  This life cycle process falls naturally into two steps. The first step can be named Possibilities Identification. Its objective is to capture and identify as many as possible of the tasks, user groups, content chunks and tool possibilities about the intended system for all potential releases.

  The next step is to analyze the possibility types identified. Relationships between possibility types and environmental factors having important effects on possibility types should be identified and the narrative recorded in this step. Then it is important to evaluate these possibilities for the next release.

  This phase ensures that usability engineers or software engineers are able to access much broader and more comprehensive context of use.

- Requirements analysis

  This phase involves identifying the new or changed usability-related information and requirements, designing the set of requirements, and evaluating their completeness and correctness using usability methods. Compared with the requirement analysis in software engineering methodologies, the processes in PUF give greater attention to usability-related requirements than to technical ones.

- Design

This phase involves choosing the significant requirements for each release and integrating them with existing requirements from previous versions of the tool being designed. A use model [29] composed of new PUF records containing the design specifications can be evaluated.

- Implementation of the releases

  This phase involves identifying the requirements of the selected design, creating the designed portions and integrating them into the existing system/prototype**.**

2.2.3 The PUF Requirement Structure

The PUF requirement structure is developed to optimize its usability both for developers and end users. The five possibility types are filled out within the PUF life cycle processes. The structure is comprised of four major sections, namely Identification Information, Linkage Information, Environment Information, and Detailed Requirements. Each section contains several questions which may or may not have sub questions.

The Identification Information section is used to identify a possibility type for a record, give a unique name and detailed description to the record. The identification information is recorded as soon as possibilities identification is done.

When the project moves on to the possibilities analysis phase, the linkage information will be added into the record. The questions in this section document other relative possibility types related to this possibility type when it is performed. Meanwhile some environmental information about this possibility type is filled out, for example, when and where this record will be used, how important it is and why it should be used.

The first three sections focus on general questions about five possibility types. Therefore, the question titles are all the same. As requirement analysis is carried out, specific usability-related information and requirements would be documented in the Detailed Requirement section. Since different possibility types have different requirements, the questions in this section are more specific than those in the first three sections. The detailed description of each PUF element is included in the appendix 6.

To better illustrate and describe the structure and the contents of a PUF record, a PUF TASK record is adopted as an example. This example is from e-Commerce application describing paying for an order for items already in a virtual shopping cart.

“Identification Information

Name: paying for ordered items using e-Commerce

Type: Task

Description: paying for an order of items already selected and currently in the customer’s virtual shopping cart.

Linkage Information

Who: customer

What: enquiring about the status of orders, ordering selected items.

How: part of an e-Commerce application

With which content: customer information, customer shopping cart contents, product information, order information.

Scenarios: new customer paying for ordered items; established customer paying for ordered items.

Environment Information

When: after ordering selected items

Where: via the Internet from at home; in an office; in an Internet café; in a store.

How much: 2 minutes per order times 1000 customer orders per day.

Why: to allow ordering from a wide range of locations is expected to increase sales by 2000 items per day.

Detailed Requirements

Task operations: customer confirms/modifies card information; customer confirms card information.

Requirement of users: must have credit card; must use supported Web browser; must understand English language.

Communications: this task involves formal interactive communications between a single user and the e-Commerce system.

Learning: the system must be self-descriptive and not require any training; the user may wish to access descriptive help while performing this task.

Error handling: the system should validate data at each step before proceeding and should help the user identify and make any required changes; the system should allow the user to edit all user input

fields prior to paying for the order; items in an established customer's virtual shopping cart should remain until the customer orders then or until they remain there for over one month.

Problem details: the system must be at least as usable as Amazon.com."[6]

The above introduction to PUF illustrates that the usability-related requirements within PUF are far beyond those specified in UML. Therefore, it is safe to say that combination of PUF specifications with those found in other methodologies could make it possible to produce a usable system. However, the PUF methodology is a complicated methodology. Many activities involved in these major processes require CASE support. If a large amount of resulting PUF records need translating into another language, we must turn to CASE support as well. MAT and UEDT depicted in section 1.2.1 will be created for meeting this requirement. Work is now underway to create these CASE tools to store PUF requirement specifications using XML.

2.3 Comparing UML & PUF

This section compares the requirement metadata in UML specification with those in PUF and identifies the possibilities of mapping relations.

Considering that UML and PUF have different purposes, other researchers have observed that they have a lot in common such as attributes (e.g., classes, user case, actor etc.). According to the conclusion of paper "Transforming Usability Engineering Requirements into Software Engineering Specifications" [6], each field in PUF records can map to some components in UML.

Figure 2.5 [6] demonstrates the relationship between PUF concepts and UML concepts. Both tasks and scenarios in PUF map to use cases in UML because of the correspondences between tasks and essential use cases and between scenarios and use case instances. Users in PUF map to actors in UML, while contents and tools in PUF correspond to attribute and operations in UML respectively.

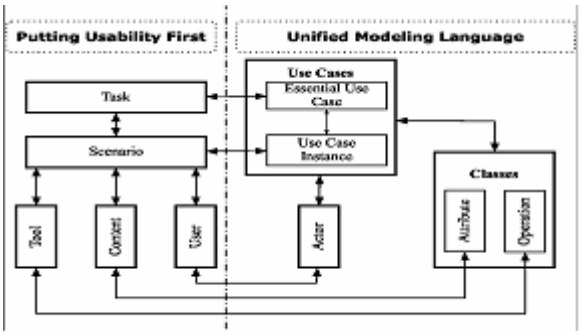

Figure 2.5: High level relationships between PUF and UML components [6]

The mappings identified in Figure 2.5 are built at the PUF record level. They provide a good starting point to do mapping. However, when it comes to the detailed requirement metadata mapping, there are some problems with these high level mappings.

1. The UML components identified in this mapping are just a small portion of UML notations. Use case and class diagrams do not capture all requirements. There is a chance that the PUF components can map to possible notations in other UML diagrams such as Interaction Diagram, Activity Diagram and State Diagrams
2. The mapping relations between the fields in the PUF possibility records and the notations in UML diagrams are not always one-to-one mappings. It could be one PUF field can map to many UML model components (called one-to-many mapping) or one UML model component can be mapped to many PUF fields (called many-to-one mapping). It is also very likely that a single PUF field ("source") can map to many UML model components ("targets") and a single model component ("target") can be mapped to many PUF fields ("sources"). This type of mapping relation can be called many-to-many mapping.

These problems point out that doing mapping between requirement metadata at the detailed level is complicated. Not only are many UML diagrams involved in the detailed requirement metadata mapping, but also there are a lot of mapping possibilities between PUF requirement metadata and UML requirement metadata. Generally speaking, mapping can be broken down into single requirement metadata mapping (one source and one target) and multiple requirement metadata mapping (more than one source or more than one target, or both). The one-to-many, many-to-one, and many-to-many mappings are three types of multiple requirement metadata mapping. Many-to-many mapping can be considered as multiple times of one-to-many mapping or many-to-one mapping. Both one-to-many mapping and many-to-one mapping can be regarded as doing single requirement metadata mapping multiple times. Therefore, all the types of multiple requirement metadata mapping can be broken into a set of various single requirement metadata mappings. This enables us to reduce the complexity of mapping relations and focus on readily identifiable mapping relations. Hence, as far as identification of the possible mapping relations is concerned, our solution focuses on single requirement metadata mapping.

For single requirement metadata mapping, I identified three possible mapping relations between a source requirement metadata and target metadata, which I defined as mapping relations. If there is no difference in the meanings of the two requirement metadata, the two requirement metadata are completely matched. The mapping relation is called **exact mapping relation.** If there is certain difference in the meanings of the PUF/UE requirement metadata from the UML/SE requirement metadata to which it is partially matched, the mapping relation is called **inclusion mapping relation.** If the meaning of the closest UML/SE XML tag does contain the meaning of the PUF/UE XML tag the mapping relation is called **non-existing mapping relation.** Figure 2.6 shows some mapping relations existing in some fields of a PUF user record and some UML model components. A double arrow line refers to an exact mapping relation. A single arrow line refers to an inclusion mapping relation. A dashed line refers to a non-existing mapping relation.

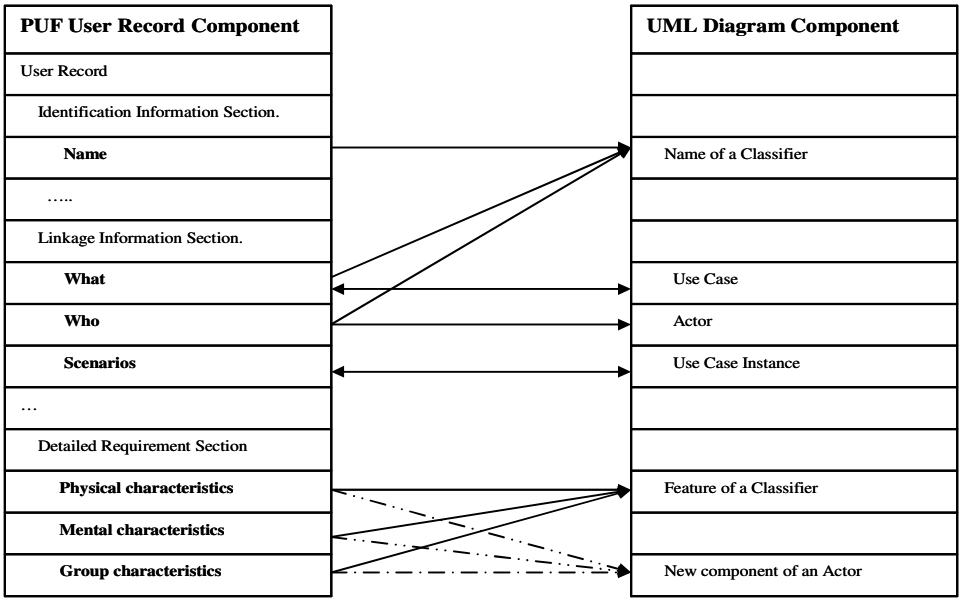

Figure 2.6: Detailed relationships between PUF fields and UML model components

Following is the detailed explanation of the relationships shown in Figure 2.6

- The *What* field in PUF user records identifies what tasks a user will do, especially focusing on how they are different from what other related user groups do. The *What* field represents the tasks which they need to accomplish. The *use case* model component in use case diagrams

describes an interaction between an actor and a system aimed at realizing a function. Thus, the *What* field and *use case* model component are identical. They can be mapped with an *exact mapping relation*.

- The *Who* field in PUF user records identifies who belongs to a given user group and can be described in terms of the characteristics that make an individual a member, especially focusing on how that user group is different from other user groups. The *actor* model component in use case diagrams could be a human being who interacts with a system or another system which interacts with the system. Therefore, the *Who* field and the *actor* model component are partially matched. They can be mapped with an *inclusion mapping relation*.
- The *Scenarios* field in PUF user records identifies scenarios which tie together sets of users, tasks, tools, and content chunks. Scenarios identify specific instantiations of specific contexts within which the user performs the tasks. The *UseCaseInstance* model component in use case diagrams represents an instance of a use case. Since PUF tasks are identical to UML use cases, the *Scenarios* field maps to the *UseCaseInstance* model component with an *exact mapping relation*.
- The *Name* field in PUF user records refers to a unique, meaningful identifier of a user group. Since actors can be regarded as a type of classifier, the *Name* field of a user record maps to the *Name* model component of a *classifier.* Besides actors, use cases are another type of classifier. So the *What* field maps to the *Name* model component of the *classifier.* It means the *Name* model component of the *classifier* contain other PUF fields which are not users' names. As a result, the mapping relation between the *Name* field of the user record and the *Name* model component of the *classifier* is an *inclusion mapping relation*.
- The *Physical characteristics, Mental characteristics, Group characteristics* fields in PUF user records identify specific design guidance related to the various user characteristics that may be unique to different user groups and suggest a method for evaluating the identified characteristics. They provide guidance and help to determine how to use this information throughout the development process. Thus, these three fields map to an *actor* model component with *non-existing mapping relations* as the actor does not contain similar requirement metadata. They

also map to the *Feature* model component of a *classifier* with an *inclusion mapping relation* as they can provide supplementary information for it.

This is an example of many-to-many mapping. The *Physical characteristic* PUF field in PUF user records can map to the *actor* model component and the *Feature* model component of a *classifier*. The *Feature* model component can be mapped to the *Physical characteristics, Mental characteristics* and *Group characteristics* fields in PUF user records.

2.5 XML

2.4.1 XML Basics

In order to combine UML and PUF requirement metadata, the same language should express both specifications of these methodologies. One possible method is to utilize XML. In this section a succinct account of XML is provided.

XML is a "W3C-recommended general-purpose specification for creating custom markup languages" [42]. XML is a way of describing data by combining markup with the data. The names of the markups describe the type or attribute of content they hold in addition to information on how to present the data. Therefore, the markups can be regarded as metadata.

XML's primary purpose is to transport and store many different kinds of data. By itself, "XML is syntax for the exchange of data and text-oriented documents" [30].

One significant use of XML is for metadata to be embedded into digital content files. It is used for reflecting the structure of particular classes of documents, such as books with chapters, user manuals, news feeds and articles incorporating explicit metadata in addition to the text. An XML document's markup structure can be defined by a schema language and validated against a definition in that language. The most widely used schema languages are the Document Type Definition (DTD) language and W3C XML Schema.

Since XML allows users to create their own tags and supports the integration of data, it is chosen as a common basis for encoding different methodology specifications.

In XML, **elements** are used to represent structured values. Element names with or without attributes are called **tags**. All **XML elements** begin with the element's start tag (formatted as <*tag*>) and close with the element's end tag (formatted as </*tag*>). The closing tag cannot be omitted.

XML syntax, elements and attributes are introduced through the following example

```
<book>
    <title> XML Tutorial </title>
    <prod id="33-657" media="paper"></prod>
    <chapter> XML Basic
        <para>XML Syntax</para>
        <para>XML Elements</para>
    </chapter>

    <chapter>XML Advanced
        <para>XML Namespaces </para>
        <para>XML CDATA</para>
    </chapter>
</book>
```

An element can have **element, attributes, text, or empty** as its content. An element can also have both **text** and other elements, called **mixed content**. In the example above, <book> has **element content**, because it includes the <title> element, the <prod> element, the <chapter> element etc. The <chapter> element has **mixed content**, while <para> has **simple content** (or **text content**) because it contains only text. <prod> has **empty content**, because it carries no information between its tags.

The power of XML is that it allows designers to create their own customized tags which actually define and describe the data that they contain. Therefore, tag names can be regarded as metadata. Furthermore, the data identified by a tag can be used for other tasks. For example, after interpreting XML tagx meaning, the tags interspersed with data can be extracted from one source, then combined with another XML document, with the combination finally output. To make labeled information reusable, XML tag name should be meaningful.

XML has very simple but well-defined syntax rules. XML has limited ability to provide what is known as “semantic transparency” [31] which means information objects in the problem domain do not correspond transparently ("one-to-one") to objects in the user's conceptual model.

2.4.2 Related XML Concepts

The XML Metadata Interchange (XMI) is regarded as a successful effort for exchanging information between UML models via XML. XMI bridges the gap between a UML model and XML by expressing UML objects using XML syntax. XMI also has capabilities to generate XML schemas from models,

specify how to tailor the XML document XMI creates, and tailor the schema representation for each UML construct in a UML model [33].

An XML schema can be used to validate a type of XML document by imposing a set of rules and constraints on their structure and content. An XML schema shows a high-level abstract view of XML documents of that type. Document Type Definition DTD [10] is one of the XML schema languages. The syntax of DTD defines the structure of a XML document and specifies the attributes and elements that make up this structure. Furthermore, DTD imposes constraints on the attributes and elements in the XML file so that the DTD can be used to validate the XML file.

2.5 XML as A Basis for UML

Mapping is facilitated by a well-structured XMI document representing UML. There are a number of versions of UML DTD available for describing UML diagrams. UML DTD 2 and UML DTD 1.4 show UML metadata structure via indention. UML requirement metadata is presented as the values of the *name* attribute of a XML element in these two versions. Thus, their tag names are not explicit about semantics and syntax of the tag.

UML DTD (version 1.3) [34] is used to define and specify the required UML 1.3 elements in its validated XMI (version 1.3) files describing models in UML 1.3. All the XMI file are consistent with UML1.3 semantics [35]. UML DTD 1.3 also lays down the rule of the presentation of the XML tags in its validated XMI (version 1.3) file. It is stipulated that the XML tags should be represented in a fully specified style so that the XML tag names explicitly specify the syntax of the tags. The XMI files can be generated by a UML modeling tool because recently most of the UML modeling tools support UML 1.3. Appendix 1 [34] shows an example XMI file which represents a UML class diagram.

Since XML tag names can be regarded as UML requirement metadata, UML requirement metadata are represented in the fully specified style as follows:

Table 2.1 UML requirement metadata structure

PackageName	**SubPackageName**	**PackageConstructName**	**ClassConstructName**
Foundation	Data_Types	Multiplicity	range
	Core	Operation	method
	Core	Attribute	associationEnd

	Extension_Mechanisms	TaggedValue	stereotype
	Extension_Mechanisms	Stereotype	baseClass
Behavioral_Elements	Common_Behavior	Link	association
	Common_Behavior	Action	stimulus
	Use_Cases	AssociationEnd	
	Use_Cases	Actor	
	Use_Cases	Use Case	
	State_Machines	Event	transition
	State_Machines	State	internalTransition
	Collaborations	Interaction	
	Collaborations	Message	Interaction
	Activity_Graphs	ActivityGraph	
	Activity_Graphs	SubactivityState	isDynamic

<PackageName.SubPackageName.PackageConstructName.ClassConstructName >

The four parts in this tag format are called **tag constructs**. The values of the four tag constructs represent the corresponding elements in the UML1.3 models. Table 2.1 lists some values of the tag constructs.

The values of **PackageName** seen in table 2.1 represent the packages in UML. They are: *Foundation*, *Behavioral_Elements*, *Model_Management* Each package has a number of sub packages. The value of SubPackageName in the *Foundation* package can be *Data_Types, Core* and *Extension_Mechanisms.* The value of SubPackageName in the *Behavioral_Elements* package can be *Common_Behavior*, *Use_Cases*, State*_Machines*, *Collaborations,* and *Activity_Graphs*.

The values of **PackageConstructName** show the constructs which are organized into a sub package. Examples of the value of PackageConstructName are *AssociationEnd*, *Actor*, and *Use Case* in the *Use_Cases* sub package.

The construct in a sub package is represented by a class. The values of **ClassConstructName** falls into two categories attribute and reference. The examples of the value of the attribute type **ClassConstructName** could be *condition, multiplicity*, and *aggregation*. As for the value of the reference type **ClassConstructName** examples are *extensionPoint*, *association*, and *qualifier.* Appendix 9 lists all the XML tags for the UML models used in this thesis and their meanings.

As far as UML mapping data is concerned, the XMI file possesses a large amount of XML tags describing UML semantics. To simplify the mapping process, the focus of the modified XMI file is on the following related packages: Core, Common Behavior, Use Cases, State Machines, Collaborations, and Activity Graphs.

2.6 XML as A Basis for PUF

MAT is currently being developed to create and store PUF specifications and/or specifications of other Usability Engineering (UE) methodologies and methods using XML. Therefore, the integration of PUF specifications with UML specifications becomes a process of XML translation from XML-based specifications for one methodology, such as PUF and/or Common Accessibility Profiles (CAP), to XMI documents. Specifically, it is a process of mapping the components in a source XML file into a target XMI file.

A PUF XML record is composed of four sections which contain a set of questions, each of which may or may not possess several sub questions. The current MAT uses number and indention to maintain the structure of PUF requirements. All the PUF requirement "building block" components (*record*, *section, question,* and *sub-question*) are numbered. Their comprehensible names are shown in the XML tags as text content.

For PUF requirements at the higher levels which are *record* and *section*, the corresponding XML tags include a *number* and a few "building block" components belonging to the 'parent' record or section. The following is an example of expressing the **Identification Information** section in a **User Record**.

```
< User Record >
  <number>1</number>
  <Identification_Information>
    <number>1.1</number>
    <question>Name
    </question>…
  </Identification_Information>
  <Linkage_Information>…
  </Linkage_Information>
```

< /User Record >

For the “building block” components at the detailed lower level, which are *question* and *sub-question*, each of them includes an XML element for numbering and one for its specific question composed by a *number* XML element, a *questionText* one and an *answer* one.

Here is the example of the **what** question.

```
<what>
   <number>2.2</number>
   <question>
      <number>2.2.1</number>
      <questionText>What does this task consist of?</questionText>
      <answer>Answer data goes here</answer>
      <answer>for as many answers as there are.</answer>
   </question>
</what>
```

The current MAT also generates the corresponding DTD to validate all the XML files.

2.7 XML as A Basis for Mapping PUF to UML

After specifying the relevant technologies to the major elements in Figure 2.1, Figure 2.7 represents those elements in a more precise way.

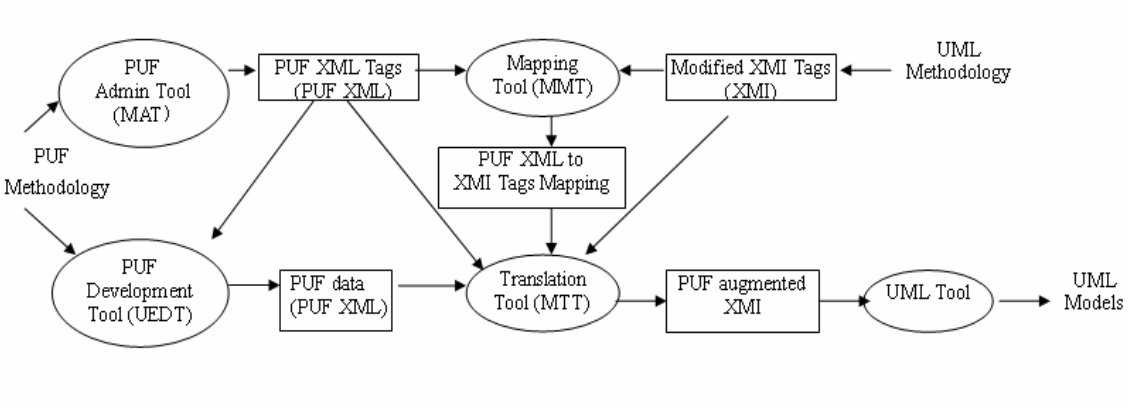

Figure 2.7: Accurate details of UE project

The discussion to this point demonstrates that both PUF requirement metadata and UML metadata can be expressed in well-formed XML files. Mapping between PUF requirement metadata and UML metadata involves identifying one or more mappings between two heterogeneous XML files. Based on the comparison of two heterogeneous XML files, the causes of heterogeneity between XML file elements can be divided into two groups based on: syntax differences and semantic differences. Since semantics is encoded in XML tags, we are actually doing XML tag mapping. To do so, mapping relations between the source and target XML tags can be identified. The mapping relations established by the Mapping Tool will drive and guide the translation process which will be done by the Translation Tool because once the location(s) of a PUF XML tag in an XMI file is determined, its PUF project requirements will move to those locations along with the PUF XML tag. As a result, PUF requirement specifications in XML format will enhance XML-based UML specifications.

CHAPTER 3

REQUIREMENTS FOR MAPPING PUF TO UML

This chapter introduces a set of requirements of performing mapping between XML-based PUF and UML requirement metadata. The requirements are composed of the following four parts:

- what tasks are required in the process of mapping,
- what are the requirements on people who perform mapping requirement metadata,
- what are the requirements on the content used for mapping,
- what are the requirements on the tool which are used for facilitating mapping requirement metadata.

3.1 Mapping Tasks

After comparing the software development processes with UML and the life cycle processes within PUF, we reach a conclusion that UE phases have corresponding ones within the SE development process. It means that most usability engineering tasks can be carried out in parallel with software development tasks. Moreover, the artifacts produced by UE tasks can be added or transformed into those of SE tasks as supplementary information. Software engineering tasks and usability engineering tasks are outside the scope of this thesis. We are concerned with how to integrate usability engineers' requirement metadata and designs with those of software engineers.

The requirement metadata of software engineering and usability engineering can be presented in XML format. Mapping is performed on the XML tags in the two heterogeneous XML files. To integrate one PUF/UE XML tag into one UML/SE XML tag, there are a number of tasks that must be accomplished. Following is the discussion of the tasks required in the process of single requirement metadata mapping. Section 2.3 explains that decomposing multiple tag mapping into single tag mapping is sufficient:

1. Determine a PUF/UE XML tag in the PUF/UE XML file as a source tag. This tag represents the PUF/UE requirement metadata that must be integrated into UML/SE requirement metadata.
2. Determine a UML/SE XML tag in the UML/SE XML file as a target XML tag. This tag represents the location to which the PUF/UE XML tag is mapped.

3. Establish mapping rules. The mapping rules define the possible mapping relation between the source and target tags. After comparison of the source and target tags, the distinctions between the meanings of the two XML tags should be identified. These semantic differences are the basis on which the mapping rules are set up. The types of the identified mapping relations should be named. The mapping rules guide in conducting the operations for integrating the PUF/UE XML tag into the UML/SE XML tag. These operations are called **transformation operations**. These transformation operations should be in accordance with the mapping relations discussed in section 2.3.
    - If the mapping has *exact mapping relation*, its corresponding transformation operation is to leave the UML/SE XML tag unchanged so that it can drive the UML CASE tool to generate UML diagrams;
    - If the mapping has *inclusion mapping relation*, the PUF/UE XML tag should go to the same location as the UML/SE XML tag and become its peer element;
    - If the mapping has *non-existing mapping relation*, its corresponding transformation operation is to add a new XML tag under the UML/SE XML tag and become its new constituent part;
4. Determine the mapping relation between the source and target tags. The semantic distinctions between the source and target XML tags should be analyzed first. Then a mapping relation should be chosen to describe their semantic relationship.
5. Generate a report of the identified mappings and a XMI file which is enhanced by adding the PUF/UE XML tag to it according to the mapping relation and the corresponding transformation operation. The report and the enhanced XMI file should be allowed to be saved or printed.

3.2 Practitioners of Mapping

Mapping between UML and PUF concepts requires practitioners to have a solid understanding of two methodologies' requirement metadata. There is some preparatory work needed to be done before mapping.

First, the meaning, purpose and use of each PUF/UE XML tag and each UML/SE XML tag should be grasped. Since it is very likely that the XML tags with the same name convey different information in

different methodologies, the meaning of an XML tag should be thoroughly understood by combining its syntactic information.

Second, the syntactic and semantic differences between a PUF/UE XML tag and a UML/SE XML tag should be identified and analyzed. One way of doing this is to conduct high level and low level comparisons. This starts with high level comparisons shown in Figure 2.5. After high level comparisons are made, the focus shifts to low level comparisons. For instance, PUF questions or PUF sub questions compare with UML package constructs or UML class constructs. Furthermore, the comparison should not be limited within one UML package. The same or similar information could be located in different locations of a XML file because some UML class construct in one sub package may relate to those in other sub packages.

Third, the mapping relation between a PUF/UE XML tag and a UML/SE XML tag should be identified. The degree of the distinction in the meaning of the two XML tags should be determined if there is any difference. The following questions should be thought about:

-- Are the two requirement metadata identical?

-- If not, do they have some overlap?

-- If so, what is the degree of overlapping?

3.3 Contents involved in mapping

The contents for mapping are a set of PUF XML tags and a set of XMI tags. The PUF XML tags represent the PUF requirement metadata in the PUF task/scenario, user, content and tool records. A task record specifies what needs to be done while a scenario record describes the execution of a task in a specific context. Project requirements recorded in the scenario record are more specific than those ones in the task record. When it comes to requirement metadata, these two types of records are similar enough that we can discuss task records without loss of generality. Therefore, PUF task, user, content and tool records should be taken into consideration in the process of the requirement metadata mapping.

As mentioned in the mapping tasks, identifying the distinction in semantics between a PUF/UE XML tag and a UML/SE XML tag should be combined with a good understanding of their syntax. Therefore, XML tag names should be represented in a fully specified XML tag format which explicitly specifies

requirement metadata's syntax and semantics so that XML tags could be easily understood and greatly facilitate mapping.

3.4 Mapping Tool

Section 2.3 discusses about how complicated detailed requirement metadata mapping is. A tool is needed to accomplish the mapping tasks elaborated in section 3.1. It is called the Mapping Tool. The Mapping Tool aims at automating the process of mapping.

The inputs to the Mapping Tool are an XMI file that describes UML models in XML format and an XML file that specifies PUF requirement metadata. The Mapping Tool should apply a set of mapping rules to the PUF XML tags in the PUF XML file so that they can be automatically mapped to and integrated into the XMI tags in the XMI file,

To serve this purpose, the Mapping Tool should meet the followings requirements:

- After MAT inputs the PUF XML file, the Mapping Tool should make PUF XML tags available so that they can be retrieved. To identify a source tag, each tag construct of a PUF XML tag in the XML file should be selected sequentially. The person performing the mapping chooses a record, then chooses a section in the record, and next chooses a question in the section and at last a sub question in the row. The Null value should be allowed to be held by the section, question and sub question levels. The Mapping Tool also should provide semantic information about each tag to help the person performing the mapping understand it so that he or she can have ideas about the possible locations in the target XMI document.
- After the UML CASE tool inputs the XMI file, the Mapping Tool should make XMI tags available so that they can be retrieved. To identify a target tag, each tag construct of an XMI tag in the XMI file should be selected sequentially. The person performing the mapping chooses a UML package, then chooses a sub package in the package, next chooses a package construct in the sub package and at lastly chooses a class construct in the package construct. The Null value should be allowed to be held by the sub package, the package construct, and the class construct levels. Similarly, semantic information about the tags should be given.
- After identifying the source and the target XML tags, the Mapping Tool should make the mapping relations of PUF and UML requirement metadata available so that they can be retrieved.

The person doing the mapping should choose an option to describe the mapping relation between the source and the target XML tags. The options on mapping relations are *exact mapping relation*, *inclusion mapping relation* and *non-existing mapping relation.*

- After the mapping is created, the mapping information including the PUF/UE XML tag and the UML/SE XML tag along with their mapping relation should be shown in a mapping list by the Mapping Tool. The person doing the mapping can save these mappings in the list or print them as a report. Furthermore, according to their mapping relation, the Mapping Tool should carry out the corresponding transformation operations to generate an enhanced XMI file.
  - If the PUF/UE XML tag directly maps to the UML/SE XML tag (*exact mapping relation*), the UML/SE XML tag is unchanged. The PUF/UE XML tag information is kept in the mapping list which will be input into the Translation Tool after mapping;
  - If the PUF/UE XML tag partially maps to the UML/SE XML tag (*inclusion mapping relation*), the PUF/UE XML tag is added to the same location in the UML/SE XML specification structure as the UML/SE XML tag so that two of them are peers. The added PUF/UE XML tag is composed of the higher level UML/SE XML tag constructs and the lowest level PUF/UE XML tag construct;
  - If the meaning of the UML/SE XML tag does not contain the meaning of the PUF/UE XML tag (*non-existing mapping relation*), a new XML tag is added under the UML/SE XML tag as a new sub element. The added PUF/UE XML tag is composed of the UML/SE XML tag and the lowest level PUF/UE XML tag construct. If the PUF/UE XML tag has sub tags, they will be added into the UML/SE XML structure as well.

Table 3.1 demonstrates the above transformation operations using the examples from section 2.3.

Table 3.1 Examples of transformation operations

PUF XML tag	**UML XML tag**	**Mapping Relation**	**New XML tag in UML/SE XML structure**
<UserRec.LnkInfoSection.Scenarios>	<Behavioral_Elements.Use_Cases.UseCaseInstance>	exact relation	Null
<UserRec.LnkInfoSection.Who>	<Foundaation.Core.Classifier.Name>	inclusion relation	<Foundaation.Core.Classifier.**Who**>
<UserRec.DetReSection.PhyCharCap>	<Foundaation.Core.Classifier.Feature>	non-existing relation,	<Foundaation.Core.Classifier.Feature.**PhyCharCap** >

Table 3.1 shows that the enhanced UML/SE XML structure contains the PUF requirement metadata if it has *inclusion mapping relation* or *non-existing mapping relation*. It is easy for the Transformation Tool to find where to put PUF project requirements in UML/SE XML structure. As for an *exact mapping relation*, the Transformation Tool relies on the mapping list to find appropriate UML locations because it keeps track of matched UML requirement metadata for all PUF requirement metadata. Therefore, the enhanced UML/SE XML structure and the mapping list combined together can guide the Transformation Tool to put PUF project requirements into appropriate locations in the UML/SE XML structure.

- The Mapping Tool should provide useful help information to make sure that the mapping tasks are performed smoothly.

CHAPTER 4

TOOL DESIGN

This chapter commences with a discussion about preparing the appropriate inputs to the Mapping Tool. Then it offers a detailed illustration of the Mapping Tool design from four aspects, namely Database design, Function design, Screen design, and Program design. The illustration starts with the database design which plays an important role in making the system work efficiently and effectively. Thereafter, it elaborates the functional design process including what functions were implemented in the Mapping Tool and how these functions were realized. The screen design is also covered by this chapter showing the layout of the Mapping Tool. The Program design specifies what technologies were used to implement the Mapping Tool.

4.1 Inputs Preparation

Section 3.3 emphasizes the importance of the fully specified tag format with regard to the functionality and usability of the Mapping Tool. The fully specified tag format allows methodology semantics to be encoded in XML syntax so that a tag name can explicitly specify its semantics and syntax.

However, the available XML tags for PUF requirement metadata and UML one are not represented in this fully specified tag format. As mentioned in section 2.6, the existing PUF XML tags provided by MAT do not comply with this fully specified style. As far as UML requirement specifications are concerned, the available XMI files are used for some specific applications. In order to make input data fully qualified. There is some preparation work needed to be done before design. This section explains why I chose the fully specified tag format for UML and PUF requirement metadata and how the ideal XML tags were obtained. These two issues are the relatively difficult problems I encountered in the process of developing the Mapping Tool.

With regard to XMI files, I did research on UML DTD 2 and UML DTD 1.4 and created databases for them before UML DTD 1.3 were chosen. From the mapping perspective, these two versions have two major disadvantages. First, it is difficult to capture requirement metadata's syntactic information from the XML tag names because UML DTD 2 and UML DTD 1.4 use indention to show the UML metadata structure. Without its location information, the meaning of a XMI tag could be variable because it is likely

that the same UML component has different meanings in different UML packages. Take *Interaction* in UML for instance. If it is a class construct in the *Message* package construct, it refers to the set of interactions that are defined during the execution of the current Message. *Interaction* could also be a package construct, which means an interaction specifies the messages sent between instances performing a specific task. Each interaction is defined in the context of collaboration. Without syntactic information, each XMI tag name is not unique and its meaning could be ambiguous. As a result, confusion may arise and consequently it will be matched wrongly. Second, it requires efforts to extract requirement metadata from the value of the *name* attribute of a XML element. Considering the above handicaps, UML DTD 2 and UML DTD 1.4 were abandoned.

The XMI tag names defined in the UML DTD 1.3 are fully qualified by explicitly specifying their semantics and syntax. In this way, each XML tag is distinguishable from other tags. That is why UML DTD 1.3 is used in this thesis as a source providing ideal XMI tags. Considering the situation that there is no XMI file recording the complete UML requirement metadata, a XMI file was generated based on the name, structure and rules specified in UML DTD 1.3. Due to the large amount of the XMI tags, it is time-consuming to include all of the XMI tags as inputs. The focus is laid on all the necessary sub elements from the same package as the starting point for mapping. The optional ones from other packages will be taken into consideration in future. The structure of the XMI file is simplified. The resultant XML tags conform to the following format. Four tag constructs in a XMI tag show the hierarchical structure of UML requirement specification.

