An executable meta-model for safety oriented software and systems development processes within the avionics domain in compliance with RTCA DO 178 B

Karthik Raja Pitchai
Master Student-Intelligent Embedded system
kpi10001@student.mdh.se

Internal Supervisor
Barbara Gallina
Post-Doc Researcher, Mälardalens Högskola
Barbara.gallina@mdh.se

External supervisor
Ross W Tsagalidis
Swedish Armed forces
wross@tele2.se

Examiner
Kristina Lundqvist
Professor, Mälardalens Högskola
kristina.lundqvist@mdh.se
Abstract:

"There are two critical points in every aerial flight—it’s beginning and its end." — Alexander Graham Bell, 1906. From beginning till the end, the safety critical software plays a vital role in avionics and hence its development and its certification are indispensable. “RTCA DO-178B- Software Considerations in Airborne Systems and Equipment Certification” provides the normative guidelines to develop such systems. In particular, this standard provides the safety protocol and processes that should be followed to achieve safe systems. The safety guideline of DO178B emphasizes more on better documentation, communication and visibility into actual process.

For realizing the guidelines of DO178B, a well-defined and collectively accepted (at least at the development team-level) interpretation of the protocol and processes is needed. To achieve such interpretation, a well-defined modeling language that models the process with safety construct is essential. The Object Management Group’s Software and System Process Engineering Metamodel SPEM 2.0 standard provides specification for modeling software and systems development processes. SPEM2.0, however, is a general purpose language and does not provide sufficient coverage in terms of language constructs to address safety concerns.

This thesis proposes S-SPEM, an extension of the SPEM2.0 to allow users to specify safety-oriented processes for the development of safety critical systems in the context of RTCA DO 178B. The DO178B is analyzed to capture the safety related process elements and SPEM 2.0 is extended to include those safety concepts. Moreover, to simulate and validate the modeled processes, S-SPEM concepts are mapped onto XML Process Definition Language (XPDL) concepts and a transformation algorithm is sketched. Finally, a case-study will illustrate the usage and effectiveness of the proposed extension.
Acknowledgements

Foremost, I would like to express my sincere gratitude to my internal supervisor Barbara Gallina for her continuous support throughout my Master thesis, for her patience, motivation, enthusiasm and immense knowledge. I could not have imagined having a better supervisor and mentor for my Master thesis.

I would like to thank my examiner Prof. Kristina Lundqvist for her excellent lecture on safety critical systems and safety standards that enlightened me to propose my master thesis on the same field.

I am most grateful to Lt.Col Arne Norlander and Ross W Tsagalidis, for accepting my thesis proposal and offering me to do my thesis work for Swedish Armed Forces.

Special thanks to my friends since childhood, Slt.Aditya, Dr.Ambika, Balaji, Deepak, Dinesh, Divya, Ej, Dr.John, Kss, Dr.Namachivayam, Naga, Rahul and Rameez for their love and immense support.

Above all, I would like to thank my parents and my sisters. My sisters Priya and Saranya have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.
# Table of Contents

1. INTRODUCTION 9

1.1 CONTEXT AND MOTIVATION 9
1.2 CONTRIBUTION 10
1.3 DOCUMENT ORGANIZATION 10

2. BACKGROUND AND RELATED WORK 12

2.1 SAFETY CRITICAL SOFTWARE SYSTEMS 12
2.2 SOFTWARE DEVELOPMENT PROCESS 13
2.2.1 SOFTWARE DEVELOPMENT PROCESS MODELS 13
2.2.2 PARADIGMS OF SOFTWARE PROCESS MODELS 14
2.3 SOFTWARE PROCESS MODELING 16
2.3.1 BASIC PROCESS MODELING ELEMENTS 16
2.3.2 CHARACTERISTICS OF PROCESS MODELING LANGUAGE 17
2.3.3 META-MODELING 17
2.3.4 MODEL TRANSFORMATION 17
2.4 SOFTWARE AND SYSTEM PROCESS ENGINEERING META-MODEL (SPEM 2.0) 18
2.4.1 MANAGED CONTENT 20
2.4.2 METHOD CONTENT AND PROCESS 21
2.4.3 BASIC MODELING ELEMENTS SPEM 21
2.4.4 CUT OF SPEM 2.0 METAMODEL 25
2.4.5 TOOLS SUPPORTING SPEM 2.0 26
2.5 XPDL- XML PROCESS DEFINITION LANGUAGE 27
2.5.1 XPDL OVERVIEW 27
2.5.2 BASIC PROCESS ELEMENTS 27
2.6 RTCA DO-178B 29
2.6.1 SOFTWARE LIFECYCLE PROCESSES- DO178 B 30
2.6.2 OVERVIEW OF PROCESSES 30
2.7 RELATED WORK 33

3. PROBLEM FORMULATION AND ANALYSIS 35

3.1 PROBLEM FORMULATION 35
3.2 PROBLEM ANALYSIS 35
Index of Figures

Figure 1: Waterfall model
Figure 2: V-model taken from [57]
Figure 3: Layered architecture of metamodel adapted from [1]
Figure 4: Layered architecture of metamodel adapted from [1]
Figure 5: Cut of Managed Content Package adapted from [2]
Figure 6: SPEM 2.0-Method Framework taken from [2]
Figure 7: Dependency of Role, Task and Work-product
Figure 8: Guidance in Process Models
Figure 9: Phases in Software Development
Figure 10: Types of Guidance adapted from [2]
Figure 11: Dependency of RoleUse, TaskUse and Work-productUse
Figure 12: Cut of SPEM 2.0 Metamodel [2]
Figure 13: XPDL Process Definition Meta-Model [3]
Figure 14: Overview of DO178B Processes.
Figure 15: Overview of DO178B Phases
Figure 16: Multiple Level Requirement adapted from [4]
Figure 17: Activity Diagram for Extending SPEM 2.0
Figure 18: Activity diagram to validate S-SPEM
Figure 19: Implementation of Model transformation
Figure 20: Software Coding Process - DO178B
Figure 21: Software Development Process - DO178B
Figure 22: Software Planning Process - DO178B
Figure 23: Software Quality Assurance Process - DO178B
Figure 24: Extended SPEM (S-SPEM Metamodel)
Figure 25: Elements of Safety Activity
Figure 26: Elements of Safety Work-Product
Figure 27: S-SPEM Extension (guidance)
Figure 28: Safety Guidance
Figure 29: Transformation of S-SPEM to XPDL
Figure 30: Sub-flow implementation type of XPDL taken from [3]
Figure 31: Quality Assurance Process model in S-SPEM
Figure 32: Quality Assurance Process model in XPDL workflow
Figure 33: Development Process in S-SPEM
Figure 34: Development Process in XPDL workflow
Index of Tables

Table 1: Types of Work product
Table 2: Failure conditions and its corresponding software levels
Table 3: Approach for extending SPEM 2.0
Table 4: Approach for validating S-SPEM
Table 5: Mapping of Roles to its corresponding XPDL element
Table 6: Mapping of Activity element to its corresponding XPDL modeling element
Table 7: Mapping of Safety Activity to its corresponding XPDL modeling element
Table 8: Mapping of Artifact to its corresponding XPDL modeling element
## Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>Configuration Management Plan</td>
</tr>
<tr>
<td>DAL</td>
<td>Design Assurance Level</td>
</tr>
<tr>
<td>EPF</td>
<td>Eclipse Process Framework</td>
</tr>
<tr>
<td>EUROCAE</td>
<td>European Organization for Civil Aviation Equipment</td>
</tr>
<tr>
<td>IPA</td>
<td>Iris Process Author</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>PML</td>
<td>Process Modeling Language</td>
</tr>
<tr>
<td>PSAC</td>
<td>Plan for Software Aspects of Certification</td>
</tr>
<tr>
<td>RMC</td>
<td>Rational Method Composer</td>
</tr>
<tr>
<td>RTCA</td>
<td>Radio Technical Commission For Avionics</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>SAS</td>
<td>Software Accomplishment Summary</td>
</tr>
<tr>
<td>SCI</td>
<td>Software Configuration Index</td>
</tr>
<tr>
<td>SCR</td>
<td>Software Conformity Review</td>
</tr>
<tr>
<td>SDP</td>
<td>Software Development Plan</td>
</tr>
<tr>
<td>SECI</td>
<td>Software Life Cycle Environment Configuration Index</td>
</tr>
<tr>
<td>SPEM</td>
<td>Software and Systems Process Engineering Meta Model</td>
</tr>
<tr>
<td>SQAR</td>
<td>Quality Assurance Records</td>
</tr>
<tr>
<td>SVP</td>
<td>Software Verification Plan</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>XPDL</td>
<td>XML Process Definition Language</td>
</tr>
</tbody>
</table>
1. Introduction

This chapter is organized as follows: context and motivation of the thesis are introduced in Section 1.1, the contribution of the thesis is explained in Section 1.2 and finally, the organization of this thesis report is explained in Section 1.3.

