Requirements Documentation for Manufacturing Systems: Template and Management Tool

Requirements Documentation for Manufacturing Systems: Template and Management Tool

By
Mahnaz Ahmadi, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of

M.Sc.
Department of Computing and Software
McMaster University

MASTER OF SCIENCE (2005)
(Computing and Software)

McMaster University
Hamilton, Ontario

TITLE: Requirements Documentation for Manufacturing Systems:
Template and Management Tool

AUTHOR: Mahnaz Ahmadi, B.Sc. (Tehran University)

SUPERVISOR: Dr. Ridha Khedri

Co-SUPERVISOR: Dr. Nejah Tounsi

NUMBER OF PAGES: xv, 122

Abstract

The experience shows that any shortcoming in defining the requirements for computer based systems imperils the deliverables of all the subsequent stages of their development. This importance is undeniable when dealing with manufacturing software-systems due to their significant role in all spheres of human life. These systems have very stringent non-functional requirements such as accuracy and real-time constraints. The development of manufacturing software-systems is very challenging and requires special caution, since any mistake might have broad impact on very expensive work-pieces or might lead to a machining accident with irreparable effects. The success of their development depends largely on the quality of their Software Requirements Specification Document (SRSD).

We propose a new requirements template specifically designed for manufacturing systems. The template is structured to reflect their characteristics including multi-constraints and multi-disciplinary problems, multi-stage processes, multi-tasking, dynamic behavior, evolutionary nature, time-varying physical characteristics, and the usage of complex scientific models. A complementary usage of goal-driven, viewpoint oriented, and scenario-based approach is adopted for structuring the template content. To provide a high quality SRSD, the template is designed to enhance unambiguity, consistency, completeness, precision, non-redundancy, and good organization of the requirements document as well as other criteria including breadth of applicability and methodology independence.

An automated tool for requirements management according to the proposed template has been designed and implemented. The requirements management tool provides relatively secure and easy-to-use capabilities for the documentation and the retrieval of the requirements. It accelerates capturing the requirements, improves system quality by enhancing the reduction of requirements errors, and helps in establishing a common understanding between the system builders and the stakeholders. In addition to a user friendly environment for changing the information, we developed a powerful dynamic report generator that can be configured by the user and that provides a simple way for retrieving the requirements and formatting them.

Both the template and the tool have been validated using the requirements for a *Tool Trajectory Planning for High Speed Machining* system developed at the *Aerospace Manufacturing Technology Centre*, Institute for Aerospace Research (Montreal).

Acknowledgments

First, I would like to express my deep gratitude to my supervisors Dr. Ridha Khedri and Dr. Nejah Tounsi for their guidance, critical insights, encouragement and many inspiration throughout my study.

I would also like to thank Dr. Alan Wassyng and Dr. Spencer Smith for their valuable advises and comments.

Special thanks to my husband Saeed Samet for his love, support, and encouragement during the years of my graduate studies.

Last but not least, thanks to my daughter Saba because of her love, understanding, and endless patience.

Contents

Abstract iii

Acknowledgments v

List of Figures xi

List of Tables xiii

1 Introduction and Motivation 1

1.1 Requirements Documentation 1

1.1.1 SRSD Quality Attributes 4

1.1.2 SRSD Templates 8

1.1.3 Requirements Management 12

1.2 Problem Statement and Scope 13

1.3 Research Objectives 14

1.4 Methods 15

1.5 Thesis Structure 16

2 Manufacturing Systems Background 17

2.1 Manufacturing Systems General Characteristics 18

2.1.1 Numerical Analysis 19

2.1.2 Optimization Problems 24

2.2 Conclusion . . . 29

3 Literature Review 31

3.1 Goals-Driven Requirements Analysis . . . 31

3.2 Viewpoint-Oriented Requirements Methods . . . 33

3.3 Scenario-Based Requirements Techniques . . . 35

3.4 Conclusion . . . 38

4 Template Design and Evaluation 39

4.1 Sources and Description of the Proposed Template . . . 40

4.2 Evaluation of the Proposed Template against Manufacturing Systems Characteristics . . . 42

4.2.1 Non-Functional Requirements . . . 42

4.2.2 Functional Requirements . . . 45

4.3 Evaluation of the Proposed Template against the SRSD Quality Attributes and two more Criteria . . . 53

4.3.1 Organization . . . 53

4.3.2 Precision . . . 53

4.3.3 Consistency . . . 54

4.3.4 Completeness . . . 54

4.3.5 Non-Redundancy . . . 56

4.3.6 Ambiguity . . . 56

4.3.7 Traceability . . . 57

4.3.8 Modifiability . . . 57

4.3.9 Methodology Independence . . . 58

4.3.10 Breadth of Applicability . . . 58

4.4 Conclusion . . . 59

5 Design, Implementation, and Validation of the Management Tool 61

5.1 System Architecture 63

5.2 Technical Decisions 69

5.2.1 XML 70

5.2.2 JAXFront 73

5.2.3 XMLSpy 73

5.2.4 Java 74

5.2.5 Eclipse 74

5.3 Validation of SMART 74

5.4 Conclusion 76

6 Conclusion 79

6.1 Concluding Remarks 79

6.2 Main Contributions 80

6.3 Future Work 81

A The Proposed SRS Template 91

A.1 Template Structure 92

A.2 Introduction 97

A.2.1 Document Purpose 97

A.2.2 Terminology Definitions, Abbreviations and Acronyms . . . 97

A.2.3 References 97

A.2.4 Document Organization 97

A.3 General System Description 98

A.3.1 System Purpose 98

A.3.2 System Scope 98

A.3.3 System Context 98

A.3.4 Operations 98

A.3.5 User Characteristics 99

A.4 Non-Functional Requirements 99

A.4.1 System Non-Functional Requirements 99

A.4.2 Partner Applications 101

A.5 System Constraints 101

A.6 Functional Requirements 102

A.6.1 Goal 102

A.6.2 Viewpoint 102

A.6.3 Scenario 103

A.6.4 Attributes and Models 103

A.6.5 Assumptions 103

A.6.6 Common Input/Output/Constant Data Items of the Models 103

A.6.7 Theoretical Models 104

A.6.8 Instanced Models 104

A.6.9 Model Description 104

A.6.10 Sensitivity of the Model 104

A.6.11 Tolerance of the Solution 105

A.6.12 Specific Input/Output/Constant Data Items of the Model . 105

A.6.13 Body of the Scenario 105

A.6.14 System Behavior 106

A.6.15 Control Flow Diagrams 106

A.6.16 Others 106

A.7 Traceability Matrixes 106

A.8 Open Issues 107

A.9 Waiting Room 107

A.10 Expected Changes 107

B The Structure of the XML Requirements Data File (XSD) 109

List of Figures

3.1 Goals Modeling Using Viewpoints and Scenarios 37

5.1 Requirements Management Tool: Technology Perspective 63

5.2 Requirements Management Tool: System Architecture 65

List of Tables

1.1 Classification and Interdependence of the SRSD Quality Attributes 6

4.1 Main Sections of the Proposed Template 41

4.2 Content of the *Non-Functional Requirements* of the Proposed Template 44

4.3 Content of the *Functional Requirements* of the Proposed Template . 47

4.4 Content of the Sections *Attributes & Models* and *Body of the Scenario* 50

4.5 Content of the Section *Instanced Models* from the Proposed Template 51

4.6 Power Conditioning Function [LFM01] 55

Chapter 1

Introduction and Motivation

1.1 Requirements Documentation

A complete cycle of developing a software product contains several phases including software requirements activities, design, implementation, testing, and maintenance. The focus of this research is on the first phase. Before illustrating the requirements activities, we describe the main aim of requirements and their importance in software development life cycle.

Requirements are the services that the product provides, or/and qualities it must satisfy [RR99]. They could be derived directly from the user needs, stated from standards, or environmental constraints. The requirements specify what should be implemented. They will be discovered by answering the questions such as: what the product must do?, what are the product properties?, and what are the process constraints?

Any difficulty in defining the system requirements will affect the later stages of developing computer based systems. In the requirements phase, the needs of users and stakeholders should be elicited and translated into software requirements understandable by software designers and managers. However, requirements do change. The changes due to omitted requirements or bad requirements are always expensive to handle. The problems that may occur if the requirements are incomplete or wrong could lead to undesirable results. These problems include latency in product delivery, end-user dissatisfaction, unreliable product, and expensive maintenance [KS98]. These undesirable consequences could be avoided by applying the software requirements activities in a rigorous systematic way.

The requirements stage is composed of the following activities: discovering, analyzing, documenting, validating, and managing the requirements. Discovering the requirements is concerned with learning about the work. It involves understanding the user and stakeholders needs, determining the software constraints, and studying the application domains.

Among the whole requirements gathered through the elicitation stage, the requirements analyst establishes a set of complete and consistent requirements, which is a translation of the user needs into product specifications.

The next activity is preparing a written statement of the requirements satisfying the quality attributes described in Section 1.1.1. They are elicited and analyzed for future use of customers, designers, testers, and managers. This document, which is called *Software Requirements Specification Document* (SRSD), is the result of a cooperation between users and the system builder, since none of them has enough knowledge about the other side activities.

The SRSD is the result of the software requirements phase and includes a complete set of the system capabilities. The benefits of providing the SRSD are as follows [IEE98, Lai04, San03]:

1. SRSD is a formal statement that works as a baseline between the customer and the system builder to ensure that technical community understands the user needs correctly. This identifies problems and possible misunderstandings

at the early stage of the software development life cycle when corrections are relatively inexpensive. The SRSD document often serves as a basis for the work contract.

2. SRSD includes the services that must be provided by the product and the system operational constraints. This constitutes a starting point for system designers.

3. SRSD works as a criterion for verifying and testing the product to ensure that the system satisfies the requirements and the stakeholders' needs.

4. Any change must be recorded first into the SRSD and then into the system. This helps to control changes and makes a standard for the maintainer of the SRSD and the system. Accordingly, a new SRSD which satisfies the requirements for the next generation of the system is being built.

The content of the SRSD is a set of requirements that describe specifications of a particular product and its interaction with the adjacent systems. The basic information that the SRSD should address are the system functionalities, and the overall constraints on the system load and its development process. The first set of information constitutes the system's functional requirements and the second set constitutes its non-functional requirements.

1. The *functional requirements* describe the functionalities or services that the expected system must provide. In [RR99], further division is presented for functional requirements as business requirements and technical requirements. Business requirements are the actions which must be taken by the product to fulfill some part of the product services, and will be described by users or business people. Technical requirements are the requirements that are needed to carry out the business requirements later. In fact, they are "requirements

for the requirements" [RR99]. We can name the evaluation of the workpiece deflection as an example of functional requirements for the *Tool Trajectory Planning* which we have chosen as our case study. One of the methods that estimates the workpiece deflection y from the longitudinal neutral axis is shown in Equation 1.1:

$$\frac{d^2y}{dx^2} = \frac{M}{EI} \tag{1.1}$$

Further technical requirements emerged by this model such as the external moment M applied on the workpiece, the Young modulus of the workpiece E, and the moment of inertia I.

2. The *non-functional requirements* are the qualities or properties that the system must satisfy, such as maintainability, usability, precision, and reliability. We can name the acceptable tolerance δ_P for the contour error resulting from individual axial errors as an example of non-functional requirements in our case study.

1.1.1 SRSD Quality Attributes

A set of properties should be associated with the SRSD to make it a quality requirements document. These qualities are related to the information going into the document and the methods of organizing the requirements. Faulk categorized the quality attributes of a software requirements document into two semantic and packaging classes as follows [Fau95]:

1. *Semantic properties* are related to the quality of the content going into the requirements document. He considers completeness, implementation independence, unambiguity, consistency, preciseness and verifiability under this class.

2. *Packaging properties* are related to the quality of the structure of the requirements document. He considers modifiability, readability, and organization for reference and review under this class.

A complete discussion about how to achieve a quality SRSD is presented in [San03]. It was described that a quality SRSD is attained in two dimensions. The first dimension refers to the quality of the content and presentation of the SRSD, while the second dimension concerns the process of specifying and documenting the requirements. To illustrate the former, we describe the classification of quality attributes presented in [San03], and to elaborate on the later, we describe techniques for documenting the requirements including requirements documentation methods and templates.

In [San03], quality attributes are classified as primary, and secondary attributes. Primary attributes are fundamental attributes, while secondary attributes depend on the primary attributes. For instance, verifiability is a secondary attribute which depends on the following primary attributes: unambiguity, consistency, completeness, and preciseness. It means, if a SRSD is complete, consistent, precise, and unambiguous, it is verifiable. Also, two further semantic and presentation categories for attributes are considered under each of the primary and the secondary classes. These attributes work as two quality gates that check the requirements for *Semantic* and *Packaging* properties presented by Faulk [Fau95]. To clarify this classification see Table 1.1 [San03].

It is described in [San03] that in assessing the content and presentation quality of the SRSD, more emphasis should be given to the primary attributes since the secondary attributes are dependent on primary ones. The complete formal and informal definitions for quality attributes are presented in [San03]. However, a summary of definitions for primary quality attributes is presented here to provide

Table 1.1: Classification and Interdependence of the SRSD Quality Attributes

			Secondary Attributes			
			Semantic Attributes		Presentation Attributes	
			Correct	Verifiable	Traceable	Modifiable
Primary Attributes	Semantic Attributes	Unambiguous	X	X		
		Consistent	X	X		
		Complete	X	X	X	
		Precise	X	X		
		Non-Redundant	X		X	X
	Presentation Attributes	Organized	X		X	X

the reader with a better understanding of these attributes.

1. Consistent: Two types of consistency named *space* and *behavior* consistency are defined. The specification is space consistent if the shared variables have the same declarations (name and type) in different parts of the SRSD. While it is behavior consistent if different parts of the SRSD do not disagree on the actions to be carried out by the system in reacting to the same triggering events. To illustrate, when different requirements scenarios have some parts in common, no conflict should exist in the behavior of the system as described by the common parts.

2. Complete: Three types of completeness named *space*, *content*, and *semantics* are defined. The specification is space complete, if it prescribes the actions to be carried out by the system at every state in the system's space domain. It is content complete if it includes all the categories of the requirements that pertain to the product. It is semantic complete if it includes all the explicit

and implicit assumptions and constraints related to the intended system. An example in Section 4.3.4 describes how semantic incompleteness may lead to inconsistency.

3. Unambiguous: Specification is unambiguous if it is not interpreted differently by readers. There is no exact measurement for examining the ambiguity of the SRSD. However, it can be reduced by documenting the requirements in a well-formed syntax and describing the system behaviors and application domains in a precise way, for example by using mathematics.

4. Precise: Two types of precisions named *coverage* and *value* are defined. Specification has coverage precision if it has narrow range of observation in defining the statements. This kind of precision is also called behavioral precision (it is a measure of the determinism of the system's functions). While it has value precision, if it supplies the tolerance of every numerical quantity of a requirement.

5. Non-redundant: Specification is non-redundant if no requirement has been documented more than once or different requirements do not exist with the same meaning.

6. Organized: Specification is organized if the principal of separation of concern is considered in the process of documenting its requirements. This means, information with a similar concern is organized under a same section.

Definitions of secondary quality attributes are provided in [San03]. It is sufficient to know that:

1. The SRSD is *correct*, if it satisfies all the primary quality attributes.

2. The SRSD is *verifiable*, if it is unambiguous, consistent, complete, and precise.

3. The SRSD is *traceable*, if it is complete, non-redundant, and organized.

4. The SRSD is *modifiable*, if it is non-redundant, and organized.

As we discussed, in addition to the quality of the content and the presentation of the SRSD, the methods of documenting the requirements and using templates also affect the quality of the SRSD. Hence, we briefly explain the techniques of documenting the requirements and then describe the role of templates in the quality of the SRSD.

The methods of documenting the requirements can be categorized into three groups as informal, semi-formal, and formal. Informal methods use natural language to describe requirements document. Although, writing the requirements document using natural language is understandable by all potential readers, it is ambiguous because it can be interpreted differently. On the other hand, formal languages reduce ambiguity. However they require more resources to use them and lead to highly technical documents. Semi-formal specification languages in practice are more accepted for documenting the requirements. A classification about the examples of formal and semi-formal languages is presented in [San03]. According to that classification, Z [Spi92, Wor96b], B [Abr98, Wor96a], Statecharts [Har87], and Petri_Nets [Rei85] are categorized as formal methods, and Unified Modeling Languages (UML) [OMG00], and Z++ [Kev90] are in the class of semi-formal methods.

1.1.2 SRSD Templates

The last factor that affects the quality of the SRSD is the use of templates in documenting the requirements. To illustrate the impact of templates on the quality of the SRSD, we first define them, then describe the advantages of using templates in documenting the requirements, and we finally introduce the properties of what is considered as a "good" template.

A template is a predefined format for recording the requirements, and a method for presenting them. It gives a starting point for documenting the requirements. There are many advantages of using templates summarized as follows [San03,Lai04]:

1. Organization: Templates organize the format of the SRSD by providing separate sections for addressing the requirements. This enhances the readability of the document and facilitates updating the information.

2. Productivity: Templates increase the productivity of the SRSD by facilitating the team work, when each part of the SRSD is assigned to an appropriate technical team.

3. Re-usability: Templates facilitate the adaption of the SRSD for the next evolution of the product, or the other products that have some characteristics in common.

4. Adequacy: Templates work as a checklist for the writer of the requirements document. This protects the specification from omitting parts of the requirements.

We assess templates against their ability to satisfy the primary quality attributes described in Section 1.1.1. Moreover, we evaluate our template against two secondary attributes named modifiability and traceability. The emphasis on these two secondary attributes is due to their impacts on the requirements management that is discussed in Section 1.1.3. We also consider two more criteria presented in [San03] for assessing the templates: breadth of applicability, and methodology independence. Accordingly, we consider the following criteria in preparing our template [San03, Lai04]:

1. Organization: The template should be organized according to the principal of separation of concern. This means sections and subsections should be

used to organize the document in a hierarchical format, where each section encapsulates the requirements which address one concern.

2. Precision: The template should provide a precise explanation about the content of the information that should be stored in each section of the template.

3. Consistency: The template should support both space and behavior consistency explained in quality attributes.

4. Completeness: The template should give a complete list of all the categories of requirements, assumptions, and constraints to be contained in the SRSD.

5. Non-redundancy: The template should impose unique identification for the requirements. This identification could be labeling the requirements by numbers or by adopting a naming convention.

6. Ambiguity: The template should present the information in a well-formed syntax and describe the system behaviors and application domains in a precise way to reduce the ambiguity. However, the syntax should be understandable for novices with the application.

7. Traceability: The source of the requirements, and their dependencies that might be affected by future changes should be identifiable in the template. This is for ease of change management, whereas ensures that changes in some requirements do not affect others adversely.

8. Modifiability: Organizing the templates and specifying the requirements that are more likely to change, are two factors that affect modifiability of a template. The second factor provides a guideline for the system designers to see them as independent components (as it is explained in Section 1.1.3), which protects the system from frequent changes of these requirements.

9. Methodology independence: A complete discussion regarding the advantages and disadvantages of the templates to be methodology dependent or independent is held in [San03]. From that discussion, it is suggested that decision making regarding the dependency of the template on a methodology should be based on the application.