<PackageName.SubPackageName.PackageConstructName.ClassConstructName >

Since the XMI tags in the XMI (version 1.3) file are used for delineating UML 1.3 models, the mappings produced in this thesis might not apply to UML 2. The general approach could be used for such mappings, if a more complicated version of the mapping tool was developed to deal with the more complex structure of the corresponding XMI.

As for the PUF XML file, after extensive investigation of the structure of the PUF XML file provided by the current version of MAT, it was abandoned because of considerations of mapping and usability. There are some problems with MAT's data structure. First of all, the original structure specifies PUF requirement metadata's syntax implicitly and semantics explicitly. Secondly, the tags are poorly

formed. Because it uses indentation to represent each PUF metadata's level, it makes the whole structure complicated. Thirdly, the PUF elements which need mapping are dispersed in different locations. Some parts are PUF XML tag names, others are text content. It is inefficient to identify mapping components from different locations and extract them. It is expected that MAT will be revised in the near future (based on inputs from this thesis) to produce tags more suitable for mapping.

Because of the above reasons, a XML file for PUF requirement metadata was created from scratch. I looked for a better way to represent PUF requirement metadata. The XML tags for PUF metadata were generated based on its location in the PUF structure. The final PUF XML tag format I designed is based on the idea of combining PUF semantics and XML syntax in XML tags to make them distinguishable between each other.

Since the PUF requirement metadata structure has four levels, the PUF XML tags are composed of four tag constructs. Each tag construct represents a syntactic level in the PUF requirement metadata structure. The tag constructs in each XML tag is given meaningful and understandable names in accordance with their names in the PUF requirement structure. Following is the tag format. Appendix 8 shows a complete list of PUF XML tags and their meanings:

< RecordName.SectionName.QuestionName.SubQuestionName>

The XML tag name conveys the information about the PUF metadata's syntax and semantics explicitly. This tag format is generic and flexible. It not only can be used for the PUF methodology, but also other UE methodologies or methods as long as they have a hierarchical requirement structure. The most important benefit is that it facilitates mapping. The mapping information can be easily obtained from the UE methodology requirement structure, which can be done either mechanically or manually. The tag constructs in <RecordName.SectionName.QuestionName.SubQuestionName> correspond to the elements in the PUF requirement structure. Same as the UML tag constructs, the values of the four tag constructs represent the corresponding elements in the UML XML structure. The combination of the four tag constructs' names shows the level of the PUF structure where a tag is located. The PUF XML tags comply with the tag format employed by the XMI tags. Doing so can make mapping easy. Since all the tag names are fully specified and meaningful, the original locations and the meaning of the PUF XML tags can still be captured via their tag names when these tags are translated.

The fully specified tag format can make each PUF XML tag and each XMI tag unique and understandable. It also facilitates mapping because both PUF and UML metadata are specified in a similar tag format that is easy to see without having to put together tags from various hierarchical levels. I chose to use the fully specified tag format is for ease of use in both the Mapping Tool and its outputs. I did a lot of work to put the existing PUF and UML metadata in this format manually. Alternately, a fully specified tag format could be dynamically generated from a hierarchical structured file of XML tags.

4.2 Database Design

From the UE project point of view, the Mapping Tool should generate a mapping specification and an enhanced XMI file based on the inputs from MAT and from UML tool. Figure 4.1 depicts the data flows between the Mapping Tool and the relevant tools in the UE project. MAT provides the PUF XML file database to the Mapping Tool as an input. UML 1.3 DTD provides the XMI File database as another input. Given the PUF XML and the XMI files, the Mapping Tool should generate outputs. One of them, the mappings between the PUF XML tags and the XMI tags, is stored in the Mapping database. Another is an enhanced XMI file. The Mapping Tool should integrate the PUF XML tags into the XMI tags according to the mapping relation. These two outputs of the Mapping Tool serve the Translation tool as inputs.

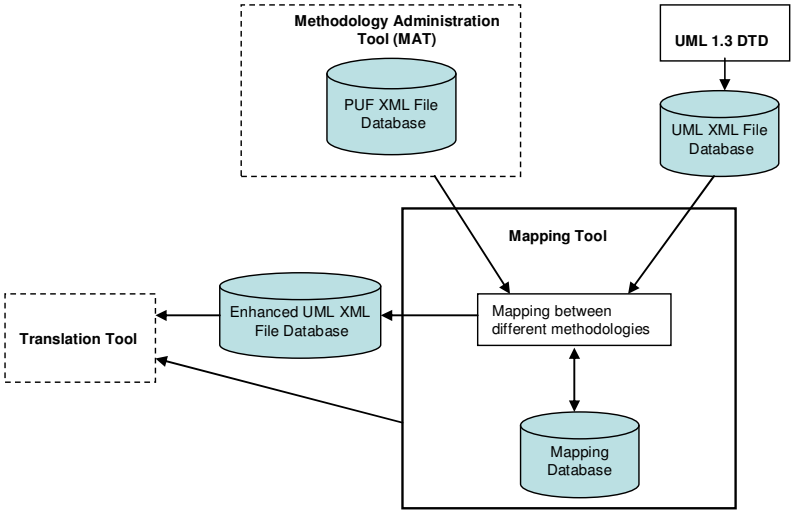

Figure 4.1: Part of UE project diagram

The relationships among the tools in the UE project provide a good point to start the database design. Figure 4.1 shows that the Mapping Tool database requires four primary components. They are a PUF XML File, a XMI File, a Mapping and an Enhanced XMI File. I designed the database for the Mapping Tool with the idea that each crucial component requires a corresponding database to make a component's

function distinguishable. Figure 4.2 shows the database design. The PUF XML File database and the XMI File database can be regarded as entity databases. An entity database is used to store semantic and syntactic information about a methodology. These two databases provide inputs to the Mapping Tool. The Mapping database can be considered as an association database which relates to two entity databases. Figure 4.2 also illustrates the relations among the required components. The Mapping database requires the data from the PUF XML File and the XMI File databases. The Enhanced XMI File database is created based on the integration of the data in the Mapping database and the XMI File database.

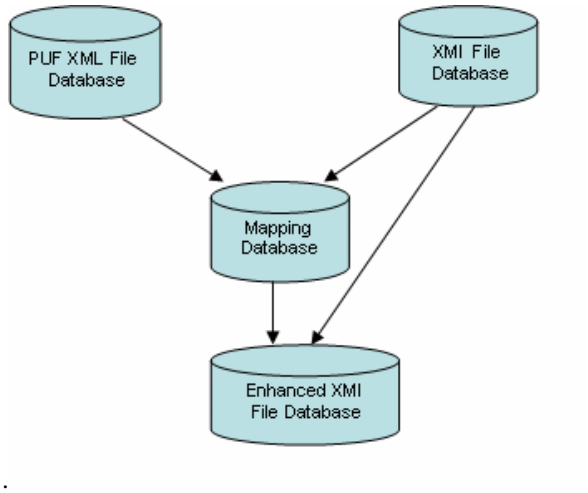

Figure 4.2: Architecture of the Mapping Tool database

4.2.1 Database Requirements

To design a usable database, the requirements placed on the Mapping database and other databases involved in the process of mapping are collected. This section illustrates these requirements for each database respectively.

1. PUF XML File

PUF XML elements information provided by the PUF XML file database must be available. To do so, the following requirements on PUF data should be fulfilled:

- PUF element name

  The database should be able to present meaningful PUF elements. Specifically, PUF XML element names should be fully specified. The names should contain semantic and syntactic information regarding PUF XML elements.

- Order of elements

  In order to make the database logical, PUF elements should be presented in a logical order. Since the structure of PUF records are generated based on a person's cognitive process of acquainting themselves with something new from the generic to the specific, the sequence of elements retained in the database should be consistent with the one in the PUF records.

- Element description

  The detailed description of PUF XML elements is needed so that they can be comprehended easily.

2. XMI File

XMI tags information which is provided by the XMI file must be available. The following requirements were found to be needed:

- UML element name

  The database should be able to present meaningful UML elements. Specifically, XMI element names should be fully specified so that XMI elements' meanings are understandable along with their structural information. XMI tags are not required in a particular sequence.

- Element description

  The detailed description of XMI elements includes their meanings and how to use them.

3. Mapping

Mapping information including PUF XML elements, XMI elements and their mapping relation must be available. To make this information comprehensible, the following requirements need to be met:

- Mapping information

  The mapping information including PUF XML element names, XMI element names, and mapping relation should be meaningful. All the element names should be fully specified by containing information about their meanings and their locations where they fit in the methodology structure. Mapping relation names should readily understandable.

- Valid PUF XML elements and XMI elements

  Since the information about PUF XML elements and XMI elements in the Mapping database each comes from their respective entity databases, any PUF XML element or XMI element in the

Mapping database should have a correspondence existing in the PUF XML File database or the XMI File database. It means the Mapping database cannot have a PUF XML element or XMI element which does not exist in the PUF XML File database or the XMI File database.

- Deletion requirements

  Deletion of a mapping should not cause the related element tags to be deleted from their corresponding entity databases.

4. Enhanced XMI File

The enhanced XMI file must be created. An Enhanced XMI file requires the following components:

- UML element name

  The database should be able to present UML elements in a XMI file including their semantic and syntactic information no matter whether they have mappings or not. They should be able to accommodate a PUF XML tag or a set of PUF XML tags when one or more PUF element integrates into the XMI file.

- Element description

  It describes the extension of a UML element. It should provide detailed description of the new added PUF XML element. The description should make sense so that the enhanced XMI tag can be understood.

4.2.2 PUF XML Tags Table

The **PUF XML tags** table is an entity table used to accommodate the semantic and syntactic information of PUF XML tags at detailed levels. It provides PUF XML tags as one input to the Mapping database. Since PUF XML tags are fully qualified, the **PUF XML tags** table was built in accordance with their tag format. As mentioned before, the tag constructs of a PUF XML tag show the hierarchical structure of the PUF requirements. In the context of a relational database table, the columns provide the structure which determines how database records are composed. The tag constructs can be regarded as the columns in this entity database which describe the semantics of the PUF requirements. Thus, the **PUF XML tags** table consists of:

- PUF element name
  - PUF_Record_Name: a name of a record type. The name is consistent with rNO.

- PUF_Section_Name: a name of a section type. The name is consistent with sNO.
- PUF_Question_Name: a name of a question type. The name is consistent with qNO.
- PUF_subQuestion_Name: a name of a sub-question. The name is consistent with sqNO.

- Order of elements:
  - rNo: a number represents a record type.
  - sNo: a number represents a section type.
  - qNo: a number represents a question type.
  - sqNo: a number represents a sub-question.
- Description: A long text explanation of the purposes of a PUF XML tag

As mentioned in Section 2.6, each PUF XML tag has a unique tag name which is composed of the *RecordName*, *SectionName*, *QuestionName*, *SubQuestionName* tag construct. Each tag construct name indicates its meaning and the level in the hierarchical PUF structure at which it is located. Take a *How* question in the *Link Information* section within a *Task* record for instance. The tag for this PUF element should be <*TaskRec.LnkInfoSection.How*>. Thus, the entire tag name identifies this PUF element's location in the PUF requirement structure. It also indicates the PUF element's semantics how the task is accomplished in general with a focus on the difference between this task and other related tasks. In order to make full use of the advantages of this tag format, each tag construct is mapped to a semantic column in the database. The *PUF_Record_Name, PUF_Section_Name, PUF_Question_Name, PUF_subQuestion_Name* columns combined together form fully specified PUF XML tag names. It ensures that the syntactic and semantic information of PUF XML tags are maintained in the database as well. Since a set of values for the four semantic columns can uniquely identify a PUF XML tag, they are considered to form a composite primary key for this table.

Since the specific individual PUF XML tags locate either at the *PUF_Question_Name* or the *PUF_subQuestion_Name* level, it is likely that the *PUF_subQuestion_Name* column have null values. To avoid violating the entity integrity rule that there should not be a null value for any column of a primary key, the columns containing null values are assigned '0' when doing one-to-one mapping. As a result, the primary key for this entity table is logical and accurate. Continuing to use <*TaskRec.LnkInfoSection.How*> as an example, instead of setting a null value to the

PUF_subQuestion_Name column, the semantic column values for the Task Record tag become 'TaskRec', 'LnkInfoSection', 'How', '0'.

The data types for these four columns are VARCHAR (80) because the number of character that these four tag construct columns can hold is variable. The maximum length of a value is 80.

The order of PUF elements is important to the PUF requirement metadata because the metadata in the requirement structure is built during the course of development. In SQL, the sort sequence depends on the data type of the sorted field. The structure of PUF elements cannot be maintained by using the character data type because they will be sorted in an alphabetical order. There is no clause in SQL for saving records in a user-defined order. I decided to use an existing SQL clause to maintain the order of the PUF requirement metadata. The solution I presented is to assign a set of *rNo, sNo, qNo, sqNo* to each PUF element to indicate its location in the PUF structure. For example, *rNo* for a Task record is 1, *sNo* for the Environment Information section is 4 and *qNo* for the Why question is 4. Thus the syntactic numbers for this tag should be 1.4.4.0 (the syntactic number 0 means there is no sub-question in the Why question).

By replacing each PUF element by a set of numbers, the PUF hierarchical structure can be represented by a large amount of sets of numbers. Therefore, sorting PUF elements in a numeric order is able to retain the PUF structure. The numbers which the syntactic columns can hold are no more than 20 and they are all integers. TINYINT (2) is the proper type for these syntactic columns. It means the maximum display width is 2 and the range of the values is 0 to 255.

The *Description* column provides a long text explanation of PUF XML tag's meaning and purposes. The data type for the Description column is VARCHAR (800).

4.2.3 XMI Tags Table

The **XMI tags** table is an entity table that accommodates the semantic and syntactic information of the XMI tags which is provided by UML 1.3 DTD. The format of the XMI tag names are in the same as that of PUF XML tags. Each tag construct represents a level in UML metamodel structure. For instance, <*Foundation.Core.Classifier.feature*> represents the *feature* class construct in the *Classifier* package construct which is in the *Core* sub package within the *Foundation* package. It specifies a list of features, like Attribute, Operation, Method, owned by the Classifier. The same is true with the **PUF XML tags**

table, each tag construct in a XMI tag corresponds to a semantic column in the **XMI tags** table. Thus, the **XMI tags** table contains the following columns:

- UML element name:
  - UML_Package_Name: a name of a package type.
  - UML_subPackage_Name: a name of a sub-package type.
  - UML_packageConstruct_Name: a name of a package construct type.
  - UML_classConstruct_Name: a name of a class construct.
- Element description: A long text explanation of the purposes of a XMI tag

The **XMI tags** table should store the detailed location in the UML structure where abstract and specific PUF XML tags can map to so that all the mapping goes to the right place. Same as the **PUF XML tags** table, if the semantic columns hold null values, they will be assigned '0' in the columns. Hence these four columns combined together also can distinguish the UML elements which makes them serve as a composite primary key in this relation. VARCHAR (80) is the proper data type for the semantic columns while the Description column needs a variable string data type with a maximum length of 800.

4.2.4 Mapping Table

The **Mapping** table can be regarded as the table that relates the **PUF XML tags** table and **XMI tags** table entities by accommodating their entity information and relation information. The **Mapping** table includes:

- PUF element name
  - PUF_Record_Name: a name of a record type. Not null.
  - PUF_Section_Name: a name of a section type. Not null.
  - PUF_Question_Name: a name of a question type. Not null.
  - PUF_subQuestion_Name: a name sub question. Not null.
- UML element name:
  - UML_Package_Name: a name of a package type. Not null.
  - UML_subPackage_Name: a name of a sub-package type. Not null.
  - UML_packageConstruct_Name: a name of a package construct type. Not null.
  - UML_classConstruct_Name: a name of a class construct. Not null.

- Mapping relation:
  - A name represents the relation between a PUF XML tag and a XMI tag. There are three possible values.

In order to make mapping information meaningful, it is better that tag names are able to convey structural information along with their meanings. From the previous discussion, the tag format used for both PUF elements and UML elements is able to satisfy this requirement metadata. Therefore, all the tag constructs of PUF XML tags and XMI tags are mapped to the eight semantic columns in the **Mapping** table. Since PUF XML tag names and XMI tag names hold their syntactic information in their semantics, it is easy to find out the locations where the PUF XML tag will fit in UML structure.

The columns for representing PUF metadata are populated by the corresponding data in the **PUF XML tags** table. These data will be used for one-to-one mapping. In other words, the Mapping Tool will map a single PUF XML tag to a specific XMI tag.

When it comes to multiple tags mapping, the Mapping Tool is able to create the mappings for a set of PUF XML tags all at once rather than map them one by one. It is an efficient way of moving them all together to a detailed location in the UML structure.

The solution I came up with is to adopt the values ‘All’ and ‘0’ to distinguish the PUF XML tags with sub XML tags from the specific PUF XML tags which have no sub XML tag. Based on the way in which I represent the PUF XML tags, those at the higher level have no textual content for the comparatively lower columns while those ones at the lower level have specific textual content for those lower columns. To retain the entity integrity rule, the null values held by the PUF XML tags with sub XML tags are replaced by the value ‘All’ , For the PUF metadata without sub metadata, the value ‘0’ are used. A PUF XML tag holding the ’All’ value means that it has sub XML tags and all of them can be moved along with it to an XMI tag when doing many-to-one mapping. Take a <*TaskRec.LnkInfoSection.How*> tag for instance. Its high level PUF XML tags should be <*TaskRec.LnkInfoSection*> and <*TaskRec*>. The semantic column values for <*TaskRec.LnkInfoSection*> should be ‘TaskRec’, ‘LnkInfoSection’, ‘All’, ’All’, and ‘TaskRec’, ‘All’, ‘All’. ‘All’ for <*TaskRec*>. All the higher level and low level PUF metadata together serve as resources for mapping.

The columns for representing XMI tags are populated by the corresponding data in the **XMI tags** table so that all the UML elements in the **Mapping** table are at a low level and have detailed locations. If there is no value held by the lowest level, it will be padded by '0'. A XML tag in the **Mapping** table serves as a target tag for mapping. Since there is no correspondence in PUF for any UML element before mapping, the corresponding PUF semantic columns for each UML element are assigned '0'. After mapping, '0' is replaced by the values for the PUF element columns from the mapped resource PUF element.

As mentioned before, there are many mapping possibilities existing between a PUF XML tag and a XMI tag. The **Mapping** table must be able to deal with the following situations:

- A PUF XML tag can be mapped to the different XMI tags even at different levels (see in Table 4.1).
- The different PUF XML tags can be mapped to the same XMI tag (see in Table 4.2).

Table 4.1: One PUF XML tag maps to different XMI tags

PUF_ Record_Name	PUF_ Sectio_Name	PUF_ Question_Name	PUF_ SubQuestion_ Name	UML_ Package_ Name	UML_ SubPackage Name	UML_ PackageConstruct_ Name	UML_ ClassConstruct_Name	Mapping Relation
UserRec	IdInfoSection	Name	0	Behavioral_ Elements	Use_Cases	Actor	0	Inclusion relation
UserRec	IdInfoSection	Name	0	Foundation	Core	ModelElement	name	Inclusion relation

Table 4.2: Different PUF XML tags maps to one XMI tag

PUF_ Record_ Name	PUF_ Sectio_Name	PUF_ Questio_Name	PUF_ SubQuestion_ Name	UML_ Package_ Name	UML_ SubPackage Name	UML_ PackageConstruct_ Name	UML_ ClassConstruct_ Name	Mapping Relation
UserRec	DetReqSection	PhysCharCap	All	Behavioral_ Elements	Use_Cases	Actor	0	Inclusion relation
UserRec	DetReqSection	MentalCharCap	All	Behavioral_ Elements.	Use_Cases	Actor	0	Inclusion relation
UserRec	DetReqSection	SocialCharCap	All	Behavioral_ Elements.	Use_Cases	Actor	0	Inclusion relation
UserRec	DetReqSection	GroupCharCap	All	Behavioral_ Elements.	Use_Cases	Actor	0	Inclusion relation

As discussed before, the way I represent both the PUF XML tag and the XMI tag ensures that there is no Null value in the **PUF XML tags** table and the **UML XML tags** table. They lay the foundations of the primary key for the **Mapping** table. It is formed by the combination of two entity tables' existing

composite primary key. Considering the cases demonstrated here, mappings are not uniquely identified until at least both the PUF XML tag and XMI tag are known. As for the values in these semantic columns, they are consistent with those ones in the entity tables. If there is null value for a tag construct name, the corresponding column will be assigned the same value as the one in its entity table. This makes sure no row has a null value for a primary key. Thus it is guaranteed that there will only be exactly one non-null row per combination of PUF's semantic columns and UML's semantic columns. At the same time, this composite primary key supports a many-to-many mapping relation.

Naturally, each part of the primary key becomes a foreign key in this relation. Referential integrity guarantees that each semantic column value in the **mapping** table references an existing, valid matching candidate key value in the **PUF XML tags** table and the **XMI tags** table. Deleting a pair of mapped PUF XML tag and XMI tag will not lead to the deletion of the related records in the underlying base relations.

The optional update rule (ON UPDATE CASCADE) specifies the UPDATE CASCADE action will be taken to the **Mapping** table when a PUF XML tag or XMI tag is updated in its entity table. It ensures that the values of these two columns in the **Mapping** table match those in their corresponding base tables. Similarly, the optional delete rule (ON DELETE CASCADE) is applied to make it sure that when a PUF XML tag or XMI tag delete from its entity table, the relating rows in the **Mapping** table are also deleted.

The data types for the semantic columns in the **Mapping** table also conform to those ones in their entity tables, using varchar (80). The data type for the *Mapping relation* column is varchar (50) because the number of character that it can hold is variable and the length of a value is no more than 50.

4.2.5 Enhanced XMI Table

The **Enhanced XMI tags** table is used to store the UML requirement metadata, which is extended to include PUF requirement metadata by adding more levels into UML metadata structure. In other words, it can be regarded as an enhanced UML metadata structure. Therefore, it is built based on the structure of the **XMI tags** table. It stores the PUF XML tags along with the XMI tags. It contains the following columns:

- UML element name:
  - UML_Package_Name: a name of a package type.
  - UML_subPackage,_Name: a name of a sub-package type.
  - UML_packageConstruct_Name: a name of a package construct type.

- UML_classConstruct_Name: a name of a class construct.
- PUF_element_Name: a name of a question type
- PUF_ subElement _Name: a sub-question name

- Description: a long text explanation about the meaning of the PUF elements.

Since the PUF elements needing mapping are at the question and sub-question levels, the new levels added in the UML structure is used to accommodate PUF tag information. The entire original UML structure will be filled in their corresponding columns in this relation. The *PUF_element_Name* and *PUF_ subElement _Name* columns will be assigned "All" values. This is to avoid null values appearing in the primary key.

4.2.5 The Combined Set of Tables

These four tables are built based on the requirements I discussed in the section 4.1.1. Both the **Mapping** table and the **Enhanced XMI tags** table have four UML metadata semantic columns. When the *PUF_element_Name* and *PUF_subElement _Name* columns are added into the **Mapping** table, the **Enhanced XMI tags** table becomes its subset. I decided to simplify multiple tables into a single virtual table by using a view which is a dynamic, virtual table. In this way, the **Enhanced XMI tags** table can be easily composed of a subset of the **Mapping** table. The advantages of using a view are:

- A view is able to reduce redundancy existing in the database which may cause data integrity issues;
- A view is able to prevent developers seeing the complexity of the database. A view enables sorting and displaying of information useful to developers in an efficient way without showing unrelated information;
- A view requires little space to store. It improves the efficiency of the Mapping Tool.

The extended **Mapping** table should contain:

- PUF element name
  - PUF_Record_Name: a name of a record type. Not null.
  - PUF_Section_Name: a name of a section type. Not null.
  - PUF_Question_Name: a name of a question type. Not null.
  - PUF_subQuestion_Name: a name sub question. Not null.

- Enhanced UML element name:
  - UML_Package_Name: a name of a package type. Not null.
  - UML_subPackage,_Name: a name of a sub-package type. Not null.
  - UML_packageConstruct_Name: a name of a package construct type. Not null.
  - UML_classConstruct_Name: a name of a class construct. Not null.
  - PUF_element_Name: a name of a question type. Not null.
  - PUF_subElement _Name: a sub-question name. Not null.
- Mapping relation:
- Description: a long text explanation about the meaning of the PUF elements.

Before mapping, the values for the two PUF semantic columns in the UML elements are assigned 'All' because the UML elements from the **XMI tags** table do not have these columns. After mapping, the values for the UML semantic columns are changed based on the mapping relations:

- *Exact mapping relation*: There is no change in the UML structure while the mapped PUF element's name will replace '0' held by the related PUF element columns.
- *Inclusion mapping relation* or *non-existing mapping relation*: The matched PUF XML tags can be added at the same level or hierarchically below the locations of the target UML tags. As mentioned before, mapping mainly happens at the lower levels in the UML structures (e.g. the *packageConstruct* and the *classConstruct* levels).

Considering single tag mapping, if a PUF XML tag partially maps to a XMI tag at the *packageConstruct* or *classConstruct* level, the value for the lowest matched PUF semantic column is assigned to the *packageConstruct* or *classConstruct* column. At the same time, the PUF XML tag names populate the corresponding columns in that row.

If no-match mapping happens at the *packageConstruct* level, the value for the lowest matched PUF semantic column is assigned to the *classConstruct* column. Another possibility is that no-match mapping happens at the *classConstruct* level, a new level is needed in the UML structure. That is why a *PUF_tagConstruct_Name* column is added into this relation. The new XMI tag can be created at the fifth level and its value is from the lowest matched PUF semantic column. All the '0's in that row will be replaced by the mapped PUF XML tag name.

With regards to multiple tags mapping, two new levels will be used to store the PUF element names. If a mapping has an *Inclusion mapping relation*, the PUF question name and its sub-question names will populate the UML element's *packageConstruct* and *classConstruct* levels or the *classConstruct* and *PUF_element_Name* levels according to the location of the UML element. If a mapping has a *non-existing mapping relation*, the PUF element will go below the UML element. Thus the PUF question name and its sub-question names will populate the UML element's the *classConstruct* and *PUF_element_Name* levels or the *PUF_element_Name* and *PUF_subElement_Name* levels according to the location of the UML element.

The **Enhanced XMI tags** view is created on the **Mapping** table by retrieving all the semantic columns in UML which are UML element names and two added PUF element names. The **MappingReport** view is needed to show all the mappings. If a UML element has the value(s) for the corresponding PUF semantic columns, it means the UML element has a mapped PUF element. The view is built by retrieving these kinds of UML elements, the mapped PUF elements and their mapping relations. Thus only three of the four tables identified previously are actually needed, since the fourth, the **Enhanced XMI tags**, is able to be created as a view. The required tables are a **PUF XML tags** table, a **XMI tag**s table, and a **Mapping** table. Figure 4.3 illustrates these along with the **Mapping Report** and **Enhanced XMI tags view** which can be generated from them. The solid boxes denote the tables in their corresponding databases. The solid lines show the relationships between them. The dashed boxes represent the views. The dashed lines are used to link these views to the tables on which they are built.

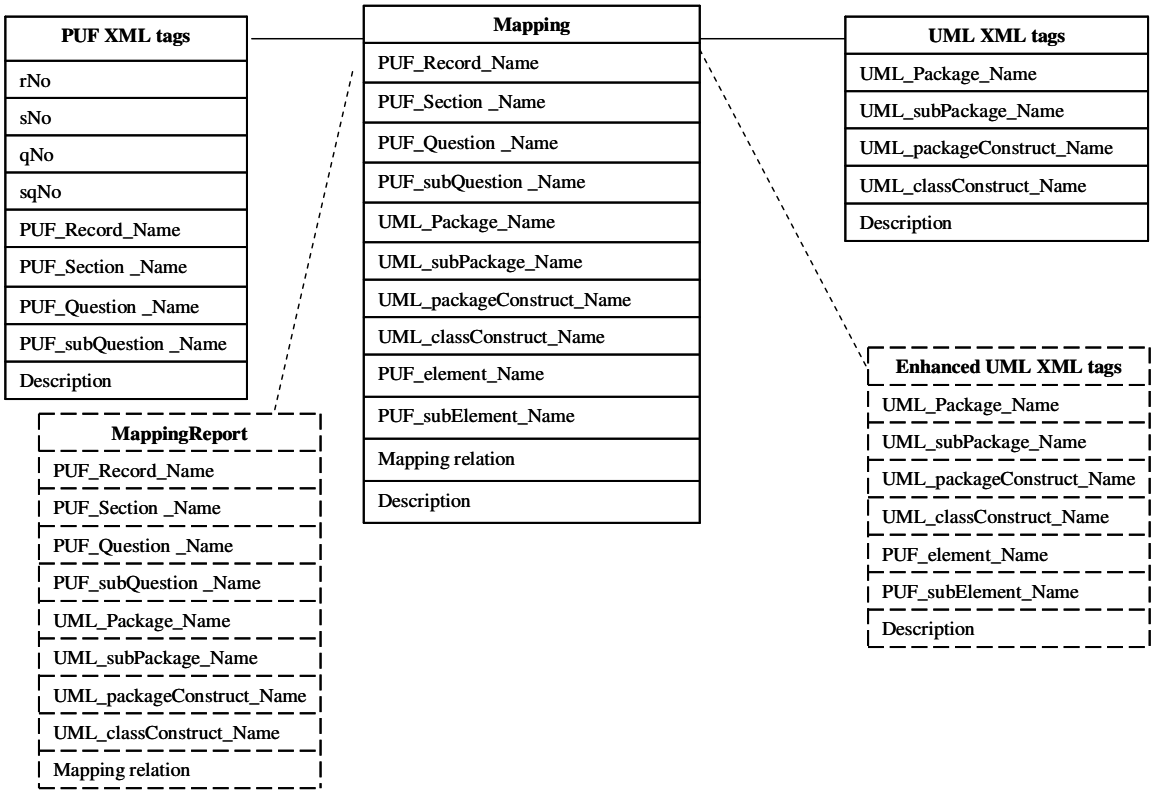

Figure 4.3: Detailed database design

4.3 Function Design

The main function of the Mapping Tool is to create the mappings between PUF requirement metadata and UML requirement metadata. From the discussion about database design, it is easy to identify the Mapping Tool's inputs and outputs. The requirements discussed in chapter 3 show that the Mapping Tool maintains a few states in the course of mapping PUF requirement metadata to UML one. So it is appropriate to use the Mapping Tool in a function-oriented approach. Because the Mapping Tool is simple, its straightforward steps for mapping discussed in the previous section can be broken down into a number of operations. These operations correspond to the specific functions. Considering the mapping possibilities existing between two sets of metadata, in addition to these functions the Mapping Tool should provide even more functions to facilitate the mapping process. This section describes this system's functions and explains how they should be realized. Before illustrating the function design, it is necessary to analyze the requirements concerning the functions of the Mapping Tool.

Input Requirements

1. Considering the requirement that the Mapping Tool should support both one-to-one mapping and many-to-one mapping, the PUF XML tag in a mapping could be abstract (at the high level) or detailed (at the low level). Thus all the generic and specific PUF XML tags should be exposed so that which PUF XML tag needs to map and which type of mapping will be carried out can be determined. The Mapping Tool should provide facilities for choosing a PUF XML tag out of all the higher level and lower level PUF XML tags as a source tag.
2. As far as the XMI tags are concerned, they should be specific (at the low level). Otherwise the mappings are not precise and accurate. The Mapping Tool should provide facilities for choosing a XMI tag out of all the lower level XMI tags as a target tag.
3. Given a PUF XML tag and a XMI tag, the mapping relation between them should be determined. The three options of mapping relations should be viewable. The Mapping Tool should provide facilities for choosing one mapping relation out of three possible choices.
4. Considering the requirement that the Mapping Tool should also support one-to-many mapping, it should provide facilities for selecting one or more UML elements and mapping relations for the chosen PUF element.
5. The Mapping Tool should provide a convenient way to modify a part of the mapping information in the existing mappings. The Mapping Tool should provide facilities for changing the chosen PUF XML tag, the XMI tag or the mapping relation.
6. Mappings should be able to be deleted if there is too much mapping information of a mapping needed to be modified. It is time-consuming to change them one by one. Therefore, the Mapping Tool should allow them to delete mappings and restart mapping from scratch.

Output Requirements

7. The current generated mappings should be kept so that they can easily resume mapping next time. The Mapping Tool should be able to generate a list of mappings which can be saved or printed.
8. There are chances that a whole picture of a UML structure with the integrated PUF information is needed during the process of mapping or at the end of mapping. The Mapping Tool should be able to generate an enhanced XMI file which can be saved or printed.

Usability Requirements

9. Help information for understanding detailed meanings of PUF elements or/and UML elements and their purposes should be available. The Mapping Tool should assist in choosing the PUF elements, UML elements and mapping relations by providing the tag description. As a result, mapping can be conducted smoothly and successfully.
10. To make it usable, the Mapping Tool should be able to make information about progress of mapping available including how many PUF XML tags have matching XMI tags and how many of them are left to map.

A set of functions is needed to be built to satisfy the above requirements. To make its purpose clear, each function aims at one requirement. Since some system functions should be realized by a number of subordinate functions, there is a hierarchy in the organization of functions. To make explicit the information about functions, a set of structure charts and a data dictionary are employed. Structure charts are a useful tool to show the decomposition hierarchy and dynamically display how these functions communicate with each other. The data dictionary gives detailed design description.

Figure 4.4 dynamically illustrates the organization of the Mapping Tool at the initial level. Generally speaking, the structure chart is constituted by the **Input Transformations** and **Output Transformations** components. The **Input Transformations** component is concerned with getting mapping data from other tools; the **Output Transformations** component is concerned with printing the reports. To make **Input Transformations** and **Output Transformations** distinguishable, they are separated in the different dotted boxes. Functions are indicated with by a rectangle. Linking rectangles with a line implies their hierarchical relation. An annotated arrow entering into a box represents input of a function. An arrow leaving a box represents output of a function. User inputs and system outputs are all shown as circles.

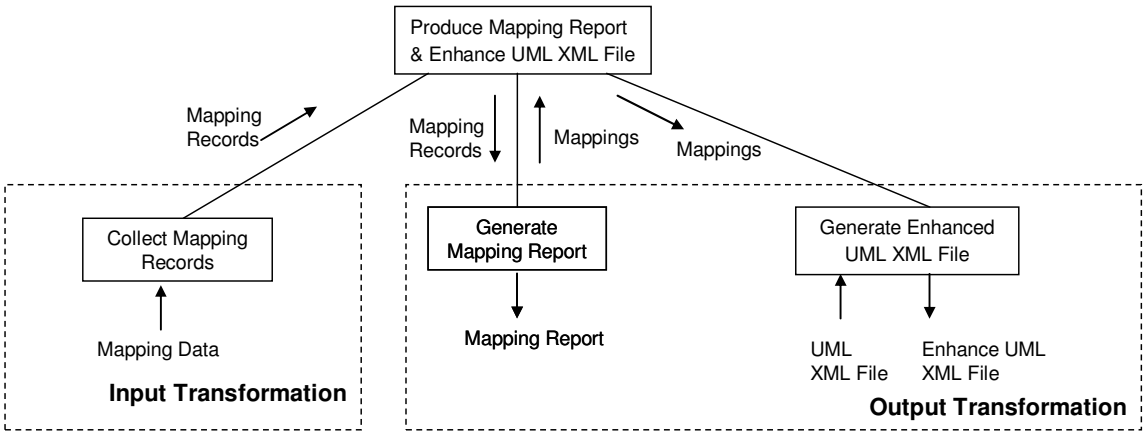

Figure 4.4: Initial structure chart for the Mapping Tool

The functions in the **Input Transformations** and **Output Transformations** components can be broken down into a number of subordinate functions. The system model at this level is shown in Figure 4.5. Databases are represented as ovals.