1.1 Context and motivation

In the context of safety critical systems, software development process plays more significant role. There are three crucial aspects for developing safety oriented software. First aspect is process engineering and management. Secondly, choosing an appropriate tool and environment for software development and thirdly, addressing any legal and regulatory requirements [5].

To develop safety oriented software, the need for safety concerns (in terms of e.g. additional process steps and or work-products) must be conveyed properly to software engineers by safety engineers. In most cases, the software engineers are not safety engineers and they might misinterpret the safety concerns. Misinterpretation in safety needs may lead to potential software failure which in turn leads to catastrophic system failure (for instance, in the avionics domain, the system failures may lead to damage of the aircraft and occupants). To avoid this issue, a proper process model that has more emphasizes on safety should be modeled, so that the process models highlights the visibility of the safety concepts. Such a process model would also help the organizations to make the documentation easier and clear, that can be easily assessed by certification authorities to certify the intended software. The process models also aims at documenting the development practices that are empirically known to have an impact on software development time, cost and quality. The vital purpose of the process models is to document and communicate processes and to enhance the reuse of processes. Some companies are familiar with traditional development processes (RUP, agile practices) and they follow the traditional development processes whereas some companies often do not follow a well defined development processes. However, for certification purposes, companies must provide evidence, process based arguments that the work products have been obtained by adopting the mandatory standard (DO-178B, IEC61508.. etc) and hence having at disposal, a modeling language that has capabilities to document as well as simulate safety processes emphasized by a specific safety standard is fundamental to obtain a safety critical system. Furthermore, the execution of process models is also essential to simulate and validate the modeled processes.

Currently there are no satisfying modeling languages that provide a clear visibility of safety concern is available. Software and Systems Process Engineering Metamodel (SPEM 2.0) is designed to model software development process. Since it is a general language, it is not designed to narrow down its use, so it lacks some essential elements needed to model few specific areas that demands safety. But SPEM 2.0 provides flexibility to extend its metamodel to meet the specific need (like safety, quality et.al) and is widely used for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes. Workflow is the automation of business process during which documents, instructions and the tasks are transferred from one participant to another for actions according to a set of procedural rules [6]. To support the SPEM process model in workflow paradigm, the process model of SPEM has to be transformed into another modeling language that supports interoperability. XPDL is one such standard process definition interface that separates the workflow definition from execution. That is, if the model is XPDL confirmed, it can be enacted in any XPDL-
compliant engine. The SPEM can be made more expressive with its capability in providing metamodel extension, while the XPDL is strong and capable of execution mechanism, the transformation of SPEM process models into XPDL workflow model will deliver an executable modeling language with the benefits of both SPEM and XPDL.

1.2 Contribution
To assure and develop safety oriented software, the safety concerns emphasized by a safety standard (RTCA DO 178B) are determined and the SPEM 2.0 is extended to accommodate the safety concerns, thus developing an adequate modeling language (S-SPEM) for modeling safety oriented software development process. This thesis also shows the modeling of safety capabilities in the proposed S-SPEM with a set of models that embodies the essential safety construct. The safety constructs are expressed with clear visual notations to provide expressiveness in the process model.

The need for providing a validation aspect to the proposed metamodel drives the thesis to provide a transformation between the S-SPEM process models to XPDL models. This thesis tabulates the transformation mapping rules between S-SPEM elements and the XPDL elements. The transformation rules and the algorithm for an effective transformation are explained briefly in this thesis. The S-SPEM process models are transformed into XPDL models because the XPDL is a standardized language that are employed at workflow engine level in order to execute and animate processes to be able to validate them. This thesis motivates the presence of proposed safety constructs in an end-to-end software development process and its validation capabilities through the application of proposed solutions.

1.3 Document Organization
A skeleton and a clear inventory of the remaining chapters are given in this subsection,

Chapter 2 is intended to present the current state of art and it introduces the background that is necessary for understanding the basic concepts of software development process, SPEM 2.0, XPDL and RTCA DO-178B for building the solution in the remaining chapters.

Chapter 3 formulates the problem addressed in this thesis and provides an extensive analysis by decomposing the problems into sub problems.

Chapter 4 explains the approach followed in this thesis to achieve the desired goal. The step by step method for the extension of SPEM 2.0 metamodel and the validation of S-SPEM are explained briefly in this chapter.

Chapter 5 elucidates the missing safety constructs to model a safety oriented software development process and proposes a safety oriented SPEM 2.0 metamodel (S-SPEM) that accommodates the safety constructs.

Chapter 6 contains the transformation of S-SPEM process models into XPDL models. This chapter also explains the description of entity mapping that is intended to transform SPEM process models to XPDL models. The algorithm for the transformation is also briefly explained in this chapter.
Chapter 7 explains the case study that portrays the S-SPEM models and its corresponding XPDL model obtained after transformation. A brief explanation to the process models are also embodied in this chapter.

Chapter 8 concludes the thesis work by manifesting the benefits as well as the limitations of this thesis work. The summary of this thesis work and potential leads for future work is briefly explained in this chapter.
2. Background and related work

This chapter provides the background information regarding the problem space and the solution space of this thesis work. More specifically, Section 2.1 explains the importance of safety assessment in safety critical software systems, and the consecutive sections introduce the software development process, modeling of software development process, XML Process Definition Language and RTCA DO178B.

2.1 Safety Critical Software Systems

Safety critical software systems are those systems whose failures could result in catastrophic events (e.g. loss of life, loss of system or damages the environment adversely). The software of a system is deemed to be safe if it never or highly unlikely produces an output that can lead to a catastrophic event. These safety oriented software is the backbone for safety critical systems [7]. Knight refers the safety oriented software as a backbone of a safety critical system because it is a part of the system safety functions that ensures that a system does not endanger human life, economics or the environment or the control of the system. The Software safety in a safety critical system exist when [8],

- Software does not fail causing or contributing a hazardous state to a system,
- Software does not fail in identifying and rectifying the system that has already reached a hazardous state.
- Software does not fail to lessen the damage during an occurrence of an accident.

The notion of safety oriented software was initially acknowledged by Mil-Std 1574A (1979) [9](Mil-STD-1574A-USAF, 1979). The Handbook of military standards issued by Department of Defense-United states of America emphasizes on eliminating the probability of software failure and developing software that is deemed to be safe when employed on a system. Since then the safety critical software plays a vital role in many fields of defense. Avionics is one among the key areas in defense where the safety critical software is expected to be safe.

Safety Assessment

Safety assessment is an important step while developing or modifying safety critical software systems. Safety assessment is similar to risk assessment (defined as carrying out an investigation in order to arrive at a judgment, based on evidence of the suitability of the product) and follows the same procedures to determine the safety concerns [11]. These safety concerns should be considered with high priority while implementing or modifying safety critical software.