10. Breadth of applicability: The applicability of the template to wide breadth of applications is a desirable characteristic. However, this criterion should not be considered as very important; it might make the template ineffective for a particular product.

There are many off-the-shelf templates. However, there are no standard templates for all products because each SRSD is unique for a specific product. The key is to tailor the existing templates to meet special needs.

We can name the following templates as well known standards:

1. IEEE std 830-1998 [IEE98]

2. Volere Requirements Specification Template of the Atlantic System Guild [RR99]

3. ESA PSS-05-0 used by European Agency [Bss91]

4. NASA-DID-P200-SW used by NASA [NAS91]

5. MIL-STD-498 and DI-IPSC-81433 used by the US Department of Defense [Def88]

6. NRL A-7E, Documented by the Naval Research Laboratory [HKSP78]

For the recent research efforts in providing templates, we refer the reader to [San03, Che03, Mer03, Lai04, Smi06, SL05, SLK05].

1.1.3 Requirements Management

The last activity in the requirements stage that we focus on is related to the requirements management. The aim of the requirements management is to demonstrate a common understanding between system builder and stakeholders to ensure that stakeholders needs are met [OOGC06]. Changes are inevitable parts of developing systems. Accordingly, the requirements management tries to minimize the difficulties following to requirements change in any stage of the software development life-cycle. Requirements change may occur due to misunderstandings, emerging new requirements, introducing new laws or regulations, technical, schedule, or cost problems [KS98]. However, changes to some requirements are slower than others. The former are called stable requirements and is referred to as the essence of a system, while the later are called volatile requirements and is referred to as the requirements of the system in a specific environment or according to particular assumptions.

In manufacturing systems, the information related to the tool material are stable requirements, while displaying the information is a volatile requirement as new ways of presentation are required or made available. The advantage of anticipating the requirements that are likely to change is providing better insight for the system designers to consider them as independent components that have less impact on the system [KS98].

The quality attributes described in Section 1.1.1 have direct impact on the efficiency of the requirements management. Traceability, modifiability and no redundancy of the requirements are essential prerequisites for the requirements management [KS98]. Traceability specifies the source of the requirements, the existence reason of the requirements, and the requirements dependencies that might be affected by the requested changes [KS98]. Therefore, keeping the parent/child relationships between requirements makes change management possible and ensures us that changes in some requirements do not affect others adversely. However, according to [San03], traceability is a secondary attribute that depends on the primary quality attributes namely, completeness, non-redundancy, and organization of the SRSD. Thus, the requirements management is not effective, if the SRSD is not qualified with respect to these primary quality attributes.

If the requirements management is performed properly, it would reduce searching time to access the information in a precise, traceable, and consistent way. The change management would control the changes in different parts and would not cause inconsistency in the SRSD. Management tools that can vary from a word processor to object-oriented databases according to the size of the information are used to facilitate this purpose. Checking the communication between all parties and the affects of changing requirements could be easily handled through management tools. The facilities of databases help to keep relation between the requirements. Also, different reports of databases help to manage the requirements. We aim at designing and implementing a requirements management tool for manufacturing systems that will be discussed in Chapter 5.

1.2 Problem Statement and Scope

We described the important role of SRSD quality attributes assessment and templates in the software requirements documentation. In Chapter 2, we express importance of applying the software requirements activities to the manufacturing software systems based on two reasons. The first reason refers to diversity in usage of these systems in all spheres of human life and the second one is concerned with their specific characteristic.

We illustrate the influence of the requirements documentation on two fundamental subareas of the manufacturing software. In that discussion, we identify the elements required to be documented for developing scientific models and optimization problems. This systematic step by step documentation defines plateaus for system designers while simplifying the real problems. In addition, applying goal driven, viewpoint oriented methods, and scenario approaches in documenting the requirements, which is described in Chapter 3, makes possible tackling complex nature of manufacturing systems. In fact, these methodologies split the system purpose into a set of system goals, study each goal from different perspectives, and consider all possible scenarios that might occur from each point of view. However, a thorough study of the literature leads to the third reason of selecting manufacturing systems as our case study. We did not find any available template that satisfies the needs of a quality requirements documentation for these systems.

In addition, the evolutionary nature of the manufacturing systems demands an automated tool, flexible enough to manage the requirements. Although, general characteristics of the manufacturing software systems are considered in presenting our template, we limit our case study to document the requirements for the *Tool Trajectory Planning for High Speed Machining* developed at *National Research Council of Canada* (NRCC). The SRSD obtained was provided to our partners at NRCC and is not included in the thesis due to a confidentiality agreement we signed with them.

1.3 Research Objectives

We pursue the following objectives in this research:

1. Elicit the requirements for the *Tool Trajectory Planning for High Speed Machining*, which we chose as our case study.

2. Design an appropriate template suitable to the requirements of manufacturing systems.
3. Design and build a requirements management tool to deal with the evolution of the requirements of manufacturing software systems.
4. Assess the provided template against the requirements elicited in the fulfillment of objective 1.

1.4 Methods

We pursue the following methodologies to achieve the research objectives:

1. Discovering the requirements through interviewing the stakeholders and investigating the functionality and the aspired qualities of the *Tool Trajectory Planning* system.
2. Designing a template which satisfies the desired quality attributes listed as the criteria for templates. Analyzing the needs of a requirements document that deals with manufacturing systems. In addition, we employ goal-driven, viewpoint-oriented, and scenario-base techniques in documenting functional requirements. These methodologies help to capture a comprehensive work perspective while all interactions of the system with the environment would be considered.
3. Applying design techniques to elaborate the design of a management tool and using *Extensible Markup Language* (XML) as a basis for implementing the tool. This provides a rich documentation structure capable to run over the web.

4. Assessing the provided management tool against the requirements of the case study by our research partner.

1.5 Thesis Structure

Chapter 1 contains the SRSD background, the motivation, statement of the problem, the contributions of the thesis and its structure.

Chapter 2 contains the manufacturing systems background.

Chapter 3 explores the use of goal-oriented analysis, viewpoint-oriented methods, and scenario-based techniques in designing the proposed template.

Chapter 4 describes the template design and the rationale of the design.

Chapter 5 describes the design, implementation, and validation of the management tool.

Chapter 6 presents our conclusion and potential future work.

Appendix A presents the proposed requirements template.

Appendix B presents the structure of the XML requirements data file (XSD).

Appendix C contains the source code of the management tool.

Chapter 2

Manufacturing Systems Background

In Chapter 1, we discussed the importance of the requirements documentation in general. In this chapter, we elaborate on the characteristics of manufacturing systems from requirements perspective. To achieve this aim, we first mention why we select manufacturing systems as our case study. Then, we describe the characteristics of manufacturing systems and challenges that are facing their developers. Finally, we elaborate on how the requirements documentation is worthwhile and cost effective in reducing inherent difficulties to manufacturing systems.

The first reason for selecting manufacturing systems as our case study refers to the significant and comprehensive role of these systems in all spheres of human life. They address a wide variety of products from apparel to very sophisticated aircrafts, and impact all aspects of life from entertainment to health issues. This diversity in usage requires special caution, since any mistake might have broad impact or cause irreparable effects. The second reason concerns with the nature of manufacturing systems, which we study their characteristics in Section 2.1.

2.1 Manufacturing Systems General Characteristics

Manufacturing systems are complex, fast, multi-stage with time-varying physical characteristics and dynamic behavior where process monitoring and automatic controls are essential. Furthermore, their ability to react quickly to changes is important in the success of a competing market place. In spite of these factors that make the development of manufacturing systems so challenging, finding techniques for delivering high quality products in the shortest time, while keeping the cost down, is the manufacturers' goal. On the other hand, usually manufacturing software systems are dealing with scientific models of the process of manufacturing a product.

The development of scientific models, consists of building mathematical models, analyzing algorithms, solving equations, and finding optimal solutions to real problems. Considering all these aspects makes the development of manufacturing systems differ from other systems, such as business systems. Any difficulties in understanding the actual needs, modeling the real problem, finding the solution, or adjusting with requested changes result in latency in product delivery, end-user dissatisfaction, unreliable product, and expensive maintenance [KS98].

In this research, we try to illustrate how providing a quality SRSD for manufacturing systems can protect them from the above mentioned difficulties at the early stage of the software development life-cycle. To pursue this aim, in Sections 2.1.1, and 2.1.2, we elaborate on the influence of numerical analysis and optimization problems as two fundamental factors in the development of manufacturing systems and on the impacts of software requirements methodology on standardizing and conducting development of manufacturing systems in a systematic way. Then in Chapter 3, we study the complementary usage of goal-driven, viewpoint-oriented, and scenario-based approaches for documenting the requirements as an efficient mechanism enhancing the quality of requirements documentation.

2.1.1 Numerical Analysis

As mentioned above, numerical analysis techniques are widely used in manufacturing software applications. The numerical analysis is the study of methods and algorithms for the numerical solution of mathematical problems [Hea02]. In many cases, it leads to a sequence of approximations in which the analysis stage tries to determine the rate of accuracy and completeness of the answer. Nowadays, the numerical analysis and the scientific computation are used interchangeably [CEJM06].

As it is illustrated in [Lai04], the scientific computation is an inevitable part of many engineering applications. Among others, the manufacturing systems, which are the focus of this research, involve many scientific models of the process of manufacturing a product. As we mentioned earlier, the development of scientific models, consists of building mathematical models, analyzing algorithms, solving equations, and optimizing the solution. Thus, understanding the factors that affect the process of building and solving scientific models, such as collecting related physical data, measuring the errors, deriving the equations, and checking the stability of algorithms has direct impact on the accuracy of the solution.

We attempt to identify these factors at the requirements stage to achieve the goal of numerical analysis, which is finding solutions to numerical problems, that are accurate, reliable, and efficient. Before describing the elements that impact the accuracy, reliability, and efficiency of numerical systems, we define these attributes more specifically for numerical systems [QSS00]:

1. The *accuracy* means that errors are small according to a certain tolerance.

2. The *reliability* means that the difference between the actual computed solution

and the physical solution (either experimental or numerical) can be warranted to be under a given tolerance.

3. The *efficiency* means that the amount of operations, time, and the required memory size that affect the computational error is as small as possible.

Efficiency is an important software quality. In the subject of numerical computation, efficiency is achievable by using optimal methods with least operations and using algorithms that use memory efficiently. Considering the methods that support efficiency of numerical computations is out of the scope of this research and should be considered at later stages of software development. Therefore, we focus on the accuracy and reliability attributes. The accuracy and reliability of numerical computations are related to computational errors, the stability of the algorithms, and the sensitivity of the problems [Lai04]. Therefore, by enhancing the non-functional requirements with the contents that identify these elements, the accuracy and reliability of computation and consequently the corresponding scientific model mapped to the mathematical model would hold.

Different classifications are presented in [QSS00, GMW81, Lai04, Wik06b] for the errors generated during numerical computations. We summarize them as data errors, modeling errors, numerical operation errors, and program errors. Later, numerical operation errors have been divided into floating point problems, truncation errors, and discretization errors. First, we give a brief explanation about these errors and then we illustrate how anticipating these issues at the requirements stage can help protect the software program from some of these errors.

1. The *data errors* are related to accuracy of measurement of the real physical data.

2. The *modeling errors* are committed because physical problems are not for-

mulated in mathematical problems properly.

3. The *problems with floating point* are referred to rounding, overflow, underflow, absorption, and cancelation. These problems arise because of the impossibility to represent all real numbers exactly on the digital computer that are in use today.

4. The *truncation errors* are due to the replacement of iterative methods with finite steps models.

5. The *discretization errors* come up when a continuous problem is approximated with a discrete problem whose solution is known.

6. The *program errors* are referred to the implementation of the numerical algorithms.

A part of the data and numerical operation errors can be reduced by employing better computers. For example, this can be achieved by using powerful computers or precise devices for measuring the data. Also, modeling errors might be controlled by selecting proper mathematical models or building new ones. These are not the issues we could deal with at the requirements stage. The issues that concern us at this stage are: specifying the source of the measurement error and the error range for each input data [Lai04], and protecting the software from propagating an error during the computation. The later is controlled by checking the stability of algorithms and sensitivity of the problems.

The concept of numerical stability varies in different situations. The definition of stability for dynamic systems differs from that of solving the partial differential equations. However, in general, an algorithm is numerically stable, if once an error is generated due to some approximation during computation, it does not grow too much to affect the result [GMW81, PC86, Lai04, Wik06b]. Moreover, not every stable algorithm leads to an accurate solution since some problems are inherently ill-conditioned.

A problem is sensitive or ill-conditioned, if small perturbation in the input data causes a big change in the solution. A quantity named condition number is associated to a numerical problem to assess whether this problem is sensitive. The condition number is the ratio of the solution changes to the input changes. Small condition number states well conditioning of the problem and vise versa.

The stability of algorithms and the insensitivity of the problems are two fundamental requirements for accuracy of numerical computations. Stable algorithms solve a nearby problem while well conditioning guarantees that the solution for nearby problem is close to that of actual problem [Lai04, Wik06b]. At the requirements stage, emphasizing on computing condition numbers would protect the software from converging to an inaccurate solution. However, stability cannot be specified at the requirements stage. What can be done is considering solution tolerance [Lai04]. Tolerance of the solution determines the level of accuracy that is expected for the solution.

In [Smi06], the author writes "validating the requirements is difficult because there are an infinite number of potential input value for the output variable is unknown a prior. In fact, the purpose of many scientific computing tools is to solve problems that are difficult or impossible to solve without the software, so in many cases the true solution is unknown." Therefore assessing whether the computed answer is correct within a tolerance is not always possible. An example in [Smi06] shows that the requirement is not always validatable even if the functional requirement is defined unambiguously. Accordingly, the SRSD template should help validating the numerical outputs by specifying the requirements that lead to a correct result [SL05]. Where it is possible to evaluate the accuracy of the solution according to a certain tolerance, first we define how the error could be measured, and then describe the backward and forward error analysis to assess the correctness of the answer. The definitions of this part are captured from [Ham06, Wik06b].

Let x be the exact answer and $\hat{x}$ be the computed value. The error in $\hat{x}$ can be measured by the absolute error $|x - \hat{x}|$. Since the size of the absolute error depends on the size of x, the notion of relative error is used instead. It is defined as $\frac{|x-\hat{x}|}{|x|}$ for $x <> 0$. let us consider a program being solved by the numerical algorithm as a function f that takes an input x and computes y such that $y = f(x)$. Let $\hat{y}$ be the computed result, then $\hat{y}$ is the exact result for the input $x + \Delta x$. Two kinds of errors might be described as follows, although the relative form of them are more natural:

1. The *forward error* is determined by $|y - \hat{y}|$. It shows how close the computed solution is to the exact solution.

2. The *backward error* is determined by the smallest $|\Delta x|$ such that:

   $\hat{y} = f(x + \Delta x)$. It indicates how well the computed solution satisfies the problem to be solved. In other words, it specifies the problem that is actually solved by the algorithm.

In backward error analysis, a bound on $\frac{|\Delta x|}{|x|}$ is considered as a criteria for evaluating the solution accuracy. However, if the exact solution is available, then $\frac{|y-\hat{y}|}{y}$ for $y <> 0$ is called forward error analysis and is used to evaluate the accuracy of the solution.

2.1.1.1 Numerical Requirements

As it is illustrated in [Lai04], most of the scientific computing problems are solved in five steps procedures including:

1. Defining the problem

2. Building a mathematical model

3. Specifying a computational method

4. Implementing a solution

5. Validating the solution

Only the first two steps are in the scope of the requirements stage. Accordingly, the issues to be considered at this stage for development of scientific models, are: posing the real problem, identifying the variables, assumptions and constraints that impact the problem, and building the mathematical representation of the real problem. In addition, according to the discussion held in Section 2.1.1, the factors needed to be documented to enhance the accuracy of the numerical solution and protect the numerical problem from propagating an error during their computation, are:

1. Specifying the source of the measurement error and the error range for each input data.

2. Computing the condition number when possible to assess the sensitivity of a problem.

3. Evaluating the accuracy of computation according to one of the forward or backward error analyses.

2.1.2 Optimization Problems

Optimization is the discipline which is concerned with finding the best of all possible solutions. For instance, this might be interpreted as maximizing efficiency, minimizing cost, or maximizing profit. Solving optimization problems means finding the maximum or minimum of the objective function depending on a set of variables and possibly some constraints. Objective function, variables, and constraints are components of optimization problems. The solution for the problem is a set of values which satisfy the constraints and make the objective function optimal [GMW81]. This purpose is so appealing in many fields of human life and it has grown in recent times as industrial applications become more complicated.

As an example of industrial systems that involve optimization, we can name optimizing the machining process using Computerized Numerical Control (CNC) machines which is used as a case study in this research. The general goal in this field is machining in the shortest time with the highest quality and at the minimum cost, while fully utilizing the capabilities of the provided machining power [CJ98]. Reaching this goal is very challenging mainly when dealing with high speed machining, high spindle speed exceeding 8000 Revolutions Per Minute (RPM) and reaching 60000 or even 100000 RPMs for ultra-modern machines, and high feed rates exceeding 50 m/min. On the other hand, the dynamic behavior of the system as well as the time-varying physical characteristics and their interactions make the situation more complex. For example, to minimize the polishing time, small path-intervals are required. However, this causes an increase in the cutting time, and consequently decreases the efficiency [CJ98]. Because of the complexity of the considered applications, decision making is no longer possible or even economically feasible to be made without using optimization models [NS96].

Optimization models arise to express the problems in mathematical terms and solve them in optimal way. Formalizing an optimization problem includes selecting optimization variables, identifying the constraints, and choosing the objective function [Bha00]. Specifying the objective function is challenging when there is more than one objective that should be minimized or maximized, specially if they are conflicting [Bha00]. As indicated in the above example: minimizing the polishing time and minimizing the cutting time. There are two possible ways in these situations. Considering the most important goal as the objective and leaving others as constraints or defining a composite objective function by assigning weights to each objective function [Bha00]. Explanation about the components of optimization problems is given in [Wik06a] as follows:

1. The *variables* are values that are controllable.
2. The *constraints* are conditions that must be satisfied to make the solution acceptable. They are the functions of one or more optimization variables [Bha00].
3. The *objective* is the value that has to be maximized or minimized. It is the function of one or more variables [Bha00].
4. The *models* are mathematical connections that relate the constraints, variables and the objective.

One general form of an optimization problem is called the Non-linear Complementarity Problem (NCP) and might be shown as [GMW81]:

$$\begin{cases} \underset{x\in\mathbb{R}^n}{min} \quad F(x) \\ subject\ to \quad c_i(x) \ = \ 0, i = 1, 2, ..., m' \\ \qquad\qquad\quad c_i(x) >= 0, i = m' + 1, ..., m \end{cases} \tag{2.1}$$

Where the objective function F and the constraint functions c_i are lumped together and referred to as an optimization problem.

Although most of the optimization problems can be expressed in the standard form or as a sequence of standard forms, it is very important that the formulation has the characteristics that enhance the efficiency of solving the optimization. Clearly, applying the methods for solving the standard form to a problem with special features tends to be complex with significant numerical effort [Bha00]. Therefore, optimization problems are categorized according to the mathematical characteristic of their objective and constraint functions [GMW81]. We briefly give the present classification for optimization problems and then we describe the issues that concern us at the requirements stage.