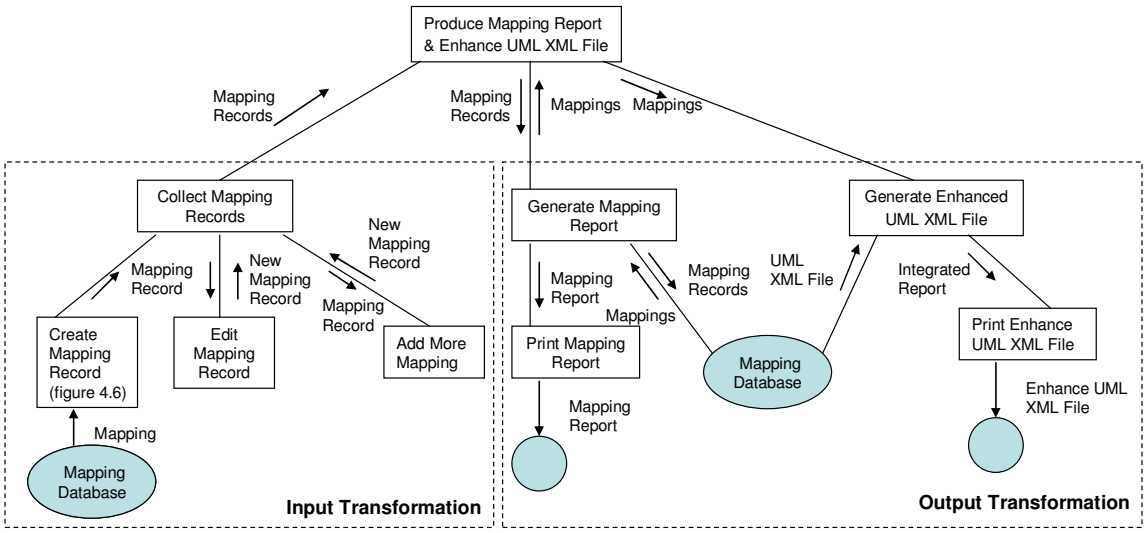

Figure 4.5: Second level structure chart for the Mapping Tool

The **Output Transformations** component stops at the *Print Mapping Report* and the *Print Enhanced UML XML File* functions. The **Input Transformations** component still needed decomposition. Figure 4.6 shows the structural chart for the **Input Transformations** component at the third level.

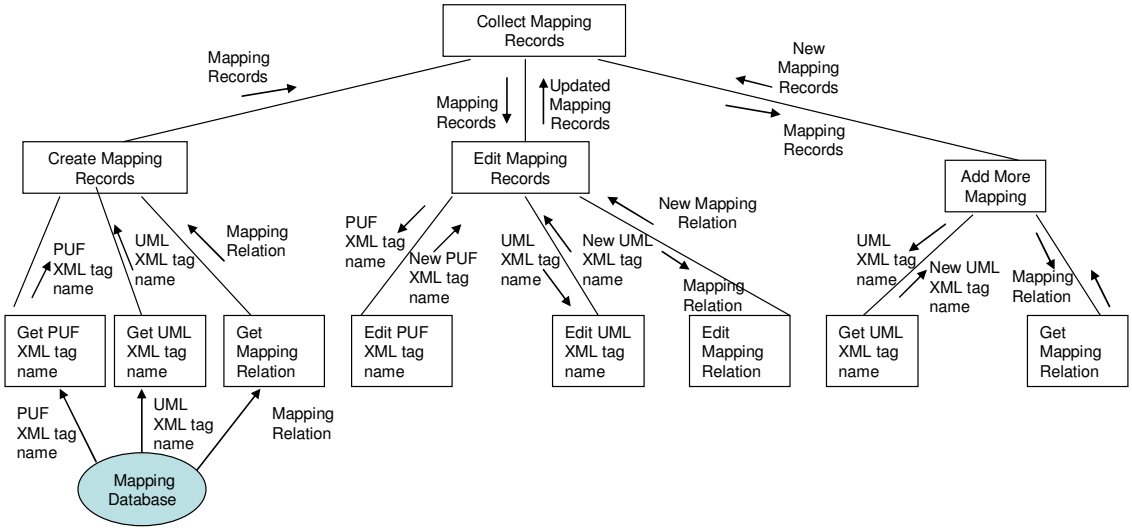

Figure 4.6: Third level structure chart for Input Transformations component

Considering that the *Create Mapping Records*, *Edit Mapping Records* and *Add More Mapping* functions are similar, the decomposition of the *Create Mapping Records* function is given as an example demonstrated in Figure 4.7.

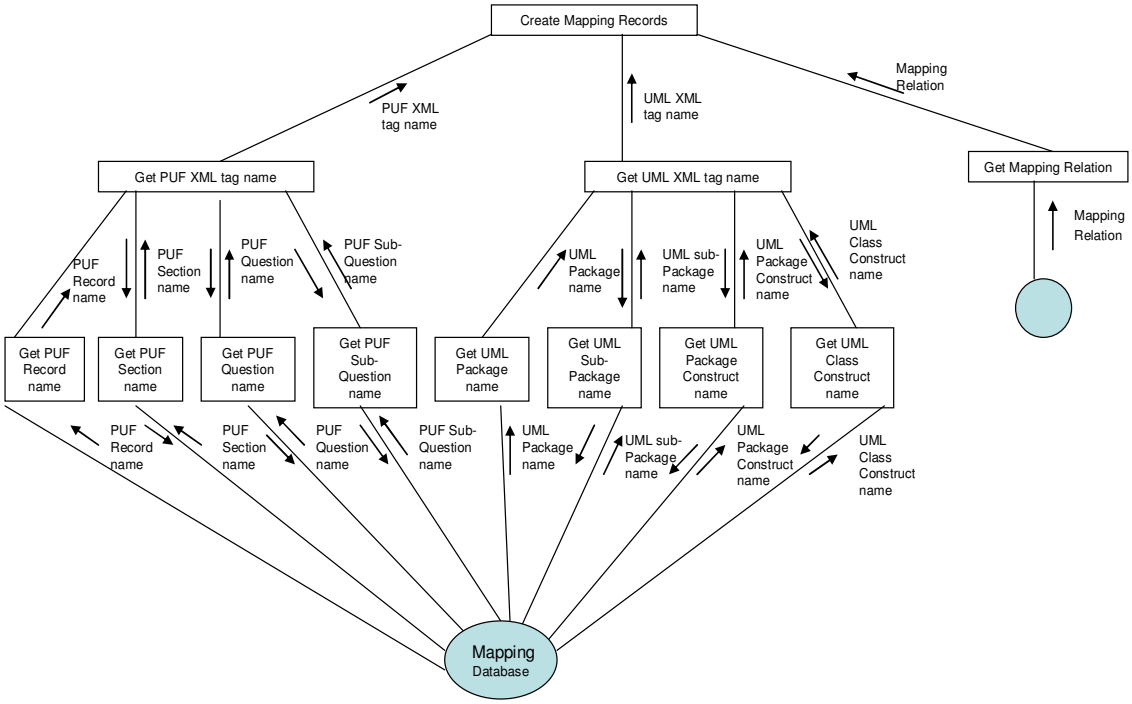

Figure 4.7: Final structure chart for the Create Mapping Records function

The above figures demonstrate the organization of the function design. What each function should do is depicted in Table 4.3. It briefly specifies the critical "building block" components and their inputs and their outputs.

Table 4.3 Data dictionary entries

Function Name	Description
Get PUF XML tag name	*Input*: PUF record name, PUF section name, PUF question name, PUF sub question name from the Mapping database *Function*: This function communicates with the user to get the name of a PUF XML tag that has been stored in the **PUF XML tags** table in the mapping database and save the PUF XML tag name in the **Mapping** database *Output:* PUF XML tag name into the Mapping database
Get PUF record name	*Input*: PUF record name from the Mapping database *Function*: This function communicates with the user to get a PUF record name that has been stored in the **PUF XML tags** table in the mapping database. *Output:* PUF record name into the Mapping database
Get PUF section name	*Input*: PUF record name from the Mapping database *Function*: Given the selected PUF record name, this function accesses the **PUF XML tags** table to find the PUF section name in that PUF record *Output*: PUF section name into the Mapping database
Get PUF question name	*Input*: PUF section name from the Mapping database *Function*: Given the selected PUF record name and a PUF section name, this function accesses the **PUF XML tags** table to find the PUF question name in the PUF section which is in that PUF record

	Output: PUF question name into the Mapping database
Get PUF sub question name	*Input*: PUF question name from the Mapping database *Function*: Given the selected PUF record name and the selected PUF section name and a PUF question name, this function accesses the **PUF XML tags** table to find PUF sub question name in the PUF question which is in the PUF section in that PUF record *Output*: PUF sub question name into the Mapping database
Get XMI tag name	*Input*: UML package name, UML sub package name, UML package construct name, UML class construct name from the Mapping database *Function*: This function communicates with the user to get the name of a XMI tag that has been stored in the **XMI tags** table in the mapping database and save the PUF XML tag name in the **Mapping** database. *Output:* XMI tag name into the Mapping database
Get UML package name	*Input*: UML package name from the Mapping database *Function*: This function communicates with the user to get a UML package name that has been stored in the **XMI tags** table in the mapping database. *Output:* PUF record name into the Mapping database
Get UML sub package name	*Input*: UML package name from the Mapping database *Function*: Given the selected UML package name, this function accesses the **XMI tags** table to find the UML sub package name in that UML package. *Output:* UML sub package name into the Mapping database
Get UML package construct name	*Input*: UML sub package name from the Mapping database *Function*: Given the selected UML package name and a

	UML sub package name, this function accesses the **XMI tags** table to find the UML package construct name in that UML sub package which is in the UML package. *Output:* UML package construct name into the Mapping database
Get UML class construct name	*Input*: UML package construct name from the Mapping database *Function*: Given the selected UML package name, the selected UML sub package name and a UML package construct name, this function accesses the **XMI tags** table to find the UML class construct name in the UML package construct in that UML sub package which is in the UML package. *Output:* UML class construct name into the Mapping database
Get mapping relation	*Input*: Mapping relation from user inputs from the Mapping database *Function*: This function communicates with the user to get the name of a mapping relation and save it in the **Mapping** table. *Output:* Mapping relation into the Mapping database
Edit PUF XML tag	*Input*: PUF XML tag in a mapping record from the Mapping database *Function*: This function communicates with the user to modify the chosen PUF XML tag in an existing mapping record which has been stored in the **Mapping** table in the mapping database *Output:* Updated PUF XML tag into the Mapping database

Reselect PUF record	Same as the *Get PUF record name* function
Reselect PUF section	Same as the *Get PUF section name* function
Reselect PUF question	Same as the *Get PUF question name* function
Reselect PUF sub question	Same as the *Get PUF sub question name* function
Edit XMI tag	*Input*: XMI tag in a mapping record from the Mapping database *Function*: This function communicates with the user to modify the chosen XMI tag in an existing mapping record which has been stored in the **Mapping** table in the mapping database *Output:* Updated XMI tag into the Mapping database
Reselect UML package	Same as the *Get UML package name* function
Reselect UML sub package	Same as the *Get UML sub package name* function
Reselect UML package construct	Same as the *Get UML package construct name* function
Reselect UML class construct	Same as the *Get UML class construct name* function
Edit mapping relation	*Input*: Mapping relation in a mapping record from the Mapping database *Function*: This function communicates with the user to modify the mapping relation in an existing mapping record which has been stored in the **Mapping** table in the mapping database *Output:* Updated mapping relation into the Mapping database
Delete mapping	*Input*: Mapping record from the Mapping database *Function*: This function communicates with the user to delete an existing mapping record from the **Mapping** table. The system will warn a user and seek confirmation prior to deletion.

	Output: Null
Add more mapping	*Input*: Mapping records *Function*: This function communicates with the user to create a new mapping for the chosen PUF XML tag in an existing mapping record which has been stored in the **Mapping** table in the mapping database *Output:* New mappings into the Mapping database
Get UML package	Same to the *Get UML package name* function
Get UML sub package	Same to the *Get UML sub package name* function
Get UML package construct	Same to the *Get UML package construct name* function
Get UML class construct	Same to the *Get UML class construct name* function
Generate mapping report	*Input*: Mapping records from the Mapping database *Function*: This function communicates with the user to produce a list of latest mapping records which are stored in the **Mapping** table. *Output:* Mapping report
Generate enhance XMI file	*Input*: Mapping records, Mappings from the Mapping database *Function*: This function communicates with the user to apply the mappings stored in the **Mapping** table to the related XMI tags in the initial **XMI tags** table, integrating the PUF XML tags into the XMI structure. The result of integration will be saved in the **Enhance XMI tags** table. *Output:* Enhanced XMI file stored on a local disk.
Print mapping report	*Input*: Mapping report from the Mapping database *Function*: This function communicates with the user to print all the mapping records which are stored in the **Mapping** table.

	Output: Mapping report
Print enhance XMI file	*Input*: Enhance XMI file from the Mapping database *Function*: This function communicates with the user to print the enhance XMI structure saved in the **Enhance XMI tags** table. *Output:* Enhance XMI file

Besides these fundamental features, the system provides assistance in conducting mapping. The *Help* function can dynamically give a detailed description about the meaning and purposes of the chosen tag if it is not understandable.

4.4 Screen Design

The functions of the Mapping Tool discussed above have been implemented. This section discusses how to present these functions in a logical, reasonable and usable way so that can be easily understood and used. The following content is required based on the functions discussed above:

- Content for the *Select* function should be viewable. It includes PUF XML tags, XMI tags and mapping relations.
- Content for the *Create Mapping Records* function should be viewable. It should add new mapping information into a list of existing mappings.
- Content for presenting all the unmapped PUF XML tags.
- Content for the *Edit Mapping Records* function should present the existing mapping information of a mapping including the PUF XML tag, the XMI tag and the mapping relation.
- Content for the *Adding More Mapping* function should present the PUF XML tag which has already had mapping(s) and all XMI tags and mapping relations.
- Content for the *Generate Mapping Report* function should present as a list of all the created mappings in a document so that it can be saved or printed.
- Content for the *Generate Enhanced XMI file* function should present an enhanced XMI file as a document so that it can be saved or printed.

- Content for the *Help* function should present the meanings of PUF XML tags, XMI tags and mapping relations.
- Warning content is needed when a mapping is deleted.

The web application was created to meet the above requirements. It is composed of a series of web pages containing the required content. Each web page is used to realize one or two more functions. They are:

- The SELECT page which is used to:
  - create a mapping by selecting a PUF XML tag, a XMI tag and a mapping relation;
  - create additional mapping for a mapped PUF XML tag by selecting another XMI tag and their mapping relation.
- The DISPLAY page which is used to:
  - show all the existing mapping information including the PUF XML tags, the XMI tags and the corresponding mapping relations;
  - link to the related pages when editing the mapping information takes place;
  - link to the SELECT page when adding one more mapping takes place;
  - delete a existing mapping;
  - show all the unmapped PUF XML tags;
  - link to the SELECT page when creating a mapping for a unmapped PUF XML tag.
- The EDIT pages which can fall into three categories according to different components of mapping information. They are respectively used to
  - modify the PUF XML tag;
  - modify the XMI tag;
  - modify their mapping relation.
- The HELP pages which are used to:
  - dynamically display help text for PUF XML tags, XMI tags and mapping relations.

To make these pages usable and accessible, the following usability-related concerns were kept in mind during the process of development.

- **Provide an efficient and convenient way of carrying out mapping tasks.** To do so, all the functions or sub functions should be presented separately and be clearly labeled. Since all the functions or sub functions are very straightforward, my major concern in presenting them is to retain that simplicity.
- **Content shown in the pages should be legible, meaningful and useful.** To accomplish this, a simple and clear layout is adopted through all the pages. Page size, the type of font, and font size which have impacts on legibility are a concern. Important information should be made to stand out. Objects used in several pages should be consistently presented.
- **Make different categories of elements distinguishable.** For instance, all PUF elements which have one or more mappings are shown in a group on the top of the screen. Those elements without mappings are displayed in another group on the bottom of the screen. Furthermore, the check box in front of each item indicates whether it has a mapping or not.
- **Give help content.** Help information is provided where UML elements and/or PUF elements are beyond developers' knowledge or if they find difficulties in doing mapping.
- **Give alerts where changes are made.** After developers make important changes to PUF or UML elements, such as creating or editing a mapping, notices will be shown on the screen letting developers know to which elements they made changes. As for the crucial actions, such as delete, there will be warnings popped up to confirm the deletion. This mechanism can help prevent developers from making changes to wrong elements or carrying out irretrievable actions

The data structure that I designed satisfied all the requirements and laid a solid foundation for the development of the page content design. The detailed description of the page design and how the pages communicate with each other will be illustrated in the following text.

From the discussion in the previous section, a mapping process begins with the SELECT page (Figure 4.8). It should identify a source tag (PUF XML tag), a target tag (XMI tag), and their mapping relation. To do so, the whole page can be broken down into three parts. One is for choosing PUF XML tags. The second part is for choosing XMI tags. The last part is for choosing the tags' mapping relation. The way I represented the PUF XML tags and XMI tags in this page is based on the data structure of the PUF XML tags that I constructed and the XMI tags from UML DTD 1.3. Each tag construct corresponds

to a drop down list. Thus, the first part of the page is comprised of four drop down lists. The same is true for the second part. The mapping relation part is a single drop down list with three options. My solution to dynamically limit the options in the drop down list is to submit the form to the server on selection of the previous drop down lists and based on the selection get the element data for the current drop down list.

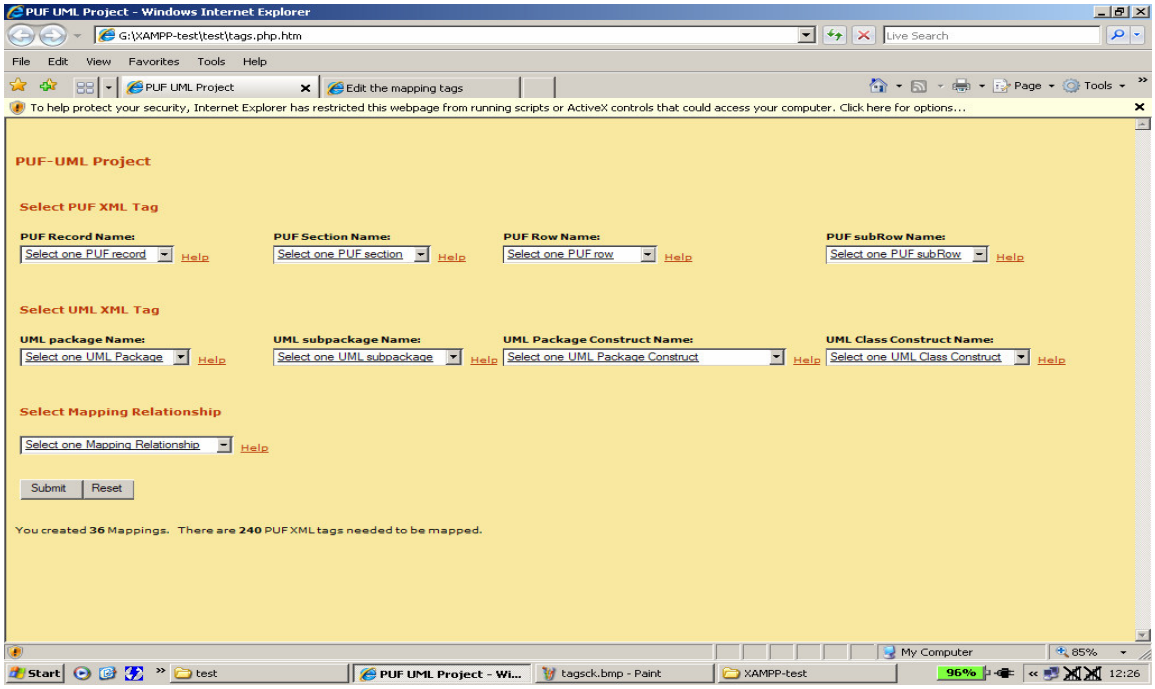

Figure 4.8: The SELECT page

The four drop down lists in the first group of drop down list are the **PUF record name**, **PUF section name**, **PUF row name list**, and **PUF subrow name** lists. These four drop down lists are populated by taking data from the **pufxmltags** table in the puf-uml database. The system can dynamically narrow down the items in the **PUF section name** drop down list based on the selected item from the **PUF record name** drop down list, restrict the options of the **PUF row name** drop down list based on the selection of the **PUF record name** and **PUF section name** drop down lists, and manage the options of the **PUF subrow name** drop down list based on the three previous drop down lists.

The four drop down lists in the second group of drop down list are the **UML package name**, **UML subpackage name**, **UML package construct name**, and **UML class construct name** lists. These four drop down lists are populated by taking data from the **umlxmltags** table in the puf-uml database. The system is also able to dynamically change the list options based on the previous selection.

The single drop down list provides the available options for the mapping relation. There are two buttons below it. One is **Submit** and another is **Reset**. When the former one is clicked, all the data in the HTML form is stored in the mapping table. It means mapping is created and this new added mapping will show in the DISPLAY page. If all the selected data needs cleaning and selecting from the very beginning, the Reset button should be chosen.

To be user-friendly, the number of existing pairs of mappings and the number of unmapped PUF XML tags is shown at the end of the web page to indicate the progress of the mapping. The system also provides help text for each XML tag and each mapping relation in case their meanings are not understood. Figure 4.9 shows the help text for the PUF XML tag <*ToolRec.EnvInfoSection.Why*>.

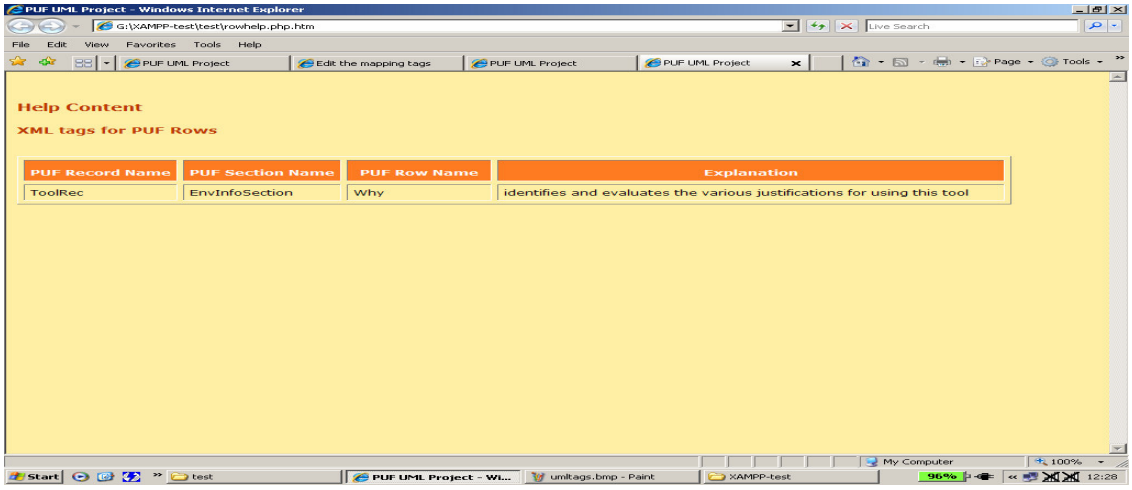

Figure 4.9 The help text for the PUF XML tag <*ToolRec.EnvInfoSection.Why*>

The purpose of the DISPLAY page (Figure 4.10) is to show the result of mapping, to facilitate carrying out operations on existing mappings and to explore PUF XML tags without mappings. The DISPLAY page is divided into two parts. The top part is for existing mappings which have checked boxes. The bottom part is for unmapped PUF XML tags which have unchecked boxes. Before mapping, the bottom part shows all PUF requirement metadata need mapping. Once a mapping is done, that particular PUF requirement metadata will move from the bottom part to the top part. In doing so, developers can immediately notice how much they have done and how much work is left.

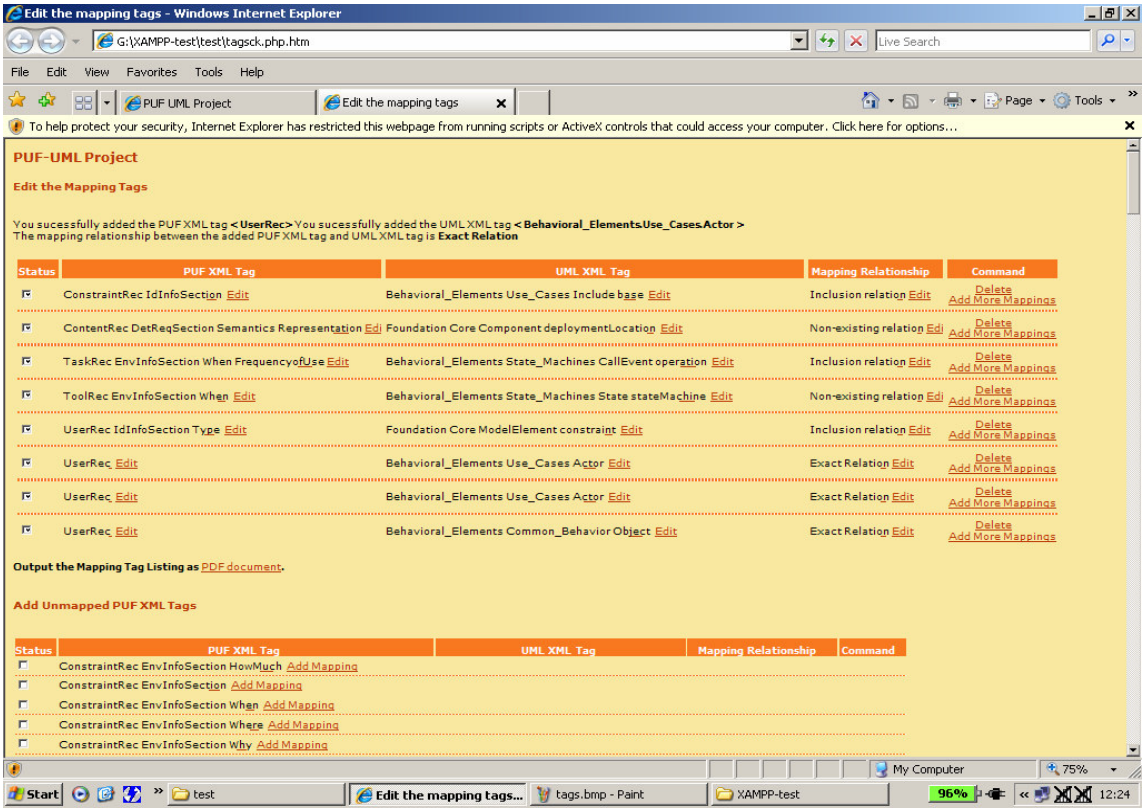

Figure 4.10: The DISPLAY page

When the DISPLAY page receives the submission from the SELECT page, there is a text shown up, specifying which PUF XML tag, which XMI tag and mapping relation are added. Simultaneously, the new mapping will be automatically added to the existing mapping form.

The existing mapping form has five columns, namely mapping status, PUF XML tag, XMI tag, mapping relation and command. All the mappings are ordered by PUF XML tag. The check boxes in the first column in this form are all checked. It indicates that all the mappings exist and are stored in the mappings table. The PUF XML tag, the XMI tag and the mapping relation columns are populated by taking the data from the mappings table. The mapping information in these three columns is changeable. The Edit link in each column leads to an EDIT page to modify that part of the mapping information. Figure 4.11 is the EDIT page for the XMI tag **<*Behavioral_Elements.Use_Cases.Actor*>.** The mapping information which is being edited is shown on the top part of the EDIT page. A group of drop down lists or a single drop down list is used for reselecting mapping information. The Update button is used to confirm the modification and it leads to replace the original mapping information in the mapping database with the new one. Meanwhile, the editing mapping information on the DISPLAY page is updated.

The command column contains two commands. One is **Delete**. It is used for deleting the selected mapping without deleting the PUF XML tag from the **PUF XML tag**s table or deleting the XMI tag from the **XMI tag**s table. A prompt window pops up to confirm the deletion. After this command is executed, the mappings below the selected mapping will be moved upwards and the PUF XML tag in the select mapping will be found in the unmapped PUF XML tags form. To cope with multiple tag mapping, the second command, **Add more mapping** shown in Figure 4.12 is created. It explores other possible mappings for the selected PUF XML tags. When this command is chosen, it links to the SELECT page and the first group of drop down lists for PUF XML tag information has already selected. The second group of drop down lists for XMI tag information and the drop down list for their mapping relation need to be determined. Figure 4.11 shows the SELECT page for adding one more mapping for the PUF XML tag **<***ToolRec.EnvInfoSection.When***>** which identifies a range of possibilities regarding when the task can be accomplished in terms of temporal attributes and includes both current and potential future ranges. When the new mapping for the selected PUF XML tag is created, it will be added below the existing mapping for the selected PUF XML tag.

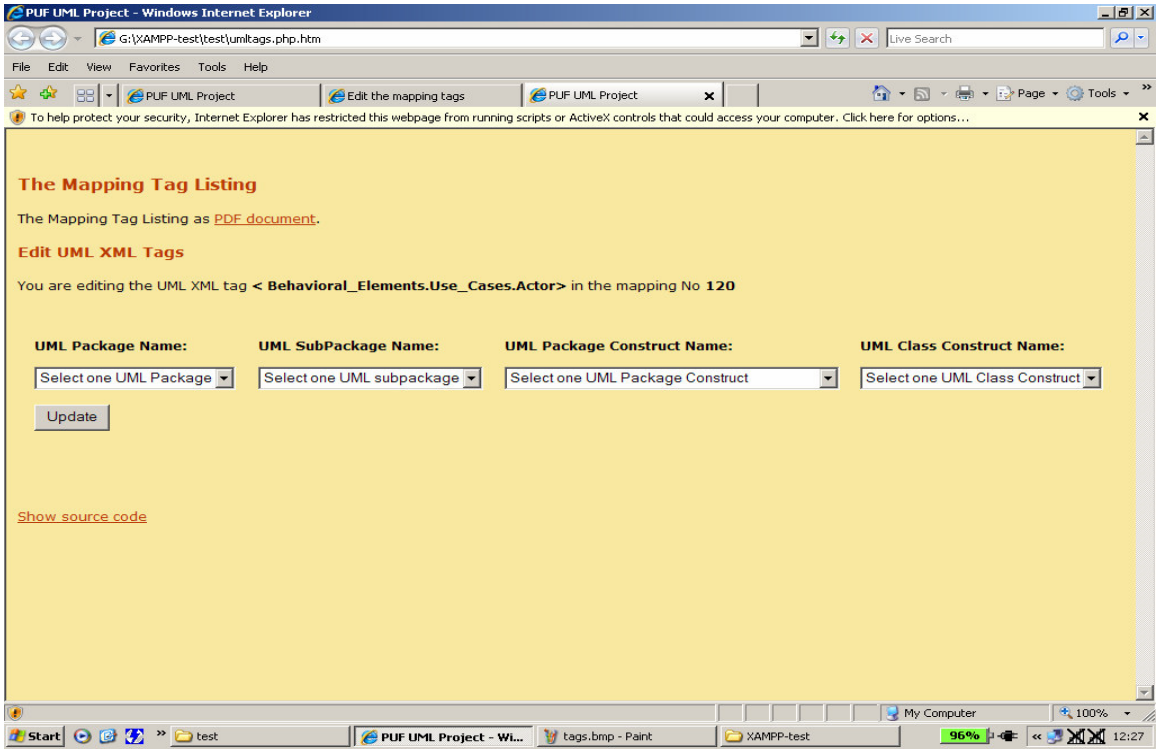

Figure 4.11: The EDIT page for the XMI tag < *Behavioral_Elements.Use_Cases.Actor*>

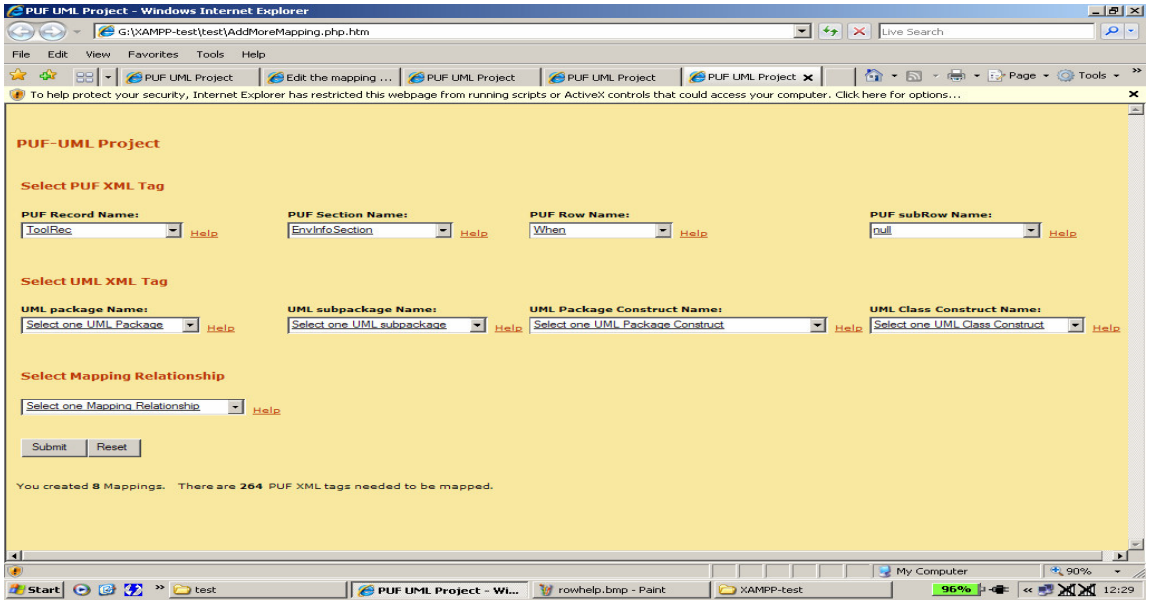

Figure 4.12: The SELECT page for adding more mapping

There is a link below the existing mapping form, outputting all the mappings in pdf format. It makes easy to save the created mappings and to print them out.

All the PUF XML tags without mappings are gathered in the unmapped PUF XML tags form. The form has two columns, mapping status and PUF XML tag. The check boxes in the mapping status column are all unchecked, representing the fact that these PUF XML tags are unmapped. The PUF XML tag column shows the name of the unmapped PUF XML tag. Next to each PUF XML tag is an Add link. This link leads to the SELECT page. Figure 4.13 gives an example of the SELECT page for adding a mapping for the PUF XML tag <*ConstraintRec.EnvInfoSection.Where*> which identifies any physical attributes that affect the constraint. After adding a UMLXML tag and a mapping relation for the selected PUF XML tag is confirmed, the new mapping is shown in the existing mapping form and the selected PUF XML tag is deleted from the unmapped PUF XML tags form.