In avionics domain, the guidelines for performing a safety assessment are outlined in the documents released by Society of Automotive Engineers (SAE) as Aerospace Recommended Practice (ARP) documents. The safety assessment emphasized in the standard DO178B specifies a set of Design Assurance Levels (DAL). The safety assessment process or team ensures that the architectural implementation meets the DAL specified by the safety assessment and that no assumption of the safety assessment has been violated by the software architecture. A constant feedback is maintained
between the software development process and safety assessment process to ensure the final product meets the DAL as well as the airworthiness requirements to obtain an approval for employing it the aircraft. The crucial aspects of developing safety oriented software includes, process modeling that has a clear visibility of safety concerns, selection of tool support to develop the software and addressing the legal and regulatory concerns to get certification. The following section discusses the initial aspect of developing the safety oriented software (i.e. software development process).

2.2 Software Development Process

A software development process is a set of partially ordered activities to be performed to transform user’s requirement into software solution. Activities can be grouped into phases. The main phases involved in the software development process include: requirement analysis, design, implementation, verification and maintenance.

Leon Osterweil states that “Software processes are software too”[12]. In his invited talk at ICSE 87, he explains his quote that there are many similarities exist between software processes and the software in terms of addressing the requirements, executability and reusability [13]. Furthermore he elucidates a software process as a vehicle for doing a job and software description is a specification of a how a job is done. A software process allows the project team to predict the output and allows continuous improvement thereby building confidence to achieve the desired output.

2.2.1 Software development process models

The software development process model is an abstract representation of a software process. Software process models are general approaches for organizing a project into activities thereby providing clarity. The main aim of software process model is to provide an unambiguous illustration of the static and dynamic structure of a software process. The success of a good software development process models emerges from the utilization of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing. The software process models embodies the following characteristics are considered to be better and

A software process models helps the project team in answering the following questions,

- What work should be done?
- Who should do the work?
- What sequence should the work be performed?

Hence an effective process models must,

- Represent the way the work is actually (or is to be) performed.
- Be easily understood and flexible
- Have capability to zoom to whatever level of detail is needed.

Finally, the process models must aim at supporting comprehensive analysis of the process through the model and should provide space for prediction to be made regarding the consequences of potential changes and improvements.
2.2.2 Paradigms of Software process models

There are several existing process models to streamline the software development process. Each process model has its own pros and cons. This section recalls a couple of prominent software process models.

Waterfall Model

The waterfall model is the first published process model that follows a sequential process modeling defined by Royce [14]. The processes in this model progress steadily downward like a waterfall and hence the name waterfall process. Figure (1) illustrates the phases in the waterfall process. The inception phase clarifies the scope, process objectives and feasibility of the intended solution whereas the planning phase deals with the ways to develop software. The requirement phase delivers a complete description of the behavior of software systems to be developed and the design phase plans for attaining the software solution by fulfilling the requirements. Note that the waterfall model start with the requirement phase, where the inception and planning phases are not included in the process model. The implementation phase includes the main coding process and integration of sub systems. The testing of the system is carried out in the verification phase and the modification of software after delivery to correct faults and improving the efficiency are carried out in maintenance phase. There are various modified waterfall model (including the Royce’s final model) that includes slight changes in the process but the main essence of the model remains the same.

Figure 1 Waterfall model

The waterfall is characterized by the fact that the output of one phase is used as an input to the next phase, implying that the second phase is not started until the former phase completes. The main advantages of the waterfall model is that it is simple and easy to use because the orderly execution of
phase is easy to understand. It works well with smaller projects, where the requirements are easily determined.

The major drawback of the waterfall model is the lack of parallelism where each phase is completed to execution before the start of next phase that creates a larger verification phase. In practice, deploying such a model in the complex project results in potentially disastrous effects on projects because most of the problems are detected in the testing phase, but has very little time for correction. The waterfall method does not provide intermediate versions and hence chances are high for the defects to be detected by the end-users after the release.

**V-Model**

The V-model is a further development of the waterfall model. The V-model was developed to overcome the issues faced in the waterfall model and hence it is considered to be an extension of it. The individual process in the V-model is almost similar to waterfall model but the process stops its linear flow at implementation phase (coding) and bent upwards forming a V shape. The reason for V shape is that, in each of the requirement and design phase, there exist counterparts of verification and validation phases that correlate to each other respectively. Figure (2) depicts the V-model.

![V-model diagram](attachment:V-model.png)

**Figure 2 V-model taken from [57]**

The main advantages of V model is that it clearly defines who is responsible for carrying out testing at each stage like acceptance testing is carried out by users, system testing is performed by system tester, unit testing is performed by programmers and integration testing is performed by team leaders. The V model mainly focuses on delivery of reliable software that fulfills the defined requirements.

The disadvantage of V model is that it is least flexible, any changes in the requirement plan demands changes in test documentation that should be updated. Even with this disadvantage, it is considered to be the most favored development model because it is simple and easy to use.
2.3 Software Process Modeling

Software process modeling is a research area aimed at defining and analyzing the significant aspects of software process in order to provide modeling elements to describe them. The main objective of process modeling is to support process modelers with modeling constructs allowing them to model the decomposition of complex software development processes into sufficient manageable activities to provide an explicit guidance and control over the process.

A process model is the description of a process represented in a suitable process modeling language. According to [15] The main objective of process model is to control the information flow and maintain the activities, and have the ability for accommodating and adapting new activities.

2.3.1 Basic process modeling elements

The main concepts related to development processes are; process, phase, activity, roles, work-product and guidance. The following list recalls their definition:

- A **process** is an activity that explains the structure for particular type of development project. A process also elucidates the way to get from one milestone (a significant activity or an event or a stage in development) to the other by defining the sequences of work, operations, or events that usually take up time.

- A significant segment of a complete software life cycle is expressed as a **phase**. The definition of a phase usually includes pre and post conditions, work products and milestones.

- An **activity** is a unit of work that is employed to produce a work-product. The activities are performed by **actors/agents** with the use of resources to obtain a desired work-product.

- A **role** describes a responsibility, rights and skills related to the actors or a group of actors or agents to perform a particular task.

- A **work-product** denotes the tangible and intangible output of a task carried out by role using the available resources. The products are usually **artifacts** that are produced or consumed by executing an activity or task by roles with the help of resources.

- A **guidance** elucidates the descriptions for effectively carrying out activities to achieve a desired product.
2.3.2 Characteristics of Process Modeling Language

Bill Curtis explains the specific goals and benefits of modeling the software process. It includes easy understanding and effective communication, process management support and control, provision for automated orientation for process performance, process improvement support [16]. The process models are brief descriptions of actual process and process modeling languages are employed to achieve a better process model. The Process Modeling Languages (PML) should satisfy the following criteria in order to design a precise and effective process model [17].

The PMLs are considered effective if they exhibit the following essential characteristics.

**Formality:** The syntax and semantics of a PML should be defined precisely and intuitively. The formality of PML should support better communication between various process participants that results in clear understanding by all the users. The process model is said to be formally defined if the PML satisfies this requirement.

**Understandability:** The model should be understandable for all the users. The users with compatible technical skills may find easy to understand the model that resembles like a programming language and those with other backgrounds may find easy to understand if the model is portrayed in graphical representation. Hence PML should be able to design a model that can be understood by all the users.

**Expressiveness:** The ability of the PML to specify and model all aspects of software process with existing features of PML. The expressiveness of the PML is considered effective if the process models are modeled with the default modeling elements of the PML without using any additional comments.

**Modularization:** The ability of PML to support the breakdown of large processes models into well defined sub-modules is considered to possess modularization. It refers to the logical partitioning of the complex software process to be manageable for the purpose of implementation and maintenance.

**Executability:** The ability of PML in defining the operational models that is easily executable and enactable. The major advantage of executable model is that the process models can be checked and validated once the model is designed.