According to modern numerical analysis, the risk of ending to a serious error or numerical instability is very high even for a simple computation if someone wants to develop his/her own method from the scratch [GMW81]. On the other hand, for each of the problem categories, enormous optimization methods are available. Thus, selecting methods from the software library or modifying techniques to fit for a specific problem is the best way to solve the optimization problems in an efficient way. However, this classification becomes more specific by taking advantages of special properties for solving complex problems.

A typical classification of problem functions could be given as linear programming (LP), second order programming (SOP), semi-definite programming (SDP), quadratic programming (QAP), and non-linear programming (NLP) problems. Although second order, semi-definite, and quadratic programming are also non-linear problems, but they are separately classified due to the special algorithms invented for the nature of these problems. There are further distinction characteristics between problems under each of the above classes according to be convex/non-convex, smooth/non-smooth, and constrained/unconstrained [BtN01, GMW81, NS96].

We clarify this part by bringing in an example of using optimization discipline in manufacturing systems. Determining the maximum contour error is one of the machining optimization problems in our case study. We borrow this example from [TE02]. In this problem, the objective is to keep the contour error lower or equal to the given tolerance. To satisfy this objective, the maximum contour error should be computed and checked if it is lower or equal than the tolerance. Variables for solving this problem are: desired and actual tool paths, and desired and actual tool orientations. Contour error refers to the distance between the desired path $C(u)$ and the actual path. The error δ is evaluated by the maximum distance between an arbitrary point M on the desired path and point k on the chord that relates two positions $P(k)$ and $P(k+1)$ on the actual path. This problem is formulated as:

$$\begin{cases} \delta_c^2(k) = \underset{u\in[u(k),1]}{max} \delta^2(u) \\ where\ \delta^2(u) = (M - P(k))^2 - ((M - P(k)).\tau(k))^2 \\ subject\ to\ : \begin{cases} 0 < (M - P(k)).\tau(k) \\ (M - P(k)).\tau(k) \leq |p(k+1) - p(k)| \end{cases} \end{cases} \tag{2.2}$$

In the above formulation, u is the path parameter which varies from zero at the starting position and one at the end position, and $\tau(k)$ is:

$$\tau(k) = \frac{d(C(u)}{du}/\|\frac{d(C(u)}{du}\|_{u=u(k)} \tag{2.3}$$

This problem is a non-linear constrained optimization problem. As it is described in [TE02], using one of the standard methods may cause numerical instability, or convergence to an undesirable solution. Therefore, it has been formulated into two subproblems as they are shown in equations 2.4 and 2.5:

$$\begin{cases} u_{max} = \underset{u\in[u(k):1]}{min} (|P(k+1) - P(k)| - ((M - P(k)).\tau(k))) \\ subject\ to:\ (|P(k+1) - P(k)| - ((M - P(k)).\tau(k))) \geq 0 \end{cases} \tag{2.4}$$

The objective of this problem is finding the closest path parameter to the current path parameter while it meets the end point $P(k+1)$ on the chord closely. This problem is solved with a non-linear constraint optimization procedure. Once the u_{max} is determined by equation 2.4, the maximum chordal deviation is computed by:

$$\begin{cases} \delta_c^2(k) = \max\limits_{u \in [u(k), u_{max}]} \delta^2(u) \\ where\ \delta^2(u) = (M - P(k))^2 - ((M - P(k)).\tau(k))^2 \end{cases} \tag{2.5}$$

2.1.2.1 Optimization Requirements

According to the above discussion, the issues which can be dealt with at the requirements stage for developing optimization problems are: posing the problem, identifying the variables, constraints, and assumptions that are involved in the problem, transferring the desired qualities into a mathematical representation, and analyzing the accuracy of the solution as it is explained in Section 2.1.1. In fact, we see the system as a black box. We refer to the objective, not the way to attain it. In this way, documenting the optimization components standardize the development of optimization problems by a systematic step-by-step process. Furthermore, as we discussed, classification of optimization problems and the available methods in each class change often while they are part of the solution. Therefore, they are not required to be a part of documentation. However, it is possible to consider them as volatile requirements to make documentation more efficient.

2.2 Conclusion

In this chapter, we justified our reasons for selecting manufacturing systems as our case study. Their diverse usage in all spheres of human life and their distinguishable characteristics motivate us for this selection. We elaborated on characteristics of manufacturing systems. According to this study, manufacturing systems involve multi-constraints and multi-disciplinary problems, multi-stage processes, and multi-tasking. They deal with scientific models and time-varying physical characteristics. They have evolutionary nature and dynamic behavior. We focused on the influence of numerical analysis and optimization problems as two fundamental knowledge fields that affect the development of manufacturing systems. Finally, we determined the issues to be considered at the requirements stage for the development of these systems.

Chapter 3

Literature Review

In this chapter, we review three techniques as a complementary approach to support requirements activities: goal-driven, viewpoint-oriented, and scenario-based. We illustrate how this complementary approach provides a framework for documenting the requirements and enhancing the completeness of the requirements. We study its influence on improving the quality of the requirements documentation by providing a proper structure for requirements elicitation. In Section 4.2.2, we elaborate on how this approach is suitable for structuring the requirements documents of manufacturing software systems.

3.1 Goals-Driven Requirements Analysis

Goals are the core of the requirements analysis. The importance of using them is given in [Lam01]. According to that discussion, goals are the objectives that the intended systems should meet. These objectives can be business, organization, or system oriented. They are originated by some stakeholders expectation. Goal driven methods identify objectives by asking: why a certain functionality is required and how it can be implemented by introducing different alternatives [Reg06]. At this stage, objectives are not formulated and accordingly non-operational. When a system is analyzed, high level goals are identified to address the systems needs. Then, they are refined to subgoals. The target of the former is strategic concerns of the intended software system and the focus of the latter is technical concerns to concretize the super goals [Lam01]. This refinement process provides a pervasive structure for the requirements document by asking: how these goals should be achieved?

The essential reasons of using goals in requirements activities are summarized as follows [Lam01, YM98, Reg06]:

1. Recognizing the requirements: Goals are the sources of the requirements recognition in a systematic elaboration process.

2. Preventing irrelevant requirements: A requirement is relevant if it is used for satisfying at least one goal. Goals provide a precise criterion for documenting relevant requirements by relating them to the system context.

3. Obtaining requirements completeness: Goals are sufficient for requirements document completeness. The system purpose will be split into a set of system goals. The completeness of the specification will be assessed against satisfying the system goals.

4. Supporting change management: In goal driven mechanism, it is clear which part of the requirements document should be manipulated for particular change.

5. Providing the requirements hierarchy: Goals provide a refinement tree between stakeholders needs and technical requirements. This supports requirements traceability from high level abstraction to low level details, while making the explanation of the requirements to the stakeholders much easier.

6. Increasing readability: Goals organize complicated requirements documents. This enhances the readability of the document.

7. Detecting conflicts: Goals are the sources for discovering the conflicts among the provided requirements. Although, conflicts may lead to elicit further information, they should be handled at early stages of requirements documentation. Resolving the conflicts at the requirements level enhances the quality of the software and reduces its development cost.

8. Connecting requirements and design phases: Goals are used to connect requirements phase to design phase by identifying desired objectives and proposing alternative ways for implementing them.

3.2 Viewpoint-Oriented Requirements Methods

The notion of viewpoint is defined differently in software requirements field, such as: interest aspects of the system, information processing entities, and service recipients [NKF94]. A further distinction between the definition of perspectives and viewpoints are presented in [Eas91]. According to the scope of this research, we accept the general notion of viewpoints seen as sources or sinks of data that address only those concerns related to their originator and ignore the others [NKF94]. We use the notion of viewpoint and perspective interchangeably to express explicit guidance for gathering the requirements.

Viewpoints are vehicles that facilitate document partitioning to support the principal of separation of concerns. Each viewpoint partially represents a specific aspect of the system. System stakeholders, partner applications, system interfaces, and other issues that affect the system functionalities such as mechanical aspects, physical aspects, computational aspects, environmental aspects, and electrical aspects are potential system viewpoints. According to the discussion held in [AC03], viewpoint-oriented methods are so crucial for documenting the requirements for large scale systems that consist of several stakeholders that may have different responsibilities and concerns. The success of these systems depend on considering all participants perspectives. Viewpoint-oriented approaches support this importance by encapsulating the information around each originator and addressing their concerns separately. This mechanism is adequate to provide suitable requirements structure in a systematic way.

Variety of reasons are proposed in the literature for using viewpoints in requirements activities [SSV98,Eas91]. We summarize them as follows:

1. Organizing the requirements: Documenting the requirements according to viewpoints, from which they are originated, enhances the organization of requirements documents.

2. Identifying the requirements: In any requirements elicitation process, requirements implicitly gathered from different views. The viewpoint-oriented method makes it explicit.

3. Detecting conflicts: Conflicts are interferences of the activities of one viewpoint with another. It is important to recognize them in early stage of software development process. There are different methods for resolving the conflicts which is out of the scope of this research. We refer the reader to [Eas91] for more information on the role of viewpoints in detecting inconsistency in requirements.

4. Providing a basis for validity tests: The specifications represent the perspective of various needs. The validation process tests if all views are addressed adequately.

5. Specification adequacy: Studying the software system from all the aspects, reduce the chance of missing critical information.

6. Requirements traceability: Identifying the viewpoints provides a traceability mechanism by linking the requirements to their sources.

Variety of information elements are introduced in the literature to associate with viewpoints [SSV98]. We itemize them as follows:

1. Name: A meaningful describer to present the viewpoint.

2. Focus: A definition for the perspective taken by the viewpoint. Three types of focuses are categorized as: perspective of the stakeholders that have direct interact with the system, perspective of the stakeholders that influence the system indirectly, and domain perspectives which encapsulate domain information.

3. Source: An originator for the requirements associated with the viewpoint.

4. History: A description about the changes made in the viewpoint.

5. Requirements: A set of the requirements from the intended viewpoint.

In the template that we propose in Chapter 4, in addition to the above elements, we associate some more elements such as *identifier*, *referenceTo*, *usedIn*, and *lastUpdate* to each requirement. A thorough discussion on the influence of the associated information follows in Section 4.2.2.

3.3 Scenario-Based Requirements Techniques

According to the literature [LY01,Sut98,Coc95,RSA98] scenarios are possible ways of using a system to fulfil desired functions. They include a sequence of interactions that might happen under certain condition to satisfy a particular result. They are introduced as system activities, system interactions, roles, real world events, or/and imaginary stories of the system-to-be. In summary, a scenario contains a single unit of meaningful work that portrays the system processes by story lines of events. Each scenario may be composed of several actions described sequentially from start to end. This is sufficient to express the behavior of complex systems. Accordingly, they systematize requirement elicitation and provide a criteria for requirements validating [HD98]. There are formal definitions to scenarios [DFKM98], but since we are not dealing with formal specifications, there is no need to introduce them.

As stated in Section 3.1, goal modeling is an effective approach in requirements documentation. However, it has some difficulties in practice. These difficulties are referred to as [RSA98]: problem in discovering goals, and fuzzy nature of goals to domain experts. In order to overcome these difficulties, it is suggested to couple goals and scenarios together [RSA98,LY01]. This complementary approach provide a two way communication between goals and scenarios. The goals help to discover the scenarios in a top-down direction and the scenarios help to identify the goals in a bottom-up approach. The former is the result of decomposing a set of primitive goals and promotes goal operationalization. The later is the result of analyzing the efficiency of the scenarios and promotes goal discovery. Further, the goals are used to structure the use cases [Coc95].

A use case is a collection of scenarios triggered from an originator to achieve a particular result corresponding to the intended goal. By this view, every action in a scenario is connected to a goal according to a particular originator. We advocate this view in Section 4.2.2 and structure functional requirements of the proposed template based on this hierarchical format. It is explained in [AMP94] that none of the top-down and bottom-up approaches solely could offer a complete view of functionalities of a software system. In this view, a scenario is a container for the

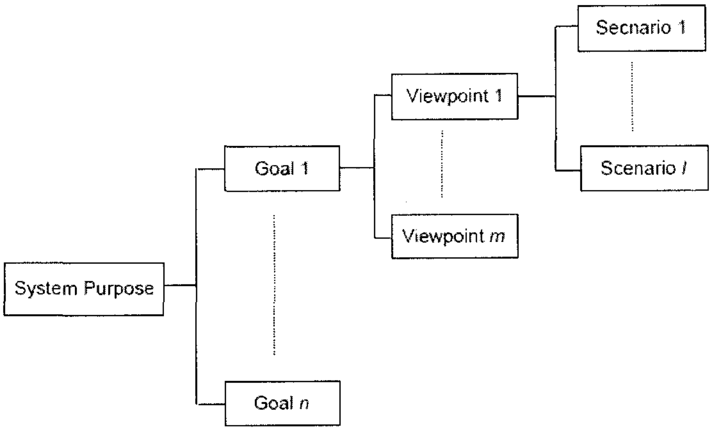


Figure 3.1: Goals Modeling Using Viewpoints and Scenarios

information required to satisfy a goal from a particular viewpoint and expresses how the intended goal can be implemented. The interactions expressed by scenarios are understandable by domain experts. Accordingly, use of goal-viewpoint-scenario complementary approach remove problems caused by goal-oriented analysis when used in isolation. The hierarchy of this approach is presented in Figure 3.1.

According to the above discussion, the purpose and usage of scenarios are considerable in all phases of requirements activities. We itemize them as follows:

1. They are used to elicit and validate system requirements.
2. They become a standard approach for test cases generation.
3. They concretize abstract description of system goals.
4. They strengthen the connection between requirements and the design phase.
5. They are essential to provide an understanding of operational concepts.
6. They help in discovering new goals.

3.4 Conclusion

In this chapter, we studied a complementary usage of goal-driven, viewpoint-oriented, and scenario-based techniques to support requirements activities. We illustrated their advantages on all aspects of requirements activities. We particulary depict the positive influence of this complementary usage for documenting the requirements of manufacturing systems in Section 4.2.2.

Chapter 4

Template Design and Evaluation

In Section 1.1.2, we explained how templates positively affect the quality of the SRSD. We also studied the advantages of using templates related to the following issues: organizing the format of SRSD, increasing the productivity of SRSD by assigning each part of SRSD to an appropriate technical team, facilitating the adaption of SRSD, and protecting the specification from omitting parts of the requirements. In addition, we defined the criteria of a "good" template and mentioned the effectiveness of product-oriented templates. We illustrated the importance of applying the software requirements activities to manufacturing software systems and our reasons for selecting this case study. According to that discussion, a template that satisfies all the needs of a quality requirements documentation for manufacturing systems does not exist in the open literature.

In this chapter, we propose a template and give the rationale for its design. We present the techniques that assist us in designing our template. This requirements template aims to be suitable for the domain of manufacturing systems while satisfying the criteria of a "good" template discussed in Chapter 1. To achieve this aim, we first present the main sections of the proposed template and specify the portions of the requirement template which are borrowed from existing templates and those which are specific to ours. Second, we discuss how the proposed template conforms with the special needs of the requirements documentation for manufacturing systems. Third, we illustrate how the proposed template satisfies the quality attributes described in Section 1.1.1.

4.1 Sources and Description of the Proposed Template

To propose the intended requirement template, we studied the following templates:

1. IEEE std 830-1998 [IEE98]
2. Volere Requirements Specification Template of the Atlantic System Guild [RR99]
3. ESA PSS-05-0 used by European Agency [Bss91]
4. NASA-DID-P200-SW used by NASA [NAS91]
5. MIL-STD-498 and DI-IPSC-81433 used by the US Department of Defense [Def88]
6. NRL A-7E, Documented by the Naval Research Laboratory [HKSP78]

In addition, we considered the recent research efforts found in [San03, Che03, Mer03, Lai04]. None of these templates can solely satisfy the needs of quality requirements documentation for manufacturing systems. Each of them addresses some aspects of manufacturing systems characteristics. For example, functional requirements in [San03] is organized in a hierarchy format based on the business events, viewpoints, and scenarios, which is similar to that we advocate in Chapter 3. However, this template does not address any of the challenges that the scientific

Table 4.1: Main Sections of the Proposed Template

1 Introduction
2 General System Description
3 Non-Functional Requirements
4 System Constraints
5 Functional Requirements
6 Traceability Matrices
7 Open Issues
8 Waiting Room
9 Expected Possible Changes

modeling of manufacturing a product presents. This weak point is common between all the studied templates except the one presented in [Lai04]. Scientific modeling is considered in this template, but this template addresses simple scientific systems. It omits to study the system from different perspectives and only considers user's point of view. Also, no classifications are considered for alternative instanced models associated to each possible scenario. For further information regarding this template see [SL05, Lai04]. In addition, manufacturing systems involve many partner applications. Taking into consideration the non-functional requirements of partner applications is so demanding. While, these requirements are disregarded by the studied templates. To sum up, we start from the existing templates, borrow some portions from them and present new structure as needed. The proposed template is composed of nine main sections which are presented in Table 4.1.

The content of sections “Introduction”, “General System Description”, “System Constraint”, “Open Issues”, “Waiting Room” and “Expected Changes” are common to off-the-shelf templates and therefore we borrowed them. However, to propose a template suitable for manufacturing systems, we present a new structure for two main sections of the requirements template named functional and non-functional requirements. In Sections 4.2.2, and 4.2.1, we provide more details on these new structures. Also, we consider some associated information to the requirements to enhance the quality of the documentation as illustrated in Sections 4.2.1. The proposed template as well as the additional information about the content of each of its elements, are given in Appendix A. In the rest of this chapter, we focus on discussing the parts of the template that are new and which are specific for documenting the requirements for manufacturing systems. We did not investigate their satisfiability for other systems.

4.2 Evaluation of the Proposed Template against Manufacturing Systems Characteristics

We propose a new structure for documenting the functional and non-functional requirements in the proposed template. We structure these two fundamental parts of the requirements document in a way to be suitable for documenting requirements of manufacturing systems. The structure of functional and non-functional requirements in the proposed template are presented in Sections 4.2.1 and 4.2.2.

4.2.1 Non-Functional Requirements

We considered two classes under non-functional requirements: system and partner applications. The former specifies the intended software system quality attributes.

The latter identifies the quality attributes for each partner application of the intended software system. Manufacturing systems involve many partner applications including software, hardware, and/or piece of technology which impact the functionalities of these systems. To support the principal of separation of concerns, we document the attributes of both classes independently and consider their influence on the functionality of the intended software system as potential system viewpoints in Section 4.2.2. This structuring enhances the ease of modifiability of the requirements document since all the attributes related to one partner are documented under one subsection.

Subclasses of both "System" and "Partner Applications" mainly come from the Volere requirements specification template [RR99]. In addition, since the software manufacturing systems might involve simulating the process of manufacturing a product, we added one subsection to the list of preserved classes for documenting the system non-functional requirements, named simulation requirements. This template subsection is intended for documenting related simulation requirements. The hierarchy of non-functional requirements is presented in Table 4.2.

The attributes of partner applications should be studied from the perspective of the intended software system. To illustrate this point, consider a "Machine" partner in the SRSD of our case study. Using safety glass by the operator is one of the security requirements for "machine" system, but it should not be documented as a security requirement under the "machine" partner. Since from the point of view of the *Optimizing Tool Trajectory Planning* software system, this is not a case. The example of security requirements that should be documented under the "machine" partner is "the maximum feed rate for having safe machining."