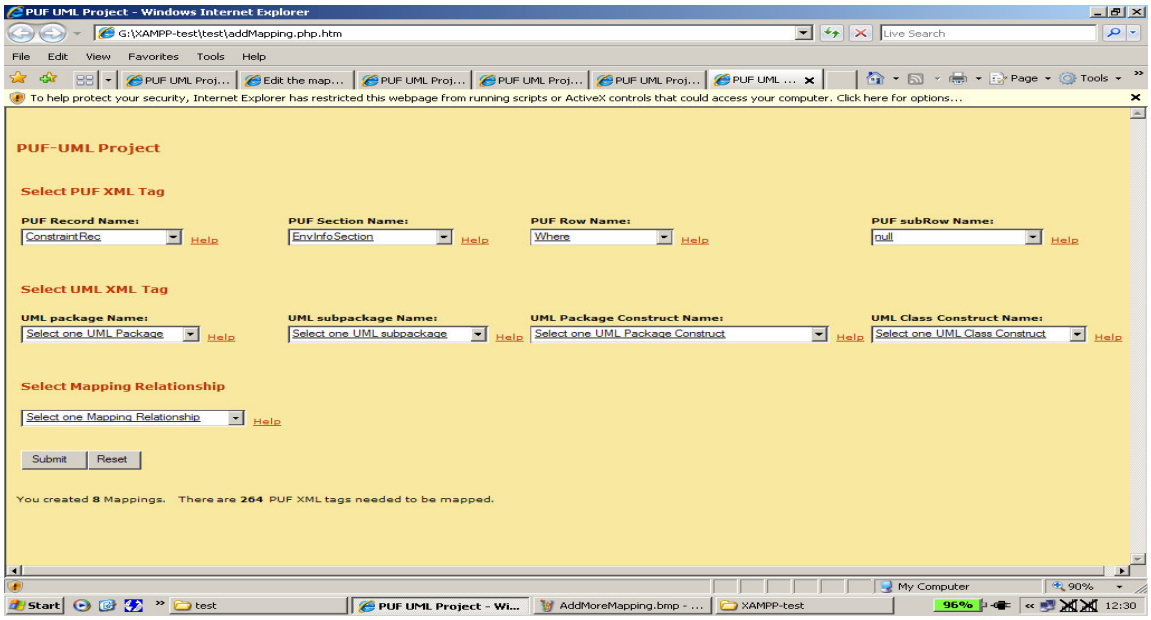

Figure 4.13: The SELECT page for adding mapping

4. 5 Program Design

In terms of implementation of the functions we discussed above, XAMPP [37] is used to realize them. XAMPP is a complete package integrated web, database, FTP server. It provides an easy approach to the agile development of a web application by working with a database and a web server. The version of XAMPP used for implementation includes Apache HTTPD 2.2.8 [38] as web server, MySQL [39] as database and PHP [40] as server-side scripting language. Apache is an efficient open-source HTTP server to develop both static web application and dynamic web application. PHP is a server-side scripting language originally used to create dynamic web pages. When PHP runs on the Apache web server, the web server will take PHP code as input and output web pages. MySQL is a very popular open source database which is notable for its high efficiency and ease of use. It is widely used not only by individual developers but also by large organizations. All of these projects are open sources.

CHAPTER 5

RESULTS

This chapter describes the preliminary requirement metadata mappings between UE requirement metadata (from the PUF methodology) and SE requirement metadata (from the UML methodology). The sections discuss the results from the Mapping Tool:

- The Mapping Tool, created as part of this thesis, was used to map PUF requirement metadata into UML specifications. Details of this mapping are presented in Appendix 2, 3, 4 and 5.
- The Mapping Tool was used to map PUF task and user records to UML. Examples of this mapping are discussed in Section 5.1.
- Mapping PUF into UML uncovered a variety of needs for adding usability related information into UML. These needs are discussed in Section 5.2.

These results attest to the feasibility of creating mappings to integrate UE requirement metadata into SE specifications.

5.1 Examples of the Mapping

This section serves to summarize the types of mapping relations and types of mapping. A number of examples are given to illustrate the possible mapping between PUF requirement metadata and UML metadata. All the mappings given as examples and created in the thesis are based on my understanding of PUF and UML requirements. They are preliminary mappings. Although I developed them very carefully to ensure their accuracy, they have not been validated so they are not definitive.

- Section 5.1.1 discusses examples of the different types of the mapping relations.
- Section 5.1.2 demonstrates examples of the various types of the mapping.
- Section 5.1.3 provides suggestions on mapping PUF components to UML components

5.1.1 Types of Mapping Relations

The results of applying the Mapping Tool demonstrate that the three possible mapping relations identified in the requirements section do exist in the mapping between a PUF XML tag and an XMI tag. Three examples given in Table 5.1, 5.2 and 5.3 demonstrate the three types of mapping relations respectively.

They also show how the matched XMI tags are enhanced. The changed tag constructs or the added ones are shown in **boldface** type.

The first mapping type is an *exact mapping relation*. This mapping relation requires that the semantics presented by the two XML tags should be identical. Table 5.1 is an example. The <*Behavioral_Elements.Use_Cases.UseCase*> tag describes "an interaction between an actor and a system aimed at realizing a function" [35]. The <*TaskRec.LnkInfoSection.What*> tag represents the tasks which they need to accomplish. These two tags are identical although their names differ from each other. Their mapping relation should be an *exact mapping relation.* Therefore, there is no new XML tag added into the UML XML structure.

Table 5.1: An example of an *exact mapping relation*

PUF XML tag	<TaskRec.LnkInfoSection.What>
XMI tag	<Behavioral_Elements.Use_Cases.UseCase>
Mapping Relation	Exact Relation
Enhanced XMI tags	<Behavioral_Elements.Use_Cases.UseCase>

The second mapping type is an *inclusion relation*. This type of mapping relation refers to two matched XML tags sharing a part of their meanings but not totally. The example of this mapping relation type is demonstrated in Table 5.2. The <*UserRec.LnkInfoSection.Who*> tag identifies who belongs to a given user group and can be described in terms of the characteristics that make an individual a member, especially focusing on how that user group is different from other user groups. The <*Behavioral_Elements.Use_Cases.Actor*> tag could be a human being who interacts with a system or other system which interacts with the system. There is a small overlap between the requirements presented by the <*UserRec.LnkInfoSection.Who*> tag and those in the <*Behavioral_Elements.Use_Cases.Actor*> tag. Therefore, they are partially matched.

Table 5.2: An example of an *inclusion mapping relation*

PUF XML tag	<UserRec.LnkInfoSection.Who>
XMI tag	<Behavioral_Elements.Use_Cases.Actor>
Mapping Relation	Inclusion relation
Enhanced XMI tags	<Behavioral_Elements.Use_Cases.Actor> …. <Behavioral_Elements.Use_Cases.**Who**>

The third mapping type is *a non-existing mapping relation*. This mapping type is used to indicate that the requirement metadata represented by a PUF XML tag cannot find equivalence in the identified UML construct. Table 5.3 demonstrates an example. The <*UserRec.DetReqSection.PhysCharCap*> tag identifies various physical limitations and impairments the users may experience. These requirements help developers design the application which is usable to the range of physical capabilities experienced by users. The <*Behavioral_Elements.Use_Cases.Actor*> tag defines "a coherent set of roles that users of an entity can play when interacting with the entity" [35]. It has no sub element in the use case diagram. Its identification information and other general information about the actor are represented by this tag's attributes rather than sub tags. As a result, the detailed PUF requirement metadata, for instance, physical characteristics, cannot find equivalence in XMI file. Their mapping relation should be a *non-existing mapping relation*. The added PUF requirement metadata supplement UML specification with user-oriented information.

Table 5.3: An example of a *non-existing mapping relation*

PUF XML tag	<UserRec.DetReqSection.PhysCharCap>
XMI tag	<Behavioral_Elements.Use_Cases.Actor>
Mapping Relation	Non-existing relation
Enhanced XMI tags	<Behavioral_Elements.Use_Cases.Actor > …. <Behavioral_Elements.Use_Cases.Actor.**PhysCharCap**>

Based on the mappings for User, Task, Content and Tool records, the percentages of each mapping relation are calculated. They are summarized in the table 5.4:

Table 5.4: Percentages of mapping relations for different types of records

	exact relation	*inclusion relation*	*non-existing relation*
Task Record	8%	24%	68%
User Record	9%	27%	64%
Content Record	18%	27%	55%
Tool Record	11%	28%	61%

Although the mappings have not been verified, table 5.4 demonstrates a reasonable estimate of the situation that there is the need for integrating more UE/PUF requirement metadata into SE/UML specification. It shows that over 60% of PUF requirement metadata has no matched information in UML specification. Approximately 26% of PUF requirement metadata has certain overlapping parts. The percentage of the exact mappings is very low compared with other mapping relations. Most of them are in the identification information section. The details about the missing PUF/UE information in UML specification will be discussed in the section 5.2

5.1.2 Types of Mappings

As mentioned before, the PUF metadata for mapping could be generic or specific. So generally speaking, the mappings between a PUF XML file and a XMI file can be summarized by two types. They are specific-to-specific mappings and abstract-to-specific mappings. In other words, the Mapping Tool supports creating a mapping for either an individual tag or a group of tags. Mapping a single PUF XML tag is considered as individual tag mapping. Creating a mapping for a PUF XML tag with a number of sub tags as a whole is regarded as group mapping. All the sub tags will follow the relative abstract tag when it is moved to the identified position in the XMI file. The target XMI tags are mainly located at the package construct level and at the class construct level.

Each mapping type can be further broken down into the one-to-one, one-to-many and many-to-one subtypes. Therefore, the mappings have six categories totally. The details of each mapping category are discussed as follows.

1. Specific-to-Specific Mapping

PUF XML tags containing specific requirement metadata are normally located at the Question level which has no sub question or at the Sub Question level. The tags at these two levels focus on the details of a PUF

record from certain perspectives. This type of mapping could be further broken down into one-to-one mapping, many-to-one mapping and one-to-many mapping. Following are the illustrations of three sub types of mapping and examples.

- One-to-One Mapping

  This refers to a single PUF XML tag which has specific requirement metadata that is mapped to a single XMI tag which also has specific requirement metadata. Table 5.5 gives an example. This table demonstrates that the equivalent to the PUF XML tag <*TaskRec.DetReqSection.TaskOper.Subtasks*> in the UML structure is the <*Behavioral_Elements.Use_Cases.Include*> tag. The <*TaskRec.DetReqSection.TaskOper.Subtasks*> tag describes "the sub-tasks that may require further investigation and any other tasks that are related to this task" [6]. It also provides "additional information including information prerequisite or post-requisite to other tasks and interfaces used to communicate with other tasks" [6]. The XMI <*Behavioral_Elements.Use_Cases.Include*> tag identifies sub use cases included in a use case. The included use cases specify the additional behaviour of the base use case and some attributes describing its features. They are individual use cases on their own. Therefore, the purposes of these two tags have certain overlap but are not identical. Hence, the PUF XML tag is added into the UML metadata as a peer tag with an *inclusion mapping relation*. The enhanced XMI tag is the combination of the higher level XMI tag constructs and the lowest PUF XML tag construct. The added tag construct is shown in bold.

Table 5.5: An example of One-to-One mapping for an individual tag

PUF XML tag	<TaskRec.DetReqSection.TaskOper.Subtasks>
XMI tag	<Behavioral_Elements.Use_Cases.Include>
Mapping Relation	Inclusion relation
Enhanced XMI tags	<Behavioral_Elements.Use_Cases.Include>…. <Behavioral_Elements.Use_Cases.**Subtasks**>

- Many-to-One Mapping

  In the specific-to-specific mapping case, many-to-one mapping refers to a couple of individual PUF XML tags that are mapped to the same XMI tag with the same or different mapping

relations. It can be considered as a particular case of multiple one-to-one mappings. Table 5.5 is an example. The source tags in the example are the detailed questions in the Identification Information section. The <*TaskRec.IdInfoSection.Type*> tag identifies a type of structure model. The <*TaskRec.IdInfoSection.Description*> tag describes "the intended accomplishment of the task, especially focusing on how the task is different from other related tasks" [6]. Both of them have no further questions. It means that they are specific. The <*Foundation.Core.ModelElement*> tag represents a model element: "a model element is an abstraction of a structural or behavioral feature of the system that you are modeling and adds semantic content to a model" [46]. It has no sub tag identical to type and description. Since use cases are a type of model element and are identical to PUF tasks at an abstract level, a task's properties, type and description, can map to the same XMI tag, <*Foundation.Core.ModelElement*>. Since the other tag in this section <*TaskRec.IdInfoSection.Name*> has a partial mapping in the Model Element package construct, the requirements of group mapping are fulfilled. Group mapping will be discussed in detail in the next sub section.

Since there are no similar or relevant elements below the mapped XMI tag for these two PUF XML tags, two new UML sub elements should be created. In this case, the <*Foundation.Core.ModelElement*> tag has two added sub elements shown in Table 5.6. Their names are generated by adding the last tag construct of the PUF XML tag to the end of the XMI tag to which it is mapped. The new added tag constructs are shown in bold.

Table 5.6: An example of Many-to-One mappings for an individual tag

PUF XML tag	<TaskRec.IdInfoSection.Type>	<TaskRec.IdInfoSection.Desccription>
XMI tag	<Foundation.Core.ModelElement>	
Mapping Relation	Non-existing relation	
Enhanced XMI tags	<Foundation.Core.ModelElement> …. <Foundation.Core.ModelElement**Type**> <Foundation.Core.ModelElement.**Desccription** >	

- One-to-Many Mappings

One-to-many mapping refers to the case in which there are a couple of different mappings in a XMI file for a single PUF XML tag. The mapping relations could be the same or different. It also can be considered as a particular case of one-to-one mapping. Table 5.7 gives an example.

The <*TaskRec.LnkInfoSection.Scenarios*> tag identifies scenarios which "tie together sets of users, tasks, tools, and content chunks including contextual information" [6]. It identifies that there are a number of relationships and interaction among these components. The <*Behavioral_Elements.Collaborations.Collaboration.interaction*> tag defines the interactions within the Collaboration. Based on their definition, the PUF XML tag covers more requirements than the XMI one. Therefore, the <*TaskRec.LnkInfoSection.Scenarios*> tag can map to the <*Behavioral_Elements.Collaborations.Collaboration.interaction*> tag with an *inclusion mapping relation*.

The <*Foundation.Core.Classifier.instance*> tag is an instance of "a model element that describes objects that are behavioral or structural features in a system"[46]. Since use cases are a type of classifier, an instance of a use case is regarded as a scenario. Thus, the <*TaskRec.LnkInfoSection.Scenarios*> tag can map to the <*Foundation.Core.Classifier.instance*> tag with an *exact mapping relation.*

Table 5.7: An example of One-to-Many mappings for an individual tag

PUF XML tag	<TaskRec.LnkInfoSection.Scenarios>	
XMI tag	<Behavioral_Elements.Collaborations.Collaboration.interaction>	<Foundation.Core.Classifier.instance>
Mapping Relation	Inclusion relation	Exact Relation
Enhanced XMI tags	<Behavioral_Elements.Collaborations.Collaboration.interaction> … <Behavioral_Elements.Collaborations.Collaboration.**Scenarios**>	<Foundation.Core.Classifier.instance>

2. Abstract -to-Specific Mapping

Abstract-to-specific mapping refers to group tag mappings. The abstract PUF XML tags normally mean those ones at the Section level which have several questions but without detailed questions at the Sub Question level or those ones at the Question level which have a number of sub questions. These two kinds of PUF XML tags represent abstract but not too general concepts in the PUF methodology. Normally, their sub elements share certain features so that they can be considered as one element for mapping. Group tag mapping is useful in the process of mapping and generating the enhanced XMI file. It is less time consuming to integrate one PUF XML tag into the target XMI tag rather than do it repeatedly for all the sub elements.

Similar to individual tag mapping, the sub types of group tag mappings also could be summarized to one-to-one, many-to-one or one-to-many.

- One-to-One Mapping

  There is only one location identified in the UML structure for a PUF XML tag which has several sub elements. Once the mapping is processed, the mapped PUF XML tag's sub elements will move together with it to the target XMI tag. Table 5.8 is used as an example showing the mapping and the result. The table demonstrates that the <*UserRec.IdInfoSection*> tag is partially mapped to the <*Foundation.Core.Classifier.name*> tag which specifies "an element that describes behavioral and structural features" [46]. So according to their mapping relation, a relatively general UML element is created first under the UML element to accommodate the mapped PUF XML tag. This new UML tag is constructed by combining the original UML tag with the last part of the PUF XML tag.

  In addition, three new UML elements are generated under the original UML element to accommodate the information about the sub elements of the <*UserRec.IdInfoSection*> tag, for instance, <*UserRec.IdInfoSection.Name*>, <*UserRec.IdInfoSection.Type*> and <*UserRec.IdInfoSection.Description*>. The names of these new tags are the integration of the tag constructs representing specific questions with the first added UML element. So even in the UML structure, the added PUF information still keeps PUF's syntax and semantics. The tag constructs from PUF are shown in bold in the Enhanced XMI tag.

Table 5.8: An example of One-to-One mappings for a group of tags

PUF XML tag	<UserRec.IdInfoSection>
XMI tag	<Foundation.Core.Classifier.name>
Mapping Relation	Inclusion relation
Enhanced XMI tags	<Foundation.Core.Classifier.name>… <Foundation.Core.Classifier.**IdInfoSection>** <Foundation.Core.Classifier.**IdInfoSection**.**Name**> <Foundation.Core.Classifier.**IdInfoSection**.**Type** <Foundation.Core.Classifier.**IdInfoSection**.**Description** >

- Many-to-One Mappings

  It is possible that several PUF XML tags having sub elements correspond to the same location in the UML structure. It is an exceptional case of the one-to-one mapping that the target XMI tags in a couple of mappings are the same ones while the mapping relations could be same or different. The sub elements belonging to each mapped PUF XML tag will go under the identified UML location with its parent PUF element as a whole. The example is shown in Table 5.9. It demonstrates that the <*TaskRec.DetReqSection.Learning*> and <*TaskRec.DetReqSection.ProblemDetail*> tags have the same match in the UML structure. It is the <*Behavioral_Elements.Collaborations*> tag. The <*TaskRec.DetReqSection.Learning*> tag identifies the learning needs and capabilities of the users because different methods, feedback, time and environments could influence the users' learning results. This information helps developers know their users better. The <*TaskRec.DetReqSection.ProblemDetail*> tag records the details about the problems which developers may encounter. This information will help developers develop a more effective and efficient design solution. The <*Behavioral_Elements.Collaborations*> tag describes "how an operation or a classifier, like a use case, is realized by a set of classifiers and associations used in a specific way" [35]. It does not have any requirement related to or similar to those ones in the above PUF XML tags. In this case, the mapping relations in these two mappings are the Non-existing mapping relation. Thus two groups of XMI tags are added under the <*Behavioral_Elements.Collaborations*> tag.

One group is used to accommodate the *<TaskRec.DetReqSection.Learning>* tag and its sub elements while another group is for the *<TaskRec.DetReqSection.ProblemDetail>* tag and its sub elements. To indicate their locations in the UML structure, the new added tags' names begin with the target XMI tag and end with their tag construct names in the PUF structure. The enhanced XMI tag is *<Behavioral_Elements.Collaborations.Learning.WhoElse>*. Since the added tags follow the original tag format, the tag name is still able to show the structural information and their meanings.

Table 5.9: An example of Many-to-One mappings for a group of tags

PUF XML tag	<TaskRec.DetReqSection.Learning>	<TaskRec.DetReqSection.ProblemDetail>
XMI tag	<Behavioral_Elements.Collaborations >	
Mapping Relation	Non-existing relation	
Enhanced XMI tags	<Behavioral_Elements.Collaborations > …. <Behavioral_Elements.Collaborations**.Learning>** <Behavioral_Elements.Collaborations**.Learning.WhoElse>** <Behavioral_Elements.Collaborations**.Learning.Prerequisite >** <Behavioral_Elements.Collaborations**.Learning.Level >** <Behavioral_Elements.Collaborations**.Learning.UsedMethod >** <Behavioral_Elements.Collaborations**.Learning.Feedback >** <Behavioral_Elements.Collaborations**.Learning.Time >** <Behavioral_Elements.Collaborations**.Learning.Environment >** <Behavioral_Elements.Collaborations**.Learning.Stress >** <Behavioral_Elements.Collaborations**.ProblemDetail>** <Behavioral_Elements.Collaborations**.ProblemDetail.ProbNature >** <Behavioral_Elements.Collaborations**.ProblemDetail.UndoneGoal >** <Behavioral_Elements.Collaborations**.ProblemDetail.Efficiency >** <Behavioral_Elements.Collaborations**.ProblemDetail.Unexpectedness >** <Behavioral_Elements.Collaborations**.ProblemDetail.IsComplex >** <Behavioral_Elements.Collaborations**.ProblemDetail.IsConstraining >** <Behavioral_Elements.Collaborations**.ProblemDetail.OtherOccuringProb >** <Behavioral_Elements.Collaborations**.ProblemDetail.SuggestedImpr >**	

- One-to-Many Mappings

This sub type refers to the situation that there are a number of locations identified in the UML structure for a PUF element with sub elements. One-to-many mappings can be regarded as another exceptional case of the one-to-one mapping when the source PUF XML tags in several mappings are the same one. Table 5.10 is an example. The table illustrates that there are a couple of locations in UML where the <*TaskRec.DetReqSection.TaskOper*> tag can be mapped, for instance, the <*Behavioral_Elements.Collaborations*> tag and the <*Foundation.Core.Operation*> tag.

The <*TaskRec.DetReqSection.TaskOper*> tag describes "operational concerns by evaluating whether the interaction meets the goal of the use case, whether there are alternatives to achieve the use case, what feedback the use case should provide and how future operations might provide improvements" [6]. The <*Behavioral_Elements.Collaborations*> tag specifies "a behavioral context for using model elements to accomplish a particular task" [35]. There is no requirement to evaluate the current operations and future improvements. The requirements from the <*TaskRec.DetReqSection.TaskOper*> tag will make a beneficial complement to this UML metadata. Therefore the source PUF tag and its sub elements are transformed into the UML structure. To accommodate the new UML sub elements, two extra levels are added in the UML structure. One is for the mapped PUF XML tag while another level is for its sub elements. All the tag constructs from the PUF structure are shown in bold.

Table 5.10: An example of One-to-Many mappings for a group of tags

PUF XML tag	<TaskRec.DetReqSection.TaskOper>	
XMI tag	<Behavioral_Elements.Collaborations>	<Foundation.Core.Operation>
Mapping Relation	Non-existing relation	Inclusion relation
Enhanced XMI tags	<Behavioral_Elements.Collaborations> … <Behavioral_Elements.Collaborations.**TaskOper**> <Behavioral_Elements.Collaborations.**TaskOper.Accessibility**> <Behavioral_Elements.Collaborations.**Task**	<Foundation.Core.Operation> … <Foundation.Core.Operation.isRoot> <Foundation.Core.Operation.isLeaf> <Foundation.Core.Operation.isAbstract> <Foundation.Core.Operation.specification>

	Oper.Alternative> <Behavioral_Elements.Collaborations.**Task Oper.Feedback>** <Behavioral_Elements.Collaborations.**Task Oper.Flexibility>** <Behavioral_Elements.Collaborations.**Task Oper.HowtoDo>** <Behavioral_Elements.Collaborations.**Task Oper.IsFormal>** <Behavioral_Elements.Collaborations.**Task Oper.IsInterruptible>** <Behavioral_Elements.Collaborations.**Task Oper.IsSharable>** <Behavioral_Elements.Collaborations.**Task Oper.NeededRedundancy>** <Behavioral_Elements.Collaborations.**Task Oper.Purpose>** <Behavioral_Elements.Collaborations.**Task Oper.Subtasks>**	<Foundation.Core.Operation.method> <Foundation.Core.**TaskOper>** <Foundation.Core.**TaskOper.Accessibility>** <Foundation.Core.**TaskOper.Alternative>** <Foundation.Core.**TaskOper.Feedback>** <Foundation.Core.**TaskOper.Flexibility>** <Foundation.Core.**TaskOper.HowtoDo>** <Foundation.Core.**TaskOper.IsFormal>** <Foundation.Core.**TaskOper.IsInterruptible>** <Foundation.Core.**TaskOper.IsSharable>** <Foundation.Core.**TaskOper.NeededRedundancy>** <Foundation.Core.**TaskOper.Purpose>** <Foundation.Core.**TaskOper.Subtasks>**

The <*Foundation.Core.Operation*> tag represents "a service that can be requested from an object to effect behavior" [35]. It specifies a series of steps to perform the task. One of the sub tags, <*Foundation.Core.Operation.HowtoDo* >, specifies how currently the task is done. The sub tag of the <*Foundation.Core.Operation*> tag, <*Foundation.Core.Operation.method*>, declares "how to realize one or a set of operations of the use case" [35]. From the tags' meanings, there is certain overlap. Their mapping relation should be an *inclusion mapping relation.* As a result, the PUF XML tag is also integrated into the <*TaskRec.DetReqSection.TaskOper*> tag by combining both the XMI tag construct and PUF XML ones, for instance <*Foundation.Core.Operation.TaskOper.Accessibility*>.

5.1.3 Results of Mappings

Using the Mapping Tool for PUF to UML mapping found that:

- All pre-identified mapping situations (identified in Chapters 3 and 4) were needed to perform the mapping
- No additional mapping situations were needed to perform the mapping
- The Mapping Tool was able to handle all the mapping needs of PUF to UML mappings

5.2 Missing Information in UML

The percentage of each *exact mapping relation*, *inclusion mapping relation* and *non-existing mapping relation* is reported in sub section 5.1.1 according to the results of the mapping shown in the Appendix 2, 3, 4 and 5. It confirms that over half of the mappings having the *non-existing mapping relation*. This observation demonstrates that PUF requirement metadata can supplement UML specification with a considerable amount of HCI/UE information. This section serves to identify the missing or incomplete HCI/UE information by analyzing and observing the results received from the Mapping Tool. The details of the observation are discussed based on the syntax of the PUF requirement metadata. It will use the possible mappings of the PUF task and user records as examples to demonstrate how UML specification could be improved by usability requirements.

5.2.1 Mapping of Identification Information

The Identification Information section in PUF requirement metadata is used to identify a type record and give a unique name and detailed description to a record. UML contains similar specifications. It can be found under the UML elements in the Core sub package which describes the static features of UML elements. Table 5.11 shows that the PUF XML tags in the Identification Information section of a User record have partially (*inclusion mapping relation*) identical matches in UML.

Table 5.11: The mappings of the PUF XML tags in the Identification Information section

PUF XML tag	**XMI tag**	**Mapping relation**
<UserRec.IdInfoSection>	<Foundation.Core.Classifier.name>	Inclusion relation
<UserRec.IdInfoSection>	<Foundation.Core.ModelElement.name>	Inclusion relation

The table demonstrates that the concepts in this PUF section are directly supported in UML. From the perspective of high level mapping, a User record is mapped to an Actor package construct in a Use case sub package. When the mapping goes further and in more detail, PUF's identification requirement metadata are added into the Core sub package because they are static information such as attributes or features.

5.2.2 Mapping of Linkage Information

For each record, a PUF Linkage Information section is used to identify its relationships with other types of records on a full-scale. The relationship between a model element and others considered in UML are not

complete. Take an Actor package construct as an example, UML illustrates its behavior and its participants. However, compared with PUF requirement metadata, some relationships, for example, the one between users and their tools are missing (*non-existing mapping relation*). The User record is used as an example to illustrate the situation. The mappings for the PUF XML tags in the linkage information section are shown in Table 5.12.

Table 5.12: The mappings of the PUF XML tags in the Linkage Information section

PUF XML tag	**XMI tag**	**Mapping relation**
<UserRec.LnkInfoSection>.	<Behavioral_Elements.Use_Cases.Actor>	Non-existing relation
<UserRec.LnkInfoSection>.	<Behavioral_Elements.Collaborations.ClassifierRole>	Non-existing relation
<UserRec.LnkInfoSection.Who>	<Foundation.Core.Classifier>	Inclusion relation
<UserRec.LnkInfoSection.What>	<Foundation.Core.Classifier>	Inclusion relation
<UserRec.LnkInfoSection.How>	<Foundation.Core.Operation>	Inclusion relation
<UserRec.LnkInfoSection.WithWhichContent>	<Foundation.Core.Classifier.feature>	Inclusion relation
<UserRec.LnkInfoSection.Scenarios>	<Behavioral_Elements.Common_Behavior.Instance>	Exact relation

From the above mapping relations, it can be sseen that most of the PUF tags have equivalences or relevant UML elements. The *Classifier* UML package construct represents all the elements having behavioral and structural features including actor, use case, data type. Hence the PUF linkage requirement metadata such as *who* and *what* could become its sub type. As for the *How* and *WithWhichContent* metadata, they identify the tools and the content chunks that may possibly be used by this user group. Both metadata can be considered as a supplement to the existing static and behavioral features possessed by this actor.

An *actor* in UML defines a coherent set of roles that users of an entity can play when interacting with the entity. There is no well-organized template to record the elements related to the *Actor* package construct which is the mapping for the User record. The same situation exists in the *ClassifierRole* package construct in the *Collaborations* sub package. Table 5.12 indicates that the PUF requirement metadata are able to repair this deficiency by integrating all the linkage metadata into the *Actor* and the

ClassifierRole package constructs. All the elements related to this actor are recorded in the same section. It facilitates the retrieval of the requirement metadata on an element's linkage information.

5.2.3 Mapping of Environmental Information

There are no very systematic and specific requirement metadata about user's usage environment, although some constraints include certain information about it. Different from UML, PUF considers all the factors from different aspects that may influence users' performance of tasks. PUF focuses on identifying a range of possibilities of when, when, why and how often this task could be accomplished rather than restricting carrying out the task. The development of a system using the PUF methodology is based on versions. Hence it is very necessary and useful to include the requirement metadata about its future versions. These requirement metadata contained in the environment section assist developers to develop a complex system from a simple one by making proper notes. They also can provide constructive suggestions for avoiding problems in the future version and for future improvement. Because of this, PUF can be used to develop a usable system which is able to meet user needs to the full. The Environment Information section in a Task record is employed as an example to demonstrate the mappings (see Table 5.13).

Table 5.13: The mappings of the PUF XML tags in the Environmental Information section

PUF XML tag	**XMI tag**	**Mapping relation**
<TaskRec.EnvInfoSection.When>	<Foundation.Core.ModelElement>	Non-existing relation
<TaskRec.EnvInfoSection.When>	<Foundation.Core.Classifier.feature>	Non-existing relation
<TaskRec.EnvInfoSection.When>	<Behavioral_Elements.Collaborations.interaction.context>	Non-existing relation
<TaskRec.EnvInfoSection.Where>	<Foundation.Core.ModelElement>	Non-existing relation
<TaskRec.EnvInfoSection.Where>	<Foundation.Core.Classifier.feature>	Non-existing relation
<TaskRec.EnvInfoSection.Where>	<Behavioral_Elements.Collaborations.interaction.context>	Non-existing relation
<TaskRec.EnvInfoSection.HowMuch>	<Behavioral_Elements.Collaborations.interaction.context>	Non-existing relation
<TaskRec.EnvInfoSection.HowMuch>	<Foundation.Core.Classifier.feature>	Non-existing relation
<TaskRec.EnvInfoSection.Why>	<Foundation.Core.ModelElement>	Non-existing relation

The *non-existing mapping relation* in Table 5.13 indicates that the PUF XML tags in the Environment Information section cannot map directly to UML concepts. The *When*, *Where* and *How much* requirement metadata specify the working environment within which users accomplish the task. Since the working environment may facilitate or constrain the performance of the task, it is necessary to include the detailed environmental information in the context within which actors interact with system. The PUF environment requirement metadata is able to help developers design an achievable use case in various situations by considering all the environmental factors which may affect the performance, for example, location, time and the frequency of use. These PUF requirement metadata can provide further information or categorize the environmental requirements in UML, for example, the *context* class construct in the *Collaborations* sub package. Thus a new level is created under the class package level in XMI structure to accommodate these PUF requirement metadata. The enhanced *Context* class construct looks like in Table 5.14. The PUF tag constructs are shown in bold.

Table 5.14: The enhanced UML XML tags with PUF environmental requirement metadata

Enhanced UML XML tags	Meanings
<Behavioral_Elements.Collaborations.interaction.context.**When.all**>	The new elements focus on various "timing concerns" [6]. It is likely that the task with high frequency of use has less usability concerns for regular users. **PeakUsageDistribution** determines "the capacity requirements of the tool used for performing the task" [6]. **SpentTime** measures "the frequency of use in terms of cumulative time used" [6].
<Behavioral_Elements.Collaborations.interaction.context.**When.FrequencyofUse**>	
<Behavioral_Elements.Collaborations.interaction.context.**When.ParticularTime**>	
<Behavioral_Elements.Collaborations.interaction.context.**When.PeakUsageDistribution**>	
<Behavioral_Elements.Collaborations.interaction.context.**When.PerformanceReq**>	
<Behavioral_Elements.Collaborations.interaction.context.**When.SpentTime**>	
<Behavioral_Elements.Collaborations.interaction.context.**Where.all**>	The new elements focus on the working environment within which the task is performed. The ergonomics of **workstation configurations** can "improve the efficiency of performing the task" [6]. **EffectOnPerf** identifies "the factors which may hinder or support accomplishing the task in those locations" [6]. **Environments** specifies "the ideal environmental conditions" [6].
<Behavioral_Elements.Collaborations.interaction.context.**Where.Configurations**>	
<Behavioral_Elements.Collaborations.interaction.context.**Where.EffectOnPerf**>	
<Behavioral_Elements.Collaborations.interaction.context.**Where.Environments**>	
<Behavioral_Elements.Collaborations.interaction.context.**Why.all**>	The new elements are used to identify "the factors that will influence the feasibility and acceptability of future
<Behavioral_Elements.Collaborations.interaction.context.**Why.HowFit**>	

<Behavioral_Elements.Collaborations.interaction.context.**Why.IsEssential**>	designs" [6]. They consider how the task fits into the overall development, whether or not it is essential and whether or not costs exceed benefits. They also specify any prerequisite relationships with other tasks...
<Behavioral_Elements.Collaborations.interaction.context.**Why.Prerequirsite**>	
<Behavioral_Elements.Collaborations.interaction.context.**Why.CostEffective**>	

Besides the mappings noted above, all the PUF concepts which have no mappings are associated with a stereotype. They will become a sub element of the <*Foundation.Extension_Mechanisms.Stereotype*>.