2.3.3 Meta-modeling

Meta-modeling is a process of modeling syntax and semantics of formalism in an expressive way [18]. Meta-model is a set of rules, frames and constraints that are illustrated into a model for useful modeling of a process. The model always conforms its unique meta-model. The meta-modeling can be defined as an analysis, construction and development of a meta-model with the rules, frames and constraints, models and theories. Generically, for the definition of new languages, the OMG prescribes an layered architecture based on four levels of abstraction that allow to distinguish the set of four concept levels that take part in modeling of a system. The concept levels include M0, M1, M2 and M3. Figure (3) depicts the layered architecture of metamodel adapted from [1]. The level M0 is the lowest layer and it indicates the System. The system is represented by a Model and it belongs to level
The models confirm to its *Metamodel* which is in level M2. The metamodel confirms it’s *Meta Metamodel* that tops the highest level M3.

The state-of-art in the area of domain specific modeling is based on a fixed meta-model. To overcome the complexity of process modeling, the environments providing flexible meta-models are considered to be more helpful (for e.g SPEM 2.0). The benefit of such a flexible meta-model is that the formalism of modeling can be freely defined and therefore the problem under consideration can be adapted in an effective way [19].

![Layered architecture of metamodel taken from[1]](image)

### 2.3.4 Model Transformation

Model-driven engineering (MDE) is a software development methodology that relies on models as first class entities and it aims to develop, maintain and evolve software by performing model transformations [20]. In MDE, the transformation is a process that converts the source models into target models by means of transformation rules. The aim model transformation is to save effort and minimize errors by automating the building and modification of models. The model transformation is carried out based on many approaches. Few approaches in model transformation are explained as follows.

*Unidirectional and bidirectional*

A unidirectional model transformation has one mode of execution (i.e.) the input and output of the unidirectional model transformation doesn’t change whereas in bidirectional model transformation, the input and output of model transformation changes.

*Technical space*

The distinction in model transformation can be distinguished with respect to the technical space. A technical space is a model management framework that contains concepts, tools, mechanisms, languages and formalism associated to particular technology [20]. The distinction can be that the source and target model belong to same technical space (e.g. Transformation in-between UML...
diagrams) or the source and target model belong to different technical space (e.g. transformation of XML documents into UML diagrams)

**Endogenous and exogenous transformation**

Both source and target model need to be expressed in some modeling language. Based on the language in which the source and target model are expressed, model transformation can be distinguished between endogenous and exogenous transformation. The transformation between the models of same language is called endogenous transformation and the transformation between models expressed in different languages is called as exogenous transformation. Exogenous transformations are also referred as translation whereas endogenous transformations are also referred as rephrasing [20].

**Horizontal and vertical transformation**

The abstraction level (ability to reflect the same concept) of source and target model determines the transformation type. In horizontal transformation, the source and target model reside at same abstraction level, whereas in vertical transformation, the source and target model reside at different abstraction level. The semantics of the source and target models corresponds in horizontal transformation whereas the model is transformed into a model of different abstraction in vertical transformation [21].

**Number of source and target model**

Depending upon the number of source and target model, the model transformation is distinguished. Some model transformation has one input and many output model (e.g. platform independent model (PIM) transformed into a number of platform-specific model (PSM). Some model transformation has many input models that are merged or combined to form a single target model (e.g. merged model)

### 2.4 Software and System Process Engineering Meta-Model (SPEM 2.0)

Software Process Engineering Meta-model (SPEM) 2.0 is a specification developed by Object Management Group OMG for defining systems and software processes[2]. SPEM 2.0 facilitates a conceptual platform for process engineers and project managers for selecting, tailoring and rapidly assembling processes for their tangible development projects [22]. SPEM 2.0 defines a language for how to describe a software processes and it does not recommends any particular process of method for designing and assembling the model. It also does not provide any guidance on model organization and tool support. SPEM lacks in providing significant depth to define precise process models and it is structured in this way intentionally for augmenting the lacking process elements to develop a domain specific meta-model with desired process elements (i.e. SPEM2.0 metamodel can be exploited to meet the domain specific modeling needs).

In context to Section 2.3.3 on meta-modeling, the level of SPEM metamodel in layered architecture is illustrated in the Figure (4). The level M0 is the lowest layer and it indicates the Process. The process is represented by a Model and it belongs to level M1. The models confirm to its Metamodel (SPEM 2.0) which is in level M2. The metamodel confirms it’s MOF that tops the highest level M3. The generic meta-model SPEM works as a template for developing models of concrete processes, such as V-Model. Therefore, SPEM is considered to be meta-model of level M2 [23].
The SPEM 2.0 metamodel is structured in seven main packages, only three of them will be elucidated in this thesis in order to justify the solution chapters. The package elements process structure and method content are explained Section 2.4.2. The package managed content is explained in the following chapter.

### 2.4.1 Managed Content

The managed content package defines the fundamental concepts for managing textual of processes and method content elements like role, workproduct, category and task (explained in section 2.4.2). Figure (5) provides an overview of the managed content package. The abstract class Describable element is an extensible element that has a metaclass classifier is introduced in this package. The describable stored the content description and stores the actual textual description. The category is a describable element that is used to group any number describable element based on user defined criteria. Guidance is a describable element that provides additional information related to describable element. This guidance element is extended in solution chapter (see chapter 5).
2.4.2 Method content and Process

The method content actually explains the questions like who, what, why and how to do a process whereas the process explains when to execute a process. It includes highly re-usable information. The method content defines the roles, tasks, work product and associated relationships. It also includes guidance and categories but does not contain any timing information. The method content element actually adds concepts of defining lifecycle and process independent reusable method content elements that provide a base of documented knowledge of software development method.

The method content element comprises of all elements that are used to model a static process model and all the elements embodied in the method content are re-useable. The dynamic process models are modeled using the MethodcontentUse that has all the elements which are not reusable. Figure (6) depicts the method content and process in SPEM 2.0. The process elucidates the modeling elements for modeling dynamic processes whereas the method content element depicts the elements for modeling static process models.

![Figure 6 SPEM 2.0- Method Framework taken from [2]](image)

The process explains the End-End sequence of phases, iterations, activities and milestone that define the development lifecycle. It also defines when the tasks are preformed via activity diagrams and/or work breakdown structures [22].

2.4.3. Basic Modeling ElementS-SPEM

The basic modeling elements Role, Work product, Task and Guidance are used for modeling a process statically, whereas the SPEM 2.0 also provides opportunity for modeling the dynamics of process by providing modeling elements like TaskUse, RoleUse, WorkproductUse.

The modeling elements are recalled as follows

**Role**

The role defines the related skills, competencies and responsibilities required to perform the task. It defines the role of one or many people involved in the project and it does not denote the individual (For e.g. Software Architect, Tester, and Developer). Roles work on the task or activity to produce the
work product. The role is responsible for producing or modifying the one or more work product. The figure (7) explains how a Role works.

![Figure 7 Dependency of Role, Task and Work-product](image)

**Work Product**

The work product usually denotes the tangible things that are used and produced in the software process. The work products are modified or produced by the task. The role exploits the work product to perform the task and thereby producing another work product while performing task. The three work products are Artifact, Deliverable, and Outcome. Figure (7) explains how the work product is produced and modified. Table 1 explains the three types of work-products.

<table>
<thead>
<tr>
<th>Work product</th>
<th>Description</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact</td>
<td>The artifacts are the tangible work product. An artifact can be composed of other artifacts. E.g. model artifact is made of model elements, which are also artifacts.</td>
<td>![Icon]</td>
</tr>
<tr>
<td>Deliverable</td>
<td>A package of other work product that are either given to internal party of external party. E.g. Stakeholder, customer deliverable</td>
<td>![Icon]</td>
</tr>
<tr>
<td>Outcome</td>
<td>Intangible work product that are usually a state or a result.</td>
<td>![Icon]</td>
</tr>
</tbody>
</table>

**Table 1 Types of Work-Product**

**Task**

The task denotes the unit of work that is assigned to role to be performed. The tasks are usually few hours or few days long in length. The task is carried out by one primary role but can be supported by additional role if the task demands it. The task modifies or produces the work product when a role
performs the task. The task is usually used as an element of defining a process. The figure (7) explains how the task works.