In addition, as it is stated in Chapter 2, the manufacturing software systems are intended to deal with scientific models of the process of manufacturing a product. According to that discussion, the development of scientific models lead to

Table 4.2: Content of the *Non-Functional Requirements* of the Proposed Template

- 1 Non-Functional Requirements
  - 1.1 System
    - 1.1.1 Accuracy Requirements
    - 1.1.2 Performance Requirements
    - 1.1.3 Security Requirements
    - 1.1.4 Maintainability Requirements
    - 1.1.5 Look and Feel Requirements
    - 1.1.6 Usability Requirements
    - 1.1.7 Portability Requirements
    - 1.1.8 Simulation Requirements
    - 1.1.9 Others
  - 1.2 Partner Applications
    - 1.2.1 Partner 1
      - 1.2.1.1 Accuracy Requirements
      - 1.2.1.2 Performance Requirements
      - 1.2.1.3 Security Requirements
      - 1.2.1.4 Maintainability Requirements
      - 1.2.1.5 Look and Feel Requirements
      - 1.2.1.6 Usability Requirements
      - 1.2.1.7 Portability Requirements
      - 1.2.1.8 Others
    - ...
    - 1.2.n Partner n

the numerical analysis and scientific computation concerns. It is summarized in Section 2.1.1 that specifying the accuracy of each numerical input data item of a model, studying the sensitivity of the problem due to a perturbation in the input data items, and evaluating the accuracy of the computation enhance the correctness of the numerical solution and protect the numerical problem from propagating an error during its computation. Accordingly in the proposed template, we preserve separate subsections for accuracy, sensitivity of the model, and tolerance of the solution. Although, these requirements present some attributes of the intended software system, we document them under functional requirements part where the scenario that describe the intended scientific model is documented. This enhances modifiability of the system, by documenting all requirements related to the mathematical representation of a problem in a single section. The template proposed in [Lai04], document these requirements under system non-functional requirements. However, manufacturing systems are complex and their development involves plenty of mathematical models, it is required to document the contents referring to the accuracy of the mathematical representation of the problems for each model. These classes are presented in Section 4.2.2 where the structures of the functional requirements are presented.

4.2.2 Functional Requirements

We organize this section to accommodate documenting manufacturing software systems. To illustrate how the proposed structure supports this purpose, we first express manufacturing systems characteristics that were taken into consideration in discussing the proposed template. Then, we present the techniques applied to support those characteristics. As stated in Chapter 2, manufacturing systems are multi-constraints and multi-disciplinary problems, involve multi-stage processes and multi-tasking. They deal with scientific models and time-varying physical characteristics. They have evolutionary nature and dynamic behavior. In the rest of this section, we describe the techniques that we used in the proposed template to support manufacturing characteristics. For some other manufacturing characteristics such as rapid response time, the template cannot directly contribute to their enhancement. However, we believe applying the software requirements activities in a systematic way could protect manufacturing systems from any insufficiency.

We advocate complementary usage of goal-driven analysis, viewpoint-oriented methods, and scenario-based techniques described in Chapter 3 as an adequate approach to tackle most difficulties raised in the requirements process of manufacturing systems. We believe this complementary approach provides a frame work for requirements activities that facilitate partitioning the vague concerns into a set of concerns with simpler structure. We follow this mechanism for structuring documentation of manufacturing systems in the proposed template.

Complex operational objectives with multi-disciplinary nature are common concerns of manufacturing systems. Hence, in the proposed template, super goals are identified and documented based on the system purpose and stakeholders needs. Then, they are refined to provide a set of primitive subgoals. This mechanism simplifies handling the complex structure of manufacturing systems where each subgoal focuses on some services that the intended software system is supposed to provide. Also, it enables the process of requirements documentation to be more clear and precise. In addition, it increases the productivity by facilitating team work, when each system goal is assigned to an appropriate technical team. However, as illustrated in Section 3.2, the success of large scale systems is not guaranteed if, the perspectives of all participants are not considered. Therefore, we study each goal from the point of view of different stakeholders that may have some concerns with regard to the system. Then, we document all scenarios triggered from one viewpoint

Table 4.3: Content of the *Functional Requirements* of the Proposed Template

1 Functional Requirements

1.1 Goal 1

1.1.1 Viewpoint 1

1.1.1.1 Scenario 1

1.1.1.1.1 Attributes and Models

1.1.1.1.2 Body of the Scenario

...

1.1.1.n Scenario n

...

1.1.m Viewpoint m

...

1.l Goal l

...

to achieve the objective of the intended goal.

Some approaches in the literature consider a fix set of viewpoints. We proposed a flexible approach where the users are allowed to define appropriate viewpoints to their application. The set of information elements which should be associated to each viewpoint is presented in Section 3.2. However, we elaborate the reasons of considering them in Section 4.3 to illustrate how these sections enhance the quality of the requirements documentation. The hierarchy of the proposed template for functional requirements is presented in Table 4.3.

To tackle the multi-stage characteristic of manufacturing systems, we follow in our template a systematic step by step documentation approach. We document needed requirements for accomplishing the task that the intended scenario is supposed to handle in a systematic way. As we illustrate in Section 3.3, scenarios are containers for the information required to satisfy goals and express how goals can be achieved. In the proposed template, presented in Table 4.3, we document these information in two main subsections labeled: *i*) *Attributes and Models* and *ii*) *Body of the Scenario.* The section *Attributes and Models* provides a clear picture of the information required for designing the task that a scenario is supposed to do. The section *Body of the Scenario* gives a detailed explanation about the intended scenario and the flow of information between its processes.

Both goal-driven analysis approach and systematic step by step documentation support multi-tasking characteristic of manufacturing systems. Tasks could be split into the set of goals or scenarios objectives, according to their nature. Then, each goal and their required information could be documented clearly in sections as presented in Table 4.4.

Dynamic behavior and time-varying physical characteristics are two other distinguished characteristics of manufacturing systems. We believe that scenario-based techniques support these characteristics. In this technique, each scenario may explain one of the system behaviors in each particular time.

As stated in Chapter 2, manufacturing software systems involve scientific modeling of the process of manufacturing a product. To support this characteristic, we focused on two fundamental factors in the development of manufacturing systems: numerical analysis and optimization problems. The information elements required to be documented in supporting these two fundamental aspects of manufacturing systems are described in Sections 2.1.1.1 and 2.1.2.1. According to that explanation, explicit sections are preserved in the proposed template for documenting the information elements related to posing the problem, identifying the variables, the constants, and the assumptions related to the problem, specifying error range for each input data, transferring the desired qualities into a mathematical representation, computing condition number, and analyzing the accuracy of the solution. This structure is presented in Table 4.4. The following are the explanations for each of its subsections:

1. The *Assumptions* are the factors that influence the requirements stated in the scenario. They are not system constraints but the assumptions taken for granted to solve the engineering problem(s).

2. The *Common Input/Output/Constant Data Items to the Models* are required data items, to and from the intended scenario. Real problems might be modeled differently. This subsection documents input/output/constant data items regardless of the model considered for modeling the real problem.

3. The *Theoretical Models* describe the set of relevant mathematical equations, axioms, or physical laws used in achieving the intended scenario goal statement.

4. The *Instanced Models* are the mathematical representation of a real problem.

Engineering problems might be modeled differently. Hence, we assign separate subsections for documenting the requirements of each alternative instanced model. For example, evaluating the workpiece deflection for *Tool Trajectory Planning*, which we have chosen as our case study, might be modeled in several ways. In equation 1.1, we propose one of those models. Also, an example is given for beam deflection problem in [SL05]. The proposed structure for the instanced models is presented in Table 4.5.

The other main subsection in the scenario structure is named *Body of the Scenario*. It is for giving a detailed explanation about the intended scenario and

Table 4.4: Content of the Sections *Attributes & Models* and *Body of the Scenario*

- 1 Attributes and Models
  - 1.1 Assumptions
  - 1.2 Common Input/Output/Constant Data Items to the Models
    - 1.2.1 Data Item Code
    - 1.2.2 Data Item Description
    - 1.2.3 Characteristics of Numerical Data item
      - 1.2.3.1 Data Type
      - 1.2.3.2 Unit
      - 1.2.3.3 Format
      - 1.2.3.4 Definition Interval
      - 1.2.3.5 Accuracy
    - 1.2.4 Mnemonic Names for the Possible Values of each Non-Numerical Data Item
  - 1.3 Theoretical Models
  - 1.4 Instanced Models
- 2 Body of the Scenario
  - 2.1 System Behavior
  - 2.2 Control Flow Diagram
  - 2.3 Others

Table 4.5: Content of the Section *Instanced Models* from the Proposed Template

1 Instanced models

- 1.1 Alternative Instanced Model 1
  - 1.1.1 Model Description
  - 1.1.2 Sensitivity of the Model
  - 1.1.3 Tolerance of the Solution
  - 1.1.4 Specific Input/Output/Constant Data Items to the Model
    - 1.1.4.1 Data Item Code
    - 1.1.4.2 Data Item Description
    - 1.1.4.3 Characteristics of Numerical Data Item
      - 1.1.4.3.1 Data Type
      - 1.1.4.3.2 Unit
      - 1.1.4.3.3 Format
      - 1.1.4.3.4 Definition Interval
      - 1.1.4.3.5 Accuracy
    - 1.1.4.4 Mnemonic Names for the Possible Values of each Non-Numerical Data Item
- 1.2 Alternative Instanced Model 2
- . . .
- 1.n Alternative Instanced Model n

the flow of information between its processes. This part is particulary required to be part of the manufacturing systems documentation. Scientific modeling of the process of manufacturing a product is complex. Hence, precise explanation of the behavior of system is suitable to provide a clear picture to the reader of the document. The following are the explanation for each of its subsections:

1. The *System Behavior* describes the dynamic functionality of the intended scenario based on the technical requirements identified in the template section titled *Attributes and Models*. This description includes recording the input data items and applying the models to generate the outputs. This technical refinement should lead to the scenario's goal and assumptions as well as the system constraints.

2. The *Control Flow Diagrams* visualizes system behavior by diagrams. It is an optional information that makes the system behavior more clear to the reader of the document. These diagrams should show major inputs/outputs to and from the scenario and the functions performed by the system in response to an input or to generate an output. These diagrams show information flow in the intended scenario.

3. The *Others* includes any requirements which cannot fit in any of the previous scenario subsections.

As stated in Section 1.1.3, changes are inevitable parts of developing systems. We advocate that templates facilitate the adaption of SRSD for evolution of the requirements. Accordingly, using templates potentially support the evolutionary nature of manufacturing systems. In addition, we believe the proposed management tool provides an extra support for this characteristic by facilitating change management to the contents of the template.

4.3 Evaluation of the Proposed Template against the SRSD Quality Attributes and two more Criteria

In this section, we evaluate the proposed template against the primarily quality attributes presented in Section 1.1.1: organization, precision, consistency, completeness, non-redundancy, and ambiguity. According to that discussion, the secondary attributes depend on the primarily attributes. However, we justify our template against two secondary attributes named modifiability and traceability. The emphasis on these two secondary attributes is due to their impacts on requirements management. In addition, we used two other criteria presented in [San03] to evaluate the proposed template: *i*) methodology independence and *ii*) breadth of applicability.

4.3.1 Organization

In the proposed template we consider the principal of separation of concern. we organize the document by using sections and subsections in a hierarchical format. Thus, the document is started by identifying super sections and then those are refined into low level subsections for holding the requirements. In this format, requirements with a similar concern are organized under the same section.

4.3.2 Precision

In the proposed template an adequate description about the contents which should go into each section, is presented. To support numerical precision, we assign a section for documenting the accuracy of each data item in the mathematical models.

Furthermore, systematic step by step documentation of scientific models is adequate for model description precision.

4.3.3 Consistency

Two types of consistency presented in Section 1.1.1: *space* and *behavior*. They address consistency in declaring the shared variables and the actions to be carried by the system in reacting to same triggering events. In the proposed template, we associate information items named *referenceTo*, and *usedIn* to each requirement. These items protect the requirements document from redundancy and the chance of defining a requirement elsewhere differently in the document. Traceability report provides the basis for both types of consistency checking. In addition, adopting a specific technique in writing the content of the template such as using tabular expressions as used in [Lai04] supports behavior consistency.

4.3.4 Completeness

Three types of completeness are presented in Section 1.1.1: *space*, *content*, and *semantics*. Specification is space complete, if it prescribes actions to be carried by the system at every state in the system's space domain. It is content complete if it includes all categories of requirements that pertain to the product. It is semantic complete if it includes all explicit and implicit assumptions and constraints related to the intended system. In Sections 4.2.1 and 4.2.2, we illustrated how the structure of the proposed template satisfies the requirements documents for manufacturing systems. According to that evaluation, the categories of the proposed template is adequate to address manufacturing needs. In Section 4.2.2, we advocated complementary usage of goal-driven analysis, viewpoint-oriented methods, and scenario-based techniques for documenting the requirements of manufacturing systems to

Table 4.6: Power Conditioning Function [LFM01]

$Power \leq K_{out}$	$K_{out} < Power < K_{in}$	$Power \geq K_{in}$
FALSE	Prev	TRUE

enhance content completeness. The specification is content complete if all the system goals are achievable. Hence, in the proposed template we aim to expect all the requirements categories lead to the statements of the goals. Furthermore, we tackle a goal statement from different perspectives, and describe all the actions which should be carried by the system through the scenarios. This approach provides a complete structure for documenting the requirements. In addition, to support semantic completeness, we consider separate sections for documenting the system constraints and required assumptions for each possible scenario. The constraints and assumptions should be considered even if we use informal methods. However, we suggest using tabular notations for describing assumptions and constraints. According to the example stated in [LFM01], semantic incompleteness may lead to inconsistency. This example explains a simple power specification system presented in Table 4.6.

There is an undocumented implicit assumption which refers to $Kout < Kin$. Consequently, if $Power = Kin$ power function is always true and if $Power = Kout$ it is always false. Therefore, the specification given in Table 4.6 is inconsistent for $Kout = Kin$.

4.3.5 Non-Redundancy

Redundancy increases maintenance effort and is a potential source for inconsistency in the requirements document. Every time a requirement changes, its duplicate should also change otherwise two versions of the requirement will be left in the document. To avoid redundancy in the documentation, we consider a unique identifier for each requirement. Then, if a requirement needs to be used in more than one location, it should be cross-referenced. It means that, the intended requirement should be located in one place and referenced from other parts of the document. Referencing is handled by an associated information named *referenceTo* to each requirement. However, referencing causes error if there is no change control. A common example is, if the content of the referenced requirement is changed, the meaning can be altered. This may affect the accuracy of the requirement which is referencing to the changed one. To avoid this inefficiency, we associate an information item named *usedIn*. This item is used to keep the identifiers of the requirements referenced to the intended one.

4.3.6 Ambiguity

It is explained in Section 1.1.1 that a specification is unambiguous, if it is not interpreted variously by different readers. In the proposed template, we preserve separate sections for introducing data definitions, specifying physical phenomena and describing system behaviors. However, the use of well-formed syntax for writing the requirements, and tabular notations for representing data definitions and system behaviors are suggested to reduce ambiguity.

4.3.7 Traceability

As described in Section 1.1.2, the requirements traceability identifies the source of requirements, and their dependencies that might be affected by future changes. This makes the change management possible, whereas ensures us that changes in some requirements do not affect others adversely. Traceability is required in all phases of software development and from one software development phase to another.

In this research we focus on satisfying requirement traceability. Four types of traceability patterns are presented in [San03]: *forward traceability*, *backward traceability*, *version traceability*, and *cross-referencing.* Forward traceability is tracing the requirements to the design. Backward traceability is mapping the requirements to their original sources. Version traceability refers to different versions of the SRSD. Cross-referencing relates the requirements within a SRSD.

In the proposed template, we associate the following information to the requirements: *identifier, name, description, source, history, lastUpdate, referenceTo, usedIn.* This information is used to support backward traceability, version traceability, and cross-referencing. The item *identifier* is important for requirements uniqueness which is the basis for referencing. Maintaining *sources* of the requirements is necessery for backward traceability. The items *history*, and *lastUpdate* are important for version traceability. The items *referenceTo*, and *usedIn* are important for cross-referencing in SRSD. In [AC03], *include*, and *expand* terms are used instead of *referenceTo* and *usedIn* respectively.

4.3.8 Modifiability

Software requirements do change. The evolutionary nature of manufacturing systems needs a special support for modifiability. Two types of modifiability are presented in [San03]: *semantic modifiability*, and *presentation modifiability.* Semantic modifiability refers to the ability to change requirements in a consistent way. This is related to the issue of non-redundancy in the requirements. Presentation modifiability depends on the organization, traceability, and non-redundancy of the SRSD. We described how the proposed template addresses each of these attributes in their corresponding sections. Also, to specify the requirements that are more likely to change, we dedicate a section to *expected possible changes* in the proposed template. However, we believe an automated management tool with a graphical user interface has a valuable impact on the modifiability of the requirements.

4.3.9 Methodology Independence

This criterion evaluates whether the template forces specific technology for documenting the requirements. A complete discussion regarding the advantages and disadvantages of the templates to be methodology dependent or independent is held in [San03]. From that discussion, it is suggested that decision making regarding dependency of the methodology should be based on the application.

In the proposed template, we advocate complementary usage of goal-driven analysis, viewpoint-oriented methods, and scenario-based techniques that we described in Chapter 3. In addition, we use a systematic step by step process for documenting the requirements, and preserve an optional template section for *Control Flow Diagrams.* However, we trade off this level of methodology dependency with the advantages discussed in Section 4.2.2 for having these methodologies.

4.3.10 Breadth of Applicability

We consider the general characteristics of manufacturing systems in organizing the proposed template. However, we believe that the proposed structure is comprehensive enough to include other systems that have some of those characteristics.

4.4 Conclusion

In this chapter, we showed that none of the templates available in the open literature could solely satisfy the needs of quality requirements documentation for manufacturing systems. Therefore, we proposed a new requirements template where two main objectives are pursued: *i*) being suitable for documenting the requirements of manufacturing systems and *ii*) satisfying the primary quality attributes as well as the two criteria of methodology independence and breadth of applicability presented in Section 1.1.1.

We described the characteristic of manufacturing systems and justified the suitability of the proposed template for each of them. We elaborated on how the proposed template supports primary quality attributes. In addition, we explained the influence of two secondary attributes, namely the traceability and modifiability, on the requirements management. We elaborated on the measure taken to satisfy these two attributes. A trade off was made between the methodology independence of documenting the requirements and the advantages of the complementary usage of goal-driven analysis, viewpoint-oriented methods, and scenario-based techniques. A systematic step by step process for documenting the requirements is also adopted. The proposed template could be applicable for other systems with characteristics similar to those of the manufacturing systems.

Chapter 5

Design, Implementation, and Validation of the Management Tool

In Section 1.1.3, we explained the role of the requirements management in the process of software development. According to that discussion, the requirements management establishes and maintains a common understanding between system builder and stakeholders, and minimizes the difficulties that emerge due to changes in requirements. However, it is illustrated that requirements management is not effective if the SRSD is not qualified with respect to the primary quality attributes.