5.2.4 Mapping of Detailed Information

As for the Detailed Information section, according to the mapping lists for the task and the user records, the UML specification is left unsatisfied. Take the Actor concept in a use case diagram for example. It is the mapping of a PUF user record at high level. As far as the XMI file is concerned, the Actor package construct in the *Use Case* sub package has no class construct. It looks like the following:

<Behavioral_Elements.Use_Cases>

<Behavioral_Elements.Use_Cases.UseCase>

<Behavioral_Elements.Use_Cases.UseCase.extend>

<Behavioral_Elements.Use_Cases.UseCase.extend2>

<Behavioral_Elements.Use_Cases.UseCase.include>

<Behavioral_Elements.Use_Cases.UseCase.include2>

<Behavioral_Elements.Use_Cases.UseCase.extensionPoint>

<Behavioral_Elements.Use_Cases.Actor>

<Behavioral_Elements.Use_Cases.UseCaseInstance>

All the information about an actor is presented by the tag's attributes. They could be sub elements in the Model Element package construct or in the Classifier package construct. For instance,

<Foundation.Core.ModelElement.name>

<Foundation.Core.ModelElement.visibility>

<Foundation.Core.ModelElement.isRoot>

……

<Foundation.Core.Classifier.feature>

<Foundation.Core.Classifier.participant>

<Foundation.Core.Classifier.instance>

……

These attributes mainly concern a user's static and structural features. They also include those ones about their behaviors. UML does not present a particular template for specifying properties of the actor and the context within which it performs the tasks.

The goal of PUF which is to develop a usable system which can be used by people with the widest range of capabilities determines that the special attention is given to users. Contrary to UML, PUF focuses on analysis of all kinds of differences between users because different users have their own specified requirements on a system. For example, users have various computer skills and knowledge about a system. So it is likely that some users need online help when they perform a task while those ones are familiars with the system may find it unnecessary. By considering users' specific requirements, there is more chance that the resulting system is able to achieve users' specific goals. Therefore, PUF specifications should keep record of user-centric information. Table 5.15 shows the details:

Table 5.15: The PUF XML tags for detail requirements of a user record

PUF XML tags	Meaning
<UserRec.DetReqSection.PhysCharCap>	The **PhysCharCap**'s sub questions identify "various physical characteristics and capabilities which the users may have". They contain four categories: -"Distinguishing characteristics, e.g. age, sex"; -"Sensing capabilities, e.g. sight, hearing, taste, smell, touch"; - "Physical capabilities, e.g. voice, touch"; - "Performing capabilities, e.g. speed, accuracy" [6].
<UserRec.DetReqSection.PhysCharCap.DistinguishingChar>	
<UserRec.DetReqSection.PhysCharCap.SensingCap>	
<UserRec.DetReqSection.PhysCharCap.PhysCap>	
<UserRec.DetReqSection.PhysCharCap.PerformingChar>	
<UserRec.DetReqSection.MentalCharCap>	The **MentalCharCap**'s sub questions identify "various mental characteristics of users which may affect the way in which they react to a variety of interactions and interfaces" [6]. They include: -"Personality traits, e.g. extroversion / introversion"; - "Emotional capabilities, e.g. attitudes, beliefs"; -"Cognitive capabilities, e.g. perception, reasoning"; - "Memory capabilities, e.g.sensory"; - "Expertise, e.g. intelligence, education" [6].
<UserRec.DetReqSection.MentalCharCap.Personality>	
<UserRec.DetReqSection.MentalCharCap.EmontionalCap>	
<UserRec.DetReqSection.MentalCharCap.CognitiveCap>	
<UserRec.DetReqSection.MentalCharCap.MemoryCap>	
<UserRec.DetReqSection.MentalCharCap.Expertise >	
<UserRec.DetReqSection.SocialCharCap>	The **SocialCharCap**'s sub questions identify "various characteristics that may influence the way in which an individual interacts with others in a group environment" [6]..
<UserRec.DetReqSection.SocialCharCap.Recognition>	
<UserRec.DetReqSection.SocialCharCap.Types>	
<UserRec.DetReqSection.SocialCharCap.Implementation>	
<UserRec.DetReqSection.SocialCharCap.Application>	
<UserRec.DetReqSection.GroupCharCap>	The **GroupCharCap**'s sub questions identify "characteristics that describe influences of groups on individual members of the group" [6].
<UserRec.DetReqSection.GroupCharCap.Membership >	

<UserRec.DetReqSection.GroupCharCap.Self-concept>	
<UserRec.DetReqSection.GroupCharCap.Orientation >	
<UserRec.DetReqSection.GroupCharCap.Relations>	

Unfortunately, the *Actor*, *Classifier* and *Model Element* package constructs make no mention of user-centric information. Numerous PUF sub questions relating to the details about users cannot find the proper mappings in UML. It is crucial to add these PUF requirement metadata about users to UML specifications on the ground of usability. Therefore, it is suggested that user needs, their knowledge, their interactions with tool, their expectations, their capabilities, their personalities, their ability to process information should be taken into consideration when UML specifications are developing. The Mapping Tool makes it possible to integrate the usability related requirements into the UML one. Following is a part of the result of the Mapping Tool. The XMI tags are enhanced by combining with PUF tag constructs. PUF tag constructs are bold.

< Foundation.Core.Classifier.feature>

< Foundation.Core.Classifier.feature.**GroupCharCap.all>**

< Foundation.Core.Classifier.feature**.GroupCharCap.Membership>**

< Foundation.Core.Classifier.feature.**GroupCharCap.Orientation>**

< Foundation.Core.Classifier.feature.**GroupCharCap.Relations>**

< Foundation.Core.Classifier.feature.**GroupCharCap.Self-concept>**

< Foundation.Core.Classifier.feature **MentalCharCap.al**l>

< Foundation.Core.Classifier.feature**.MentalCharCap.CognitiveCap>**

< Foundation.Core.Classifier.feature.**MentalCharCap.EmontionalCap>**

< Foundation.Core.Classifier.feature.**MentalCharCap.Expertise>**

< Foundation.Core.Classifier.feature.**MentalCharCap.MemoryCap>**

< Foundation.Core.Classifier.feature.**MentalCharCap.Personality>**

< Foundation.Core.Classifier.feature**.PhysCharCap.al**l>

< Foundation.Core.Classifier.feature.**PhysCharCap.DistinguishingChar>**

< Foundation.Core.Classifier.feature.**PhysCharCap.PerformingChar>**

< Foundation.Core.Classifier.feature**.PhysCharCap.PhysCap>**

< Foundation.Core.Classifier.feature.**PhysCharCap.SensingCap>**

< Foundation.Core.Classifier.feature.**SocialCharCap.all>**

< Foundation.Core.Classifier.feature.**SocialCharCap.Application>**

< Foundation.Core.Classifier.feature.**SocialCharCap.Implementation>**

< Foundation.Core.Classifier.feature.**SocialCharCap.Recognition>**

< Foundation.Core.Classifier.feature.**SocialCharCap.Types**>

5.2.5 Suggestions on Mapping PUF Components to UML Components

During the process of creating the mapping via the Mapping Tool, I explored and identified more mapping possibilities between high level PUF components and UML ones. In this sub-section, the suggestions I make extend and supplement Figure 2.5. To differentiate the new mappings from the original ones, the thick lines in Figure 5.1 are used to highlight the new mappings found in my research.

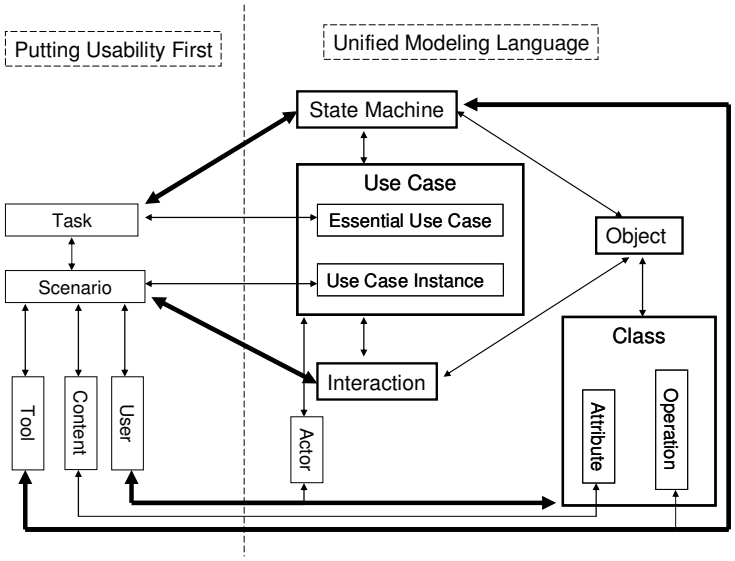

Figure 5.1: Extended high level relationships between PUF and UML components

Figure 2.3 depicts the relationships between a use case diagram and other types of diagrams. Figure 5.1 shows that the PUF components can also be mapped to UML components in the other types of diagrams besides use case diagrams.

- A state machine diagram can describe the internal operation of a single task. Thus a task relates to a state machine diagram.
- The interaction diagram describes a scenario by showing the messages passed between the system and anything that interacts with it. Therefore, a scenario can be modelled using an interaction diagram to show the communication among tools, users and contents.
- Users (actors) can relate to a class diagram because an actor is a type of classifier which can be represented by the class diagram.

- A state machine diagram can define the behaviour of one or more operations which any class has. Therefore, each operation in a class could have an associated state machine. Since a tool is mapped to a operation, the tool can also be described by a state machine diagram.

The above analysis and discussion about the results from the Mapping Tool lead to the conclusion that UML elements do not have certain information which can be provided by the PUF XML file. Integrating PUF XML tag information into UML can efficiently enrich a XMI file and, to take a step further, may lead to better UML models with UML CASE tools which are able to convert XML files to UML diagrams.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

This thesis provides an important start towards automating the integration of UE requirement metadata within SE metadata.

- Section 6.1 summarizes what was done for this thesis.
- Section 6.2 emphasizes the importance and the benefits of the general approach presented in the thesis.
- Section 6.3 summarizes the contribution of the research.
- Section 6.4 makes recommendations for related work in the field.
- Section 6.5 addresses the directions for future research, including suggestions for improvements.

6.1 Summary

This thesis addressed one of the UML methodology's weaknesses, that it is function-centered and technology-centered without sufficient HCI/UE requirements. I developed a general solution for integrating HCI/UE requirement metadata with SE metadata via XML. The significance of the generality of this method will be discussed in the section 6.2. This thesis also presented a specific approach for PUF and UML to prove the feasibility of this general solution. The Mapping Tool was developed to carry out a series of tasks to map and combine PUF requirement metadata with UML requirement metadata. The results of the Mapping Tool were analyzed to show what kind of usability-related requirement metadata is left out in UML specification and how PUF requirement metadata can improve it.

6.2 A General Approach for Integrating HCI Requirements into SE Specification

In this thesis, I designed a general approach to solve the problem that UML is lacking sufficient HCI/UE requirements and presented it in section 2.7. The solution is to use XML as a base for mapping between HCI/UE requirement metadata and SE requirement metadata so that SE requirement metadata can be enhanced with usability metadata. Rather than being limited to PUF and UML, the approach I designed can be applied to any XML-based HCI/UE and/or SE methodologies which have hierarchical requirement structures. This generality has the benefit of using XML to encode both sets of specifications. Firstly, XML enables us to perform mapping between two methodologies' specifications by doing so between two

different XML files. Secondly, this approach can be applied to a variety of HCI/UE and/or SE methodologies as long as they are presented in a fully specified XML tag format showing the syntax and the semantics of the methodologies. The fully specified XML tag format facilitates creating accurate mappings between a HCI/UE XML file and a SE XML file and making the enhanced SE XML tags meaningful as well.

The Mapping Tool was developed to test the feasibility of this generic method by integrating a PUF XML file to a XMI file (or even any two sets of XML based specifications). It provides facilities to assist developers to explore as many mappings as possible and allows them to produce mappings efficiently. It also supports editing, printing and saving existing mappings. The secondary outcome of the Mapping Tool is the enhanced the XMI file by integrating the PUF XML tag into the XMI tag according to the mapping relation.

6.3 Contributions

The following is a summary of the major contributions which this research makes:

- The development of a general approach for integrating HCI/UE requirement metadata into SE metadata is the major contribution of this thesis. This general approach can make full use of all kinds of existing HCI/UE methodology specifications to improve SE specifications by combining HCI/UE requirement metadata into SE metadata. Because a large amount of usability issues are addressed in the HCI/UE specifications, the enhanced SE specifications are more usable than the original ones. This approach can assist software engineers struggling to balance a function-centered perspective and a user-centered one. SE specifications supplemented with sufficient usability information can lay a solid foundation for designing a usable system and reduce the likelihood that usability problems occur in the resulting system. The end result is that usability of the system may be increased.
- The Mapping Tool I developed for PUF requirements and UML specifications is another important contribution of this thesis. The Mapping Tool is placed in the USERLab Website and available at http://userlab.usask.ca/UE-SE-mapping.html. This web page includes a general introduction to mapping UE requirement metadata to SE requirement metadata and contains links to the Mapping Tool which has its own page http://userlab.usask.ca/mapping-tool.html and

links to the guidance of doing good mappings which has its own page http://userlab.usask.ca/mapping-tool-help.html. The details about the design are discussed in chapter 4. It serves as a demonstration of the feasibility of the generic approach for integrating HCI/UE requirement metadata into SE metadata. The Mapping Tool supports all the operations of this method, including generating the mappings between PUF requirement metadata and UML requirement metadata, documenting the mappings and producing an enhanced UML metadata integrated with the PUF metadata according to their mapping relations. Besides, the Mapping Tool provides useful information for assisting developers with generating and managing the mappings. It ensures that all the PUF requirement metadata are taken into consideration and informs the person doing the mapping of the progress of the mapping via its user-friendly user interface. The results of the Mapping Tool, e.g. the preliminary mappings between PUF and UML requirement metadata and the enhanced UML requirement metadata, will show developers the way of gathering usability-related project requirements. So far, the Mapping Tool specifically works for PUF and UML. The Mapping Tool can be evolved to support any HCI/UE or SE methodology in future as long as its requirements have a hierarchical structure.

- I investigated and analyzed the meanings and the structure of UML elements and PUF ones in order to find the correspondences between them at both high and low levels. I expanded the original mapping relation at a high level defined by J.Carter by identifying more UML concepts to which PUF concepts can be mapped. Appendix 8 and 9 show the complete list of PUF XML tags, XMI tags, their meanings and what they are used for. Figure 5.1 shows additional UML concepts which I identified that PUF concepts at a high level can be mapped to. As for the mapping at the low level, the preliminary PUF-to-UML requirement metadata mappings were developed although they need to be validated in future. Appendix 10 provides a guide for doing good mappings using the Mapping Tool and suggests a procedure for validating mappings. These detailed mappings are not limited to use case diagram. More UML concepts describing the static and behavioral features of model elements are involved in mappings, for instance, the elements in the *Core* sub package, those ones in the *State Machine* sub package and in the

Collaboration sub package. By doing so, the usability-related information is included in various UML models so that they can assist software engineer with the development of a system at different stages. The research carried out at both high and low levels makes the preliminary mappings more accurate and more specific and makes the method suggested in this thesis more robust. Appendix 4, 5, 6 and 7 show the preliminary mappings I created for the PUF task, user, content and tool records. Since the UML requirement metadata is based on the UML 1.3 models, these preliminary mappings might not apply to UML 2. The contribution my mappings make includes:

  - Demonstrate the need for mapping UE/HCI requirements to SE specification and the feasibility of doing mapping.
  - Find that there is approximately 12% exact mapping, 26% inclusion mapping, 62% non-existing mapping. Although the mappings identified in this thesis are not verified, they provide a reasonable estimate of the situation that there is the need for more usability-related information in UML 1.3 models.

- I evaluated possible mappings from the Mapping Tool and gave a full and detailed discussion of the possible types of the identified mappings in chapter 5. I also demonstrated what aspects of usability can apply to various UML concepts and diagrams and how to perform those applications using the Mapping Tool.

6.4 Recommendations

Based on the observations of the results of the Mapping Tool, there is a need for UML, XMI and the Mapping Tool to further evolve.

- Although my approach presented in the thesis is intended to be generic, as far as my database design is concerned, it specifically serves PUF and UML. The same is true for the Mapping Tool. However, the results of the Mapping Tool show that this design is a good start. It would be useful to evolve the current specific database design into a generic one so that the corresponding Mapping Tool can be usable for differently structured sets of methodology requirement metadata. To make the database design generic, several rules are recommended as follows:

- HCI/UE and SE requirement specifications should have hierarchical structures so that the XML tags used to represent their requirement metadata can be transformed into a fully specified style. The fully specified tag format is my design in this thesis and it is also used by some UML CASE tools.
- The fully specified XML tag format used in this thesis should be extended to include HCI/UE or SE requirement methodology names. Therefore, the new XML tag format will be composed by five tag constructs. The first one is for identifying a methodology and the rest of them are for identifying a specific requirement metadata in the methodology. After the tag style is determined, a new column should be added into the corresponding table in the database to keep track of methodology names.
- The columns in the database for accommodating HCI/UE and SE XML tag construct names should be given more generic names so that they can be applied to represent hierarchical structures of methodologies. For instance, these tag construct names could be 'Category', 'SubCategory', 'Division', and 'SubDivision' so the whole tag format can expressed as follows:

  <MethodologyName.Category.SubCategory, Division.SubDivision>

  They also can be more abstract as follows:

  <MethodologyName.Level1.Level2, Level3.Level4>

- Since the existing UML specifications do not contain enough HCI/UE information, it should be extended to include a variety of usability information. The UML stereotypes mechanism can be adopted as a basic structure to accommodate the specific usability-related requirement metadata.
- UML models require further investigation and standardization. The UML elements in the current XMI file fall into two main parts. One is the Foundation package which contains structural or static modeling concepts. Another one is the Behavioral Elements package which contains behavioral or dynamic modeling concepts. It consists of the Common Behavior package, the Collaborations package, the Use Cases package, the State Machines package and the Activity Graphs package. The reason why these UML packages were chosen is that they have certain relationships with the use case diagram to which most of PUF records are mapped at the high

level. Based on the research of this thesis, it is necessary to add more UML elements from other UML diagrams, for instance the Actions package and Data type packages into the XMI file. Furthermore, XMI tags' attributes are not included in the XMI file although they provide very general requirement metadata. They could be potential options for performing mapping in the future. They can be added into the XMI file as sub elements of a XMI tag. Normally, these attributes are different from other UML packages so their names totally differ from those ones of sub elements. As a result, it is impossible to group the elements just by their tag components' names. Alternative mechanisms for grouping and sorting must be applied to the XMI tags table of the database. The solution suggested here is to introduce artificial orderings by adding a set of number attributes into the table. These numbers are employed in the same way as the syntactic number in the PUF XML tags table. They contain structural information. Hence sorting by the numbers is able to keep XMI tags in the user-defined order and to show their structural hierarchies.

- The mappings identified in the thesis have not been validated. There is the need for validating them in future. This work requires the collaboration between software engineers and usability engineers. Following are some of my recommendation based on my experience of mapping:
  - Usability engineers should recognize what UML diagrams are involved in each development stage and what UML diagrams are most frequently used. Software engineers should assist them with finding out this information. Doing so makes sure that UE/HCI requirements are always available for software engineers in the course of development. They can prepare documentation in case software engineers require more detailed information. Usability engineers should ensure that all the UE/HCI requirement metadata are mapped.
  - As for software engineers, they should comprehend UE/HCI requirement metadata correctly. They can examine the mappings by using them in a specific project. With detailed project requirements, software engineers can better understand the meanings and functions of UE/HCI requirement metadata. If the mappings are not held true, they can communicate with usability engineers suggesting more appropriate locations in UML specifications.

- Usability engineers and software engineers can do mapping separately. Comparing and analyzing their results finds out the mappings which are consistent. Usability engineers and software engineers should discuss those one which are contentious and come to compromise mappings. It would be better to have developers who have both UE/HCI and SE background verify mappings.

6.5 Future Work

The following discussion identifies the areas for possible future work:

- Implement new features to make the Mapping Tool more functional and more efficient.
- Develop additional tools, especially the Translation tool

6.5.1 Further Implement Features in the Mapping Tool

There are still a few problems needed to be resolved and some features needed to be developed for future use.

The current file names are hardwired into the application. Although creating multiple mapping files at a time is doable, it would be useful to load their methodology requirement metadata files to create mappings for various PUF records. Work is now underway to construct a new PUF MAT which will provide methodology metadata in the fully specified format and its corresponding DTD used to validate XML files. The Mapping Tool will potentially support to choose different fully qualified methodology requirement metadata to do mapping.

The second improvement needed in the Mapping Tool is the presentation of a large amount of unmapped PUF XML tags. It takes too long to look for an unmapped PUF XML tag in the SELECT page. The recommended solution is to create an expandable tree. The tree can be fully expanded or fully collapsed if necessary. It would make navigation easy among the PUF elements. Furthermore, the tree would clearly show the hierarchy of all the records to provide a big picture of the PUF records. It would facilitate understanding of the structure of the PUF records.

6.5.2 Develop Additional Tools

From the perspective of the whole UE project, more tools are needed to complete the whole process of integration in the future, for instance, a Translation Tool and a UML CASE tool. XML should be used as basis for all CASE tools to allow mapping and integration.

The purpose of the Translation Tool is to apply the outcomes from the Mapping Tool to the PUF project instance so that UE/PUF requirements can be integrated in SE/UML specifications. It should be able to translate project requirements at least once per project, after the Mapping Tool generates the mapping list and the enhanced XMI file. The mapping of requirement metadata shows the default and validated locations in SE/UML specifications where the specific project requirements should go. It could be the basis of translation. Considering the variations of project requirements, besides the basic translation, the Translation Tool should support adding project-dependant HCI/UE project requirements into SE/UML specifications in the light of specific conditions, which change from time to time and from place to place. The Translation Tool should also allow developers to finish translating one project in as many sessions as they require and to update the added project requirements if the HCI/UE project requirements are changed. Similar to mapping, translation requires that developers to have very broad and solid knowledge about both usability engineering and software engineering methodologies. Since it is very likely that usability engineers and software engineers are not very familiar with each other's research area, it is recommended that they should work together to translate HCI/UE project requirements into SE ones.

Furthermore, there is a need to explore an approach to diagram usability-related requirements specification into UML diagrams. The approach suggested here is to use a UML tool to transform XML files into UML diagrams, for instance, a use case diagram, state machine diagram, or interaction diagram. Currently, the available UML tool is only able to transform a XML file into a class diagram. In the future, we will find this kind of UML tool or develop one by ourselves.

BIBLIOGRAPHY

[1] International Organization for Standardization. *ISO International Standard 9241-11Ergonomic requirements for office work with visual display terminals (VDTs) —Part 11: Guidance on Usability,* page 6. ISO, 1998.

[2] International Organization for Standardization, *ISO International Standard 13407 Human-centered design processes for interactive systems,* page 7. ISO, 1999.

[3] International Organization for Standardization, *ISO International Standard 9241-110 Ergonomics of Human-system Interaction—Part 110 Dialogue principles,* page 11. ISO.2005.

[4] IBM's User-Centered Design Available at https://www-01.ibm.com/software/ucd/ucd.html#ucdprinciples -Accessed on March 17, 2009.

[5] Raïssa, K.H.. Ten Guidelines for User-Centered Web Design. In *Usability Interface*, Vol 5, No. 1. Available at http://www.stcsig.org/usability/topics/articles/ucd%20_web_devel.html -Accessed on Accessed on March 17, 2009, July 1998

[6] Carter, J.A., Liu, J, Schneider, K, and Fourney, D. *Transforming Usability Engineering Requirements into Software Engineering Specifications- From PUF to UML.* 2004.

[7] Carter, J.A. Putting usability first in the design of Web sites, Proceedings of WebNet'97, (Toronto), 142-148. Nov. 1997.

[8] Mayhew, D. J. *The usability engineering lifecycle: a practitioner's handbook for user interface design*, San Francisco, Calif.: Morgan Kaufmann Publishers, 1999

[9] Norman, D. A., Draper, S. W.*, User centered system design: new perspectives on human-computer interaction***,** Hillsdale, N.J. : L. Erlbaum Associates, 1986

[10] Bray, T., J. Paoli, C. M., and Maler, E. (editors) Extensible Markup Language (XML) 1.0 (Four Edition). W3C recommendation available at: http://www.w3.org/TR/REC-xml/#dt-doctype - Accessed on August 13, 2008.

[11] Chak, A. *Strategy Product Review | Usability Tools: A Useful Start (Web Techniques).* Available at: http://www.ddj.com/dept/architect/184413683-Accessed on August 13, 2008.

[12] Dong, J., Martin, S., and Waldo. P. A user Input and Analysis Tool for Information Architecture. *CHI* 2001 • 31 MARCH - 5 APRIL. Available at: http://www.stcsig.org/usability/topics/articles/EZSortPaper.pdf-Accessed on August 13, 2008.

[13] Object Management Group (OMG), UML 2.1 XSD files. Available at: http://www.omg.org/cgi-bin/doc?ptc/2006-04-05-Access on August 13, 2008.

[14] Object Management Group. OMG XML Metadata Interchange (XMI) Specification. Version 1.1. Available at: www.omg.org/cgi-bin/doc?formal/2000-11-02. November 2000- Accessed on August 13, 2008.

[15] Gentleware, Poseidon for UML Community edition. Available at: http://www.gentleware.com/index.php?id=ce- Accessed on August 13, 2008.

[16] Tigris. Argouml Project home. Available at: http://argouml.tigris.org/- Accessed on August 13, 2008.

[17] Omondo. EclipseUML. Available at: http://www.eclipsedownload.com/- Accessed on August 13, 2008.

[18] Object Management Group, UML® Resource Page, Available at: http://www.uml.org/- Accessed on August 13, 2008.

[19] Wikipedia, Unified Modeling Language. Available at: http://en.wikipedia.org/wiki/Unified_Modeling_Language - Accessed on August 13, 2008.

[20] Holt, J. *UML for Systems Engineering: Watching the Wheels, Second Edition* The Institute of Electrical Engineering, London, United Kingdom 2004.

[21] Eriksson, H.E. and Penker, M. *Business Modeling with UML: Business Patterns at Work,* John Wiley & Sons, 2000.

[22] Fowler, M. *UML Distilled: a brief guide to the standard object modeling language*. – 3rd Ed. Reading, MA: Addison-Wesley. 2003.

[23] Pinheiro da Silva, P. and Paton, N. W., *User Interface Modelling with UML.* Avaiable at http://www.ksl.stanford.edu/people/pp/papers/PinheirodaSilva_IMKB_2000.pdf -Access on August 13 2008

[24] Dobing, B. and Parsons, J. How UML is used, *Communications of the ACM* Volume 49, Issue 5 (May 2006) Two decades of the language-action perspective Pages: 109 – 113, 2006

[25] Greco de Paula, M. Conveying human-computer interaction concerns to software engineers through an interaction mode. In *Proceedings of the 2005 Latin American conference on Human-computer interaction CLIHC '05*. October 2005.

[26] Pinheiro da Silva, P. and Paton, N. W. UMLi: The Unified Modeling Language for Interactive Applications. Available at http://www.ksl.stanford.edu/people/pp/papers/PinheirodaSilva_SOFTWARE_2003.pdf- Access on August 13 2008.

[27] Nunes, N. J. *Object Modeling for User-centered Development and User-interface Design: The Wisdom Approach*, 301 pags., Tese de Doutoramento em Engenharia de Sistemas, Especialidade de Informática, UMa, Jul. 2001,

[28] Kruchten, P., Ahlqvist, S., and Bylund, S. User Interface Design in the Rational Unified Process. In: *Object Modeling and User Interface Design*, London: Addison Wesley Longman, 2001, pp. 161-196.

[29] Rubenstein, R. and Hersh, H. *The Human Factor: Designing Computer Systems for People*, Maynard, MA: Digital Press, 1984.

[30] Carlson, D. *Modeling XML applications with UML : practical e-business applications,* Boston, Mass. Addison-Wesley, c2001

[31] Uche Ogbuji, *Thinking XML: XML meets semantics, Part 1.* Available at: http://www.ibm.com/developerworks/xml/library/x-think1.html- Access on August 13, 2008.

[32] Cover, R. *XML and Semantic Transparency,* Available at: http://xml.coverpages.org/xmlAndSemantics.html - Access on August 13, 2008

[33] Grose, T. J., Doney, G. C. and Brodsky, S. A. *Mastering XMI: Java Programming with XMI, XML, and UML.*New York: John Wiley & Sons, 2002

[34] Generic Understanding Programs (GUPRO), UML DTD. Available at: http://www.gupro.de/mirror/xig/xmi.the_uml_dtd/xmi_the_uml_dtd.htm- Accessed on August 13, 2008

[35] Object Management Group. UML1.3 Semantics. Available at: http://www-inf.int-evry.fr/COURS/UML/UML1.3.pdf - Access on August 13, 2008

[36] Booch, G., Rumbaugh, J. and Jascobson, I. *The Unified Modeling Language User Guide*, Addison-Wesley Longman, 1998

[37] Seidler, K.O. Apache Friends*,* Available at: http://www.apachefriends.org/en/index.html - Access on August 13, 2008

[38] The Apache Software Foundation, Apache HTTP Server Project, Available at: http://httpd.apache.org/ - Access on August 13, 2008

[39] MySQL AB. MySQL. Available at: http://www.mysql.com/ - Access on August 13, 2008

[40] The PHP Group. PHP. Available at: http://www.php.net/ - Access on August 13, 2008

[41] Dray, S. M. The Importance of Designing Usable Systems In: *interactions* magazine, January, 1995 (Volume 2, issue 1), pages 17 - 20. Available at: http://www.dray.com/articles/usablesystems.html-Access on August 13, 2008

[42] Wikipedia, XML. Available at: http://en.wikipedia.org/wiki/XML - Accessed on August 13, 2008.

[43] Usernomics**,** User Interface Design, Available at: http://www.usernomics.com/user-interface-design.html - Accessed on October 13, 2008.

[44] International Organization for Standardization. *ISO Technical Specification 16982:Ergonomics of human-system interaction – Usability methods supporting human centered design*. ISO, 2002.

[45] Wikipedia. Methodology (software engineering). Available at: http://encyclopedia.thefreedictionary.com/Methodology+(software+engineering) - Accessed on October 13, 2008.

[46] Help-IBM WebSpare Help System, Model elements. Available at: http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=/com.ibm.xtools.modeler.doc/topics/cme.html - Accessed on November 30, 2008.

APPENDIX 1

Below is the XMI for a UML class diagram.

```
<?xml version="1.0" encoding="UTF-8"?>
<XMI xmi.version="1.0">

    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Novosoft UML Library</XMI.exporter>
            <XMI.exporterVersion>0.4.19</XMI.exporterVersion>
        </XMI.documentation>
        <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
    </XMI.header>

    <XMI.content>
      <Model_Management.Model xmi.id="xmi.1" xmi.uuid="-119--63--45--20-b34b1:e806184ce1:-8000">
            <Foundation.Core.ModelElement.name>untitledModel</Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
            <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
            <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
            <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
            <Foundation.Core.Namespace.ownedElement>

              <Foundation.Core.Class xmi.id="xmi.2" xmi.uuid="-119--63--45--20-b34b1:e806184ce1:-7fff">
                   <Foundation.Core.ModelElement.name>class1</Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
                  <Foundation.Core.Class.isActive xmi.value="false"/>
                  <Foundation.Core.ModelElement.namespace>
                      <Foundation.Core.Namespace xmi.idref="xmi.1"/>
                  </Foundation.Core.ModelElement.namespace>
               </Foundation.Core.Class>

              <Foundation.Core.Class xmi.id="xmi.3" xmi.uuid="-119--63--45--20-b34b1:e806184ce1:-7ffc">
                  <Foundation.Core.ModelElement.name>class2</Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
                  <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
                  <Foundation.Core.Class.isActive xmi.value="false"/>
                  <Foundation.Core.ModelElement.namespace>
```

```
                    <Foundation.Core.Namespace xmi.idref="xmi.1"/>
                </Foundation.Core.ModelElement.namespace>
            </Foundation.Core.Class>

            <Foundation.Core.Class xmi.id="xmi.4" xmi.uuid="-119--63--45--20-b34b1:e806184ce1:-7ffb">
                <Foundation.Core.ModelElement.name>class3</Foundation.Core.ModelElement.name>
                <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
                <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
                <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
                <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
                <Foundation.Core.Class.isActive xmi.value="false"/>
                <Foundation.Core.ModelElement.namespace>
                    <Foundation.Core.Namespace xmi.idref="xmi.1"/>
                </Foundation.Core.ModelElement.namespace>
            </Foundation.Core.Class>

        </Foundation.Core.Namespace.ownedElement>
    </Model_Management.Model>
</XMI.content>

</XMI>
```

APPENDIX 2

Below is the complete mapping list for the Task record. The mappings included in this appendix are preliminary mappings. They need to be validated in the future.

Mapping Tags List

2008, Fei Huang

PUF XML Tag	UML XML Tag	Mapping Relationship
TaskRec IdInfoSection	Foundation Core ModelElement	Inclusion relation
TaskRec IdInfoSection	Foundation Core Classifier	Inclusion relation
TaskRec IdInfoSection	Behavioral_Elements Use_Cases UseCase	Inclusion relation
TaskRec IdInfoSection Name	Foundation Core ModelElement name	Inclusion relation
TaskRec IdInfoSection Type	Foundation Core ModelElement	Non-existing relation
TaskRec IdInfoSection Desccription	Foundation Core ModelElement comment	Exact Relation
TaskRec LnkInfoSection	Behavioral_Elements Common_Behavior Instance	Non-existing relation
TaskRec LnkInfoSection Who	Behavioral_Elements Use_Cases Actor	Inclusion relation
TaskRec LnkInfoSection What	Behavioral_Elements Use_Cases UseCase	Non-existing relation
TaskRec LnkInfoSection How	Behavioral_Elements Common_Behavior Action	Inclusion relation
TaskRec LnkInfoSection How	Behavioral_Elements Collaborations Collaboration	Non-existing relation
TaskRec LnkInfoSection WithWhichContent	Behavioral_Elements Collaborations ClassifierRole availableContents	Inclusion relation
TaskRec LnkInfoSection Scenarios	Behavioral_Elements Use_Cases UseCaseInstance	Exact Relation
TaskRec LnkInfoSection Scenarios	Foundation Core Classifier instance	Exact Relation
TaskRec LnkInfoSection Scenarios	Behavioral_Elements Collaborations Collaboration	Non-existing relation
TaskRec EnvInfoSection When	Behavioral_Elements Collaborations interaction context	Inclusion relation
TaskRec EnvInfoSection When	Foundation Core ModelElement	Non-existing relation
TaskRec EnvInfoSection When	Foundation Core Classifier feature	Non-existing relation
TaskRec EnvInfoSection Where	Foundation Core Classifier feature	Non-existing relation
TaskRec EnvInfoSection Where	Behavioral_Elements Collaborations interaction	Inclusion relation
TaskRec EnvInfoSection Where	Foundation Core ModelElement	Non-existing relation
TaskRec EnvInfoSection HowMuch	Foundation Core ModelElement constraint	Non-existing relation
TaskRec EnvInfoSection HowMuch	Foundation Core Classifier feature	Non-existing relation
TaskRec EnvInfoSection Why	Behavioral_Elements Use_Cases UseCase	Non-existing relation
TaskRec EnvInfoSection Why	Foundation Core Classifier	Non-existing relation
TaskRec EnvInfoSection Why	Foundation Core ModelElement	Non-existing relation
TaskRec DetReqSection TaskOper	Behavioral_Elements Collaborations Collaboration representedOperation	Non-existing relation
TaskRec DetReqSection TaskOper	Behavioral_Elements Common_Behavior Action	Non-existing relation
TaskRec DetReqSection ReqOfUsers	Behavioral_Elements Collaborations ClassifierRole availableFeature	Non-existing relation
TaskRec DetReqSection ReqOfUsers	Behavioral_Elements Collaborations ClassifierRole availableFeature	Non-existing relation
TaskRec DetReqSection ReqOfUsers	Behavioral_Elements Use_Cases Actor	Non-existing relation
TaskRec DetReqSection Communications	Behavioral_Elements Collaborations interaction context	Non-existing relation
TaskRec DetReqSection Learning	Behavioral_Elements Collaborations Collaboration	Non-existing relation
TaskRec DetReqSection ErrorHandling	Behavioral_Elements Collaborations Collaboration	Non-existing relation
TaskRec DetReqSection ErrorHandling	Behavioral_Elements State_Machines StateMachine	Non-existing relation
TaskRec DetReqSection ProblemDetail	Behavioral_Elements Collaborations Collaboration	Non-existing relation
TaskRec DetReqSection ProblemDetail	Behavioral_Elements State_Machines StateMachine	Non-existing relation

APPENDIX 3

Below is the complete mapping list for the User record. The mappings included in this appendix are preliminary mappings. They need to be validated in the future.