**Guidance**

The main aim of guidance is to explain “how to” while a task explain “what to be done”. It provides the guidelines and may be associated with roles, tasks and work products. The various types of guidance includes checklist, concepts, example, guideline, estimate, consideration, practice, report, reusable asset, roadmap, supporting material, template, term definition, tool mentor and whitepaper. The Figure (8) explains the usage of guidance in a model. A Role performs the task and outputs a work-product which is checked by a guidance to make sure the work-product is an intended output. Various types of guidance are shown in Figure (9)

![Figure 8 Guidance in Process Models](image)

**Figure 8 Guidance in Process Models**

There are 13 types of guidance contained in the guidancekind. The type of guidance shown in solid boxes (guidelines, checklist, supporting material) is extended in solution chapter (S-SPEM). The guidelines provides an additional information to perform a particular task, checklist is a specific type of guidance that identifies a series of items that need to be completed or verified and supporting material is a catch-all for all other type of guidance not specifically defined elsewhere.

![Figure 9 Types of Guidance adapted from[2]](image)

**Figure 9 Types of Guidance adapted from[2]**
Process

A process is a special activity that defines how to reach from one milestone (an action or event indicating a significant change or stage in development) to the next by defining sequence of work, operations or events and produce outcome. Figure (14) in section 2.6 illustrates the process.

Phase

A significant period in a project, ending with major management checkpoint, milestone, or set of deliverables is called as phase. In SPEM2.0 phase is categorized as a special activity, because of its significance in defining breakdowns. Figure (10) depicts the SPEM entity phase to explain the phases in software development. Each phase ends with a pre determined milestone to start the next phase.

![Figure 10 Phases in Software Development](image)

RoleUse

RoleUses represent a role dynamically in the context of a specific activity. It is the performer of TaskUses and defines a set of related skills, competencies and responsibilities of a set of individuals[24]. Figure (11) shows dependencies of RoleUse, TaskUse and WorkproductUse

![Figure 11 Dependency of RoleUse, TaskUse and Work-ProductUse](image)
TaskUse

TaskUse defines a unit of work dynamically and it represents a proxy for a Task Definition in the content of one specific activity. Therefore one Task Definition can represent by many Task Uses each within the context of an activity with its own set of relationships[2]. Task Use has a clear purpose in which the performing roles achieve a well defined goal. The complete step by step explanation for doing all the works to achieve the desired goal is provided by TaskUse [24]. Figure (11) shows the TaskUse in the process model dynamically.

WorkProductUse

A WorkProductUse represent a WorkProductDefinition dynamically in the context of one specific activity. WorkProductUses are the artifacts, produced, consumed or modified by TaskUses. Figure (11) explains the WorkProductUse that is exploited by RoleUse by performing a TaskUse.

2.4.4 Cut of SPEM 2.0 Metamodel

The modeling elements explained in the previous section are engineered in a process model by SPEM metamodel explicitly. Figure (12) depicts the cut of SPEM2.0 Metamodel. The breakdown element is an abstract generalization for any type of process element that is a part of breakdown structure. The WorkBreakdownElement is a special breakdown element that provides specific properties of breakdown element to represent the work. The RoleUse and WorkProductUse are the special breakdown elements that represent the dynamic view or a process model. The Milestone represents a significant event in the software development. The Milestone is a WorkBreakdownElement that indicates events like major decision, completion of project and delivery time. The sequencing of WorkBreakdownElements is carried out by WorkSequence that determines the start/finish of a Workbreakdown Element that allows the begin/ end of the next WorkBreakdownElement in an order. As Activity is BreakdownElement themselves, and hence they can nested inside other activities as well. This SPEM 2.0 metamodel is exploited in Chapter 5 that will address all the limitation of metamodel and usability of SPEM model is explained in Chapter 6.

Figure 12 Cut of SPEM 2.0 Metamodel adapted from [2]
2.4.5. Tools Supporting SPEM2.0

This section discusses the tools that support the modeling within SPEM2.0.

**IRIS Process Author**

IRIS Process Author (IPA) is a software process automation tool developed by a Canadian software firm Osellus. The IPA is a visual process management system that helps the organization to model, improve and automate the software processes [25]. The main advantage of IPA is that process validation is carried out automatically. With the large inventory of process data in a complex workflow model, the process validation would become tedious. IPA plays a vital role in such instances where the validation process is carried out automatically so that process authors need not validate it manually. Further, the validation of the processes ensures end-to-end integrity[26].

**EPF Composer**

Eclipse Process Framework composer is a process modeling tool platform and extensible conceptual framework in compliance with SPEM [27]. EPF is employed for authoring maintaining and customizing software development processes. EPF composer facilitates three sample process frameworks and provide flexibility for users to choose and customize existing three process frameworks. It also provides options for creating a new process framework [28]. The three process frameworks provided by EPF are OpenUP/Basic, Extreme Programming and Scrum.

The main aim of EPF composer is to serve two purposes [28]. Firstly, to provide knowledge base of intellectual capital to development practitioners that allow them to browse, manage and deploy content. Secondly, to provide catalogs of pre-defined processes for various projects that can be modified for individual needs. The process developed from EPF composer can be published and deployed as websites.

**StarUML**

StarUML is an open source UML model development tool that supports the newly adopted SPEM2.0 version. StarUML develops fast, flexible, extensible, and freely-available UML/MDA platform and it runs on Win32 platform. The models developed using StarUML can be extracted into XMI files which contain the model information. The usability plays a vital role in software process development. StarUML is designed to deliver many user-friendly features such as quick dialog, keyboard manipulation. Drag and drop options, diagram overview and etc.

All the process modeling (SPEM 2.0) in this thesis is carried out in StarUML tool because StarUML is a open source UML model development tool (IPA is closed source) and it provides easy drag and drop options (EPF lacks drag and drop options). The StarUML also provides options for extending the modeling notation and since the notation extension of this tool serves the purpose of this thesis partly, it is chosen to process the models in SPEM 2.0 in this thesis [56].
2.5 XPDL- XML Process Definition Language

This chapter gives the overview of XPDL and introduces the XPDL basic language constructs.

2.5.1 XPDL Overview

XML Process Definition Language (XPDL) is a formal standard process definition language proposed by the Workflow Management Coalition (WfMC)[3]. XPDL serves as an exchange language between different modeling languages thereby providing interoperability [29]. The main goal of XPDL is to exchange the process definition, both the graphics and the semantics between different modeling languages. XPDL includes the elements to hold the graphical information, such as the X and Y position of the nodes, as well as executable aspects, which would be used to run a process.

The XPDL specification uses XML as the mechanism for process definition interchange. The process definition consist of a network of activities and their relationship, criteria to indicate the start and termination of process and information about the individual activities such as participants, etc [30]. The XPDL specification provides the process definition interface that defines a common interchange format. The interface also defines a formal separation between the development and run-time environments, enabling a process definition, generated by one modeling tool, to be used as an input to various workflow runtime products.

2.5.2 Basic Process Elements

The XPDL metamodel defines a basic set of elements, shown in Figure (13) used in the exchange of process definitions. For a Process Definition, the following entities must be defined, either explicitly at the level of the process definition, or by inheritance directly or via cross reference from surrounding package. For the clarity of process, only few important entities that are used in the future chapters are explained in this section. The top level elements are as follows [3]

Process Definition

The process definition element provides contextual information that applies to other elements within the process. It contains the process itself and provides information associated with administration and the information used during the process execution.

Process Activity

A process activity represents the work, which will be carried out by a combination of resources (for e.g. Participants) specified by application assignment and/or computer applications (for e.g. Supporting tools) specified by application assignment. An internal process includes one or more activities, each comprising a logical and self-contained unit of work. The scope of an activity is localized to a specific process definition.