In Section 4.3, we elaborated on how the proposed template and its associated information such as requirements identifiers, requirements references, source of requirements, and history of changes, support quality attributes. In addition, they ensure that changes in some requirements do not adversely affect others. The structure of the proposed template and its associated information make requirements management possible. However, without an automated tool for tracing the requirements, managing changes and enhancing requirements consistency become tedious.

In this chapter, we focus on designing, implementing and validating a System for MAnagement of Requirements according to the proposed Template for manufacturing systems (SMART). We present the design, discuss some technologies that we adopted in implementing SMART, and comment on its validation by our client. SMART aims to create a relatively secure, manageable and easy-to-use application for documenting and retrieving requirements. It accelerates capturing the requirements and improves system quality while reducing errors.

Protecting the valuable and sensitive information from unauthorized access is a must, especially for the environments that involves many temporary persons. This capability for giving the access permission to the right persons, protects the document from entering the requirements that are likely to be wrong and enhances consistency among the requirements. In SMART, we have two levels of security, one for the administrator and another for general users. The later has limited access for the information as well as the operation that can be performed.

The user-friendly interface of our management tool increases modifiability by providing a more comfortable environment for the users. The evolutionary nature of the manufacturing systems requires such an environment for updating the requirements and controlling the change-impact. In addition to a user friendly environment for changing the information, using Extensible Markup Language (XML) as a management base provides an extensible structure for SMART as described in Section 5.2.1.

A powerful dynamic report generator that can be configured by the user provides a simple way for retrieving the requirements and their dependencies. A simple tick-list mechanism enables the user to build a wide range of reports directly in PDF and RTF formats. Furthermore, we offer the option of using Latex for writing the mathematical formula and drawing tables in the corresponding text areas of

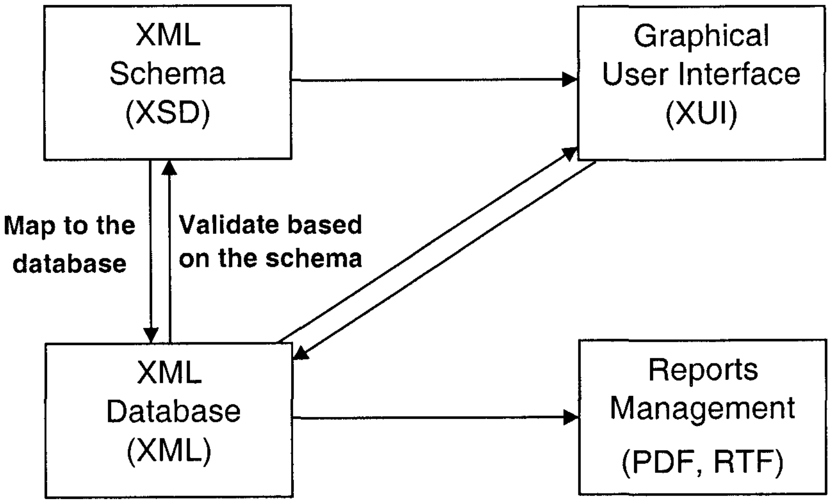

XML Schema (XSD)
Graphical User Interface (XUI)
Map to the database
Validate based on the schema
XML Database (XML)
Reports Management (PDF, RTF)

Figure 5.1: Requirements Management Tool: Technology Perspective

the graphical user interface. This enables users to add desirable formats to the generated report. Figure 5.1 shows SMART from a technology perspective.

In the rest of this chapter, we describe how the system performs these tasks by decomposing the systems tasks into modules and then explaining the services each module should provide. We elaborate on the technologies that helped us in implementing the management tool and we give the rationale behind our choices. We also illustrate the validation of SMART by our research partner.

5.1 System Architecture

To simplify the system, we decompose its functionality to a set of manageable modules. Figure 5.1 gives the module decomposition hierarchy for SMART. It depicts the modules and their relationships. The arrow line from module A to module B determines that accomplishing tasks in module A requires some results from module B.

Modular design is useful in four ways: *i*) it reduces complexity by splitting the system into small parts, *ii*) it facilitates maintenance by encapsulating expected changes in separate modules, *iii*) it facilitates implementation through parallel tasks, *iv*) and it facilitates testing by localizing the objects that should be tested together. Information hiding is a module decomposition technique that is carried out by identifying the expected changes, and it encapsulates each expected change called module's secret [HS95]. Modules can be divided into three groups according to the information they hide [HS95]:

1. *Behavior-hiding modules*: They hide input formats, screen formats, and text messages.

2. *Software decision-hiding modules*: They hide internal data structure, and algorithms.

3. *Machine-hiding modules*: They hide the characteristic of the hardware machines or virtual machines.

The following is the module guide we have derived for SMART. For each module we gave a name, a service and a secret. The service specifies the function that the module should provide, while the secret identifies the change that the module hides. A secret type has been associated with each secret. The secret types include data structure, algorithm, virtual machine, input formats, and text message.

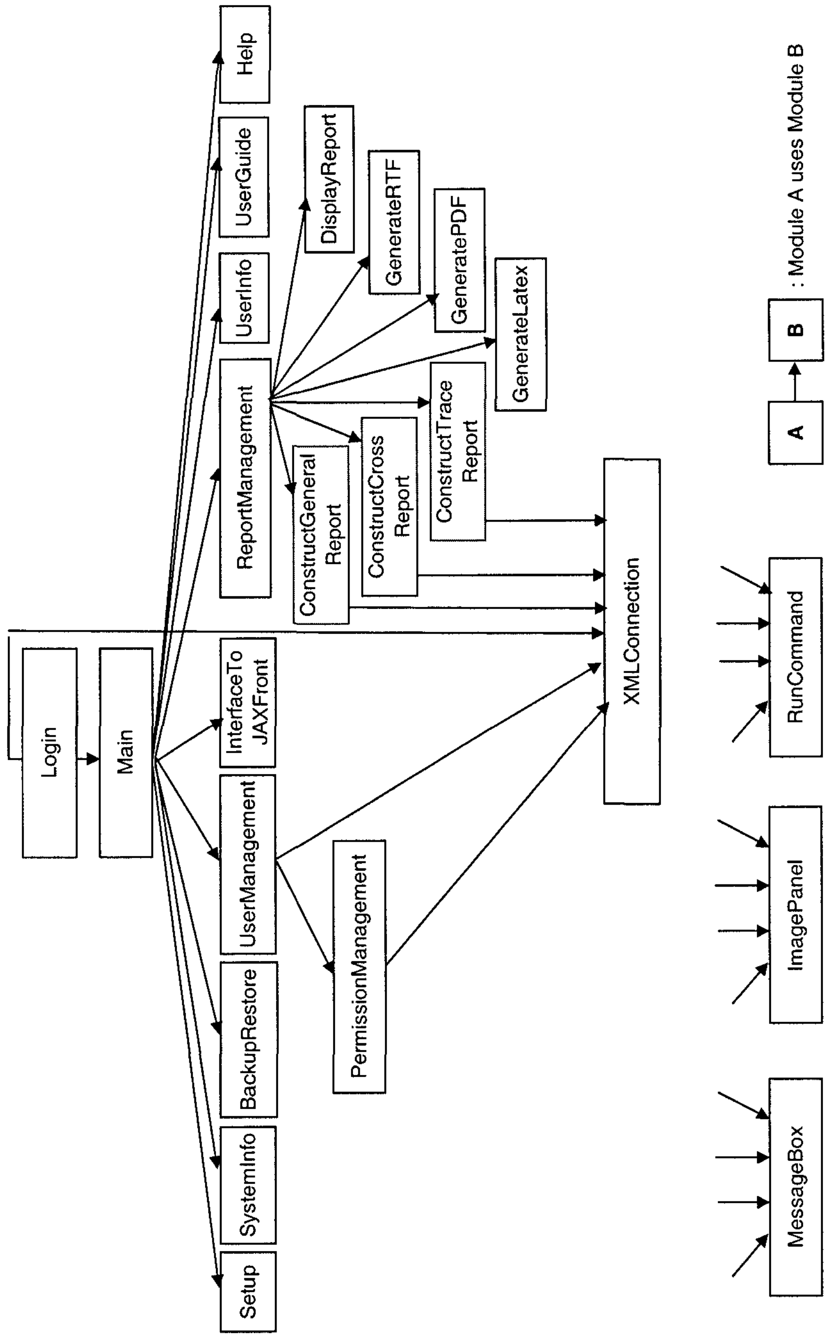


Figure 5.2: Requirements Management Tool: System Architecture

(1) *Name:* **Login Module**

Service: Verifies users authorizations.

Secret: The data structure needed to store users information. Secret type: data structure.

(2) *Name:* **Main Module**

Service: Integrates together all of the functions of the system except login function.

Secret: The sequence of the modules to call. Secret type: algorithm.

(3) *Name:* **Setup Module**

Service: Allows user to setup the system's directories.

Secret: The input format of the directories for set up. Secret type: input formats.

(17) *Name:* **SystemInfo Module**

Service: Displays the system information including the data and program directories that system has been set up.

Secret: The data structure to represent the current system setup. Secret type: data structure.

(4) *Name:* **BackupRestore Module**

Service: Allows user to provide backup and restore from the selected directories.

Secret: The input format of the directories for backup. Secret type: input formats.

(5) *Name:* **UserManagement Module**

Service: Provides a set of available users of the system to update them.

Secret: The data structure to represent the available users. Secret type: data structure.

(6) *Name:* **PermissionManagement Module**

Service: Provides a set of requirements sections that can be accessed by the intended user.

Secret: The set of all the sections of requirements document. Secret type: data structure.

(7) *Name:* **InterfaceToJAXFront Module**

Service: Interfaces to the JAXFront technology for updating the requirements.

Secret: The language to call JAXFront. Secret type: virtual machine.

(8) *Name:* **ReportManagement Module**

Service: Integrates all the functions related with providing the requirements output.

Secret: The algorithm to integrate the functions for providing requirements output. Secret type: algorithm.

(9) *Name:* **ConstructGeneralReport Module**

Service: Constructs a dynamic report according to the user selection.

Secret: The set of all the sections of requirements document. Secret type: data Structure.

(10) *Name:* **ConstructCrossReport Module**

Service: Constructs a dynamic report according to the user selection.

Secret: The set of all the sections of requirements document. Secret type: data Structure.

(11) *Name:* **ConstructTraceReport Module**

Service: Constructs a dynamic report according to the user selection.

Secret: The set of all the sections of requirements document. Secret type: data Structure.

(12) *Name:* **GenerateLatex Module**

Service: Provides latex format from extracted information.

Secret: The command to make Latex file. Secret type: virtual machine.

(13) *Name:* **GeneratePDF Module**

Service: Provides PDF format from the generated Latex file.

Secret: The command to make PDF file. Secret type: virtual machine.

(14) *Name:* **GenerateRTF Module**

Service: Provides RTF format from the generated Latex file.

Secret: The command to make RTF file. Secret type: virtual machine.

(15) *Name:* **DisplayReport Module**

Service: Displays the content of the PDF or RTF files on the screen.

Secret: The command to display PDF or RTF files. Secret type: virtual machine.

(16) *Name:* **UserGuide Module**

Service: Displays a guideline for the users who work with the requirements management tool.

Secret: The command to display PDF file. Secret type: virtual machine.

(17) *Name:* **UserInfo Module**

Service: Displays the user information including user name, Id, and his/her access level.

Secret: The data structure to represent the current user information. Secret type: data structure.

(18) *Name:* **Help Module**

Service: Displays a guideline about the content of each element of the template.

Secret: The command to display PDF file. Secret type: virtual machine.

(19) *Name:* **XMLConnection Module**

Service: Returns the root of XML file to the calling module.

Secret: The command to connect to the XML file. Secret type: virtual machine.

(20) *Name:* **MessageBox Module**

Service: Displays the message information.

Secret: The format of the messages. Secret type: text message.

(21) *Name:* **ImagePanel Module**

Service: Provides an image for a panel background.

Secret: The format of the image. Secret type: text message.

(22) *Name:* **RunCommand Module**

Service: Run an external command.

Secret: The format of the command. Secret type: text message.

5.2 Technical Decisions

Some technical decisions are made to provide an integrated and powerful environment for designing and implementing SMART. Discussion held in Section 5.2.1 justifies our choice for selecting XML to structure and document the content of our SRSD. XMLSpy technology is introduced in Section 5.2.3 to visually design XML Schema. To provide a convenient interface between the end-users and SMART, we present a powerful technology for implementing Graphical User Interface (GUI) in Section 5.2.2. Finally, we present our justification for selecting Java for implementing SMART and Eclipse environment for developing the Java project in Sections 5.2.4, and 5.2.5.

5.2.1 XML

The extraordinary growth of demand for distributing electronic documents over the web has fueled the development of Extensible Markup Language (XML) for applications that need more functionality than current Hyper Text Markup Languages (HTML) can provide. Development of XML is started in 1996 [Lib06]. XML is designed by taking the best parts of Standard Generalized Markup Language (SGML) defined by ISO 8879 and experienced gained from HTML [Lib06]. It is designed for structuring, describing, storing, and passing the data. It is strongly believed that XML will be the most important tool for manipulating and transmitting data [W3S06]. An XML file consists of a plain text and some tags. Tags are sequence of characters enclosed in angle brackets ('<' and '>'). Like HTML, XML uses tags to specify the logical components of document, which delimit the information such as formatting and specification about the document.

The mechanism of identifying structures in a document is called a markup language. However, unlike HTML, XML tags are not predefined. The XML user is allowed to define her/his tags and leaves the interpretation of data to the application reading these data. To be more specific, XML is a set of rules for describing the data by adding markup for documenting and designing text formats. Accordingly, any software that can work with text file can handle XML files. This makes the major role of XML in connecting heterogenous databases. It is ideal not only for transferring data from server to browser but also for sending the data from application to application and machine to machine [Lib06]. The function of XML could be simulated by the function that ASCII did for unifying the representation of the letters on the computer many years ago [SF06].

Incompatible format between computer systems and databases is one of the most challenges in computer technology. Converting the data between incompatible formats is very time consuming. Neutral characteristic of XML helps to reduce this complexity. The plain text format of data created by XML can be read by almost all types of applications. It also facilitates upgrading a system to new operating system, server and/or browser.

The structure of an XML is described by XML Schema Definition (XSD). XSD defines building blocks of XML documents which are elements and attributes. Elements are specified by start tag and end tag, shown by <element-name>, </element-name>. Attributes are attached to the elements to associate values to an element. Any search in XML document is done by looking for start tags and end tags. Accordingly, adding more elements to the structure of the XML do not cause any break or crash to the application that extract the data from XML document.

In addition, XSD defines child-parent relationship between the elements, defines default and fixed values for elements, and defines data types for elements. It allows the author to create her/his own data types. The strength about XSD schemas is that, they are written in XML. Hence, there is no need to learn new language to work with them. Also, they could be edited and parsed with the same editor and parser used for XML file. XSD increases usability of the system by referencing the other schemas and reusing by other schemas.

The reasons for advocating XML for documenting data could be summarized as follows: [W3S06, Lib06, Stu06, Wal98]:

1. XML provides a rich documentation structure capable to run over the web. The syntax rules of XML are very simple. Accordingly, it is easy to use and manipulate by generic applications.

2. XML is a meta language that allows the author to define her/his tags and document structure. Its modularity allows one to define new document by

combining other schemas.

3. XML is platform independent and license free. That means, it is software and hardware independent. The syntax of an XML describes the relationships among the elements. Hence, no prior information about the sender is required. Using the XML as a basis for the project reduces the cost of storing, converting, and transferring the data. It shipped with internet explorer 5 and higher and has the features for many languages such as VB, Java, C++, Perl.

4. Documenting the system by XML enhances usability of the system by making the data accessible to all kinds of reading machines. Accordingly, more data could be available for blind people or people with other disability.

5. The XML document is extensible to carry out more information. New elements do not affect the search engine for previous elements.

6. It is possible to validate the correctness of XML data against its XSD. An XML document is valid if it obeys the syntax of XML and obeys the constraints that is defined by its XSD, such as: attribute values have the correct type, child-parent relationship is valid, etc.

7. XML is a family of technologies. In spite of being a new technology, widespread technologies and tools such as XSL, CSS, DOM, TurboXML, Stylus Studio, VisualScript, Altova(XMLSpy), JAXFront are available to support XML.

The above characteristics of XML justify our choice for selecting XML to structure and document the content of our SRSD. The structure of requirements specification document based on the proposed template is presented in appendix B.

5.2.2 JAXFront

As illustrated in Section 5.2.1, the XML document is extensible to carry out more information. Handling extensibility in a visual environment is high demanding. Java Enabled XML Frontends (JAXFront) is a technical tool facilitating XML document manipulation visually. JAXFront provides an interactive Extensible User Interface (XUI) to a valid XML schema (XSD). This graphical user interface allows editing the underlying XML data or the creation of new one. Any changes in the XML schema are reconstructed directly to the GUI, and any modifications done through GUI will be validated against the XML schema. Only valid changing will affect the underlying XML document. Other strong factors of JAXFront convincing us to choose this technology for developing our graphical user interfaces are: platform independence, multiple natural languages supportance, ability to add plug-ins, functions, and actions to the XUI. We refer the reader to [JAX06] for more information on this technology.

5.2.3 XMLSpy

Although XML schema can be made by using solely Microsoft technologies, we used XMLSpy technology to visually design the structure of our requirement specification document based on the template proposed in Chapter 4. XMLSpy is a pervasive tool for developing projects with XML base. We benefited of some of the features this software provides such as designing the structure of the document visually, providing XML document according to the XSD, and validating the XML document against the XSD. We refer the reader to [XML06] for more information on this technology.

5.2.4 Java

We opted Java as the programming language to implement SMART for the following reasons:

1. Java is Platform independent
2. Many Java classes are available to parse and manipulate XML documents
3. Programming in Java provides wide accessibility to other technologies implemented for manipulating XML documents. For example, it is easy to handle the communication between Java programs and JAXFront explained in Section 5.2.2 and we chose Java to implement the graphical user interface for SMART. JAXFront accepts Java plug-ins and can be embedded in Java programs easily.

5.2.5 Eclipse

Eclipse is a powerful, standard-based, and free Integrated Development Environment (IDE). Recently Eclipse came out as superior IDE, which enables us to develop Java projects with some facilities such as syntax highlighting, popup windows for different Java classes, debugging, refactoring, auto-compiling, and packaging the whole project in a jar file [Inn06]. Eclipse platform offers facilities to add plug-ins implemented by other companies. Specifically, we used JAXFront plug-in to communicate with JAXFront through Eclipse in our project.

5.3 Validation of SMART

We partially elicit the requirements needed for developing a software simulation for the *Tool Trajectory Planning for High Speed Machining.* To describe this process, we first briefly explain the process of manufacturing a product [CJ98] and then explain the role of the *Tool Trajectory Planning* in this process.

Machining a part is an important step in manufacturing of a new product. It requires producing master models by the skilled hands of artisans or virtual design tools such as Computer Aided Design (CAD). The geometric information is digitally saved. A process planning and tool path strategies will be elaborated either manually or using the Computer Aided Manufacturing (CAM) tools. The resulting data will be post-processed to create the program for the Computer Numerical Controlled (CNC) cutting machines. The CNC machine removes the material of a workpiece according to the provided program.