Mapping Tags List
2008, Fei Huang

PUF XML Tag	UML XML Tag	Mapping Relationship
UserRec IdInfoSection	Foundation Core Classifier	Inclusion relation
UserRec IdInfoSection	Foundation Core ModelElement	Inclusion relation
UserRec IdInfoSection Name	Foundation Core ModelElement name	Inclusion relation
UserRec IdInfoSection Type	Foundation Core ModelElement	Non-existing relation
UserRec IdInfoSection Desccription	Foundation Core ModelElement	Inclusion relation
UserRec LnkInfoSection	Behavioral_Elements Use_Cases Actor	Non-existing relation
UserRec LnkInfoSection Who	Foundation Core Classifier	Inclusion relation
UserRec LnkInfoSection Who	Behavioral_Elements Use_Cases Actor	Exact Relation
UserRec LnkInfoSection What	Foundation Core Classifier	Inclusion relation
UserRec LnkInfoSection How	Foundation Core Classifier feature	Non-existing relation
UserRec LnkInfoSection WithWhichContent	Foundation Core Classifier feature	Non-existing relation
UserRec LnkInfoSection Scenarios	Behavioral_Elements Common_Behavior Instance	Exact Relation
UserRec EnvInfoSection	Foundation Core ModelElement	Inclusion relation
UserRec EnvInfoSection	Foundation Core Classifier feature	Non-existing relation
UserRec DetReqSection PhysCharCap	Foundation Core Classifier feature	Non-existing relation
UserRec DetReqSection PhysCharCap	Behavioral_Elements Use_Cases Actor	Non-existing relation
UserRec DetReqSection MentalCharCap	Behavioral_Elements Use_Cases Actor	Non-existing relation
UserRec DetReqSection MentalCharCap	Foundation Core Classifier feature	Non-existing relation
UserRec DetReqSection SocialCharCap	Foundation Core Classifier feature	Non-existing relation
UserRec DetReqSection SocialCharCap	Behavioral_Elements Use_Cases Actor	Non-existing relation
UserRec DetReqSection GroupCharCap	Behavioral_Elements Use_Cases Actor	Non-existing relation
UserRec DetReqSection GroupCharCap	Foundation Core Classifier feature	Non-existing relation

APPENDIX 4

Below is the complete mapping list for the Content record. The mappings included in this appendix are preliminary mappings. They need to be validated in the future.

Mapping Tags List
2008, Fei Huang

PUF XML Tag	UML XML Tag	Mapping Relationship
ContentRec	Foundation Extension_Mechanisms Stereotype	Non-existing relation
ContentRec IdInfoSection Name	Foundation Core Attribute initialValue	Inclusion relation
ContentRec IdInfoSection Type	Foundation Core Attribute	Non-existing relation
ContentRec IdInfoSection Desccription	Foundation Core Attribute	Non-existing relation
ContentRec LnkInfoSection	Behavioral_Elements Common_Behavior Link	Non-existing relation
ContentRec LnkInfoSection	Foundation Core Attribute associationEnd	Inclusion relation
ContentRec LnkInfoSection Who	Behavioral_Elements Use_Cases Actor	Inclusion relation
ContentRec LnkInfoSection What	Behavioral_Elements Use_Cases UseCase	Exact Relation
ContentRec LnkInfoSection How	Foundation Core Operation specification	Inclusion relation
ContentRec LnkInfoSection WithWhichContent	Foundation Core Attribute	Exact Relation
ContentRec LnkInfoSection Scenarios	Behavioral_Elements Use_Cases UseCaseInstance	Exact Relation
ContentRec LnkInfoSection Scenarios	Foundation Extension_Mechanisms Stereotype	Non-existing relation
ContentRec EnvInfoSection	Behavioral_Elements State_Machines StateMachine context	Non-existing relation
ContentRec EnvInfoSection	Behavioral_Elements State_Machines StateMachine context	Non-existing relation
ContentRec EnvInfoSection	Behavioral_Elements Collaborations interaction context	Non-existing relation
ContentRec DetReqSection Structure	Foundation Core Attribute	Non-existing relation
ContentRec DetReqSection Structure	Foundation Core Generalization	Non-existing relation
ContentRec DetReqSection Semantics	Foundation Core Constraint body	Inclusion relation
ContentRec DetReqSection Semantics	Foundation Core Attribute	Non-existing relation
ContentRec DetReqSection ReqOfUsers	Behavioral_Elements Use_Cases Actor	Non-existing relation
ContentRec DetReqSection HowBeHandled	Foundation Core Operation	Inclusion relation
ContentRec DetReqSection WhenBeUsed	Behavioral_Elements State_Machines TimeEvent when	Exact Relation

APPENDIX 5

Below is the complete mapping list for the Tool record. The mappings included in this appendix are preliminary mappings. They need to be validated in the future.

Mapping Tags List
2008, Fei Huang

PUF XML Tag	UML XML Tag	Mapping Relationship
ToolRec IdInfoSection Name	Foundation Core Operation	Inclusion relation
ToolRec IdInfoSection Type	Foundation Core Operation	Non-existing relation
ToolRec IdInfoSection Desccription	Foundation Core Operation specification	Exact Relation
ToolRec LnkInfoSection Who	Behavioral_Elements Use_Cases Actor	Inclusion relation
ToolRec LnkInfoSection What	Behavioral_Elements Use_Cases UseCaseInstance	Inclusion relation
ToolRec LnkInfoSection How	Foundation Core Operation	Inclusion relation
ToolRec LnkInfoSection WithWhichContent	Foundation Core Attribute	Inclusion relation
ToolRec LnkInfoSection Scenarios	Behavioral_Elements Use_Cases UseCaseInstance	Exact Relation
ToolRec EnvInfoSection	Foundation Core Operation	Non-existing relation
ToolRec DetReqSection ToolOper	Behavioral_Elements State_Machines Event	Non-existing relation
ToolRec DetReqSection ToolOper	Behavioral_Elements Common_Behavior Action	Non-existing relation
ToolRec DetReqSection ToolOper	Behavioral_Elements Use_Cases UseCase	Non-existing relation
ToolRec DetReqSection ToolOper	Behavioral_Elements State_Machines Event	Non-existing relation
ToolRec DetReqSection ReqOfUsers	Foundation Core Operation	Non-existing relation
ToolRec DetReqSection Communications	Foundation Core Operation	Non-existing relation
ToolRec DetReqSection Learning	Foundation Core Operation	Non-existing relation
ToolRec DetReqSection ErrorHandling	Foundation Core Operation	Non-existing relation
ToolRec DetReqSection ProblemDetail	Foundation Core Operation	Non-existing relation

APPENDIX 6

Below is the enhanced XMI file integrated with PUF requirement metadata according to the Task record' mapping list.

Enhanced UML XML File
2008, Fei Huang

Enhanced UML XML Tag
Behavioral_Elements
Behavioral_Elements Collaborations AssociationEndRole
Behavioral_Elements Collaborations AssociationEndRole base
Behavioral_Elements Collaborations AssociationEndRole collaborationMultiplicity
Behavioral_Elements Collaborations AssociationRole
Behavioral_Elements Collaborations AssociationRole base
Behavioral_Elements Collaborations AssociationRole multiplicity
Behavioral_Elements Collaborations ClassifierRole
Behavioral_Elements Collaborations ClassifierRole availableContents
Behavioral_Elements Collaborations ClassifierRole availableContents WithWhichContent
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers Acceptance
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers all
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers EaseofUse
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers MentalCap
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers PhysCap
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers PotentialMisuse
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers Responsibility
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers Skill
Behavioral_Elements Collaborations ClassifierRole availableFeature ReqOfUsers Speed
Behavioral_Elements Collaborations ClassifierRole base
Behavioral_Elements Collaborations ClassifierRole multiplicity
Behavioral_Elements Collaborations Collaboration constrainingElement
Behavioral_Elements Collaborations Collaboration How
Behavioral_Elements Collaborations Collaboration interaction
Behavioral_Elements Collaborations Collaboration representedClassifier Who
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Accessibility
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper all
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Alternative
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Feedback
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Flexibility
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper HowtoDo
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper IsFormal
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper IsInterruptible
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper IsSharable
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper NeededRedundancy
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Purpose
Behavioral_Elements Collaborations Collaboration representedOperation TaskOper Subtasks
Behavioral_Elements Collaborations Collaboration representedOperation What
Behavioral_Elements Collaborations Collaboration Scenarios
Behavioral_Elements Collaborations interaction context Communications all
Behavioral_Elements Collaborations interaction context Communications FlowDirection
Behavioral_Elements Collaborations interaction context Communications Frequency
Behavioral_Elements Collaborations interaction context Communications HowManyUser
Behavioral_Elements Collaborations interaction context Communications IsCentralized
Behavioral_Elements Collaborations interaction context Communications IsFormal
Behavioral_Elements Collaborations interaction context Communications IsOneWay

Enhanced UML XML Tag
Behavioral_Elements Collaborations interaction context Communications IsRecorded
Behavioral_Elements Collaborations interaction context Communications Language
Behavioral_Elements Collaborations interaction context Communications Medium
Behavioral_Elements Collaborations interaction context Communications Security
Behavioral_Elements Collaborations interaction context HowMuch
Behavioral_Elements Collaborations interaction context When all
Behavioral_Elements Collaborations interaction context When FrequencyofUse
Behavioral_Elements Collaborations interaction context When ParticularTime
Behavioral_Elements Collaborations interaction context When PeakUsageDistribution
Behavioral_Elements Collaborations interaction context When PerformanceReq
Behavioral_Elements Collaborations interaction context When SpentTime
Behavioral_Elements Collaborations interaction context Where all
Behavioral_Elements Collaborations interaction context Where Configurations
Behavioral_Elements Collaborations interaction context Where EffectOnPerf
Behavioral_Elements Collaborations interaction context Where Environments
Behavioral_Elements Collaborations Message
Behavioral_Elements Collaborations Message activator
Behavioral_Elements Collaborations Message base
Behavioral_Elements Collaborations Message interaction
Behavioral_Elements Collaborations Message predecessor
Behavioral_Elements Collaborations Message receiver
Behavioral_Elements Collaborations Message sender
Behavioral_Elements Common_Behavior
Behavioral_Elements Common_Behavior Action
Behavioral_Elements Common_Behavior Action actionSequence
Behavioral_Elements Common_Behavior Action actualArgument
Behavioral_Elements Common_Behavior Action isAsynchronous
Behavioral_Elements Common_Behavior Action recurrence
Behavioral_Elements Common_Behavior Action script
Behavioral_Elements Common_Behavior Action stimulus
Behavioral_Elements Common_Behavior Action target
Behavioral_Elements Common_Behavior ActionSequence
Behavioral_Elements Common_Behavior ActionSequence action
Behavioral_Elements Common_Behavior CallAction
Behavioral_Elements Common_Behavior CallAction operation
Behavioral_Elements Common_Behavior Instance
Behavioral_Elements Common_Behavior Instance attributeLink
Behavioral_Elements Common_Behavior Instance classifier
Behavioral_Elements Common_Behavior Instance linkEnd
Behavioral_Elements Common_Behavior Link
Behavioral_Elements Common_Behavior Link association
Behavioral_Elements Common_Behavior Object
Behavioral_Elements Common_Behavior SendAction
Behavioral_Elements Common_Behavior SendAction signal
Behavioral_Elements State_Machines
Behavioral_Elements State_Machines CallEvent
Behavioral_Elements State_Machines CallEvent operation
Behavioral_Elements State_Machines CompositeState
Behavioral_Elements State_Machines CompositeState isConcurrent

Enhanced UML XML Tag
Behavioral_Elements State_Machines CompositeState subvertex
Behavioral_Elements State_Machines Event
Behavioral_Elements State_Machines Event state
Behavioral_Elements State_Machines Event transition
Behavioral_Elements State_Machines SignalEvent
Behavioral_Elements State_Machines SignalEvent signal
Behavioral_Elements State_Machines SimpleState
Behavioral_Elements State_Machines State
Behavioral_Elements State_Machines State deferrableEvent
Behavioral_Elements State_Machines State doActivity
Behavioral_Elements State_Machines State entry
Behavioral_Elements State_Machines State exit
Behavioral_Elements State_Machines State internalTransition
Behavioral_Elements State_Machines State stateMachine
Behavioral_Elements State_Machines StateMachine
Behavioral_Elements State_Machines StateMachine context
Behavioral_Elements State_Machines StateMachine subMachineState
Behavioral_Elements State_Machines StateMachine top
Behavioral_Elements State_Machines StateMachine transitions
Behavioral_Elements State_Machines SubmachineState
Behavioral_Elements State_Machines SubmachineState submachine
Behavioral_Elements State_Machines TimeEvent
Behavioral_Elements State_Machines TimeEvent when
Behavioral_Elements State_Machines Transition
Behavioral_Elements State_Machines Transition effect
Behavioral_Elements State_Machines Transition guard
Behavioral_Elements State_Machines Transition source
Behavioral_Elements State_Machines Transition state
Behavioral_Elements State_Machines Transition stateMachine
Behavioral_Elements State_Machines Transition target
Behavioral_Elements State_Machines Transition trigger
Behavioral_Elements Use_Cases
Behavioral_Elements Use_Cases A_base_extend2
Behavioral_Elements Use_Cases A_base_extend2 base
Behavioral_Elements Use_Cases A_base_extend2 extend2
Behavioral_Elements Use_Cases A_extensionPoint_useCase
Behavioral_Elements Use_Cases A_extensionPoint_useCase extend
Behavioral_Elements Use_Cases A_extensionPoint_useCase extensionPoint
Behavioral_Elements Use_Cases A_extension_extend
Behavioral_Elements Use_Cases A_extension_extend extend
Behavioral_Elements Use_Cases A_extension_extend extension
Behavioral_Elements Use_Cases A_include_addition
Behavioral_Elements Use_Cases A_include_addition addition
Behavioral_Elements Use_Cases A_include_addition include
Behavioral_Elements Use_Cases Extend
Behavioral_Elements Use_Cases Extend base
Behavioral_Elements Use_Cases Extend condition
Behavioral_Elements Use_Cases Extend extension
Behavioral_Elements Use_Cases Extend extensionPoint

Enhanced UML XML Tag
Behavioral_Elements Use_Cases ExtensionPoint
Behavioral_Elements Use_Cases ExtensionPoint extend
Behavioral_Elements Use_Cases ExtensionPoint location
Behavioral_Elements Use_Cases ExtensionPoint useCase
Behavioral_Elements Use_Cases Include
Behavioral_Elements Use_Cases Include addition
Behavioral_Elements Use_Cases Include base
Behavioral_Elements Use_Cases UseCase extend
Behavioral_Elements Use_Cases UseCase extend2
Behavioral_Elements Use_Cases UseCase extensionPoint
Behavioral_Elements Use_Cases UseCase include
Behavioral_Elements Use_Cases UseCase include2
Foundation Core Association
Foundation Core Association connection
Foundation Core AssociationClass
Foundation Core AssociationEnd
Foundation Core AssociationEnd aggregation
Foundation Core AssociationEnd association
Foundation Core AssociationEnd multiplicity
Foundation Core AssociationEnd ordering
Foundation Core AssociationEnd qualifier
Foundation Core AssociationEnd specification
Foundation Core AssociationEnd type
Foundation Core Attribute
Foundation Core Attribute associationEnd
Foundation Core Attribute initialValue
Foundation Core A_constrainedElement_constraint
Foundation Core A_constrainedElement_constraint constrainedElement
Foundation Core A_constrainedElement_constraint constraint
Foundation Core Class
Foundation Core Class isActive
Foundation Core Classifier
Foundation Core Classifier feature HowMuch
Foundation Core Classifier feature TaskOper Accessibility
Foundation Core Classifier feature TaskOper all
Foundation Core Classifier feature TaskOper Alternative
Foundation Core Classifier feature TaskOper Feedback
Foundation Core Classifier feature TaskOper Flexibility
Foundation Core Classifier feature TaskOper HowtoDo
Foundation Core Classifier feature TaskOper IsFormal
Foundation Core Classifier feature TaskOper IsInterruptible
Foundation Core Classifier feature TaskOper IsSharable
Foundation Core Classifier feature TaskOper NeededRedundancy
Foundation Core Classifier feature TaskOper Purpose
Foundation Core Classifier feature TaskOper Subtasks
Foundation Core Classifier feature When all
Foundation Core Classifier feature When FrequencyofUse
Foundation Core Classifier feature When ParticularTime
Foundation Core Classifier feature When PeakUsageDistribution

Enhanced UML XML Tag
Foundation Core Classifier feature When PerformanceReq
Foundation Core Classifier feature When SpentTime
Foundation Core Classifier feature Where all
Foundation Core Classifier feature Where Configurations
Foundation Core Classifier feature Where EffectOnPerf
Foundation Core Classifier feature Where Environments
Foundation Core Classifier participant Who
Foundation Core Classifier Scenarios
Foundation Core Comment
Foundation Core Comment annotatedElement
Foundation Core Component
Foundation Core Component deploymentLocation
Foundation Core Component residentElement
Foundation Core Constraint
Foundation Core Constraint body
Foundation Core Constraint constrainedElement
Foundation Core Dependency
Foundation Core Dependency client
Foundation Core Dependency supplier
Foundation Core Generalization
Foundation Core Generalization child
Foundation Core Generalization discriminator
Foundation Core Generalization parent
Foundation Core Method
Foundation Core Method body
Foundation Core Method specification
Foundation Core ModelElement comment
Foundation Core ModelElement constraint HowMuch
Foundation Core ModelElement Desccription
Foundation Core ModelElement Name
Foundation Core ModelElement Type
Foundation Core ModelElement When all
Foundation Core ModelElement When FrequencyofUse
Foundation Core ModelElement When ParticularTime
Foundation Core ModelElement When PeakUsageDistribution
Foundation Core ModelElement When PerformanceReq
Foundation Core ModelElement When SpentTime
Foundation Core ModelElement Where all
Foundation Core ModelElement Where Configurations
Foundation Core ModelElement Where EffectOnPerf
Foundation Core ModelElement Where Environments
Foundation Core ModelElement Why all
Foundation Core ModelElement Why CostEffective
Foundation Core ModelElement Why HowFit
Foundation Core ModelElement Why IsEssential
Foundation Core ModelElement Why Prerequirsite
Foundation Core Operation
Foundation Core Operation concurrency
Foundation Core Operation isAbstract

Enhanced UML XML Tag
Foundation Core Operation isLeaf
Foundation Core Operation isRoot
Foundation Core Operation method
Foundation Core Operation specification
Foundation Extension_Mechanisms
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint constrainedElement2
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint Constraint
Foundation Extension_Mechanisms Stereotype
Foundation Extension_Mechanisms Stereotype baseClass
Foundation Extension_Mechanisms Stereotype extendedElement
Foundation Extension_Mechanisms Stereotype requiredTag
Foundation Extension_Mechanisms Stereotype stereotypeConstraint
Foundation Extension_Mechanisms TaggedValue
Foundation Extension_Mechanisms TaggedValue modelElement
Foundation Extension_Mechanisms TaggedValue stereotype
Foundation Extension_Mechanisms TaggedValue tag
Foundation Extension_Mechanisms TaggedValue value

APPENDIX 7

Below is the enhanced XMI file integrated with PUF requirement metadata according to the User record' mapping list.

Enhanced UML XML File
2008, Fei Huang

Enhanced UML XML Tag
Behavioral_Elements
Behavioral_Elements Collaborations AssociationEndRole
Behavioral_Elements Collaborations AssociationEndRole base
Behavioral_Elements Collaborations AssociationEndRole collaborationMultiplicity
Behavioral_Elements Collaborations AssociationRole
Behavioral_Elements Collaborations AssociationRole base
Behavioral_Elements Collaborations AssociationRole multiplicity
Behavioral_Elements Collaborations ClassifierRole
Behavioral_Elements Collaborations ClassifierRole availableContents
Behavioral_Elements Collaborations ClassifierRole availableFeature
Behavioral_Elements Collaborations ClassifierRole base
Behavioral_Elements Collaborations ClassifierRole multiplicity
Behavioral_Elements Collaborations Collaboration
Behavioral_Elements Collaborations Collaboration constrainingElement
Behavioral_Elements Collaborations Collaboration interaction
Behavioral_Elements Collaborations Collaboration representedClassifier
Behavioral_Elements Collaborations Collaboration representedOperation
Behavioral_Elements Collaborations Message
Behavioral_Elements Collaborations Message activator
Behavioral_Elements Collaborations Message base
Behavioral_Elements Collaborations Message interaction
Behavioral_Elements Collaborations Message predecessor
Behavioral_Elements Collaborations Message receiver
Behavioral_Elements Collaborations Message sender
Behavioral_Elements Common_Behavior
Behavioral_Elements Common_Behavior Action
Behavioral_Elements Common_Behavior Action actionSequence
Behavioral_Elements Common_Behavior Action actualArgument
Behavioral_Elements Common_Behavior Action isAsynchronous
Behavioral_Elements Common_Behavior Action recurrence
Behavioral_Elements Common_Behavior Action script
Behavioral_Elements Common_Behavior Action stimulus
Behavioral_Elements Common_Behavior Action target
Behavioral_Elements Common_Behavior ActionSequence
Behavioral_Elements Common_Behavior ActionSequence action
Behavioral_Elements Common_Behavior CallAction
Behavioral_Elements Common_Behavior CallAction operation
Behavioral_Elements Common_Behavior Instance
Behavioral_Elements Common_Behavior Instance attributeLink
Behavioral_Elements Common_Behavior Instance classifier
Behavioral_Elements Common_Behavior Instance linkEnd
Behavioral_Elements Common_Behavior Link
Behavioral_Elements Common_Behavior Link association
Behavioral_Elements Common_Behavior Object
Behavioral_Elements Common_Behavior SendAction
Behavioral_Elements Common_Behavior SendAction signal

Enhanced UML XML Tag
Behavioral_Elements State_Machines
Behavioral_Elements State_Machines CallEvent
Behavioral_Elements State_Machines CallEvent operation
Behavioral_Elements State_Machines CompositeState
Behavioral_Elements State_Machines CompositeState isConcurrent
Behavioral_Elements State_Machines CompositeState subvertex
Behavioral_Elements State_Machines Event
Behavioral_Elements State_Machines Event state
Behavioral_Elements State_Machines Event transition
Behavioral_Elements State_Machines SignalEvent
Behavioral_Elements State_Machines SignalEvent signal
Behavioral_Elements State_Machines SimpleState
Behavioral_Elements State_Machines State
Behavioral_Elements State_Machines State deferrableEvent
Behavioral_Elements State_Machines State doActivity
Behavioral_Elements State_Machines State entry
Behavioral_Elements State_Machines State exit
Behavioral_Elements State_Machines State internalTransition
Behavioral_Elements State_Machines State stateMachine
Behavioral_Elements State_Machines StateMachine
Behavioral_Elements State_Machines StateMachine context
Behavioral_Elements State_Machines StateMachine subMachineState
Behavioral_Elements State_Machines StateMachine top
Behavioral_Elements State_Machines StateMachine transitions
Behavioral_Elements State_Machines SubmachineState
Behavioral_Elements State_Machines SubmachineState submachine
Behavioral_Elements State_Machines TimeEvent
Behavioral_Elements State_Machines TimeEvent when
Behavioral_Elements State_Machines Transition
Behavioral_Elements State_Machines Transition effect
Behavioral_Elements State_Machines Transition guard
Behavioral_Elements State_Machines Transition source
Behavioral_Elements State_Machines Transition state
Behavioral_Elements State_Machines Transition stateMachine
Behavioral_Elements State_Machines Transition target
Behavioral_Elements State_Machines Transition trigger
Behavioral_Elements Use_Cases
Behavioral_Elements Use_Cases Actor GroupCharCap all
Behavioral_Elements Use_Cases Actor GroupCharCap Membership
Behavioral_Elements Use_Cases Actor GroupCharCap Orientation
Behavioral_Elements Use_Cases Actor GroupCharCap Relations
Behavioral_Elements Use_Cases Actor GroupCharCap Self-concept
Behavioral_Elements Use_Cases Actor MentalCharCap all
Behavioral_Elements Use_Cases Actor MentalCharCap CognitiveCap
Behavioral_Elements Use_Cases Actor MentalCharCap EmontionalCap
Behavioral_Elements Use_Cases Actor MentalCharCap Expertise
Behavioral_Elements Use_Cases Actor MentalCharCap MemoryCap
Behavioral_Elements Use_Cases Actor MentalCharCap Personality
Behavioral_Elements Use_Cases Actor PhysCharCap all

Enhanced UML XML Tag
Behavioral_Elements Use_Cases Actor PhysCharCap DistinguishingChar
Behavioral_Elements Use_Cases Actor PhysCharCap PerformingChar
Behavioral_Elements Use_Cases Actor PhysCharCap PhysCap
Behavioral_Elements Use_Cases Actor PhysCharCap SensingCap
Behavioral_Elements Use_Cases Actor SocialCharCap all
Behavioral_Elements Use_Cases Actor SocialCharCap Application
Behavioral_Elements Use_Cases Actor SocialCharCap Implementation
Behavioral_Elements Use_Cases Actor SocialCharCap Recognition
Behavioral_Elements Use_Cases Actor SocialCharCap Types
Behavioral_Elements Use_Cases A_base_extend2
Behavioral_Elements Use_Cases A_base_extend2 base
Behavioral_Elements Use_Cases A_base_extend2 extend2
Behavioral_Elements Use_Cases A_extensionPoint_useCase
Behavioral_Elements Use_Cases A_extensionPoint_useCase extend
Behavioral_Elements Use_Cases A_extensionPoint_useCase extensionPoint
Behavioral_Elements Use_Cases A_extension_extend
Behavioral_Elements Use_Cases A_extension_extend extend
Behavioral_Elements Use_Cases A_extension_extend extension
Behavioral_Elements Use_Cases A_include_addition
Behavioral_Elements Use_Cases A_include_addition addition
Behavioral_Elements Use_Cases A_include_addition include
Behavioral_Elements Use_Cases Extend
Behavioral_Elements Use_Cases Extend base
Behavioral_Elements Use_Cases Extend condition
Behavioral_Elements Use_Cases Extend extension
Behavioral_Elements Use_Cases Extend extensionPoint
Behavioral_Elements Use_Cases ExtensionPoint
Behavioral_Elements Use_Cases ExtensionPoint extend
Behavioral_Elements Use_Cases ExtensionPoint location
Behavioral_Elements Use_Cases ExtensionPoint useCase
Behavioral_Elements Use_Cases Include
Behavioral_Elements Use_Cases Include addition
Behavioral_Elements Use_Cases Include base
Behavioral_Elements Use_Cases Scenarios
Behavioral_Elements Use_Cases UseCase extend
Behavioral_Elements Use_Cases UseCase extend2
Behavioral_Elements Use_Cases UseCase extensionPoint
Behavioral_Elements Use_Cases UseCase include
Behavioral_Elements Use_Cases UseCase include2
Behavioral_Elements Use_Cases UseCase WithWhichContent
Behavioral_Elements Use_Cases What
Behavioral_Elements Use_Cases Who
Foundation Core Association
Foundation Core Association connection
Foundation Core AssociationClass
Foundation Core AssociationEnd
Foundation Core AssociationEnd aggregation
Foundation Core AssociationEnd association
Foundation Core AssociationEnd multiplicity

Enhanced UML XML Tag
Foundation Core AssociationEnd ordering
Foundation Core AssociationEnd qualifier
Foundation Core AssociationEnd specification
Foundation Core AssociationEnd type
Foundation Core Attribute
Foundation Core Attribute associationEnd
Foundation Core Attribute initialValue
Foundation Core A_constrainedElement_constraint
Foundation Core A_constrainedElement_constraint constrainedElement
Foundation Core A_constrainedElement_constraint constraint
Foundation Core Class
Foundation Core Class isActive
Foundation Core Classifier feature GroupCharCap all
Foundation Core Classifier feature GroupCharCap Membership
Foundation Core Classifier feature GroupCharCap Orientation
Foundation Core Classifier feature GroupCharCap Relations
Foundation Core Classifier feature GroupCharCap Self-concept
Foundation Core Classifier feature MentalCharCap all
Foundation Core Classifier feature MentalCharCap CognitiveCap
Foundation Core Classifier feature MentalCharCap EmontionalCap
Foundation Core Classifier feature MentalCharCap Expertise
Foundation Core Classifier feature MentalCharCap MemoryCap
Foundation Core Classifier feature MentalCharCap Personality
Foundation Core Classifier feature PhysCharCap all
Foundation Core Classifier feature PhysCharCap DistinguishingChar
Foundation Core Classifier feature PhysCharCap PerformingChar
Foundation Core Classifier feature PhysCharCap PhysCap
Foundation Core Classifier feature PhysCharCap SensingCap
Foundation Core Classifier feature SocialCharCap all
Foundation Core Classifier feature SocialCharCap Application
Foundation Core Classifier feature SocialCharCap Implementation
Foundation Core Classifier feature SocialCharCap Recognition
Foundation Core Classifier feature SocialCharCap Types
Foundation Core Classifier feature WithWhichContent
Foundation Core Classifier participant
Foundation Core Comment
Foundation Core Comment annotatedElement
Foundation Core Component
Foundation Core Component deploymentLocation
Foundation Core Component residentElement
Foundation Core Constraint
Foundation Core Constraint body
Foundation Core Constraint constrainedElement
Foundation Core Dependency
Foundation Core Dependency client
Foundation Core Dependency supplier
Foundation Core Generalization
Foundation Core Generalization child
Foundation Core Generalization discriminator

Enhanced UML XML Tag
Foundation Core Generalization parent
Foundation Core Method
Foundation Core Method body
Foundation Core Method specification
Foundation Core ModelElement comment
Foundation Core ModelElement constraint
Foundation Core ModelElement Desccription
Foundation Core ModelElement Name
Foundation Core ModelElement Type
Foundation Core Operation concurrency
Foundation Core Operation How
Foundation Core Operation isAbstract
Foundation Core Operation isLeaf
Foundation Core Operation isRoot
Foundation Core Operation method
Foundation Core Operation specification
Foundation Extension_Mechanisms
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint constrainedElement2
Foundation Extension_Mechanisms A_constrainedElement2_stereotypeConstraint Constraint
Foundation Extension_Mechanisms Stereotype
Foundation Extension_Mechanisms Stereotype baseClass
Foundation Extension_Mechanisms Stereotype extendedElement
Foundation Extension_Mechanisms Stereotype requiredTag
Foundation Extension_Mechanisms Stereotype stereotypeConstraint
Foundation Extension_Mechanisms TaggedValue
Foundation Extension_Mechanisms TaggedValue modelElement
Foundation Extension_Mechanisms TaggedValue stereotype
Foundation Extension_Mechanisms TaggedValue tag
Foundation Extension_Mechanisms TaggedValue value

APPENDIX 8

Below is the complete list of PUF XML tags and their meanings cited from *Transforming Usability Engineering Requirements into Software Engineering Specifications- From PUF to UML.*[6].