Transition Information

Activities are related to each other via flow control conditions which are called as transition
information. Each individual transition includes three elementary properties, the from-activity, the to-activity and the condition under which the transition is made.

**Participant Declaration**

The descriptions of resources that can act as a performer and carry out various activities in the process definition are provided by participant declaration. The participant declaration does not necessarily refer to a human or a single person, but may also refer to set of people with appropriate skill or responsibility, or machine automata resource rather than human.

**Swimlane**

Swimlanes are used to provide a graphical layout of processes and the activities they contain. The participant information can be designated to it at process level and the performer information at activity level.

**Application Declaration**

This provides descriptions of the IT application or interfaces which may be invoked by the service to support, or wholly automate, the processing associated with each activity, and identified within the activity by an application assignment attribute or attributes.

**Relevant Data**

This defines the data that is created and used within each process instance during process execution.

---

**Figure 13 XPDL Process Definition Meta-Model** taken from [3]

28
**System and Environment Data**

This is data which is maintained by the process or workflow management system or the local system environment, but it can also be accessed by activities or used by the process or workflow management system in the evaluation of conditional expressions and assignments in the same way as relevant data fields.

**Resource Repository**

The resource repository accounts for the fact that participants can be humans, programs, or machines.

### 2.6 RTCA DO-178B

RTCA DO-178B, “Software Consideration in Airborne Systems and Equipment Certification” is a standard published by Radio Technical Commission for Aeronautics (RTCA, Inc) and Europe Organization for Civil Aviation Equipment (EUROCAE) that deals with the safety of software employed in airborne systems[31]. In the history of commercial aviation, it has been reported that not even one passenger has lost life due to software failure in avionics control system. This standard proves to be a milestone in the development of software for avionics domain [32].

The document RTCA DO 178B, serves as a universal specification for avionics software process management. The standard provides guidelines for developing software for airborne systems and equipment that can function with the level of confidence in safety. As an acknowledged universal standard in the avionics industry for developing software, DO-178B is applied in many aviation projects by the US military [33]. The standard classifies the software into 5 levels of criticality. Each level of software criticality contributes to the amount of effort required to establish compliance with certification requirement that varies with the failure condition.

The Design assurance level (DAL) is determined during the application of the safety assessment process and hazard analysis by analyzing the effect of avionics software failure in the system. The failure conditions are determined with the misbehavior in intended action of software that affects aircraft, passenger and crew. DAL has to be documented in the System Safety Assessment Process (SSA). The certification authorities require and DO178B emphasizes to determine and establish the correct DAL by using comprehensive analysis methods to establish software level A-E. The system safety assessments are driven by software safety analysis to determine the appropriate DAL, so that an appropriate level of rigor can be established in DO178B. The failure conditions and its corresponding software level are shown in Table 2.

<table>
<thead>
<tr>
<th>Failure Condition</th>
<th>Software level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Level A</td>
</tr>
<tr>
<td>Hazardous</td>
<td>Level B</td>
</tr>
<tr>
<td>Major</td>
<td>Level C</td>
</tr>
<tr>
<td>Minor</td>
<td>Level D</td>
</tr>
<tr>
<td>No Effect</td>
<td>Level E</td>
</tr>
</tbody>
</table>

**Table 2 Failure conditions and its corresponding software level**
The level A is most critical and applies to the software; which if it fails to perform the intended function could lead to catastrophic failure such as loss of life. The levels B to E are decreasing in the level of criticality where failure of intended function of software level B leads to hazardous effect of passenger and aircraft. The failure of software with level E contributes to the major injuries or discomfort to the passengers and also spoils the aircraft. The failure of software with level D has very minor consequences on aircraft and passengers and the failure of software with Level-E has no significance towards to aircraft and passengers. These software levels are used in this thesis to reflect its significance in its respective process (refer chapter 5).

2.6.1. Software lifecycle processes- DO178 B


2.6.2. Overview of processes

The DO-178B defines a set of software related activities as well as objectives for each process. The use of standard processes and compliance with the objectives and activities defined in each process helps to prevent common mistakes during software development for avionics domain.

The three software related process of DO178B is shown in Figure (14). The phases comprised in each process are shown in Figure (15) and it is briefly explained in the following sections.

![Figure 14 Overview of DO178B Processes](image)

The plan documents are developed and the coordination of activities is planned in the planning process. The development activities that include requirement analysis, design, implementation and integration are carried out in development process. The assurance that the development and planning process have been followed is carried out in software integral process. The crucial software verification process is also carried out in software integral process[34].
All the processes mentioned in DO 178B have inputs, outputs, and transition criteria. The traceability of processes is given high priority in DO-178B standard.

**Planning Process:**

The standard defines objectives for development and integral processes, that demands the developers to start documentation in the planning process to meet the objectives mentioned by the standard in future processes. The standard allows any unavoidable variations in software plan, while developing a software as long as the objectives emphasized in software development processes are satisfied [4]. The main purpose of planning process is to determine the appropriate means of developing software that will satisfy the system requirements and delivers the level of confidence which is consistent with airworthiness requirements. The standard DO-178B demands 5 plans that have to be documented to meet the objectives in next consecutive processes. The 5 plans include:

- Plan for Software Aspects of Certification (PSAC)
- Quality Assurance Plan (QA)
- Configuration Management Plan (CMP)
- Software Development Plan (SDP)
- Software Verification Plan (SVP)

**Development Process:**

The DO-178B allows flexibility by providing various methodologies for handling software requirements and design. The development process is composed of three vital phases (Requirement Phase, Design Phase, Coding and Integration Phase). The standard facilitates developing multiple levels of requirements mainly high level requirements (HLR) and low level requirements (LLR). The high level requirements are directly developed through the analysis of system requirements and system architecture. These high level requirements are further developed in the design phase to produce one or more low level requirements. The low level requirement is provided to system safety assessment process that validates the low level requirements with the system safety requirements. Hence the development process plays a vital role in providing safety to address the airworthiness requirements. The intention of the standard is to create an overlap between HLR and design [4]. Figure (16) depicts the multiple level requirements facilitated by DO178B standard.
Figure 16 Multiple Level Requirement adapted from [4]

The implementation is carried out in coding phase. During the coding process, if source code is developed directly from high level requirements then the high level requirements are also considered as low-level requirements. The guidelines for low level requirements are applied for the same.

Integral Process:

The correctness, control and confidence of the software life cycle processes and their outputs are ensured in this integral process. Integral process consists of four phases that are explained briefly in the following.

Testing and Verification Phase

Software testing in DO-178B includes both black box testing (Requirement based testing) and white box testing (Coverage analysis). The DO-178B uses the term “verify” instead of “test” to emphasize more on the process that explicitly contain a combination of review, analysis and results. The verification process actually helps to show the absence of errors unlike testing process and hence the standard employs the term verification. The DO-178B elucidates three requirements based testing methods- low level, software integration and hardware/software integration. The Normal range test case selection and robustness test case selection are the two categories of requirement based test case selection. The test cases with regular inputs are carried out in Normal range testing to determine the ability of software to respond to regular inputs and conditions. The test cases with abnormal inputs and conditions are carried out in Robustness testing [35].

Configuration Management Phase

Configuration management phase plays an vital role for establishing and maintaining consistency of a product requirement, product’s configuration information and physical and functional attributes throughout its life [36]. It provides assurance that configuration of product is determined and reflected in the product information that facilitates support and product maintenance [37]. The Software Configuration Index (SCI) and Software life cycle environment configuration index (SECI) are the documents developed and maintained in configuration management process of DO-178B. These documents record the problem reports, changes and related activities. The configuration management phase elucidated in DO-178B provides a detailed inventory and revision identification of software development environment and software integration tool.