The *Tool Trajectory Planning for High Speed Machining* aims to optimize tool trajectory for removing material. The objective is to maximize the amount of material to be removed while guaranteeing the quality of the product and the safety of machining. This is a vital objective especially for high speed machining where the spindle speed exceeds 8000 Revolution Per Minutes (RPM) and could reach 60000 and even 100000 RPMs for ultra-modern machines.

Along this thesis we illustrate the role of the requirements activities to support manufacturing process. We elicit a sample requirements from National Research Council of Canada (NRCC). We feed the gathered requirements to the management tool and validate the result by our research partner in NRCC. A demo of the management tool, including the elicited requirements had been represented to the users of the tool. However, due to the confidentially agreement signed with NRCC, we are not supposed to reveal the information gathered to non-signatory partners. The purpose for gathering this information was to enable us to propose a template adequate for manufacturing systems and to enable our client to validate both the template and the requirements management tool.

Our client found that the new structures for the functional and non-functional requirements are adequate for documenting the requirements for manufacturing systems. The trade-off made between the complementary usage of goal-driven, viewpoint oriented, and scenario-based approaches and the criteria of the methodology independence of the template is appropriate and justifiable due to the nature of the manufacturing systems. The systematic step-by-step documentation is crucial to tackle the complexity of manufacturing systems.

Our client is convinced that the association of additional information, such as code, source, history, lastUpdate, referenceTo, and usedIn, with the requirements leads to enhancing the quality of the requirements document. As far as the management tool is concerned, our client found that the functionalities provided by SMART responds to the important basic needs for security and dynamic reporting. SMART went beyond the original agreement with our client where fixed maximum numbers of goals, viewpoints, and scenarios are to be considered. SMART is providing non-limited dynamic lists for these items.

Our client team appreciated the conviviality and the ease of use of SMART, noticed when feeding some simple case studies. This will encourage them to use it and assess its performance for more substantial case studies. Being extensible, SMART constitutes an important step-forward toward developing a sophisticated management tool that would include automatic space consistency checking and other features.

5.4 Conclusion

In this chapter, we described the role of management tools in the requirements activities in general and specifically the features provided by the proposed tool (SMART). The services provided by SMART are: a friendly user interface, permission check, and dynamic reporting. We decomposed the system into modules to simplify the system development and explained the service and secret of each module. We explained and justified XML, XMLSpy, JAXFront, Java, and Eclipse as the technologies that helped us in implementing our management tool. Finally, we reported on the validation of SMART with a case study to document the requirements for the *Tool Trajectory Planning for High Speed Machining* system developed at *National Research Council of Canada* (NRCC).

Chapter 6

Conclusion

The goal of this research is to provide a requirement documentation guidance for manufacturing systems. A management tool is implemented to mechanize the process of documenting, retrieving, and tracing the requirements in a secure, flexible, and user friendly environment. We present our concluding remarks in Section 6.1, main contribution in Section 6.2, and propose some potential future work in Section 6.3.

6.1 Concluding Remarks

In this thesis, we described the influence of the requirements phase in the software development life-cycle in general and made specific attention to structuring and managing the requirements for manufacturing systems. We advocated using templates as an intuitive way for structuring the requirements documents. We proposed a new template aiming at two main objectives: being suitable for documenting the requirements for manufacturing systems and satisfying SRSD primary quality attributes. To achieve these goals we focused on two related issues. We described the criteria for a "good" template according to SRSD primary quality attributes, and elaborated on distinguished characteristics of manufacturing systems. According to this study, manufacturing systems involve multi-constraint problems, multi-disciplinary problems, multi-stage processes, and multi-tasking. They have evolutionary nature, dynamic behavior, and deal with scientific models and time-varying physical characteristics.

6.2 Main Contributions

We reviewed the literature for the well known templates and the recent research efforts on structuring software requirements. We started from existing templates then we restructured documenting the functional and non-functional requirements in order to achieve the intended objectives.

The essential methodology we advocated for structuring the proposed template is the complementary usage of goal-driven, viewpoint-oriented, and scenario-based approach. We also followed a systematic step by step documentation for documenting manufacturing systems requirements. In addition, we associated some information to the requirements to enhance the quality of the requirements documents and improve the traceability and modifiability of these requirements. These information include: *identifier, name, description, source, history, lastUpdate, referenceTo*, and *usedIn*. An evaluation process is presented to ensure the capabilities of the proposed template.

To demonstrate a common understanding between the system builder and the stakeholders, enhance modifiability and traceability of requirements documents and make change management possible, a management tool named SMART is designed and implemented. SMART aims to create a relatively secure, manageable, and easy-to-use application for documenting and retrieving the requirements. We elaborated on the technologies that provide an integrated and powerful environment for designing and implementing SMART. We used XML to structure and document the content of our SRSD to reduce the cost of storing, converting, and transferring the data. Our reasons for this selection are:

1. Platform independency
2. Capability to run over the web
3. Extensibility to carry out more information
4. Being a part of a family of technologies

We opted for Java as the programming language to implement SMART due its platform independency and its ability to provide wide accessibility to other technologies implemented for manipulating XML documents. We developed our Java tool through Eclipse, a free integrated development environment. Other complementary technologies that we used are:

1. XMLSpy to visually design the structure of our requirement specification document.
2. JAXFront, a technical tool facilitating XML document manipulation visually.

We assessed SMART with a case study to document the requirements for the *Tool Trajectory Planning for High Speed Machining* system developed at *National Research Council of Canada* (NRCC).

6.3 Future Work

This thesis encourages further research in two dimensions: upgrade SMART and improve the template. We propose them as follows:

1. Enrich SMART by adding more report options.

2. Integrate automatic space consistency checking to the tool which would lead to a more quality specification document. However, behavior consistency will require a quite considerable amount of work.

3. Improve the system by adding an automatic change management module which handles changes applied to the requirements, and checks their dependencies.

4. Connect the requirements to the design level to support forward traceability.

5. Make SMART more practical by automatically adding an identifier to each requirements item. In the current version of SMART, users enter the identifier for requirements items. We expect an automatic way for generating requirements identifier to protect the requirements management tool from user's mistakes.

6. Validate the management tool by handling formal requirements.

7. Additional validation of the template by using more substantial case studies.

Bibliography

[Abr98] J. R. Abrial. B user manual. Technical report, Programming Research Group, Oxford University, 1998.

[AC03] Joao Araujo and Paulo Coutinho. Identifying aspectual use cases using a viewpoint-oriented requirements method. In *Workshop of 2nd Int. Conf. on Aspect- Oriented Software Development (AOSD)*, Boston, USA, March 2003. Aspect-Oriented Requirements Engineering and Architecture Design.

[AMP94] Annie I. Antn, W. Michael McCracken, and Colin Potts. Goal decomposition and scenario analysis in business process reengineering. In *6th international conference on Advanced information systems engineering*, 0-387-58113-8, pages 94 – 104, Utrecht, The Netherlands, 1994. Springer-Verlag New York, Inc. Secaucus, NJ, USA.

[Bha00] M. Asghar Bhatti. *Practical Optimization Methods.* Springer-Verlag New York, Inc., 2000.

[Bss91] ESS BssC. Software project documents. In *ESA PSS-05-0-Issue 2.* European Space Agency, February 1991.

[BtN01] Aharon Ben-tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization.* MPS/SIAM Series on Optimization, Philadelphia, PA, USA, 2001.

[CEJM06] C.C. Christara, W.H. Enright, K.R. Jackson, and R.A. Mathon. Numerical analysis. http://www.cs.toronto.edu/DCS/Groups/numerical.html, 2006. (Last posted: 22/8/2006).

[Che03] C. Chen. A software engineering approach to developing a mesh generator. Master's thesis, McMaster University, Hamilton, Ontario, Canada, September 2003.

[CJ98] Byoung K. Choi and Robert B. Jerard. *Sculptured Surface Machining.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[Coc95] Alistair Cockburn. Structuring use cases with goals. Technical Report 84121, Human and Technology, Salt Lake City, Utah, 1995. HaT.TR.95.1.

[Def88] Defense system software development. In *DOD Standard 2167A*. US Department of Defense, February 1988.

[DFKM98] Jules Desharnais, Marc Frappier, Ridha Khedri, and Ali Mili. Integration of sequential scenarios. *IEEE Transactions on Software Engineering*, 24(9):695–708, September 1998.

[Eas91] Steve Easterbrook. *Elicitation of Requirements from Multiple Perspectives.* PhD thesis, Department of Computing, Imperial College, London, UK., June 1991.

[Fau95] S. R. Faulk. Software requirements. Technical Report NRL/MR/5546-95-7775, Naval Research Laboratory, Washington DC, 1995. A tutorial.

[GMW81] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press INC, San Diego, CA 92101, 1981.

[Ham06] Sven Hammarling. Backward error analysis and numerical software. http://www.cerfacs.fr/algor/PastWorkshops/IndustrialDays2000/absf-ormer.html, 2006. (Last posted: 7/3/2006).

[Har87] D. Harel. Statechart: Visual formalism for complex system. In *Science of Computer Programming*, 8:231-274. 1987.

[HD98] Patrick Heymans and Eric Dubois. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. *Requir. Eng.*, 3(3/4):202–218, 1998.

[Hea02] Michael Heath. *Scientific Computing An Introductory Survey.* McGraw-Hill Publishing Company, 2nd edition, 2002.

[HKSP78] K. Heninger, J. Kallander, J. Shore, and D. Parnas. Software requirements specification for a-7e aircraft. Technical Report memo-3876, Naval Research Laboratory, Washington DC, November 1978.

[HS95] Daniel M. Hoffman and Paul A. Strooper. *Software design, automated testing, and maintenance.* International Thomson Computer Press, 1995.

[IEE98] IEEE. IEEE recommended practice for software requirements specifications. In *IEEE Standard*, pages 830–1998. New York, US, June 1998.

[Inn06] Object Innovation. Using eclipse overlays with oi java courses. http://www.objectinnovations.com/Guides/EclipseOverlays.html, 2006. (Last posted: 22/6/2006).

[JAX06] JAXFront. http://www.jaxfront.com/pages/home.html, 2006. (Last posted: 22/6/2006).

[Kev90] L. Kevin. Z++: An object-oriented extension to z. In Workshops in Computing, editor, *Z User Workshop*, pages 151–172. Springer-Verlag, 1990.

[KS98] G. Kotonya and I. Sommerville. *Requirements Engineering.* 1998.

[Lai04] L. Lai. Requirements documentation for engineering mechanics software: Guidelines, template and a case study. Master's thesis, McMaster University, Hamilton, Ontario, Canada, September 2004.

[Lam01] Axel Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings 5th IEEE International Symposium on Requirements Engineering*, pages 249–262, Toronto, Canada, August 27-31 2001. Institue of Electrical and Electronic Engineers, Inc.

[LFM01] M. Lawford, P. Froebel, and G. Moum. Application of tabular methods to the specification and verification of a nuclear reactor shutdown system. Ontario Power Generation, 700 University Ave., Toronto, ON, CANADA M5G 1X6, August 20 2001.

[Lib06] WebDevelope's Virtual Library. Structuring data for the web. http://wdvl.internet.com/Authoring/Languages/XML/Intro/benefits.html, 2006. (Last posted: 22/6/2006).

[LY01] Lin Liu and Eric Yu. From requirements to architectural design - using goals and scenarios. In *ICSE-2001*, pages 22–30, Toronto, Canada, May 14 2001. From Software Requirements to Architectures (STRAW 2001).

[Mer03] K. Meridji. Documentation and validation of the requirements specification an xml approach. Master's thesis, Concordia University, Montreal, Quebec, Canada, August 2003.

[NAS91] NASA. Nasa software documentation standard. National Aeronautics and Space Administration, Washington, DC 20546, July 1991.

[NKF94] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Software Eng.*, 20(10):760–773, 1994.

[NS96] Stephen G. Nash and Ariela Sofer. *Linear and Nonlinear Programing.* The McGraw-Hill Companies, Inc., 1996.

[OMG00] OMG. Unified modeling language. http://www.omg.org/uml, March 2000. (Last posted 8/4/2006).

[OOGC06] Office Of Government Commerce. http://www.ogc.gov.uk/sdtoolkit/deliveryteam/briefings/businesschange/newline/reqments_mgmt.htm, 2006. (Last posted 24/3/2006).

[PC86] C. Phillips and B. Crnelius. *Computational Numerical Methods.* Chichester, 1986.

[QSS00] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics.* Springer, 2000.

[Reg06] Gil Regev. Goal driven requirements engineering overview. http://lamswww.epfl.ch, RE05 conference, 2006. (Last posted: 18/6/2006).

[Rei85] W. Reisig. Petri nets: An introduction. In *Number 4 in EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[RR99] S. Robertson and J. Robertson. *Mastering the Requirements Process.* 1999.

[RSA98] Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding goal modeling using scenarios. In *IEEE Transactions on Software Engineering archive*, volume 24 of *0098-5589*, pages 1055–1071, Centre de Recherche en Inf., Paris I Univ., December 1998. IEEE Press Piscataway, NJ, USA.

[San03] B. Sanga. Assessing and improving the quality of software requirements specification documents. Master's thesis, McMaster University, Hamilton,Ontario, Canada, August 2003.

[SF06] Quentin Stafford-Fraser. http://www.qandr.org/quentin/writings/xml-importance.html, 2006. (Last posted: 22/6/2006).

[SL05] W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

[SLK05] W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, Accepted June 2005.

[Smi06] W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, St. Paul, Minnesota, Accepted April 2006.

[Spi92] J. M. Spivey. *Z Notation.* Prentice Hall, 1992.

[SSV98] I. Sommerville, P. Sawyer, and S. Viller. Viewpoints for requirements elicitation: A practical approach. In *3rd Int. Conf. Requirements Eng.*, pages 74–81, Los Alamitos, Calif., March 1998. IEEE Computer Society Press.

[Stu06] Stylus Studio. A technical introduction to xml. http://www.xml.com /pub/a/98/10/guide0.html, 2006. (Last posted: 22/6/2006).

[Sut98] Alistair Sutcliffe. Scenario-based requirements analysis. volume 3 of *0947-3602*, pages 48–65. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1998.

[TE02] N. Tounsi and M. A. Elbestawi. Optimization feed scheduling in three axes machining. part I. *International Journal of Machine Tool and Manufacture*, (43 (2003)):253–267, October 4 2002.

[W3S06] W3Schools. Xml tutorials. http://www.w3schools.com/xml/xml_ used-for.asp, 2006. (Last posted: 22/6/2006).

[Wal98] Norman Walsh. http://www.xml.com/lpt/a/98/10/guide0.html, 1998. (Last posted: 22/6/2006).

[Wik06a] Wikipedia. Multidisciplinary design optimization. http://en.wikipedia.org/wiki/Multidisciplinary_design_optimization, 2006. (Last posted: 25/2/2006).

[Wik06b] Wikipedia. Numerical analysis. http://en.wikipedia.org/wiki/Numerical_method, 2006. (Last posted: 20/2/2006).

[Wor96a] J. B. Wordsworth. *Software Development with B.* Addison-Wesley, 1996.

[Wor96b] J. B. Wordsworth. *Software Development with Z.* Addison-Wesley, 1996.

[XML06] Altova XMLSpy. http://www.altova.com/download/xml_suite/xml_tools_enterprise.html, 2006. (Last posted: 22/6/2006).

[YM98] Eric Yu and John Mylopoulos. Why goal-oriented requirements engineering. In *Foundation for Software Quality (RESFQ'98)*, pages 15–22, Pisa, Italy, Jun 1998. 4th International Workshop on Requirements Engineering.

Appendix A

The Proposed SRS Template

In this appendix, we present the structure of the proposed template that is suitable for documenting the requirements of manufacturing systems. We organize the document by using sections and subsections in a hierarchical format. Thus, the document is started by identifying super sections which are refined into low level subsections. In this format, information with similar concerns is organized under the same section. In addition, we associate the following attributes to each requirement: *identifier, name, description, source, history, lastUpdate, referenceTo, usedIn.* The definitions of these attributes are as follows:

1. Identifier: A unique label for each requirement.

2. Name: A distinguishable word or words for each requirement.

3. Description: An explanation to identify what the requirement is about.

4. Source: An originator for the requirement.

5. History: A description about the changes made in the requirements.

6. LastUpdate: The last date that a change is made to the requirement.

7. ReferenceTo: List of other requirements that this requirement includes.

8. UsedIn: List of other requirements that this requirement extends.

In proposing our template, we start from the existing templates, borrow some portions from them and present new structure as needed. We also describe the information to be located in each part of the proposed template. The structure and the content description of sections *Introduction*, *General System Description*, *System Constraint*, *Open Issues*, *Waiting Room* and *Expected Changes* are common to off-the-shelf templates [IEE98, RR99, Bss91, NAS91, Def88, HKSP78, San03, Che03, Mer03, Lai04] and therefore we borrowed them. However, to propose a template suitable for manufacturing systems, we present new structure for two main sections of the proposed template named functional and non-functional requirements. The basic idea of structuring the functional parts comes mainly from the current research efforts presented in [San03, Lai04]. However, we trim those templates to make them suitable for requirements documentation for manufacturing systems. In addition, the description of non-functional requirements comes mainly from Volere Requirements Specification Template [HKSP78]. The proposed template as well as the additional information about the content of each of its elements, are given as follows:

A.1 Template Structure

1. Introduction

   1.1 Document Purpose

   1.2 Terminology Definitions, Abbreviations and Acronyms

   1.3 References

   1.4 Document Organization

2. General System Description

2.1 System Purpose

2.2 System Scope

2.3 System Context

2.4 Operations

2.5 User Characteristics

3. Non-Functional Requirements

3.1 System

3.1.1 Accuracy Requirements

3.1.2 Performance Requirements

3.1.3 Security Requirements

3.1.4 Maintainability Requirements

3.1.5 Look and Feel Requirements

3.1.6 Usability Requirements

3.1.7 Portability Requirements

3.1.8 Simulation Requirements

3.1.9 Others

3.2 Partner Applications

3.2.1 Partner 1

3.2.1.1 Accuracy Requirements

3.2.1.2 Performance Requirements

3.2.1.3 Security Requirements

3.2.1.4 Maintainability Requirements

3.2.1.5 Look and Feel Requirements

3.2.1.6 Usability Requirements

3.2.1.7 Portability Requirements

3.2.1.8 Others

....

3.2.m Partner m

....

4. System Constraints

5. Functional Requirements

5.1 Goal 1

5.1.1 Viewpoint 1

5.1.1.1 Scenario 1

5.1.1.1.1 Attributes and Models

5.1.1.1.1.1 Assumptions

5.1.1.1.1.2 Common Input/Output/Constant Data Items of the Models

5.1.1.1.1.2.1 Data Item Code

5.1.1.1.1.2.2 Data Item Description

5.1.1.1.1.2.3 Characteristics of Numerical Data Item (Data Type, Unit, Format, Definition Interval, Accuracy)

5.1.1.1.1.2.4 Mnemonic Names for the Possible Values of each Non-Numerical Data Item

5.1.1.1.1.3 Theoretical Models

5.1.1.1.1.4 Instanced Models

5.1.1.1.1.4.1 Alternative Instanced Model 1

5.1.1.1.1.4.1.1 Model Description

5.1.1.1.1.4.1.2 Sensitivity of the Model

5.1.1.1.1.4.1.3 Tolerance of the Solution

5.1.1.1.1.4.1.4 Specific Input/Output/Constant Data Items of the Model

5.1.1.1.1.4.1.4.1 Data Item Code

5.1.1.1.1.4.1.4.2 Data Item Description

5.1.1.1.1.4.1.4.3 Characteristics of Numerical Data Item (Data Type, Unit, Format, Definition Interval, Accuracy)

5.1.1.1.1.4.1.4.4 Mnemonic Names for the Possible Values of each Non-Numerical Data Item

....