Record Name	Section Name	Question Name	SubQuestion Name	Description
<TaskRec >				It describes theTask structure model which is used to identify tasks and helps developers with analyzing and designing them.
<TaskRec	IdInfoSection>			It identifies a task/scenario type record and give a unique name and detailed description to the record
<TaskRec	IdInfoSection	Name>		It is a unique, meaningful identifier
<TaskRec	IdInfoSection	Type>		It is a type of structure model
<TaskRec	IdInfoSection	Description >		It describes the intended accomplishment of the task, especially focusing on how the task is different from other related tasks
<TaskRec	LnkInfoSection >			It identifies the set of linkages between that possibility and all other related possibilities
<TaskRec	LnkInfoSection	Who>		It identifies who belongs to a given user group can be described in terms of the characteristics that make an individual a member, especially focusing on how that user group is different from other user groups
<TaskRec	LnkInfoSection	What>		It identifies what the intended accomplishment of the task, especially focusing on how that task is different from other related tasks
<TaskRec	LnkInfoSection	How>		It identifies how the task is accomplished in general (for all the users, tasks, and content for which it can be used without reference to differences in use for different users, tasks, or content), especially focusing on how that task is different from other related tasks.
<TaskRec	LnkInfoSection	WithWhich Content>		It identifies which are the main components of the content chunk, especially focusing on how that content chunk is different from other related content chunks
<TaskRec	LnkInfoSection	Scenarios>		It identifies scenarios which tie together sets of users, tasks, tools, and content chunks
<TaskRec	EnvInfoSection			It provides environmental information about this task/scenario
<TaskRec	EnvInfoSection	When>		It identifies a range of possibilities of when this task could be accomplished, in terms of temporal attributes, and includes both current and potential future ranges
<TaskRec	EnvInfoSection	When	FrequencyofUse >	What is the frequency of use. It is a measure of use in terms of number of separate uses. A task that is performed frequently is likely to have less usability concerns for regular users, since the users are likely to have mastered even difficult parts of the task through repeated practice
<TaskRec	EnvInfoSection	When	SpentTime>	How much time is spent by users. It is another measure of use in terms of cumulative time used
<TaskRec	EnvInfoSection	When	ParticularTime>	What particular times is the task performed/used. They are some timing concerns, including time of day, day of week, time of month or year. The reason we have these concerns is because the problems introduced due to

				interactions or competition with other tasks and/or memory lapses are most pronounced when the task is tied to a particular time of using.
<TaskRec	EnvInfoSection	When	PeakUsageDistribution>	what is the distribution of peak usage. The distribution profile of peak usage of a task is especially important in determining the capacity requirements of tools to be used for the tasks.
<TaskRec	EnvInfoSection	When	PerformanceReq>	what are the performance / execution requirements. The task may introduce various performance related constraints on its execution. Additional constraints may be added based on potential interactions with other tasks.
<TaskRec	EnvInfoSection	Where>		The use case may be limited by its environment, or might be used in a broader environment to achieve some greater benefits. If the use case is used in a different location, different frequency of use, and/or different distribution of peak usage, the design for the use case will be different. Designers should know how to design the use case to make it still achievable in various situations
<TaskRec	EnvInfoSection	Where	Configurations>	what workstation configurations may be used. Workstations can be considered extensions of the main tools (the Web site or other interactive system) we are developing. They are also the interface between those tools and their environment.
<TaskRec	EnvInfoSection	Where	Environments>	what environments may the task have. Ideal environmental conditions have been specified by an International standard. The environmental factors include quality of space considerations, lighting / heat / air considerations, distractions considerations.
<TaskRec	EnvInfoSection	Where	EffectOnPerf>	how does the location effect performance. This involves identifying: the locations where the task is required, or may be required, or may be desired to be accomplished; the workstation and environmental factors that are needed to support the task in the locations; the factors that might inhibit accomplishing the task in those locations and develop design(s) to overcome those inhibiting factors
<TaskRec	EnvInfoSection	HowMuch>		an estimate of the frequency and quantity of the performance of the task
<TaskRec	EnvInfoSection	Why>		Justification is an important predictor of potential future success. If a use case does not fit the overall development, or costs of the use case exceed either the benefits of serving it or available resources, it will be impractical to develop special tools for the use case. Developers should know the factors that will influence the feasibility and acceptability of possible designs
<TaskRec	EnvInfoSection	Why	HowFit>	how does it fit into the overall development. While individual tasks contribute towards the release's overall justification, they then may be (circularly) justified by being part of the release
<TaskRec	EnvInfoSection	Why	IsEssential>	is it essential. The ways by which a given task may be considered essential involve both internal factors such as

				being part of the fundamental definition of an application and external factors such as legal requirements.
<TaskRec	EnvInfoSection	Why	Prerequirsite>	is it in a prerequisite relationship with other TASKS. It intends to identify tasks that are prerequisite to other tasks or tasks that have other prerequisites. This question should be answered by examining the relationships that are identified in “what” linkages
<TaskRec	EnvInfoSection	Why	CostEffective>	do benefits exceed costs. Costs and benefits can be identified with the following questions: tangible benefit concerns, intangible benefit concerns, tangible costs concerns, intangible costs concerns.
<TaskRec	DetReqSection>			The detailed information gained in analysis considers what essential limitations or requirements they place on design, It will be essential in designing a new system.
<TaskRec	DetReqSection	TaskOper>		This PUF field describes operational concerns. Elaboration is needed to understand the operations of essential use cases to evaluate how well current operations work and how future operations might provide improvements. When designers start interaction design, they should know whether the interaction meets the goal of the use case, whether there are alternatives to achieve the use case, and what feedbacks the use case should provide.
<TaskRec	DetReqSection	TaskOper	Purpose>	what is the purpose of the task. It describes the general tasks that are identified in “what” linkages.
<TaskRec	DetReqSection	TaskOper	IsFormal>	is this a formal or informal task. Formal tasks are the ones readily identified as part of an application or application system. Informal tasks are those tasks which are seldom acknowledged as necessary part of an application (or of the user's work) but which the user may accomplish along with the formal tasks of the application or work.
<TaskRec	DetReqSection	TaskOper	HowtoDo>	how is the task done. It tries to identify the common, basic factors of use that are essential to the task or tool. An analysis of how tasks are currently performed may provide a starting point for identifying the relevant operations parameters for performing the task(s)
<TaskRec	DetReqSection	TaskOper	Subtasks>	are there identifiable subtasks. It describes the sub-tasks that are identified in “what” linkages.
<TaskRec	DetReqSection	TaskOper	IsInterruptible>	can the task be interrupted. Some tasks cannot be interrupted without causing them to fail. Some tasks have certain discrete points where they can be interrupted. Other tasks can be interrupted at any point. it is desirable to maximize the potential for interruptions to occur without causing problems.
<TaskRec	DetReqSection	TaskOper	Alternative>	what are the alternatives to the task. when and why will the users choose the alternatives and when and why will they choose our task.
<TaskRec	DetReqSection	TaskOper	Flexibility>	how flexible / adaptable is the task. Flexibility is important to be able to meet the needs of various types of users. Generally the more flexible the task, the more users can accomplish the task. Flexibility and

				adaptability help expand the applicability of the task.
<TaskRec	DetReqSection	TaskOper	NeededRedundancy>	what redundancies are needed. If properly designed, redundancies can provide the user with increased usability. This section includes information about the costs / benefits of retaining redundancy
<TaskRec	DetReqSection	TaskOper	Feedback>	what feedback does the task provide. Feedback is important to reassure the user both that a task was accomplished and that it was accomplished correctly.
<TaskRec	DetReqSection	TaskOper	IsSharable>	is the task sharable / concurrently usable. Some tasks seem inherently individual and some seem inherently group tasks. Some individual tasks can benefit from help and some are hindered by interference.
<TaskRec	DetReqSection	TaskOper	Accessibility>	how accessible is the task. Accessibility refers to the lack of limitations to usability. Where accessibility is limited or denied, usability becomes nonexistent to certain users in certain or in all circumstances.
<TaskRec	DetReqSection	ReqOfUsers>		It is important to recognize the requirements that use cases place on actors. The developers should know how to design the interfaces or interactions for the use case to meet users' current skills and mental and physical capabilities. This information may require additional use cases or tools to help users reduce the impact of these factors.
<TaskRec	DetReqSection	ReqOfUsers	Speed>	what speed is required of the user. Speed is most essential for tasks that need to occur in "real time" which depends on the nature of the task. Speed of the user is dependent on the means of communication employed in the assignment and investigation.
<TaskRec	DetReqSection	ReqOfUsers	MentalCap>	what mental workload capability is required of the user. All tasks place certain cognitive demands that make up part of the mental workload of a user.
<TaskRec	DetReqSection	ReqOfUsers	Skill>	what skills are required of the user. The skills may be gained via training or experience with the given task or via transfer from similar tasks. The need for these skills is solely a function of the current tasks and may exist in spite of the typical skills of existing or potential users.
<TaskRec	DetReqSection	ReqOfUsers	EaseofUse>	what ease of use is needed by the user. It is useful for a requirements analysis to identify important usability issues that should influence any resulting designs. These issues can be positive, negative, or neutral.
<TaskRec	DetReqSection	ReqOfUsers	Responsibility>	what responsibilities / authority is required of user. Individuals may participate in a task at various levels. Different users will complete tasks which may be delegated in different manners.
<TaskRec	DetReqSection	ReqOfUsers	PotentialMisuse>	what potential is there for misuse. In the real world these tasks will be subject to a variety of misuses including errors in identifying the correct task made accidentally by valid users; slips in carrying out a task made accidentally by valid users; misuse performed intentionally by valid users; misuse performed intentionally by other than valid users
<TaskRec	DetReqSection	ReqOfUsers	Acceptance>	what is the level of user acceptance / satisfaction. Acceptance deals with recognizing that something has to

				be the way it is presented and done. Satisfaction generally implies acceptance, especially in cases where the user feels that a more satisfying alternative is possible.
<TaskRec	DetReqSection	ReqOfUsers	PhysCap>	what physical capabilities are required of the user. While many tasks that are suitable for computerized assistance do not appear to require special physical capabilities, there are many tools that may introduce such requirements. These physical requirements often are based on a desired mode of communication between the computer and the user.
<TaskRec	DetReqSection	Communications>		When users interact with the application, communications take place. The task may require users to communicate with other users or tools. Developers should know how to design the current use case for different language, different frequency, different media, and different security levels in communications or whether to create some new potential use cases to better serve these communications
<TaskRec	DetReqSection	Communications	IsFormal>	is the communication formal and/or informal. Formal communications are those which are mandated as part of accomplishing the task, regardless of whether or not their medium of communication (discussed below) is specified. Informal communications are those which are not mandated.
<TaskRec	DetReqSection	Communications	Language>	what is the language of communication. It refers to a shared language must be used for communication
<TaskRec	DetReqSection	Communications	IsOneWay>	is the communication one way or two way. One way communications usually transfer data or information either into the task or out of it, often on a scheduled basis. Two way communications often involve the user (or users) in making decisions.
<TaskRec	DetReqSection	Communications	Frequency>	what is the frequency of communication
<TaskRec	DetReqSection	Communications	Medium>	what is the medium of communication. Communications can take place via various general media types including: meeting, phone, computer network / Internet and printed material (memo, letter, report)
<TaskRec	DetReqSection	Communications	IsRecorded>	are records kept. Communications can be considered transactions that change our current state of data, information, knowledge, and/or wisdom. The task-related issue is whether or not records of particular communications needs to be kept.
<TaskRec	DetReqSection	Communications	Security>	what security / privacy does the communication need
<TaskRec	DetReqSection	Communications	IsCentralized>	are the users centralized and/or decentralized. The location of users has a considerable impact on many of the other aspects of user to user communications and to a lesser extent on user to system communications.
<TaskRec	DetReqSection	Communications	HowManyUser>	how many users communicate at a time. For communications to happen, there needs to be concern over synchronization of the communication, so that all required participants are available.

<TaskRec	DetReqSection	Communications	FlowDirection>	does the communication flow horizontal / vertical / other. For communications to take place effectively, it is important for all participants to know what their role is in the communication.
<TaskRec	DetReqSection	Learning		To accomplish a use case, users need to learn how to interact with the application. This implies that there might be a new use case for training. Training learning through different methods, feedback, time and environments, create different learning outcomes. Developers should consider the learning needs and capabilities of the intended users. Developers should know how to design a usable learning system and be aware that different methods, feedback, time and environments could influence the users' learning results.
<TaskRec	DetReqSection	Learning	WhoElse>	who else is involved in the learning. Learning can directly involve a number of people in addition to the learner, including: other learners, instructors /coaches and mentors/ advisors/ consultants
<TaskRec	DetReqSection	Learning	Prerequisite>	what are the prerequisites to learning. The developer needs to identify those prerequisites which have the greatest impact on the success of learning the particular task. One way is to categorize potential prerequisites
<TaskRec	DetReqSection	Learning	Level>	what different levels of learning are there. . Three stereotypical levels should be defined regardless of the particular subject of the learning: beginner / novice / apprentice / learner, regular user / journeyman / trained and expert / master / trainer.
<TaskRec	DetReqSection	Learning	UsedMethod>	what methods are used to learn. Learning can occur via a variety of methods, each with its own strengths and limitations such as Apprenticeship / on-the-job training, Consulting reference material, Completing a hands-on tutorial,
<TaskRec	DetReqSection	Learning	Feedback>	what kind of evaluation / feedback is provided to the learner. Feedback is part of learning. including: identifying the occurrence of errors and correct actions; identifying the specific error that occurred [knowledge]; explaining why it is an error [comprehension]; helping to correct the error [application]; helping to analyze what caused the error [analysis] planning a course of training to avoid similar errors
<TaskRec	DetReqSection	Learning	Time>	how does time relate to learning. There are a number of ways in which the demands of time and learning interact with each other, such as the frequency of users starting to learn, the variability in the length of time
<TaskRec	DetReqSection	Learning	Environment>	what is the environment for learning. There are some learning specific environmental concerns.
<TaskRec	DetReqSection	Learning	Stress>	what stresses are placed on the learning
<TaskRec	DetReqSection	ErrorHandling		Use cases should acknowledge where and when errors may occur. Developers should recognize these situations and determine how to help users avoid or handle them.
<TaskRec	DetReqSection	ErrorHandling	HelpForDifficulty>	what help exists for general difficulties, A system can help the user in a variety of ways including providing: documentation, on-line help, understandable error

				messages, help to diagnose cause of error and automatic error correction.
<TaskRec	DetReqSection	ErrorHandling	HelpForProblem>	what help exists for major problems. Some of problems may be identified in a risk analysis while others may relate to particulars of applications or systems not commonly associated with traditional risks.
<TaskRec	DetReqSection	ErrorHandling	IsControllable>	does the user have control of the task.It is especially important where the user is expected to fix errors.
<TaskRec	DetReqSection	ErrorHandling	ErrorAvoidance>	what error avoidance is provided. Each task needs to be analyzed for particular errors that may occur. The impact of these errors should be evaluated (based on risk management) and an appropriate management strategy should be chosen for each potential error.
<TaskRec	DetReqSection	ProblemDetail>		When developers design the solution for problems, the problem details should be thoroughly known. This information will help developers develop a more effective and more efficient design solution.
<TaskRec	DetReqSection	ProblemDetail	ProbNature>	what is the nature of the problems. Investigating the nature of a problem involves identifying the causes and symptoms of the problem as well as any special circumstances that control its occurrence.
<TaskRec	DetReqSection	ProblemDetail	UndoneGoal>	what purposes are not achieved. By focusing on purposes that are not achieved developers can balance the costs of fixes against the cost of allowing the problem to continue.
<TaskRec	DetReqSection	ProblemDetail	Efficiency>	how efficient are users at the task. Low levels of efficiency may indicate opportunities for improvement.
<TaskRec	DetReqSection	ProblemDetail	Unexpectedness>	what isn't working as planned. Differences between what was planned and what is happening do not necessarily mean that there is a problem in what is happening.
<TaskRec	DetReqSection	ProblemDetail	IsComplex>	is it too complex. Complexity often results from an attempt to be all things to all people. This can result in designs intended for generic users who never exist and which do not suit those real users who do exist.
<TaskRec	DetReqSection	ProblemDetail	IsConstraining>	are constraints too constraining. Constraints should not inhibit normal processing and should be flexible enough to allow the recovery from errors.
<TaskRec	DetReqSection	ProblemDetail	OtherOccuringProb>	what other problems are occurring. The developer needs to do this with great care in order not to turn this question into a promise that all problems will be addressed.
<TaskRec	DetReqSection	ProblemDetail	SuggestedImpr>	what improvements have been suggested. Developers should ask separately ask what improvements could be made to existing tasks and applications.
<UserRec>				It is a clear and unambiguous possibility record for each user group. The developer should investigate each user group by directly interviewing or otherwise obtaining information from a representative range of members of each identified group..
<UserRec	IdInfoSection			It identifies a user type record and give a unique name and detailed description to the record

<UserRec	IdInfoSection	Name>		a unique, meaningful identifier
<UserRec	IdInfoSection	Type>		user
<UserRec	IdInfoSection	Description>		describes the identifying characteristics of membership in this group, especially focusing on how the user group is different from other related user groups
<UserRec	LnkInfoSection			It identifies the set of linkages between that possibility and all other related possibilities
<UserRec	LnkInfoSection	Who>		It identifies other possible user groups related to this user group
<UserRec	LnkInfoSection	What>		It identifies possible tasks that may be performed by this user group
<UserRec	LnkInfoSection	How>		It identifies tools that may possibly be used by this user group
<UserRec	LnkInfoSection	WithWhichContent>		It identifies possible content chunks that may be used by this user group
<UserRec	LnkInfoSection	Scenario>		It identifies scenarios which tie together sets of users, tasks, tools, and content chunks
<UserRec	EnvInfoSection			It provides environmental information about this user group.
<UserRec	EnvInfoSection	When>		It identifies a range of possibilities of when this user group may perform related tasks, in terms of temporal attributes, and includes both current and potential future ranges
<UserRec	EnvInfoSection	Where>		Different users may operate in different environments. Each different environment, may involve different usability and accessibility challenges that need to be handled by a system for it to successfully meet the needs of that group of users
<UserRec	EnvInfoSection	HowMuch>		Quantifies the size and importance of this user group and includes both current and potential future measures. Different users may have differing levels of involvement with different use cases. High levels of involvement generally mean that users will stay familiar with the operations of systems used for the use case. Infrequent involvement may suggest the need for refresher style retraining before performing a use case or higher levels of help to assist in their performance.
<UserRec	EnvInfoSection	Why>		It identifies and evaluates the various justifications for serving this user group. Justification is an important predictor of potential future success. If a user's needs do not fit the overall development, or the cost of serving the user exceeds the resources available or the benefits of such service
<UserRec	DetReqSection>			This section is to identify specific design guidance related to the various user characteristics that may be unique to different user groups and to suggest a method for evaluating the identified characteristics and guidance and determining how to use this information throughout the development process.
<UserRec	DetReqSection	PhysCharCap>		There are various physical limitations and impairments the users may experience. Identifying this information, developers should consider how to design the application to fit the range of physical capabilities

				experienced by intended users, and whether they should design some new tools for users to reduce the impact of any physical limitations.
<UserRec	DetReqSection	PhysCharCap	DistinguishingChar>	Distinguishing characteristics , including: age, gender, race, physical size
<UserRec	DetReqSection	PhysCharCap	SensingCap>	Sensing capabilities including: sight, hearing, taste, smell, touch
<UserRec	DetReqSection	PhysCharCap	PhysCap>	Physical capabilities. including: voice, touch
<UserRec	DetReqSection	PhysCharCap	PerformingChar>	Performing, such as movement, characteristics, including speed, accuracy, strength
<UserRec	DetReqSection	MentalCharCap>		Users' mental characteristics influence how they typically react to a variety of interactions and interfaces. Developers should consider whether to create new tools and how to design the application to fit or change actors' mental capabilities.
<UserRec	DetReqSection	MentalCharCap	Personality>	Personality including: extroversion / introversion, sensory / intuitive, thinking / feeling, judging / perceiving
<UserRec	DetReqSection	MentalCharCap	EmontionalCap>	Emotional capabilities including: attitudes, beliefs, motivation, satisfaction, risk taking/aversion
<UserRec	DetReqSection	MentalCharCap	CognitiveCap>	Cognitive capabilities including: perception, reasoning, problem solving, decision making, adaptability, creative thinking, multitasking
<UserRec	DetReqSection	MentalCharCap	MemoryCap>	Memory capabilities including: sensory, working and long term memory, and memory caching
<UserRec	DetReqSection	MentalCharCap	Expertise>	Expertise including: intelligence, education, application experience, task experience, general computing experience
<UserRec	DetReqSection	SocialCharCap>		Users may come from various social communities with different social backgrounds. This information is important for designers to determine how to design the interfaces and interaction sequences for users who have the cultural and/or linguistic differences with each other.
<UserRec	DetReqSection	SocialCharCap	Recognition>	Recognizing the Effects of Social Relationships
<UserRec	DetReqSection	SocialCharCap	Types>	Major Types of Social Relationships
<UserRec	DetReqSection	SocialCharCap	Implementation>	Implementations of Social Relationships
<UserRec	DetReqSection	SocialCharCap	Application>	Applying a Usability First Approach to Dealing with Social Relations
<UserRec	DetReqSection	GroupCharCap>		User groups will generally recognize the predominance of a particular social group, as a support for the needs of individuals in the user group. Additional social groups may have an influence (intended or otherwise) on individual actions
<UserRec	DetReqSection	GroupCharCap	Membership>	Membership includes group type, membership criteria, cohesion. Membership in a group and or acting as a representative of a group may influence a user's actions. Developers should be made aware of group membership situations that may influence the actions of a user.
<UserRec	DetReqSection	GroupChar	Self-concept>	Self-concept including: past, present, and future

		Cap		concepts
<UserRec	DetReqSection	GroupChar Cap	Orientation>	Orientation including: activity, emphasis, directedness, flexibility
<UserRec	DetReqSection	GroupChar Cap	Relations>	Relations including: independence, interdependence, permeability, coordination, communication
<ContentRec>				It describes the Content structure model which is used to identify content and helps developers with analyzing and designing them.
<ContentRec	IdInfoSection>			This section serves to identify the basic content chunks to be analyzed. It identifies a content type record and give a unique name and detailed description to the record
<ContentRec	IdInfoSection	Name>		It is a unique, meaningful identifier
<ContentRec	IdInfoSection	Type>		It is a type of structure model
<ContentRec	IdInfoSection	Description>		It describes the intended accomplishment of the tool, especially focusing on how the tool is different from other related tasks
<ContentRec	LnkInfoSection			
<ContentRec	LnkInfoSection	Who>		It identifies possible user groups of this content chunk
<ContentRec	LnkInfoSection	What>		It identifies possible tasks that may use this content chunk
<ContentRec	LnkInfoSection	How>		It identifies tools that may possibly use this content chunk
<ContentRec	LnkInfoSection	WithWhich Content>		It identifies other possible content chunks that may be related to this content chunk
<ContentRec	LnkInfoSection	Scenario>		It identifies scenarios which tie together sets of users, tasks, tools, and content chunks
<ContentRec	EnvInfoSection			It provides environmental information about this content chunk
<ContentRec	EnvInfoSection	When>		Attributes may need to be handled differently in different temporal and environmental situations. For example, some situations may call for precise details while others may prefer summary data. Each different situation may involve different usability and accessibility challenges that need to be handled by a system for it to successfully work with an attribute.
<ContentRec	EnvInfoSection	Where>		It identifies a range of possibilities of where this content chunk may be used, in terms of physical attributes, and includes both current and potential future ranges
<ContentRec	EnvInfoSection	HowMuch>		It quantifies the size and importance of this content chunk and includes both current and potential future measures. Attributes may be used in a system at considerably different frequencies of use. High frequencies of use generally mean that users will stay familiar with the meaning, format, and use of attributes. Infrequent use may suggest the need for higher levels of assistance in working with particular attributes.
<ContentR	EnvInfoSection	Why>		It identifies and evaluates the various justifications for

ec				using this content chunk. If the cost of including an attribute exceeds the resources available or the benefits of such inclusion, it will be impractical to consider it for inclusion.
<ContentRec	DetReqSection>			This section serves to identify detailed content structures and other related requirements
<ContentRec	DetReqSection	Structure>		Interface designers should consider where it is necessary to organize several linked attributes or whether they should create some new attributes to design more meaningful information for the users.
<ContentRec	DetReqSection	Structure	DataAttributes>	what are the data attributes of the content chunk. This question deals with identifying and describing different individual data attributes of content that are combined to produce the content chunk.
<ContentRec	DetReqSection	Structure	Volume>	what volume of content is typically accessed at a single time. If a content chunk is trivial, then accessing and using it on its own may often be a waste of the user's time and effort. If a content chunk is too large, the user may only be able to consider only a portion of the chunk at a time.
<ContentRec	DetReqSection	Structure	IndividualInst>	how many individual instances are there of the content chunk. This includes identifying the typical and/or average expected number of instances of a content chunk and identifying the maximum and minimum numbers of possible instances.
<ContentRec	DetReqSection	Structure	RelatedInst>	how are individual instances of the content chunk related to each other. this question concerns the use of multiple content chunks, either in series or in parallel.
<ContentRec	DetReqSection	Semantics>		It is detailed analysis of the "what" of content chunks which focus on establishing the identity of the content chunk to the level of detail that all persons involved in the development share a common understanding of its purpose and validity.
<ContentRec	DetReqSection	Semantics	Representation>	what does the content chunk represent. The content chunks should have a more general purpose of representing some data, information, knowledge, or wisdom, which can exist apart from any application.
<ContentRec	DetReqSection	Semantics	HowFitIn>	how does it fit in the overall application. Content can fit into an application in a number of ways including: providing background, being used directly in accomplishing some tasks for some users, providing help / clarification and providing enrichment / further details.
<ContentRec	DetReqSection	Semantics	Reliability>	how reliable is the content chunk. This question informs the user of the relative level of reliability and/or factors that can help the user to objectively decide hoe much to rely on the content chunk.
<ContentRec	DetReqSection	Semantics	Alternative>	what are the alternatives to the content chunk. It is important to identify alternate sources of content to help users: place new content into context, handle difficulties in new content and avoid confusion and potential errors.
<ContentR	DetReqSection	Semantics	Privacy>	what are the needs for privacy. Privacy refers to the

ec				protection of information about an individual and/or organization from being accessed by unauthorized persons and/or organizations.
<ContentRec	DetReqSection	ReqOfUsers>		This section focuses on the relationships between content and its different user groups, including general relationship such as ownership and understanding that exist for all user - content pairings and specific relationships that exist only for certain user - content pairings.
<ContentRec	DetReqSection	ReqOfUsers	Interaction>	What interactions does the content expect of users. There are a wide variety of potential relationships between content and users. They can be identified from analyzing either the users or the content
<ContentRec	DetReqSection	ReqOfUsers	Understandability>	How well do different users understand the content. This question serves to recognize the actual level of understanding of the content by each group of users. Differing levels of understanding may be appropriate for different types of content.
<ContentRec	DetReqSection	ReqOfUsers	PotentialConfusion>	What is the potential for confusion regarding the content. Confusion can arise because of poorly presented content, content that is mistaken for similar but different content, content that conflicts with other content or content that conflicts with what the user “knows”.
<ContentRec	DetReqSection	ReqOfUsers	Permission>	what type of ownership/permission is exercised over the content
<ContentRec	DetReqSection	HowBeHandled>		This section is the elaboration of how the content is used. It serves to recognize current limitations placed on its current use and identify future potential for greater use and usability.
<ContentRec	DetReqSection	HowBeHandled	HowAccess>	how is the content accessed / used. Different users may have different levels of access (read only, read and write, and/or full update).
<ContentRec	DetReqSection	HowBeHandled	HowStore>	how is the content stored. The purpose of considering how the content is stored is to identify additional potential uses of the content that currently are impractical due to limitations resulting from its storage.
<ContentRec	DetReqSection	HowBeHandled	HowProtect>	how is the content protected. It is necessary to limit different types of use of some content
<ContentRec	DetReqSection	HowBeHandled	RepresentationStorage>	what forms of representation and storage are suitable for the content. Different users will have different preferences for how the same content is presented
<ContentRec	DetReqSection	WhenBeUsed>		This section elaborates to identify time dependent characteristics of content and its use. It considers both typical use and exceptional uses that may be difficult due to time related circumstances.
<ContentRec	DetReqSection	WhenBeUsed	Frequency>	what is the frequency of use of content chunks. The frequency of overall use by an application provides a measure of the influence of improving the usability of the content.
<ContentRec	DetReqSection	WhenBeUsed	SpentTime>	how much time is spent on the content. The amount of time spent on a content chunk is dependent both on the

				needs of the users and on the volume of the content in the chunk
<ContentRec	DetReqSection	WhenBeUsed	TimeforProfile>	what is the time of use profile of the content. The time of use profile can affect the availability of a chunk of content, and thus its usability. This question provides information on typical and peak usage and distributions of each.
<ContentRec	DetReqSection	WhenBeUsed	Unavailable>	when is the content not available. This question serves to identify other reasons not to make content available to users at certain times besides updating or limitations in the number of concurrent users
<ContentRec	DetReqSection	WhenBeUsed	Update>	when is the content updated. Content needs to be kept current to be reliable in its use.
<ContentRec	DetReqSection	WhereObtained&Used>		
<ContentRec	DetReqSection	WhereObtained&Used	WhereFrom>	where does the content come from. This question serves to identify what is the source of the content chunk and how this may impact the usability of the content.
<ContentRec	DetReqSection	WhereObtained&Used	WhereUsed>	where will the content be used. Content is typically used in a variety of environments both within and apart from the application / system. It considers the content needs that are within a system's boundaries and additional needs that may provide constraints on the content.
<ToolRec>				It describes theTool structure model which is used to identify tools and helps developers with analyzing and designing them.
<ToolRec	IdInfoSection>			It identifies a tool type record and give a unique name and detailed description to the record
<ToolRec	IdInfoSection	Name>		It is a unique, meaningful identifier
<ToolRec	IdInfoSection	Type>		It is a type of structure model
<ToolRec	IdInfoSection	Description>		It describes the intended accomplishment of the tool,, especially focusing on how the tool is different from other related tasks
<ToolRec	LnkInfoSection>			It identifies the set of linkages between that possibility and all other related possibilities
<ToolRec	LnkInfoSection	Who>		It identifies who belongs to a given user group can be described in terms of the characteristics that make an individual a member, especially focusing on how that user group is different from other user groups
<ToolRec	LnkInfoSection	What>		It identifies what the intended accomplishment of the tool, especially focusing on how that task is different from other related tasks
<ToolRec	LnkInfoSection	How>		It identifies how the tool is used in general (for all the users, tasks, and content for which it can be used without reference to differences in use for different users, tasks, or content), especially focusing on how that tool is different from other related tools
<ToolRec	LnkInfoSection	WithWhich Content>		It identifies which are the main components of the content chunk, especially focusing on how that content chunk is different from other related content chunks
<ToolRec	LnkInfoSection	Scenarios>		It identifies scenarios which tie together sets of users,

				tasks, tools, and content chunks
<ToolRec	EnvInfoSection			It provides environmental information about this tool
<ToolRec	EnvInfoSection	When>		It identifies a range of possibilities of when this tool could be used, in terms of temporal attributes, and includes both current and potential future ranges
<ToolRec	EnvInfoSection	When	FrequencyofUse >	What is the frequency of use. It is a measure of use in terms of number of separate uses. A tool that is used frequently is likely to have less usability concerns for regular users, since the users are likely to have mastered even difficult parts of the task through repeated practice
<ToolRec	EnvInfoSection	When	SpentTime>	How much time is spent by users. It is another measure of use in terms of cumulative time used
<ToolRec	EnvInfoSection	When	ParticularTime>	What particular times is the tool used. They are some timing concerns, including time of day, day of week, time of month or year. The reason we have these concerns is because the problems introduced due to interactions or competition with other tools and/or memory lapses are most pronounced when the tool is tied to a particular time of using.
<ToolRec	EnvInfoSection	When	PeakUsageDistribution>	what is the distribution of peak usage. The distribution profile of peak usage of a tool is especially important in determining the capacity requirements of tools to be used for the tasks.
<ToolRec	EnvInfoSection	When	PerformanceReq >	what are the execution requirements. The tool may introduce various performance related constraints on its execution. Additional constraints may be added based on potential interactions with other tools.
<ToolRec	EnvInfoSection	Where>		It identifies a range of possibilities of where this tool could be used, in terms of physical attributes, and includes both current and potential future ranges
<ToolRec	EnvInfoSection	Where	Configurations>	what workstation configurations may be used. Workstations can be considered extensions of the main tools (the Web site or other interactive system) we are developing. They are also the interface between those tools and their environment.
<ToolRec	EnvInfoSection	Where	Environments>	what environments may the tool have. Ideal environmental conditions have been specified by an International standard. The environmental factors include quality of space considerations, lighting / heat / air considerations, distractions considerations.
<ToolRec	EnvInfoSection	Where	EffectOnPerf>	how does the location effect performance. This involves identifying: the locations where the tool is required, or may be required, or may be desired to be accomplished; the workstation and environmental factors that are needed to support the tool in the locations; the factors that might inhibit using the tool in those locations and develop design(s) to overcome those inhibiting factors
<ToolRec	EnvInfoSection	HowMuch >		quantifies how often this tool might be used, and includes both current and potential future uses
<ToolRec	EnvInfoSection	Why>		It identifies and evaluates the various justifications for developing and using this tool. Justification is an important predictor of potential future success. If a use

				case does not fit the overall development, or costs of the use case exceed either the benefits of serving it or available resources, it will be impractical to develop special tools for the use case. Developers should know the factors that will influence the feasibility and acceptability of possible designs
<ToolRec	EnvInfoSection	Why	HowFit>	how does it fit into the overall development. While individual ttools contribute towards the release's overall justification, they then may be (circularly) justified by being part of the release
<ToolRec	EnvInfoSection	Why	IsEssential>	is it essential. The ways by which a given tool may be considered essential involve both internal factors such as being part of the fundamental definition of an application and external factors such as legal requirements.
<ToolRec	EnvInfoSection	Why	Prerequirsite>	is it in a prerequisite relationship with other tools. It intends to identify tools that are prerequisite to other tools or tools that have other prerequisites. This question should be answered by examining the relationships that are identified in "how" linkages
<ToolRec	EnvInfoSection	Why	CostEffective>	do benefits exceed costs. Costs and benefits can be identified with the following questions: tangible benefit concerns, intangible benefit concerns, tangible costs concerns, intangible costs concerns.
<ToolRec	DetReqSection>			The detailed information gained in analysis considers what essential limitations or requirements they place on design, It will be essential in designing a new system.
<ToolRec	DetReqSection	ToolOper>		This PUF field describes operational concerns. Elaboration is needed to understand the operations of essential use cases to evaluate how well current operations work and how future operations might provide improvements. When designers start interaction design, they should know whether the interaction meets the goal of the use case, whether there are alternatives to achieve the use case, and what feedbacks the use case should provide.
<ToolRec	DetReqSection	ToolOper	Purpose>	what is the purpose of the tool. It describes the general tools that are identified in "how" linkages.
<ToolRec	DetReqSection	ToolOper	IsFormal>	is this a formal or informal tool. Formal tools are the ones readily identified as part of an application or application system. Informal tools are those tools which are seldom acknowledged as necessary part of an application (or of the user's work) but which the user may use along with the formal tools of the application or work.
<ToolRec	DetReqSection	ToolOper	HowtoDo>	how is the tool done. It tries to identify the common, basic factors of use that are essential to the task or tool. An analysis of how tools are currently used may provide a starting point for identifying the relevant operations parameters for performing the task(s)
<ToolRec	DetReqSection	ToolOper	SubTools>	are there identifiable subtools. It describes the sub-tools that are identified in "how" linkages.
<ToolRec	DetReqSection	ToolOper	IsInterruptible>	can the tool be interrupted. Some tools impose further

				limitations on being interrupted that are not required by the actual task. It is desirable to maximize the potential for interruptions to occur without causing problems.
<ToolRec	DetReqSection	ToolOper	Alternative>	What are the alternatives to the tool? When and why will the users choose the alternatives and when and why will they choose our tool.
<ToolRec	DetReqSection	ToolOper	Flexibility>	how flexible / adaptable is the tool. Flexibility is important to be able to meet the needs of various types of users. Generally the more flexible the tool, the more users can accomplish the use the tool. Flexibility and adaptability help expand the applicability of the tool.
<ToolRec	DetReqSection	ToolOper	NeededRedundancy>	what redundancies are needed. If properly designed, redundancies can provide the user with increased usability. This section includes information about the costs / benefits of retaining redundancy
<ToolRec	DetReqSection	ToolOper	Feedback>	what feedback does the tool provide. Feedback is important to reassure the user both that a tool was used and that it was used correctly.
<ToolRec	DetReqSection	ToolOper	IsSharable>	is the tool sharable / concurrently usable. Some tools seem inherently individual and some seem inherently group tools. Some individual tools can benefit from help and some are hindered by interference.
<ToolRec	DetReqSection	ToolOper	Accessibility>	how accessible is the tool. Accessibility refers to the lack of limitations to usability. Where accessibility is limited or denied, usability becomes nonexistent to certain users in certain or in all circumstances.
<ToolRec	DetReqSection	ReqOfUsers		Different tools require different skills and abilities to operate them successfully. It is important to recognize the abilities and skills that will be necessary for a given tool and then to compare them with the skills and abilities of the various proposed users
<ToolRec	DetReqSection	ReqOfUsers	Speed>	what speed is required of the user. Speed is most essential for tasks that need to occur in “real time” which depends on the nature of the task. Speed of the user is dependent on the means of communication employed in the assignment and investigation.
<ToolRec	DetReqSection	ReqOfUsers	MentalCap>	what mental workload capability is required of the user. All tools place certain cognitive demands that make up part of the mental workload of a user.
<ToolRec	DetReqSection	ReqOfUsers	Skill>	what skills are required of the user. The skills may be gained via training or experience with the given tools or via transfer from similar tools. The need for these skills is solely a function of the current tools and may exist in spite of the typical skills of existing or potential users.
<ToolRec	DetReqSection	ReqOfUsers	EaseofUse>	what ease of use is needed by the user. It is useful for a requirements analysis to identify important usability issues that should influence any resulting designs. These issues can be positive, negative, or neutral.
<ToolRec	DetReqSection	ReqOfUsers	Responsibility>	what responsibilities / authority is required of user
<ToolRec	DetReqSection	ReqOfUsers	PotentialMisuse>	what potential is there for misuse. In the real world these tools will be subject to a variety of misuses including errors in identifying the correct tool made accidentally