Quality Assurance Phase

The reviews and audits of software development is carried out in Quality Assurance Phase to show the compliance with DO-178B standard. The Quality Assurance Phase outputs 3 documents which are: Software Quality Assurance Records (SQAR), Software Accomplishment Summary (SAS), and Software Conformity Review (SCR). The audit results and evidence of completion of software
conformity review that are approved by certification authority should be submitted as part of certification application. Hence quality assurance phase is considered to be vital process to obtain certification.

*Certification Liaison Phase*

The aim of certification liaison phase is to establish communication and understanding between the applicant and certification authority throughout the software lifecycle to assist the certification process[31]. The minimum documents that has to be submitted to certification authority for certification includes, Plan for Software Aspect of Certification, Software Configuration Index, and Software Accomplishment Summary.

### 2.7 Related work

Over the last two decades, the need for a formally defined software process has emerged and it is found that, although the software processes are rich in describing approaches to a variety of task/activities that take place during the process, it lacks in portraying safety concerns in the software processes [38]. To overcome this, approaches for expressing concepts related to safety in software process are emanating and [39] discusses about one such approach to support safety in process modeling of embedded system applications. To support safety in process modeling, a safety oriented process meta-model was proposed to meet all the requirements of safety in software process [37]. Based on this notion, there are many researches and practices for augmenting safety into the software process are currently developing in these years.

To determine the safety concerns, many safety standards like IEC61508, RTCA DO178B are widely used as safety guideline for developing safety critical software processes[40][41]. To express the safety concerns in the process model, the Software Process Engineering Meta-model (SPEM 2.0) modeling standard offers solution for modeling software processes by customizing the SPEM 2.0 meta-model [42]. SPEM2.0 provides an opportunity for extending the meta-model for implementing a new approach for software process reusing based on software architectures. The PPFS (Process based Pattern Fundamental Structure ) designed by European project TERESA proposes a new metamodel with the concepts of SPEM meta-model to specify safety oriented processes [43]. The PPFS Metamodel was proposed by developing a fresh metamodel using the basic concepts of SPEM and including the vital safety concepts like SIL, checks and safety relationships. Based on this viewpoint, S-SPEM in this thesis follows a similar approach as PPFS Metamodel but instead of proposing a new metamodel, S-SPEM is built by extending the existing SPEM metamodel and introduces few more safety concepts that are compliant to safety standard under consideration. Besides PPFS Metamodel, many practices and approaches in extending the SPEM meta-model for augmenting characteristics like quality, process line variability, executability and representing agent oriented methodologies in SPEM 2.0 has been carried out in [44][45][46][47][58].

Though there are many researches in these years for extending SPEM 2.0, this thesis focuses on extending SPEM 2.0 to express safety concerns based on a well defined safety standard like DO-178B. This thesis follows the similar approach of [36] in which IEC 61508 was considered for extending SPEM meta-model. Furthermore, many researches on transforming the process models for attaining the usability and executability were carried out in recent years [48][49][30]. An approach on transforming the models between well standardized meta-model like SPEM meta-model, UML to an
XPDL workflow model elucidates the essence of validation [50][51]. SPEM to XPDL approach carried out in SoftPM project performs the transformation using transformation rules, algorithm and an engine was to realize the transformation [48]. Based on this notion, this thesis provides a similar approach for sketching transformation rules and algorithm but the transformation is carried out manually for transforming the process models of extended SPEM 2.0 (S-SPEM) onto a XPDL workflow process to depict the usability of the S-SPEM process models.
3. Problem formulation and analysis

This chapter explains the problem addressed in the thesis and analyses the problem to refine into manageable sub-problems. Section 3.1 formulates the essence of the problems briefly and section 3.2 elucidates the problems with a logical analysis.

3.1 Problem Formulation

Software failure in eminent domains like aerospace, defense and medicine makes the headlines due to the potentially fatal consequences. The software development has particularly serious implications for such software failures [52].

Currently, it has become vital to optimize safety in software development process to develop a safety critical software system. The safety standards like DO178B, IEC 61508 and EN50126 emphasizes to maintain an evidential chain to ensure that the software is developed with safety concerns. This evidential chain assures better documentation, communication, certification and reuse of safety critical software. The safety critical software developed for domains like aerospace that demands safety, should comply with the safety standard DO178B to be certified for airworthiness. A modeling language for modeling a safety oriented software development process is relevant to provide evidence that the process followed to develop the software is compliant with the standard.

For developing such an adequate modeling language for modeling a safety oriented software development process, SPEM 2.0 is investigated in this thesis. However, SPEM 2.0 is a general purpose language and lacks modeling constructs to address safety concerns. This thesis investigates to propose an approach for extending the SPEM 2.0 to accommodate the safety concerns for modeling a safety oriented software development process. Furthermore, it is also essential to check and validate the software process model modeled using the extended SPEM 2.0. An avenue for validation of software process model has to be implemented by transforming the process model developed from extended SPEM 2.0 to another standardized metamodel. It is essential that process model transformation to confirm that the semantics are preserved during transformation. This thesis investigates in validating the software process models by transforming the SPEM 2.0 process models into a XPDL workflow model and also aims at preserving the semantics during the transformation.

3.2 Problem Analysis

To realize the goal of this thesis, the problems are decomposed into sub problems and analyzed in this section. The problems concerning the extension of metamodel with the safety concerns and the problem concerning the executability of process models are broken down into sub problems and analyzed in this section.

Identification of Safety-related process elements emphasized in DO178B

The vital safety features for modeling a safety oriented software development process lies on determining the safety concepts that can be expressed well in the process model. The software safety
standard for the appropriate domain must be considered for research and evaluating the crucial safety concerns emphasized in the safety standards. Since, this thesis focuses on avionics domain; DO178B safety standard should be evaluated to capture the safety related process elements. The process objectives and the activities enlisted in the DO178B standard should be analyzed to understand the safety concerns emphasized in the standard. A better augmentation of safety concerns to the metamodel depends on determining an effective modeling construct to express the safety concerns so that the safety properties can be well manifested.

**Selection of a modeling languages to model the safety oriented process mandated within DO178B**

An extensive study on various predominant UML based languages for software process modeling is essential to select an appropriate modeling language (language 1) to model the safety oriented software process mandated within DO178B. Various aspects of process modeling language like semantic richness, conformity to UML, graphical representation, executability, modularity, formality and tooling support are considered for choosing a better modeling language to model safety oriented process model.

**Evaluation of the expressiveness of the selected modeling language (language 1) with respect to safety and understandability**

Expressiveness of a modeling language is an ability to depict the process model with the modeling constructs without any additional comments. The modeling constructs of the metamodel have to be explored to realize the modeling capabilities that are at disposal and the missing modeling capabilities for modeling a safety oriented software development process. An evaluation of missing modeling capabilities of selected modeling language is essential to determine the competence for effective expressiveness and understandability of safety in process model.

**Is the modeling language executable?**

The selected modeling language for modeling safety oriented software development process is checked for ability to execute. If the selected modeling language (language 1) is not executable, then a tool supported process definition language (language 2) has to be selected for providing transformation rules, so that the models from language 1 can be transformed into models in language 2. An ontological study on both the metamodel is essential for mapping the modeling constructs for transformation rule. Furthermore, determining correctness of the transformed workflow model by case studies establishes reliability on mapping rules and transformation algorithm.
4. Method

This chapter explains the step by step method for extending SPEM 2.0 metamodel and an approach for validating the process model by transforming the S-SPEM process model into and XPDL workflow model. Section 4.1 explains the approach for extending the SPEM 2.0 and the Section 4.2 explains the approach for validating the S-SPEM.

4.1 Approach for extending SPEM 2.0

The analysis of the sub-problem mentioned in Section 3.2 results in choosing SPEM 2.0 over the other modeling language based on the ability of the SPEM towards tool support, flexibility for extension, satisfying characteristics of process modeling language mentioned in background Chapter and the existing research work on SPEM metamodel. The S-SPEM is developed by actualizing the following approach. The activity diagram that depicts the step by step approach is provided in the Figure (17). The explanation of the activity diagram is provided in Table 3.