5.1.1.1.1.5.l Alternative Instanced Models l

....

5.1.1.1.2 Body of the Scenario

5.1.1.1.2.1 System Behavior

5.1.1.1.2.2 Control Flow Diagrams

5.1.1.1.2.3 Others

....

5.1.1.m Scenario m

....

5.1.n Viewpoint n

....

4.p Goal p

....

6. Traceability Matrices

7. Open Issues

8. Waiting Room

9. Expected Possible Changes

A.2 Introduction

This section should provide an overall view to the whole template.

A.2.1 Document Purpose

This subsection should describe the purpose of the current SRSD and state its intended audience.

A.2.2 Terminology Definitions, Abbreviations and Acronyms

This subsection should provide the definition of all technical terms as used in the domain area of the system, and express the abbreviations and acronyms in full terms. This reduces ambiguity in the documents and lets the reader properly interpret the SRSD.

A.2.3 References

This subsection should provide an annotated list of documents referenced in the SRSD as a source of additional information.

A.2.4 Document Organization

This subsection should provide a summary of the information contained in the SRSD and explain how they are organized in the document.

A.3 General System Description

This section should provide a background about the system and should describe general factors that affect the requirements defined in Sections A.4, A.5, and A.6.

A.3.1 System Purpose

This subsection should describe the problem domain and delineate the reason that the system is being developed.

A.3.2 System Scope

This subsection should provide an executive summary of the software system being developed including: software system name, general description of its functionality, and its objectives.

A.3.3 System Context

This subsection should provide a high level view of the software system and its integration with other factors that have a direct interface with the system including: people, partner applications, and organizations. If the system is embedded in a larger system it should be stated here. The important system interfaces should be defined by drawing diagrams to depict system boundaries and fitness of the software system in the context.

A.3.4 Operations

This subsection should provide a list of routine operations required to be done by users including: back up and recovery operations, initiated operations, and data processing operations.

A.3.5 User Characteristics

This subsection should define characteristics of the potential users of the software system including: educational level, technological expertise, experience, age, gender, or physical abilities. The software designer should take them into consideration to make targeted system conform with its user characteristics.

A.4 Non-Functional Requirements

This section should specify the software system overall quality attributes. These attributes describe the qualities or properties of the system and its partner applications. Attributes of partner applications should be considered from perspective of the software system.

A.4.1 System Non-Functional Requirements

This section should specify the qualities or properties of the intended software system.

A.4.1.1 Accuracy Requirements

This subsection should state the possible source of measurement errors and error range applied to each input data of the software system. It is usually specified by the number of significant digits.

A.4.1.2 Performance Requirements

This subsection should specify the performance requirements including: response time, speed, capacity, reliability, and availability requirements.

A.4.1.3 Security Requirements

This subsection should specify the factors that protect the software system information from unauthorized accesses. They include specification of the persons who have been permitted to work with different parts of the system, the circumstances under which a permission is given, the required checks for auditing the system, and/or the required rules to preserve the system integrity.

A.4.1.4 Maintainability Requirements

This subsection should specify the maintenance that the system needs. The required time for fixing a damage or adding a new feature, and any attributes that affect the maintenance of the software system.

A.4.1.5 Portability Requirements

This subsection should specify the attributes that determine how easy the intended software system should be to port to others host machines with different platforms. This may include the percentage of code which depends on the host, or use of particular portable language, compiler, or operating system.

A.4.1.6 Look and Feel Requirements

This subsection should state the requirements relating to the appearance of the intended software system including style, color, and so on. These characteristics should be taken into consideration by designer.

A.4.1.7 Usability Requirements

This subsection should describe clients aspiration regarding: how easy the intended software system should be for its user to learn it and operate it.

A.4.1.8 Simulation Requirements

This subsection should provide non-functional requirements specific to the simulation issues. For example, simulation time, the laws governing the simulation, simulation condition, etc.

A.4.1.9 Other Non-Functional Requirements

This subsection should describe the attributes which affect the intended software system but cannot fit in any of the non-functional requirements categorized in Sections A.4.1.1 to A.4.1.8.

A.4.2 Partner Applications

This subsection should identify quality attributes categorized in Subsections A.4.1.1 to A.4.1.7, and A.4.1.9 for each of the partner applications of the intended software system. Partner applications can be considered as software, hardware, or piece of technology which impact the system. Required attributes should be studied from the perspective of the intended software system.

A.5 System Constraints

This section should provide a list of general constraints that apply to the whole system and affect the next development stage. This includes implementation environment, financial budget, system deadline, rules, and constraints on the solution.

A.6 Functional Requirements

This section should describe all the services that the software system has to provide in various situations. This explanation should be at a level of detail that provides the design team with a clear understanding. It should contain software system inputs, and the processes performed by the system to generate outputs. This section should be organized properly to be suitable for the intended software system. We organize functional requirements based on the system goals. Then we consider each goal from different perspectives. Third, we specify all scenarios that might occur from each point of view. Finally, we follow a systematic step by step process for documenting the requirements needed for describing each scenario.

A.6.1 Goal

Goals are the objectives that the intended systems should meet. These objectives can be business, organization, or system oriented that originated by some stakeholders expectation. This subsection should describe portion of the system objectives that are supported by the intended goal.

A.6.2 Viewpoint

Viewpoints are sources or sinks of data that address only those concerns related to their originator and ignore the others. They are vehicles that facilitate document partitioning in order to support the principal of separation of concerns. System stakeholders, partner applications, system interfaces, and other issues that affect the system functionalities such as mechanical aspects, physical aspects, computational aspects, environmental aspects, and electrical aspects are potential system viewpoints. This subsection should describe the viewpoint that the intended goal is studied under.

A.6.3 Scenario

scenarios are possible ways of using a system to fulfil desired functions. They include a sequence of interactions that might happen under certain condition to satisfy a particular result. They are introduced as system activities, system interactions, roles, real word events, or/and imaginary stories of the system-to-be. Every action in a scenario connected to a goal according to a particular originator. Thus, scenarios are containers for the information required to satisfy the intended goal from a particular viewpoint and express how the intended goal can be implemented. This subsection should describe the scenario objective and the services handled by that.

A.6.4 Attributes and Models

This subsection should express the assumptions, scenario data items, theoretical models, and instanced models of the intended scenario. This explanation provides a clear picture of the information required for designing the task that the intended scenario describes.

A.6.5 Assumptions

This subsection should list the factors that influence the requirements stated in the scenario. They are not system constraints but the assumptions taken for granted to solve the engineering problem(s).

A.6.6 Common Input/Output/Constant Data Items of the Models

Real problems might be modeled differently. This subsection should identify input/output/constant data items of and from the intended scenario regardless of the model considered for modeling the real problem. Provided for each data item are data item description, characteristic of each numerical data including type, unit, format of the data representation, accuracy of the input data, and definition interval, or mnemonic names for possible values of each non-numerical data item.

A.6.7 Theoretical Models

This subsection should describe the set of relevant mathematical equations, axioms, or physical laws used in achieving the intended scenario goal statement.

A.6.8 Instanced Models

Instanced models are the mathematical representations of a real problem. Engineering problems might be modeled in several ways. We assign separate sections for documenting the requirements of each alternative instanced model. The information required to be documented for each alternative instanced model is categorized in Sections A.6.9 to A.6.12

A.6.9 Model Description

This subsection should represent the mathematical model of the problem defined in Section A.6.3.

A.6.10 Sensitivity of the Model

This subsection should check sensitivity of the numerical model expressed in Section A.6.9. Sensitivity of the model causes big changes in the solution due to perturbations in the input data. Accordingly, inevitable computation errors propagate during the computation. To protect the software system from this inaccurate result, sensitivity of the model should be tested. Calculating the condition number might be used for evaluating sensitivity of the model. However, in practice the exact amount of condition number is unknown. In this case, an approximation of condition number over the input domain is required.

A.6.11 Tolerance of the Solution

Solution accuracy is one of the most challenges in scientific computation. This subsection should determine the level of accuracy that is expected for the solution. The solution is assessed by evaluating closeness of the computed solution to the exact solution or how well the computed solution satisfies the problem to be solved.

A.6.12 Specific Input/Output/Constant Data Items of the Model

Inspite of Section A.6.6 that is reserved for documenting common input/output/constant data items regardless of the specific model considered for modeling the real problem, this section should identify input/output/constant data items of and from each alternative instanced model. Provided for each data item are data item description, characteristic of each numerical data including type, unit, format of the data representation, accuracy of the input data, and definition interval, or mnemonic names for possible values of each non-numerical data item.

A.6.13 Body of the Scenario

This subsection should give a detailed explanation about the intended scenario and the flow of information between its processes.

A.6.14 System Behavior

This subsection should describe dynamic functionality of the intended scenario based on the technical requirements identified in Sections A.6.5 to A.6.12. This description includes recording the input data items and applying the models to generate the outputs. This technical refinement should end up to the scenario goal while satisfies the scenario assumptions and system constraints.

A.6.15 Control Flow Diagrams

This subsection should visualize system behavior by diagrams to make it more clear to the reader of the document. This diagram should show major inputs/outputs handled by the scenario and the functions performed by the system in response to an input or to generate an output. This diagram shows information flow in the intended scenario.

A.6.16 Others

This subsection should include any requirements which cannot fit in any of the previous scenario subsections.

A.7 Traceability Matrixes

This section should show the relationships between the requirements. These relationships could identify the entities most likely to be reused, the sources of information, the existent reason of the requirements, and requirements dependencies. Keeping the parent/child relationships between requirements makes change management possible, whereas ensures that changes in some requirements do not affect others adversely.

A.8 Open Issues

This section should include the issues that have been raised but a final decision has not been reached.

A.9 Waiting Room

This section should contain the potential requirements that might be included in the next evolution of the software system.

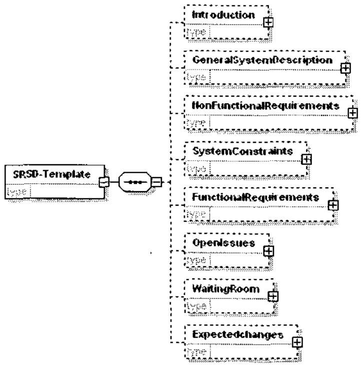
A.10 Expected Changes

This section should characterize the requirements that are more likely to change in the near future. This provides a guideline for the system designer to see them as independent components, to protect the system from frequent changes of these requirements. The expected changes might be related to changes in assumptions, hardware, functions, or/and interfaces.

Appendix B

The Structure of the **XML** Requirements Data File (**XSD**)

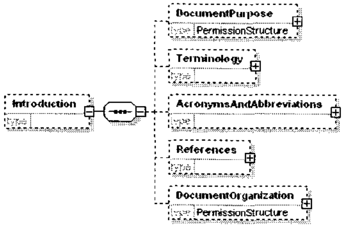
element **SRSD-Template**



```
<xs:element name="SRSD-Template">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Introduction" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="DocumentPurpose" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="Terminology" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Terms" type="CodeStructureItem" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="AcronymsAndAbbreviations" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="AcrAbb" type="CodeStructureItem" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="References" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Reference" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="DocumentOrganization" type="PermissionStructure" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
```

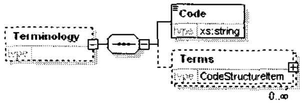
```
   </xs:element>
   <xs:element name="GeneralSystemDescription" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="SystemPurpose" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="SystemScope" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="SystemContext" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="Operations" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="UserCharacteristics" type="PermissionStructure" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="NonFunctionalRequirements" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="SystemNonFunctional" type="SystemNonFunctional" minOccurs="0"/>
      <xs:element name="PartnerNonFunctional" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="PartnerApplications" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Code" type="xs:string"/>
            <xs:element name="Name" type="xs:string" minOccurs="0"/>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:element name="Source" type="xs:string" minOccurs="0"/>
            <xs:element name="History" type="xs:string" minOccurs="0"/>
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Partners" type="PartnerNonFunctional" minOccurs="0"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="SystemConstraints" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="SystemConstraints" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="FunctionalRequirements" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Goals" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Name" type="xs:string" minOccurs="0"/>
         <xs:element name="Description" type="xs:string" minOccurs="0"/>
         <xs:element name="Source" type="xs:string" minOccurs="0"/>
         <xs:element name="History" type="xs:string" minOccurs="0"/>
         <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
         <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="Viewpoints" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Code" type="xs:string"/>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:element name="Source" type="xs:string" minOccurs="0"/>
            <xs:element name="History" type="xs:string" minOccurs="0"/>
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Scenario" minOccurs="0" maxOccurs="unbounded">
             <xs:complexType>
              <xs:sequence>
               <xs:element name="Code" type="xs:string"/>
               <xs:element name="Name" type="xs:string" minOccurs="0"/>
               <xs:element name="Description" type="xs:string" minOccurs="0"/>
               <xs:element name="Source" type="xs:string" minOccurs="0"/>
               <xs:element name="History" type="xs:string" minOccurs="0"/>
               <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
               <xs:element name="ReferenceTo" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="AttributesAndModels" minOccurs="0">
                <xs:complexType>
                 <xs:sequence>
                  <xs:element name="Assumptions" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="AlternativeInstancedModels" minOccurs="0"
maxOccurs="unbounded">
                   <xs:complexType>
                    <xs:sequence>
                     <xs:element name="DataCode" type="xs:string"/>
                     <xs:element name="Name" type="xs:string" minOccurs="0"/>
                     <xs:element name="Description" type="xs:string" minOccurs="0"/>
                     <xs:element name="Source" type="xs:string" minOccurs="0"/>
                     <xs:element name="History" type="xs:string" minOccurs="0"/>
                     <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
                     <xs:element name="RefrenceTo" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                     <xs:element name="UsedIn" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                     <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
                     <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
                     <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                   </xs:complexType>
                  </xs:element>
                 </xs:sequence>
                </xs:complexType>
               </xs:element>
               <xs:element name="BodyOfScenario" minOccurs="0">
                <xs:complexType>
                 <xs:sequence>
                  <xs:element name="SystemBehaviour" type="CodeStructure"/>
                  <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
                  <xs:element name="Others" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
                 </xs:sequence>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="OpenIssues" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="OpenIssues" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="WaitingRoom" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="WaitingRoom" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="Expectedchanges" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Expectedchanges" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/Introduction**



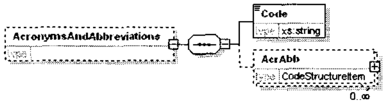
```
<xs:element name="Introduction" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="DocumentPurpose" type="PermissionStructure" minOccurs="0"/>
   <xs:element name="Terminology" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Terms" type="CodeStructureItem" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="AcronymsAndAbbreviations" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="AcrAbb" type="CodeStructureItem" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="References" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Reference" type="CodeStructure" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="DocumentOrganization" type="PermissionStructure" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/Introduction/Terminology**



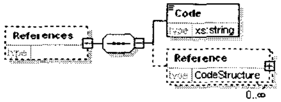
```
<xs:element name="Terminology" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Terms" type="CodeStructureItem" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/Introduction/AcronymsAndAbbreviations**

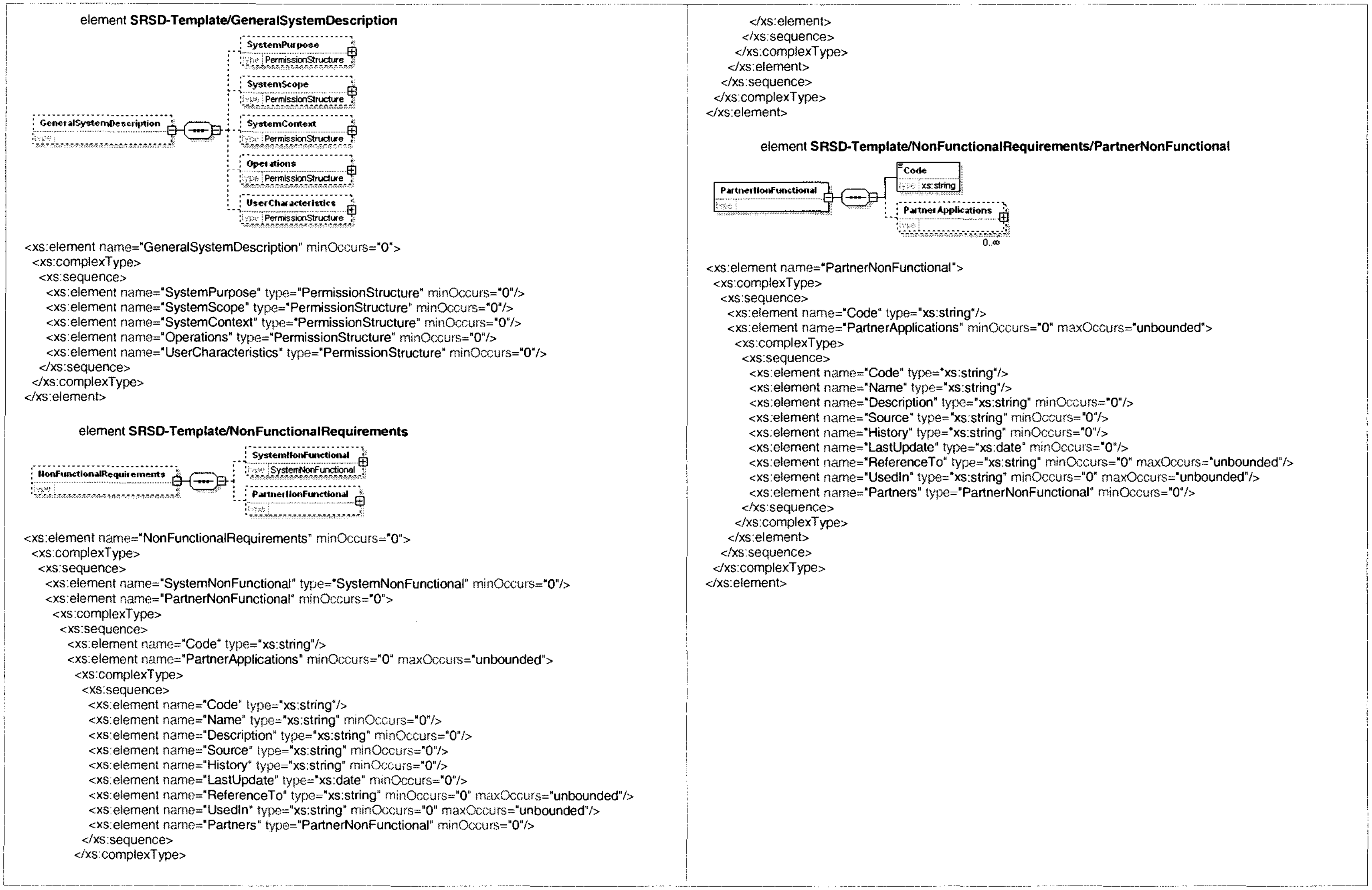