				by valid users; slips in using a tool made accidentally by valid users; misuse performed intentionally by valid users; misuse performed intentionally by other than valid users
<ToolRec	DetReqSection	ReqOfUsers	Acceptance>	what is the level of user acceptance / satisfaction. Acceptance deals with recognizing that something has to be the way it is presented and done. Satisfaction generally implies acceptance, especially in cases where the user feels that a more satisfying alternative is possible.
<ToolRec	DetReqSection	ReqOfUsers	PhysCap>	what physical capabilities are required of the user. While many tasks that are suitable for computerized assistance do not appear to require special physical capabilities, there are many tools that may introduce such requirements. These physical requirements often are based on a desired mode of communication between the computer and the user.
<ToolRec	DetReqSection	Communications>		Developers should consider the potential impact of different media and methods that might be used to communicate with users and with other linked tools. This involves recognizing the effectiveness of various current media and methods.
<ToolRec	DetReqSection	Communications	IsFormal>	is the communication formal and/or informal. Formal communications are those which are mandated as part of using the tool, regardless of whether or not their medium of communication (discussed below) is specified. Informal communications are those which are not mandated.
<ToolRec	DetReqSection	Communications	Language>	what is the language of communication. It refers to a shared language must be used for communication
<ToolRec	DetReqSection	Communications	IsOneWay>	is the communication one way or two way. One way communications usually transfer data or information either into the task or out of it, often on a scheduled basis. Two way communications often involve the user (or users) in making decisions.
<ToolRec	DetReqSection	Communications	Frequency>	what is the frequency of communication
<ToolRec	DetReqSection	Communications	Medium>	what is the medium of communication. Communications can take place via various general media types including: meeting, phone, computer network / Internet and printed material (memo, letter, report)
<ToolRec	DetReqSection	Communications	IsRecorded>	are records kept. Communications can be considered transactions that change our current state of data, information, knowledge, and/or wisdom. The tool-related issue is whether or not records of particular communications needs to be kept.
<ToolRec	DetReqSection	Communications	Security>	what security / privacy does the communication need
<ToolRec	DetReqSection	Communications	IsCentralized>	are the users centralized and/or decentralized, The location of users has a considerable impact on many of the other aspects of user to user communications and to a lesser extent on user to system communications.
<ToolRec	DetReqSection	Communic	HowManyUser>	how many users communicate at a time. For

		ations		communications to happen, there needs to be concern over synchronization of the communication, so that all required participants are available.
<ToolRec	DetReqSection	Communications	FlowDirection>	does the communication flow horizontal / vertical / other. For communications to take place effectively, it is important for all participants to know what their role is in the communication. Tools to assist with these communications should not violate this authority / responsibility relationship.
<ToolRec	DetReqSection	Learning>		Developers should consider the learning needs and capabilities of the intended users. It is possible that additional tools must be built to help users learn tools being developed that serve application-related use cases.
<ToolRec	DetReqSection	Learning	WhoElse>	who else is involved in the learning. Learning can directly involve a number of people in addition to the learner, including: other learners, instructors /coaches and mentors/ advisors/ consultants
<ToolRec	DetReqSection	Learning	Prerequisite>	what are the prerequisites to learning. The developer needs to identify those prerequisites which have the greatest impact on the success of learning the particular task. One way is to categorize potential prerequisites
<ToolRec	DetReqSection	Learning	Level>	what different levels of learning are there. Three stereotypical levels should be defined regardless of the particular subject of the learning: beginner / novice / apprentice / learner, regular user / journeyman / trained and expert / master / trainer.
<ToolRec	DetReqSection	Learning	UsedMethod>	what methods are used to learn. Learning can occur via a variety of methods, each with its own strengths and limitations such as Apprenticeship / on-the-job training, Consulting reference material, Completing a hands-on tutorial,
<ToolRec	DetReqSection	Learning	Feedback>	what kind of evaluation / feedback is provided to the learner. Feedback is part of learning. including: identifying the occurrence of errors and correct actions; identifying the specific error that occurred [knowledge]; explaining why it is an error [comprehension]; helping to correct the error [application]; helping to analyze what caused the error [analysis] planning a course of training to avoid similar errors
<ToolRec	DetReqSection	Learning	Time>	how does time relate to learning. There are a number of ways in which the demands of time and learning interact with each other, such as the frequency of users starting to learn, the variability in the length of time
<ToolRec	DetReqSection	Learning	Environment>	what is the environment for learning. There are some learning specific environmental concerns.
<ToolRec	DetReqSection	Learning	Stress>	what stresses are placed on the learning
<ToolRec	DetReqSection	ErrorHandling>		Tools should acknowledge what errors might occur from the user-side and tool-side. Developers should recognize these situations and determine how to help users avoid or handle them.
<ToolRec	DetReqSection	ErrorHandling	HelpForDifficulty>	what help exists for general difficulties. A system can help the user in a variety of ways including providing: documentation, on-line help, understandable error

				messages, help to diagnose cause of error and automatic error correction
<ToolRec	DetReqSection	ErrorHandling	HelpForProblem>	what help exists for major problems. Some of problems may be identified in a risk analysis while others may relate to particulars of applications or systems not commonly associated with traditional risks.
<ToolRec	DetReqSection	ErrorHandling	IsControllable>	does the user have control of the tool. It is especially important where the user is expected to fix errors.
<ToolRec	DetReqSection	ErrorHandling	ErrorAvoidance>	what error avoidance is provided. Each tool needs to be analyzed for particular errors that may occur. The impact of these errors should be evaluated (based on risk management) and an appropriate management strategy should be chosen for each potential error.
<ToolRec	DetReqSection	ProblemDetail>		This field/property is used to document known problems with this tool and tools that it is designed to replace.
<ToolRec	DetReqSection	ProblemDetail	ProbNature>	what is the nature of the problems. Investigating the nature of a problem involves identifying the causes and symptoms of the problem as well as any special circumstances that control its occurrence.
<ToolRec	DetReqSection	ProblemDetail	UndoneGoal>	what purposes are not achieved. By focusing on purposes that are not achieved developers can balance the costs of fixes against the cost of allowing the problem to continue.
<ToolRec	DetReqSection	ProblemDetail	Efficiency>	how efficient are users at the tool. Low levels of efficiency may indicate opportunities for improvement.
<ToolRec	DetReqSection	ProblemDetail	Unexpectedness>	what isn't working as planned . Differences between what was planned and what is happening do not necessarily mean that there is a problem in what is happening.
<ToolRec	DetReqSection	ProblemDetail	IsComplex>	is it too complex. Complexity often results from an attempt to be all things to all people. This can result in designs intended for generic users who never exist and which do not suit those real users who do exist.
<ToolRec	DetReqSection	ProblemDetail	IsConstraining>	are constraints too constraining. Constraints should not inhibit normal processing and should be flexible enough to allow the recovery from errors.
<ToolRec	DetReqSection	ProblemDetail	OtherOccuringProb>	what other problems are occurring. The developer needs to do this with great care in order not to turn this question into a promise that all problems will be addressed.
<ToolRec	DetReqSection	ProblemDetail	SuggestedImpr>	what improvements have been suggested. Developers should ask separately ask what improvements could be made to existing tools and applications.
<ConstraintRec>				It describes the constraints structure model which is used to identify constraints and helps developers with analyzing and designing them.
<ConstraintRec	IdInfoSection>			It identifies a constraints type record and give a unique name and detailed description to the record
<ConstraintRec	IdInfoSection	Name>		It is a unique, meaningful identifier
<Constrai	IdInfoSection	Type>		It is a type of structure model

ntRec				
<Constraintrec	IdInfoSection	Description>		It describes the intended accomplishment of the constraints
<Constraintrec	LnkInfoSection			
<Constraintrec	LnkInfoSection	Who>		It identifies effected user groups
<Constraintrec	LnkInfoSection	What>		It identifies effected tasks
<Constraintrec	LnkInfoSection	How>		It identifies effected tools
<Constraintrec	LnkInfoSection	WithWhichContent>		It identifies effected content chunks
<Constraintrec	LnkInfoSection	Scenario>		It identifies effected scenarios
<Constraintrec	LnkInfoSection	Constraint>		It identifies related constraints
<Constraintrec	EnvInfoSection>			
<Constraintrec	EnvInfoSection	When>		It identifies any temporal attributes that effect the constraint
<Constraintrec	EnvInfoSection	Where>		It identifies any physical attributes that effect the constraint
<Constraintrec	EnvInfoSection	HowMuch>		It discusses the potential consequences of the constraint
<Constraintrec	EnvInfoSection	Why>		It provides further information on why the constraint is relevant to current and/or future developments

APPENDIX 9

Below is the complete list of UML XML tags and their meanings from UML semantics [35].

Package Name	SubPackage Name	Package Construct Name	Class Construct Name	Description
Foundation				The Foundation package defines structural or static modeling concepts. It is often called the language infrastructure, and consists of the Data Types package, the Core package and the Extension Mechanisms package.
Foundation	Core>			Core sub package specifies the concepts required for an elementary metamodel and defines an architectural backbone for attaching additional language constructs, such as metaclasses, metaassociations, and metaattributes.
Foundation	Core	Element		An element is an atomic constituent of a model.
Foundation	Core	ModelElement		A model element is an element that is an abstraction drawn from the system being modeled.
Foundation	Core	ModelElement	Name	An identifier for the ModelElement within its containing Namespace.
Foundation	Core	ModelElement	constraint	A set of Constraints affecting the element.
Foundation	Core	ModelElement	comment	
Foundation	Core	Classifier>		A classifier is an element that describes behavioral and structural features; it comes in several specific forms, including class, data type, interface, and others that are defined in other metamodel packages
Foundation	Core	Classifier	feature	A list of Features, like Attribute, Operation, Method, owned by the Classifier.
Foundation	Core	Classifier	participant	Inverse of specification on association to AssociationEnd.
Foundation	Core	Class>		A class is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics
Foundation	Core>			Core sub package specifies the concepts required for an elementary metamodel and defines an architectural backbone for attaching additional language constructs, such as metaclasses, metaassociations, and metaattributes.
Foundation	Core	Class	isActive	Specifies whether an Object of the Class maintains its own thread of control. If true, then an Object has its own thread of control and runs concurrently with other active Objects. If false, then Operations run in the address space and under the control of the active Object that controls the caller
Foundation	Core	AssociationEnd		An association end is an endpoint of an association, which connects the association to a classifier. Each association end is part of one association; the association-ends of each association are ordered

Foundation	Core	AssociationEnd	ordering	When placed on a target end, specifies whether the set of links from the source instance to the target instance is ordered.
Foundation	Core	AssociationEnd	aggregation	When placed on a target end, specifies whether the target end is an aggregation with respect to the source end.
Foundation	Core	AssociationEnd	association	the association which the association end connects
Foundation	Core	AssociationEnd	multiplicity	When placed on a target end, specifies the number of target instances that may be associated with a single source instance across the given Association
Foundation	Core	AssociationEnd	qualifier	An optional list of qualifier Attributes for the end. If the list is empty then the Association is not qualified
Foundation	Core	AssociationEnd	type	Designates the Classifier connected to the end of the Association.
Foundation	Core	AssociationEnd	specification	
Foundation	Core	Constraint		A constraint is a semantic condition or restriction.
Foundation	Core	Constraint	body	A BooleanExpression that must be true when evaluated for an instance of a system to be well formed.
Foundation	Core	Constraint	constrainedElement	A ModelElement or list of ModelElements affected by the Constraint.
Foundation	Core	Association		An association defines a semantic relationship between classifiers; the instances of an association are a set of tuples relating instances of the classifiers.
Foundation	Core	Association	connection	An Association consists of at least two AssociationEnds, each of which represents a connection of the association to a Classifier.
Foundation	Core	Attribute		An attribute is a named slot within a classifier that describes a range of values that instances of the classifier may hold.
Foundation	Core	Attribute	initialValue	An Expression specifying the value of the attribute upon initialization. It is meant to be evaluated at the time the object is initialized.
Foundation	Core	BehavioralFeature		
Foundation	Core	BehavioralFeature	isQuery	Specifies whether an execution of the Feature leaves the state of the system unchanged. True indicates that the state is unchanged; false indicates that side-effects may occur.
Foundation	Core	BehavioralFeature	parameter	An ordered list of Parameters for the Operation.
Foundation	Core	Operation		An operation is a service that can be requested from an object to effect behavior.
Foundation	Core	Operation	concurrency	Specifies the semantics of concurrent calls to the same passive instance,
Foundation	Core	Operation	specification	Description of the effects of performing an Operation, stated as an Expression.
Foundation	Core	Operation	method	a Method is a declaration of a named piece of behavior in a Classifier and realizes one or a set of Operations of the Classifier.
Foundation	Core	Method		A method is the implementation of an operation.

				It specifies the algorithm or procedure that effects the results of an operation.
Foundation	Core	Method	body	The implementation of the Method as a Procedural Expression.
Foundation	Core	Method	specification	Designates an Operation that the Method implements. The Operation must be owned by the Classifier that owns the Method or be inherited by i
Foundation	Core	Generalization		A generalization is a taxonomic relationship between a more general element and a more specific element
Foundation	Core	Generalization	discriminator	Designates the partition to which the Generalization link belongs.
Foundation	Core	Generalization	child	Designates a GeneralizableElement that is the specialized version of the supertype GeneralizableElement.
Foundation	Core	Generalization	parent	Designates a GeneralizableElement that is the generalized version of the subtype
Foundation	Core	AssociationClass		An association class is an association that is also a class. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not any of the classifiers
Foundation	Core	Component		A component is a reusable part that provides the physical packaging of model elements.
Foundation	Core	Component	deploymentLocation	The set of Nodes the Component is residing on.
Foundation	Core	Component	residentElement	
Foundation	Core	Comment		A comment is an annotation attached to a model element or a set of model elements.
Foundation	Core	Comment	annotatedElement	theModelElements which the Comment associated with
Foundation				
Foundation	Core	Dependency		A dependency states that the implementation or functioning of one or more elements require the presence of one or more other elements.
Foundation	Core	Dependency	client	The ModelElement or set of ModelElements that require the presence of the supplier.
Foundation	Core	Dependency	supplier	The ModelElement or set of ModelElements whose presence is required by the client.
Foundation				
Foundation	Core	A_constrainedElement_constraint		The model element have arbitrary constraints
Foundation	Core	A_constrainedElement_constraint	constrainedElement	An ordered list of elements subject to the constraint. The constraint applies to their instances
Foundation	Core	A_constrainedElement_constraint	constraint	A constraint that must be satisfied for instances of the model element
Foundation	.Extension_Mechanisms			The Extension Mechanisms package specifies how model elements are customized and extended with new semantics.

Foundation	.Extension_Mechanisms	Stereotype		The stereotype concept provides a way of classifying (marking) elements so that they behave in some respects as if they were instances of new "virtual" metamodel constructs
Foundation	.Extension_Mechanisms	Stereotype	baseClass	Species the name of a UML modeling element to which the stereotype applies, such as Class, Association, Refinement, Constraint, etc.
Foundation	.Extension_Mechanisms	Stereotype	requiredTag	Specifies a set of tagged values, each of which specifies a tag that an element classified by the stereotype is required to have
Foundation	.Extension_Mechanisms	Stereotype	extendedElement	Designates the model elements affected by the stereotype. Each one must be a model element of the kind specified by the baseClass attribute.
Foundation	.Extension_Mechanisms	Stereotype	stereotypeConstraint	Designates constraints that apply to elements bearing the stereotype.
Foundation	.Extension_Mechanisms	TaggedValue		A tagged value is a (Tag, Value) pair that permits arbitrary information to be attached to any model element.
Foundation	.Extension_Mechanisms	TaggedValue	tag	A name that indicates an extensible property to be attached to ModelElements.
Foundation	.Extension_Mechanisms	TaggedValue	value	An arbitrary value. The value must be expressible as a string for uniform manipulation
Foundation	.Extension_Mechanisms	TaggedValue	stereotype	Designates at most one stereotype that further qualifies the UML class (the base class) of the modeling element.
Foundation	.Extension_Mechanisms	TaggedValue	modelElement	Any model element have arbitrary tagged values
Foundation	.Extension_Mechanisms	A_constrainedElement2_stereotypeConstraint	constrainedElement2	An ordered list of elements subject to the constraint. The constraint applies to their instances.
Foundation	.Extension_Mechanisms	A_constrainedElement2_stereotypeConstraint	Constraint	A constraint that must be satisfied for instances of the model element
Behavioral_Elements				The Behavioral Elements package defines behavioral or dynamic modeling concepts, often called the language superstructure, and consists of The Common Behavior package, The Collaborations package,The Use Cases package ,The State Machines package, The Activity Graphs package and The Actions package
Behavioral_Elements	Common_Behavior			Common Behavior specifies the core concepts required for behavioral elements.
Behavioral_Elements	Common_Behavior	Instance		The instance construct defines an entity to which a set of operations can be applied and which has a state that stores the effects of the operations.
Behavioral_Elements	Common_Behavior	Instance	classifier	The set of AttributeLinks that holds the attribute values of the Instance.
Behavioral_Elements	Common_Behavior	Instance	attributeLink	The set of AttributeLinks that holds the attribute values of the Instance.
Behavioral_Elements	Common_Behavior	Instance	linkEnd	The set of AttributeLinks that holds the attribute values of the Instance.
Behavioral_Elements	Common_Behavior	Action		An action is a specification of an executable statement that forms an abstraction of a computational procedure that results in a change in

				the state of the model, realized by sending a message to an object or modifying a value of an attribute.
Behavioral_Elements	Common_Behavior	Action	recurrence	An Expression stating how many times the Action should be performed.
Behavioral_Elements	Common_Behavior	Action	target	An ObjectSetExpression which determines the target of the Action.
Behavioral_Elements	Common_Behavior	Action	isAsynchronous	Indicates whether ot not the caller waits for the completion of the execution of the Operation but continues immediately.
Behavioral_Elements	Common_Behavior	Action	actualArgument	A sequence of Expressions which determines the actual arguments needed when evaluating the Action
Behavioral_Elements	Common_Behavior	Action	actionSequence	An action sequence is a collection of actions.
Behavioral_Elements	Common_Behavior	Action	stimulus	
Behavioral_Elements	Common_Behavior	Object		An object is an instance that originates from a class.
Behavioral_Elements	Common_Behavior	Link		The link construct is a connection between instances.
Behavioral_Elements	Common_Behavior	Link	association	The Association the is the declaration of the Link.
Behavioral_Elements	Common_Behavior	Link	connection	
Behavioral_Elements	Common_Behavior	CallAction		A call action is an action resulting in an invocation of an operation on an instance.
Behavioral_Elements	Common_Behavior	CallAction	operation	an operation on an instance is invocated
Behavioral_Elements	Common_Behavior	SendAction		A send action is an action that results in the (asynchronous) sending of a signal.
Behavioral_Elements	Common_Behavior	SendAction	signal	A signal is a specification of an asynchronous stimulus communicated between instances. The receiving instance handles the signal by a state machine.
Behavioral_Elements	Common_Behavior	ActionSequence		An action sequence is a collection of actions.
Behavioral_Elements	Common_Behavior	ActionSequence	action	A sequence of Actions performed sequentially as an atomic unit.
Behavioral_Elements	Collaborations			The Collaborations package specifies a behavioral context for using model elements to accomplish a particular task
Behavioral_Elements	Collaborations	Collaboration		A collaboration describes how an operation or a classifier, like a use case, is realized by a set of classifiers and associations used in a specific way.
Behavioral_Elements	Collaborations	Collaboration	interaction	The set of Interactions that are defined within the Collaboration.
Behavioral_Elements	Collaborations	Collaboration	representedClassifier	The Classifier the Collaboration is a realization of.
Behavioral_Elements	Collaborations	Collaboration	representedOperation	The Operation the Collaboration is a realization of.
Behavioral_Ele	Collaboration	Collaboration	constrainingE	The ModelElements that add extra constraints, like Generalization and Constraint, on the

ments	s		lement	ModelElements participating in the Collaboration.
Behavioral_Elements	Collaborations	ClassifierRole		A classifier role is a specific role played by a participant in a collaboration.
Behavioral_Elements	Collaborations	ClassifierRole	multiplicity	The number of Instances playing this role in a Collaboration.
Behavioral_Elements	Collaborations	ClassifierRole	base	A ClassifierRole that is a projection of a Classifier.
Behavioral_Elements	Collaborations	ClassifierRole	availableFeature	The subset of Features of the Classifier which is used in the Collaboration
Behavioral_Elements	Collaborations	ClassifierRole	availableContents	the content of the messages
Behavioral_Elements	Collaborations	AssociationRole		An association role is a specific usage of an association needed in a collaboration.
Behavioral_Elements	Collaborations	AssociationRole	multiplicity	The number of Links playing this role in a Collaboration.
Behavioral_Elements	Collaborations	AssociationRole	base	An AssociationRole that is a projection of an Association.
Behavioral_Elements	Collaborations	AssociationEndRole		An association-end role specifies an endpoint of an association as used in a collaboration.
Behavioral_Elements	Collaborations	AssociationEndRole	collaborationMultiplicity	The number of LinkEnds playing this role in a Collaboration.
Behavioral_Elements	Collaborations	AssociationEndRole	base	An AssociationEndRole is a projection of an AssociationEnd.
Behavioral_Elements	Collaborations	AssociationEndRole	availableQualifier	
Behavioral_Elements	Collaborations	Message		A message defines how a particular request is used in an interaction.
Behavioral_Elements	Collaborations	Message	base	The specification of the Message.
Behavioral_Elements	Collaborations	Message	interaction	The set of Interactions that are defined during the execution of the current Message
Behavioral_Elements	Collaborations	Message	activator	The Message that called the operation whose method contains the current Message.
Behavioral_Elements	Collaborations	Message	sender	The role of the Instance that sends the Message and possibly receives a response.
Behavioral_Elements	Collaborations	Message	receiver	The role of the Instance that receives the Message and reacts to it.
Behavioral_Elements	Collaborations	Message	predecessor	The set of Messages whose completion enables the execution of the current Message
Behavioral_Elements	Collaborations	Interaction		An interaction specifies the messages sent between instances performing a specific task. Each interaction is defined in the context of collaboration.
Behavioral_Elements	Collaborations	Interaction	message	The Messages that specify the communication in the Interaction.
Behavioral_Elements	Collaborations	Interaction	context	The Collaboration which defines the context of the Interaction.
Behavioral_Elements	Use_Cases			The Use Case package specifies behavior using actors and use cases.
Behavioral_Elements	Use_Cases	UseCase		The use case construct is used to define the behavior of a system or other semantic entity without revealing the entity's internal structure.

				Each use case specifies the interaction between a actor and the system itself by specifying a sequence of actions which the system takes
Behavioral_Elements	Use_Cases	UseCase	extensionPoint	A list of strings representing extension points defined within the use case. An extension point is a location at which the use case can be extended with additional behavior.
Behavioral_Elements	Use_Cases	Actor		An actor defines a coherent set of roles that users of an entity can play when interacting with the entity. An actor has one role for each use case with which it communicates.
Behavioral_Elements	Use_Cases	UseCaseInstance		A use case instance is the performance of a sequence of actions being specified in a use case.
Behavioral_Elements	Use_Cases	Extend		Extends relationship indicates the behaviours modeled in an extended use case can be supplemented by the extending use cases. In other words, it adds extra functionality to the base use case while the base use case is complete on its own.
Behavioral_Elements	Use_Cases	Extend	condition	The requirements which must be satisfied to establish an extends relationship
Behavioral_Elements	Use_Cases	Extend	base	The parent (“base”) use case which is extended.
Behavioral_Elements	Use_Cases	Include		Includes relationship shows the common behaviors modeled in a use case is included in another use case. Includes relationship makes the functionality of reuse possible. The base use case is not a valid use case on its own.
Behavioral_Elements	Use_Cases	Include	addition	The additional behavior which is added into the base use case.
Behavioral_Elements	Use_Cases	Include	base	The parent (“base”) use case which is completed
Behavioral_Elements	State_Machines			The State Machines package defines behavior using finite-state transition systems.
Behavioral_Elements	State_Machines	StateMachine		A state machine is a behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.
Behavioral_Elements	State_Machines	StateMachine	context	An association to a ModelElement constrained to be a Classifier or a BehavioralFeature.
Behavioral_Elements	State_Machines	StateMachine	top	Designates the top level State directly owned by the StateMachine.
Behavioral_Elements	State_Machines	StateMachine	transitions	Associates the StateMachine with its Transitions
Behavioral_Elements	State_Machines	StateMachine	subMachineState	Semantically equivalent to composite states, submachine States have substates that are contained within a submachine.
Behavioral_Elements	State_Machines	Event		An event is the specification of a significant occurrence that has a location in time and space. An instance of an event can lead to the activation of a behavioral feature in an object.
Behavioral_Elements	State_Machines	Event	state	a condition or situation during which is waits for some event.
Behavioral_Elements	State_Machines	Event	transition	The transition may take place during the occurrence of this type of event

Behavioral_Elements	State_Machines	State		A State is a condition or situation during the life of an object during which is satisfies some condition, performs some activity, or
Behavioral_Elements	State_Machines	State	entry	An optional ActionSequence that is executed when the State is entered.
Behavioral_Elements	State_Machines	State	exit	An optional ActionSequence that is executed when the State is exited.
Behavioral_Elements	State_Machines	stateMachine		A state machine is a behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.
Behavioral_Elements	State_Machines	State	deferrableEvent	A list of Events the effect of whose occurrence during the State is postponed until the owner enters a State in which they are not deferred, at which time they may trigger Transitions as if they had just occurred.
Behavioral_Elements	State_Machines	State	internalTransition	A set of Transitions that occur entirely within the State
Behavioral_Elements	State_Machines	TimeEvent		A TimeEvent is a subtype of Event for modeling event instances resulting from the expiration of a deadline.
Behavioral_Elements	State_Machines	TimeEvent	when	Specifies the corresponding time deadline.
Behavioral_Elements	State_Machines	CallEvent		A call event is the reception of a request to invoke an operation. The expected result is the execution of the operation.
Behavioral_Elements	State_Machines	CallEvent	operation	Designates the operation whose invocation is requested.
Behavioral_Elements	State_Machines	SignalEvent		A SignalEvent represents events that result from the reception of a signal by an object.
Behavioral_Elements	State_Machines	SignalEvent	signal	Designates the Signal whose reception by the state owner may trigger a Transition..
Behavioral_Elements	State_Machines	Transition		A Transition is a relationship between a source state vertex and a target state vertex.
Behavioral_Elements	State_Machines	Transition	guard	Predicate that must evaluate to true at the instant the Transition is triggered.
Behavioral_Elements	State_Machines	Transition	effect	Specifies an ActionSequence to be performed when the Transition fires.
Behavioral_Elements	State_Machines	Transition	state	a condition or situation during the transition
Behavioral_Elements	State_Machines	Transition	trigger	Specifies the single Event which activates it
Behavioral_Elements	State_Machines	Transition	stateMachine	StateMachine where the transition takes place.
Behavioral_Elements	State_Machines	Transition	source	Designates the StateVertex affected by firing the Transition.
Behavioral_Elements	State_Machines	Transition	target	Designates the StateVertex that results from a firing of the Transition when the StateMachine was originally in the source State. After the firing the StateMachine is in the target State.
Behavioral_Elements	State_Machines	CompositeState		A composite state is a state that consists of substates.
Behavioral_Ele	State_Machin	CompositeState	isConcurrent	A boolean value that specifies the decomposition

ments	es			semantics. If this attribute is true, then the composite state is decomposed directly into two or more orthogonal conjunctive components (usually associated with concurrent execution). If this attribute is false, then there are no direct orthogonal components in the composite.
Behavioral_Elements	State_Machines	CompositeState	subvertex	Designates a set of States that constitute the substates of a CompositeState.
Behavioral_Elements	State_Machines	SimpleState		A SimpleState is a state that does not have substates.
Behavioral_Elements	State_Machines	SubmachineState		A SubmachineState represents a nested state machine.
Behavioral_Elements	State_Machines	SubmachineState	submachine	Represents the substate machine where the SubmachineState belongs

APPENDIX 10

This appendix serves to guide users of the Mapping Tool in doing good mappings and to suggest a validation procedure.

The following information illustrates a suggested procedure of creating meaningful mappings according to the Mapping Tool design. Some steps can be skipped based on developer's experience. The sequence of the steps is changeable.

1. **Determine a PUF XML tag and a XMI one**. The first part constituting the SELECT page is for choosing a PUF XML tag as a source tag of the mapping. Four drop down lists correspond to four tag constructs composing the PUF XML tag, namely **PUF Record Name, PUF Section Name, PUF Question Name and PUF Sub Question Name**. This set of drop down lists can narrow down the options so that users do not have to locate a tag out of over a hundred of options. Developers can easily determine a tag by choosing its tag constructs from the four drop down lists successively.

   The second part of the SELECT page is for choosing a XMI tag as a target tag. Similar to the first part, it also composed by four drop down lists, namely, **UML package Name, UML Sub Package Name, UML Package Construct Name,** and **UML Class Construct Name.** Developers can determine a XMI tag by choosing each tag construct from its corresponding drop down list.

   This sub step is crucial to the correctness of the mapping. Here are some suggestions on identifying the proper XMI tag:

   - Doing mappings starts from the high level mapping. Developer can refer to Figure 5.1 in the thesis to identify the UML diagram (UML sub package in XMI file) to which they should map.
   - As the mapping goes into the detailed level, developers are required to know what this XMI tag does and identify the semantic differences between the source tag and the XMI tag. It requires developers have certain knowledge about both methodologies. To meet this requirement, they can either turn to reference materials about PUF and UML before they do mappings or refer to the assistant information

provided by the Mapping Tool while they doing them. The latter approach will be illustrated in Step 2.

- The PUF XML tags in the Identification Information section are suggested to map to the *Classifier* or *ModelElement* package construct in the *Core* sub package. Those ones in the Detailed Requirement section are suggested to map to the XMI tag identified in the high level mapping.

The Mapping Tool allows developers to do individual tag mapping and group tag mapping by giving 0 or ALL values. The 0 value represents that the chosen tag has no sub tag so it would be individual tag mapping if it is chosen. The ALL value shows the chosen tag represents all its sub tags so it would be group tag mapping if it is chosen. Users are able to determine if it is one-to-one mapping or many-to-one mapping by finding the values in the four drop down lists.

2. **Understand the PUF XML file and the XMI file from syntactic and semantic perspectives**. Considering software engineers may be not familiar with the PUF methodology or usability engineers may not know UML very well, this step requires the developer doing the mapping to possess basic knowledge about two methodology requirements. The HELP pages created for each PUF XML tag and each XMI tag serves to provide tags' meanings, their sub tags and their syntactic locations in the hieratical methodology requirements. To access to the help information, the developer should choose a PUF XML tag or XMI tag. The detailed description of the procedure is given in Step 1. The Help link next to the tag construct leads to the HELP page for the chosen tag. The characteristics of the tag and how to use the tag will show on the HELP page.
3. **Determine a mapping relation between the PUF XML tag and the XMI one.** After having identified the syntactic and semantic differences between the PUF XML tag and the XMI tag, developers should choose a mapping relation from the drop down list shown in the third part of the SELECT page. There are three options in the drop down list, namely *Exact Relation, Inclusion Relation* and *Non-existing Relation.* If two tags' meanings and functions are identical, their mapping relation should be *Exact Relation*. If their meanings and functions have

certain overlap, their mapping relation should be *Inclusion Relation.* If their meanings and functions have no any overlap, their mapping relation should be *Non-existing Relation.*

Once the PUF XML, the XMI tags are their mapping relation are chosen in the SELECT page, developers can click the **Submit** button to create this new added mapping which will show in the DISPLAY page,

4. **Edit/Delete/Add more mappings.** The DISPLAY page is composed of two parts. The top part is for showing the mapping information about the existing mappings. The bottom one is a list of unmapped PUF XML tags.

   The top part not only contains the mapping information, which components are PUF XML tag, XMI tag, Mapping Relation, but also the commands for performing operations on the existing mappings. The commands are **Edit**, **Delete**, **Add more mappings.** How they work will be illustrated in the following text respectively.

   The **Edit** command is used to make changes on each component of the mapping information. Therefore, this command is placed right next to that specific component so that developers clearly know which component they are working on. This command lead to the EDIT pages which are created for PUF XML tags and XMI tags respectively. If editing the PUF XML tag, the corresponding EDIT page looks like the choosing-a-PUF-XML-tag part of the SELECT page. If editing the XMI tag, the corresponding EDIT page looks like the choosing-a-XMI-tag part of the SELECT page. If editing the mapping relation, the corresponding EDIT page looks like the choosing-a-Mapping-Relation part of the SELECT page. The procedures of editing these mapping information components are the same as those of the selection in Step 1. After choosing a new mapping information component, click the **Update** button to confirm the change. It will go back the DISPLAY page where the mapping information component has been modified.

   The **Delete** and **Add more mappings** commands are the operations on the entire mapping. Hence they are located at the end of each mapping. The **Delete** command is used to delete a mapping where the command is placed. If the mapping is deleted, the PUF XML tag in the

mapping will be automatically shown the unmapped PUF XML tags list so that developers can create a new mapping for it.

The **Add more mappings** command is used to create another mapping for the specific PUF XML tag in the row where this command is placed. This command also leads to the SELECT page. The difference is that the choosing-a-PUF-XML-tag part has been filled out because it is for doing mapping the specific PUF XML tag. The rest of the procedure refers to Step 1. After clicking the **Submit** button, it will go back the DISPLAY page where a new mapping for the PUF XML tag has been added into the existing mappings list.

5. **Save or print all the identified mapping information and the enhanced XMI file**. The Mapping Tool offers a useful functionality of transforming the mapping list and the enhanced XMI file to pdf format. Developers can save or print out the lists.

   After finishing generating the mapping for all the PUF elements, a complete mapping list will be shown in the DISPLAY page. This list can be transformed into pdf format that could be saved or printed by clicking the **Mapping List** link. Appendix 2, 3, 4 and 5 show the complete mapping list for the task, user, content and tool records. Another outcome of the Mapping Tool is an enhanced XMI file which is the result of applying the mappings to the target XMI file. The enhanced XMI pdf file is generated by clicking the **Enhance XMI File** link. Appendix 6 and 7 show the enhanced XMI files for the task record and the user record respectively. Figure 4.14 and 4.15 show the RESULT pages of the Mapping Tool. They are the screen shot of the Mapping list for the user record and the enhanced XMI file for the task record.

To generate definitive, verified mappings needs a validation procedure. Validation of mappings involves constructing a comparison between the mappings created by software engineers and those ones by usability engineers. The procedure requires performing the following validation steps.

1. It should have a group of software engineers do mappings. Software engineers can start thinking over what types of diagrams or what components they use frequently and what kind of usability requirements they can think of are needed in these frequently used diagrams. Then they can explore PUF requirement metadata to see if some of them can meet their needs. They also can

think about if it is possible to create mappings for each UML concept. By doing so, it is very likely that they can identify some mappings which usability engineers might miss.

2. It should have a group of usability engineers do mappings. To begin with, they can consult Figure 5.1 in the thesis, the extended high level relationships between PUF and UML components, about high level mapping. Then they can do mapping at more detailed level based on their understanding of PUF and UML requirement metadata.
3. Compare the results from software engineers and usability engineers and differentiate them. Since software engineers and usability engineers perform mapping in different ways, an amount of controversial mappings might be found. Then it requires software engineers and usability engineers work together or turn to developers with both software engineering and usability engineering backgrounds to find out a compromise mapping.