```
<xs:element name="AcronymsAndAbbreviations" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="AcrAbb" type="CodeStructureItem" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/Introduction/References**



```
<xs:element name="References" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Reference" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```



element **SRSD-Template/GeneralSystemDescription**

```
<xs:element name="GeneralSystemDescription" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SystemPurpose" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="SystemScope" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="SystemContext" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="Operations" type="PermissionStructure" minOccurs="0"/>
      <xs:element name="UserCharacteristics" type="PermissionStructure" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

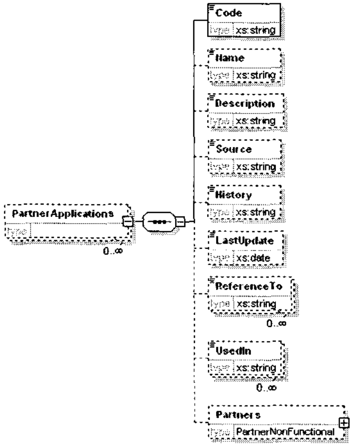
element **SRSD-Template/NonFunctionalRequirements**

```
<xs:element name="NonFunctionalRequirements" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SystemNonFunctional" type="SystemNonFunctional" minOccurs="0"/>
      <xs:element name="PartnerNonFunctional" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Code" type="xs:string"/>
            <xs:element name="PartnerApplications" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Code" type="xs:string"/>
                  <xs:element name="Name" type="xs:string" minOccurs="0"/>
                  <xs:element name="Description" type="xs:string" minOccurs="0"/>
                  <xs:element name="Source" type="xs:string" minOccurs="0"/>
                  <xs:element name="History" type="xs:string" minOccurs="0"/>
                  <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
                  <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Partners" type="PartnerNonFunctional" minOccurs="0"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **SRSD-Template/NonFunctionalRequirements/PartnerNonFunctional**

```
<xs:element name="PartnerNonFunctional">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="PartnerApplications" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Code" type="xs:string"/>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:element name="Source" type="xs:string" minOccurs="0"/>
            <xs:element name="History" type="xs:string" minOccurs="0"/>
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Partners" type="PartnerNonFunctional" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

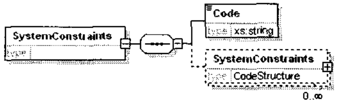
element **SRSD-Template/NonFunctionalRequirements/PartnerNonFunctional**



/PartnerApplications

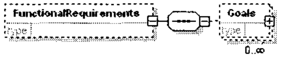
```
<xs:element name="PartnerApplications" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Description" type="xs:string" minOccurs="0"/>
   <xs:element name="Source" type="xs:string" minOccurs="0"/>
   <xs:element name="History" type="xs:string" minOccurs="0"/>
   <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
   <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Partners" type="PartnerNonFunctional" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/SystemConstraints**



```
<xs:element name="SystemConstraints">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="SystemConstraints" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

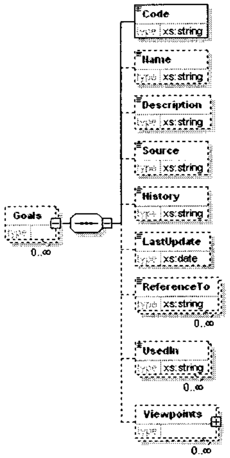
element **SRSD-Template/FunctionalRequirements**



```
<xs:element name="FunctionalRequirements" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Goals" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="Source" type="xs:string" minOccurs="0"/>
      <xs:element name="History" type="xs:string" minOccurs="0"/>
      <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
      <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Viewpoints" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Name" type="xs:string"/>
         <xs:element name="Description" type="xs:string" minOccurs="0"/>
         <xs:element name="Source" type="xs:string" minOccurs="0"/>
         <xs:element name="History" type="xs:string" minOccurs="0"/>
         <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
         <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="Scenario" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Code" type="xs:string"/>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:element name="Source" type="xs:string" minOccurs="0"/>
            <xs:element name="History" type="xs:string" minOccurs="0"/>
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="AttributesAndModels" minOccurs="0">
             <xs:complexType>
              <xs:sequence>
               <xs:element name="Assumptions" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element name="AlternativeInstancedModels" minOccurs="0"
maxOccurs="unbounded">
```

```
                <xs:complexType>
                 <xs:sequence>
                  <xs:element name="DataCode" type="xs:string"/>
                  <xs:element name="Name" type="xs:string" minOccurs="0"/>
                  <xs:element name="Description" type="xs:string" minOccurs="0"/>
                  <xs:element name="Source" type="xs:string" minOccurs="0"/>
                  <xs:element name="History" type="xs:string" minOccurs="0"/>
                  <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
                  <xs:element name="RefrenceTo" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="UsedIn" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
                  <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
                  <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
                 </xs:sequence>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
            <xs:element name="BodyOfScenario" minOccurs="0">
             <xs:complexType>
              <xs:sequence>
               <xs:element name="SystemBehaviour" type="CodeStructure"/>
               <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
               <xs:element name="Others" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

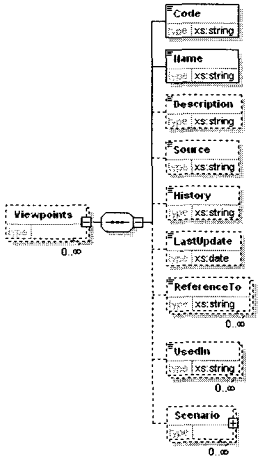
element **SRSD-Template/FunctionalRequirements/Goals**



```
<xs:element name="Goals" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Description" type="xs:string" minOccurs="0"/>
   <xs:element name="Source" type="xs:string" minOccurs="0"/>
   <xs:element name="History" type="xs:string" minOccurs="0"/>
   <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
   <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Viewpoints" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="Source" type="xs:string" minOccurs="0"/>
      <xs:element name="History" type="xs:string" minOccurs="0"/>
      <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
      <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Scenario" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Code" type="xs:string"/>
         <xs:element name="Name" type="xs:string" minOccurs="0"/>
         <xs:element name="Description" type="xs:string" minOccurs="0"/>
```

```
          <xs:element name="Source" type="xs:string" minOccurs="0"/>
          <xs:element name="History" type="xs:string" minOccurs="0"/>
          <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
          <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="AttributesAndModels" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Assumptions" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
             <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
             <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
             <xs:element name="AlternativeInstancedModels" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="DataCode" type="xs:string"/>
                <xs:element name="Name" type="xs:string" minOccurs="0"/>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="Source" type="xs:string" minOccurs="0"/>
                <xs:element name="History" type="xs:string" minOccurs="0"/>
                <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
                <xs:element name="RefrenceTo" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
                <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
                <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="BodyOfScenario" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="SystemBehaviour" type="CodeStructure"/>
             <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
             <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

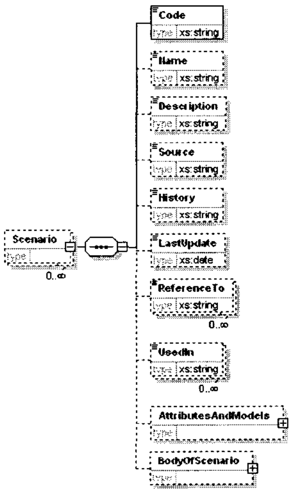
element **SRSD-Template/FunctionalRequirements/Goals/Viewpoints**



```
<xs:element name="Viewpoints" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Name" type="xs:string"/>
   <xs:element name="Description" type="xs:string" minOccurs="0"/>
   <xs:element name="Source" type="xs:string" minOccurs="0"/>
   <xs:element name="History" type="xs:string" minOccurs="0"/>
   <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
   <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Scenario" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:string"/>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="Source" type="xs:string" minOccurs="0"/>
      <xs:element name="History" type="xs:string" minOccurs="0"/>
      <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
      <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="AttributesAndModels" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Assumptions" type="CodeStructure" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
```

```
maxOccurs="unbounded"/>
         <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="AlternativeInstancedModels" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="DataCode" type="xs:string"/>
            <xs:element name="Name" type="xs:string" minOccurs="0"/>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:element name="Source" type="xs:string" minOccurs="0"/>
            <xs:element name="History" type="xs:string" minOccurs="0"/>
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="RefrenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
            <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
            <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems"
minOccurs="0" maxOccurs="unbounded"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="BodyOfScenario" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="SystemBehaviour" type="CodeStructure"/>
         <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
         <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

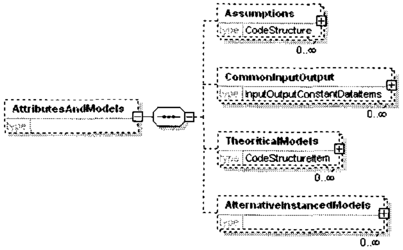
element **SRSD-Template/FunctionalRequirements/Goals/Viewpoints/Scenario**



```
<xs:element name="Scenario" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Description" type="xs:string" minOccurs="0"/>
   <xs:element name="Source" type="xs:string" minOccurs="0"/>
   <xs:element name="History" type="xs:string" minOccurs="0"/>
   <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
   <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="AttributesAndModels" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Assumptions" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="AlternativeInstancedModels" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="DataCode" type="xs:string"/>
         <xs:element name="Name" type="xs:string" minOccurs="0"/>
         <xs:element name="Description" type="xs:string" minOccurs="0"/>
         <xs:element name="Source" type="xs:string" minOccurs="0"/>
         <xs:element name="History" type="xs:string" minOccurs="0"/>
```

```
            <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
            <xs:element name="RefrenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
            <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
            <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
maxOccurs="unbounded"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="BodyOfScenario" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="SystemBehaviour" type="CodeStructure"/>
         <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
         <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
```

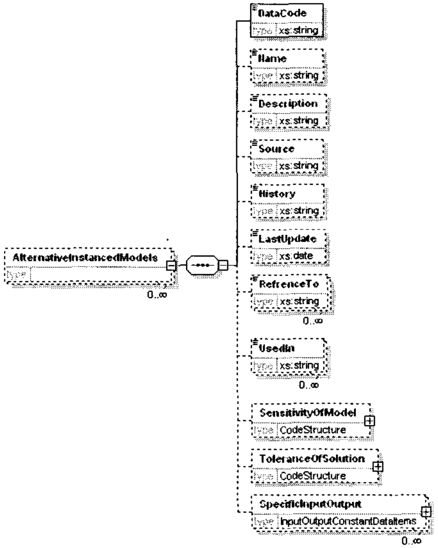
element **SRSD-Template/FunctionalRequirements/Goals/Viewpoints/Scenario /AttributesAndModels**



```
<xs:element name="AttributesAndModels" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Assumptions" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="CommonInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
maxOccurs="unbounded"/>
   <xs:element name="TheoriticalModels" type="CodeStructureItem" minOccurs="0"
maxOccurs="unbounded"/>
   <xs:element name="AlternativeInstancedModels" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="DataCode" type="xs:string"/>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
```

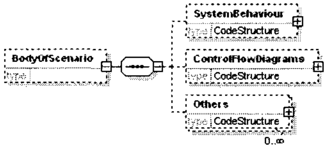
```
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="Source" type="xs:string" minOccurs="0"/>
      <xs:element name="History" type="xs:string" minOccurs="0"/>
      <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
      <xs:element name="RefrenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
      <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
      <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/FunctionalRequirements/Goals/Viewpoints/Scenario /AttributesAndModels/AlternativeInstancedModels**



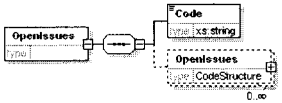
```
<xs:element name="AlternativeInstancedModels" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="DataCode" type="xs:string"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Description" type="xs:string" minOccurs="0"/>
   <xs:element name="Source" type="xs:string" minOccurs="0"/>
   <xs:element name="History" type="xs:string" minOccurs="0"/>
   <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
   <xs:element name="RefrenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="SensitivityOfModel" type="CodeStructure" minOccurs="0"/>
   <xs:element name="ToleranceOfSolution" type="CodeStructure" minOccurs="0"/>
   <xs:element name="SpecificInputOutput" type="InputOutputConstantDataItems" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/FunctionalRequirements/Goals/Viewpoints/Scenario/BodyOfScenario**



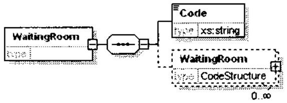
```
<xs:element name="BodyOfScenario" minOccurs="0">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="SystemBehaviour" type="CodeStructure" minOccurs="0"
/>
   <xs:element name="ControlFlowDiagrams" type="CodeStructure" minOccurs="0"/>
   <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/OpenIssues**



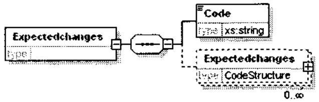
```
<xs:element name="OpenIssues">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="OpenIssues" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/WaitingRoom**



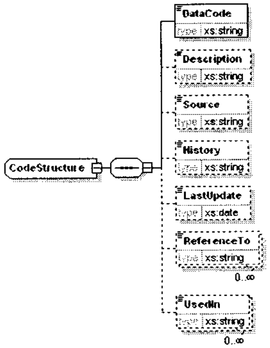
```
<xs:element name="WaitingRoom">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="WaitingRoom" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **SRSD-Template/Expectedchanges**



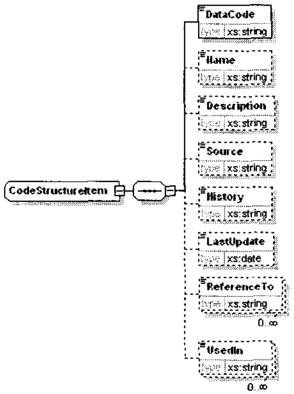
```
<xs:element name="Expectedchanges">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Code" type="xs:string"/>
   <xs:element name="Expectedchanges" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

complexType **CodeStructure**



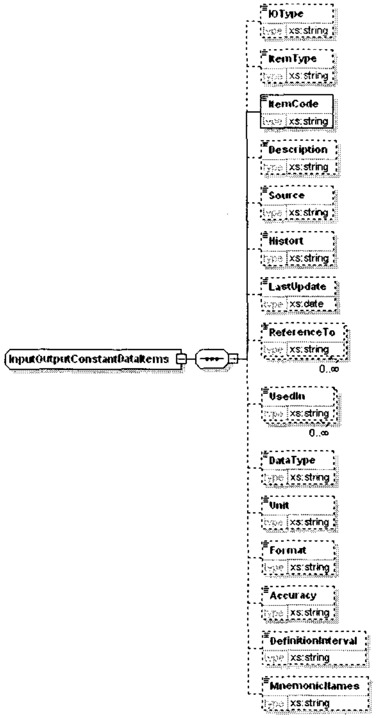
```
<xs:complexType name="CodeStructure">
 <xs:sequence>
  <xs:element name="DataCode" type="xs:string"/>
  <xs:element name="Description" type="xs:string" minOccurs="0"/>
  <xs:element name="Source" type="xs:string" minOccurs="0"/>
  <xs:element name="History" type="xs:string" minOccurs="0"/>
  <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
  <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence></xs:complexType>
```

complexType **CodeStructureItem**



```
<xs:complexType name="CodeStructureItem">
 <xs:sequence>
  <xs:element name="DataCode" type="xs:string"/>
  <xs:element name="Name" type="xs:string" minOccurs="0"/>
  <xs:element name="Description" type="xs:string" minOccurs="0"/>
  <xs:element name="Source" type="xs:string" minOccurs="0"/>
  <xs:element name="History" type="xs:string" minOccurs="0"/>
  <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
  <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

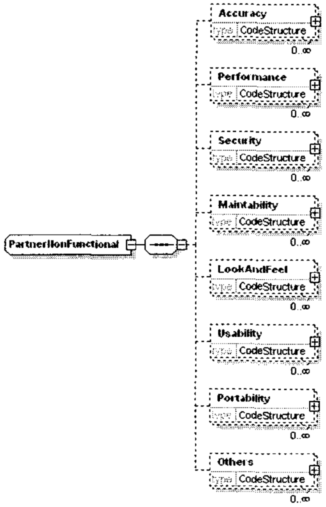
complexType **InputOutputConstantDataItems**



```
<xs:complexType name="InputOutputConstantDataItems">
 <xs:sequence>
  <xs:element name="IOType" minOccurs="0">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="Input"/>
     <xs:enumeration value="Output"/>
     <xs:enumeration value="Input/Output"/>
     <xs:enumeration value="Constant"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:element>
  <xs:element name="ItemType" minOccurs="0">
   <xs:simpleType>
```

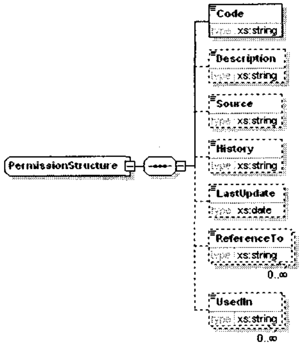
```
        <xs:restriction base="xs:string">
         <xs:enumeration value="Numerical"/>
         <xs:enumeration value="NonNumerical"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ItemCode" type="xs:string"/>
    <xs:element name="Description" type="xs:string" minOccurs="0"/>
    <xs:element name="Source" type="xs:string" minOccurs="0"/>
    <xs:element name="Histort" type="xs:string" minOccurs="0"/>
    <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
    <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="DataType" type="xs:string" minOccurs="0"/>
    <xs:element name="Unit" type="xs:string" minOccurs="0"/>
    <xs:element name="Format" type="xs:string" minOccurs="0"/>
    <xs:element name="Accuracy" type="xs:string" minOccurs="0"/>
    <xs:element name="DefinitionInterval" type="xs:string" minOccurs="0"/>
    <xs:element name="MnemonicNames" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

complexType **PartnerNonFunctional**



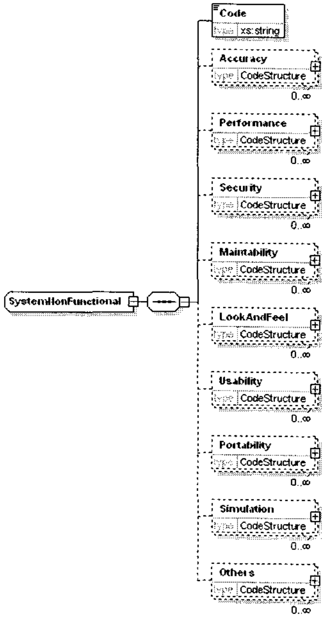
```
<xs:complexType name="PartnerNonFunctional">
 <xs:sequence>
  <xs:element name="Accuracy" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Performance" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Security" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Maintability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="LookAndFeel" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Usability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Portability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence></xs:complexType>
```

complexType **PermissionStructure**



```
<xs:complexType name="PermissionStructure">
 <xs:sequence>
  <xs:element name="Code" type="xs:string"/>
  <xs:element name="Description" type="xs:string" minOccurs="0"/>
  <xs:element name="Source" type="xs:string" minOccurs="0"/>
  <xs:element name="History" type="xs:string" minOccurs="0"/>
  <xs:element name="LastUpdate" type="xs:date" minOccurs="0"/>
  <xs:element name="ReferenceTo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="UsedIn" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

complexType **SystemNonFunctional**



```
<xs:complexType name="SystemNonFunctional">
 <xs:sequence>
  <xs:element name="Code" type="xs:string"/>
  <xs:element name="Accuracy" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Performance" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Security" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Maintability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="LookAndFeel" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Usability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Portability" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Simulation" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="Others" type="CodeStructure" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

7426_60