Figure 17 Activity diagram for extending SPEM 2.0

The extension of SPEM 2.0 starts with analyzing the standards that emphasize more on safety in software development process. The RTCA DO178B standard is selected for further analysis to determine to safety concepts enlisted in the standard. The background Section 2.5 provides a brief explanation of the standard RTCA DO178B.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
</table>
| A-1      | Analysis of DO 178B standard  
The DO 178B standard is studied and the safety-related concepts that are emphasized in the standard are identified. Those safety-related concepts that are amicable for software process modeling are then identified. |
| A-2      | Study of SPEM 2.0  
The SPEM 2.0 is examined with SPEM specification and existing literature, for missing crucial safety modeling constructs and the ways to accommodate the safety related concepts that are emphasized in DO178B. |
| A-3      | Extension of SPEM 2.0 metamodel  
The SPEM 2.0 metamodel is extended to provide new modeling capabilities to model the safety oriented software development process in compliance to DO178B standard. The extended metamodel with safety attributes is called S-SPEM. |
| A-4      | Usage of S-SPEM to model DO178B Safety processes  
The process models are modeled using the S-SPEM to demonstrate the enhancement in terms of expressiveness. |

**Table 3 Approach for extending SPEM 2.0**

The output of activity A-4 is assessed to check whether the S-SPEM process models satisfactorily enhances the development process of safety-critical software and the expressiveness of the modeling constructs. If the results are not satisfactory, then the step A-1 is carried out and the process is iterated to obtain a best result.

**4.2 Approach for validating S-SPEM**

XML process definition language (XPDL) is chosen based on the analysis of sub-problem explained in Section 3.2. The validation of S-SPEM process models is carried out by transforming the process models of S-SPEM into an XPDL workflow model. The S-SPEM to XPDL approach aims at supporting S-SPEM model enactment in workflow paradigm. The transformation consists of semantics of the S-SPEM process model, the semantics of XPDL workflow model, transformation rules and the algorithm for implementing the transformation.

The Figure (18) explains the step by step approach for transforming the S-SPEM process models into an XPDL workflow model. The first step in the transformation process is to carry out an extensive study.
on XPDL specification. The familiarity in XPDL is necessary to find the ontological match of the modeling construct for S-SPEM modeling elements. The second step includes the transformation mapping that finds the XPDL elements to match the suitable modeling construct to reflect the behavior of modeling construct in S-SPEM process model.

Figure 18 Activity diagram to validate S-SPEM

The transformation rules are developed using the semantics of the metamodels. Each transformation rule represents a mapping between the modeling constructs that portrays similar characteristics from source (S-SPEM) and target metamodel (XPDL). These transformation rules and the transformation algorithm are used to perform the S-SPEM to XPDL transformation, which is illustrated in the Figure (19). The correctness of the transformation is examined by case studies, where transformations are implemented at various scenarios to check for uniqueness in the transformation and the sustainability of semantics during the transformation. The transformation of S-SPEM process model into a XPDL
workflow model is explained in chapter 6 and the demonstration of manual transformation with pseudo code is explained in case study at chapter 7.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
</table>
| A-1      | Study of XPDL  
An extensive study on XPDL specification is carried out and the expressiveness of the XPDL model and its ability to reflect the S-SPEM modeling elements during transformation are analyzed. |
| A-2      | Transformation mapping  
The mapping between modeling elements of S-SPEM and the XPDL modeling elements is developed. The mapping is done by examining ontological similarities between the modeling elements of both the metamodels. |
| A-3      | Transformation algorithm  
The step by step approach for sequencing the transformation S-SPEM process into XPDL workflow model is developed. This transformation algorithm provides a clear guideline for the transformation of S-SPEM process models. |
| A-4      | Manual transformation of DO-178B process models  
The S-SPEM process models are transformed manually into XPDL workflow model using the transformation rules and the transformation algorithm. |

**Table 4 Approach for validating S-SPEM**

The output of the activity A-4 is examined for correctness of transformation. If the transformed XPDL workflow model fails to depict the information of S-SPEM process models in the XPDL workflow model, then the whole approach is repeated.

![Implementation of Model transformation](image)

**Figure 19 Implementation of Model transformation**
5. S-SPEM

The background chapter has elucidated the basic process conceptual elements and modeling capabilities of SPEM 2.0 with a brief explanation of its modeling limitations. This chapter develops a solution to overcome the limitations by extending the SPEM metamodel with the missing safety concerns and proposes S-SPEM (Safe SPEM) that emphasizes more on safety oriented software development process.

5.1 Modeling safety within SPEM 2.0

The goal of modeling safety within SPEM 2.0 is to identify and include the crucial safety elements that are missing in the SPEM 2.0. The meta-model proposed within this chapter allows the development teams to communicate, monitor, and analyze the integration of safety within the software process effectively. This thesis has investigated how well the traditional SPEM 2.0 metamodel can support the safety in the software processes, so that the missing safety concerns can be augmented easily within the SPEM 2.0 by extending the SPEM 2.0. The thesis is carried out in a step by step approach where the crucial safety elements mentioned in DO-178B are identified and the SPEM metamodel is extended to accommodate the safety elements mentioned in DO-178B standard. The safety concepts emphasized in the DO-178B standard are explained and motivated in Section 5.2.

5.2 Process-related safety concepts within DO178B

The process-related safety concepts that can be retrieved from DO178B are:

- The consistency of the software process has to be maintained.
- The software development process should be traceable and verifiable.
- The transition criterion between the software processes has to be compliant with the DO178B standard.
- The impact of system safety assessment process on the software process has to be monitored and the configuration of the software has to be maintained
- The software process must be compliant to DO178B standard by following the objective and activities emphasized for each processes in the standard.

The safety related concepts determined above are then accommodated in the SPEM2.0 metamodel to provide new modeling capabilities. The metamodel proposed with these safety related concepts can be used to model a safety related process in a more expressive way, and hence the name S-SPEM (Safe SPEM). The following section will explain the safety construct that are used to extend the traditional SPEM 2.0 metamodel.

5.3 Modeling with safety constructs

The process related safety concepts mentioned in the section 5.2 and the missing crucial safety modeling constructs in SPEM2.0 that are analyzed by an extensive study on SPEM2.0 are considered to develop a set of new modeling constructs that commits the safety criteria in the software process.
The new modeling safety constructs are *Safety Guidance, Checks, Safety report, Reviews, and Audit.* These modeling constructs are explained below.

**Safety Guidance**

The RTCA DO178B standard consists of objectives and activities for each processes and phases. The objective and activities mentioned in the DO178B explains the steps that have to be carried out to develop software for airworthiness certification. Hence the safety construct *safety guidance* proposed in this thesis denotes the safety steps mentioned in the objectives and activities that has to be essentially followed to develop safety oriented software. Furthermore, each process delivers documents that act as a guideline for consecutive process. For instance, planning process of DO178B outputs documents like Plan for Software Aspects of Certification (PSAC), Software Development Plan (SDP) and 7 other documents, which must be used as guidelines while developing and verifying the software for airworthiness. Therefore these safety documents delivered from each process are also enlisted under the safety guidance to show a visibility between normal guidelines and safety guidelines.

To this concept, the following icon (graphical concrete syntax) is associated:

*Icon associated to the safety guidance concept:*

The icon built by enriching the guidance element available in SPEM 2.0 (See Section 2.4.2) with a safety hat to show the importance of safety in the guidelines enlisted in the *safety guidance* modeling construct.

*Element implication:*

The *safety guidance* modeling construct are used by safety activities like *checks, reviews and audits* (will be explained in following sections) or/and by normal activities/tasks to perform work according to the safety guidelines mentioned in DO178B or/and the safety document produced during the process. To show the usability of safety guidance, the coding phase from software development process is considered. Figure (20) shows the role of *SafetyGuidance* in coding phase of software development process mentioned in DO178B.

![Figure 20 Software Coding Process- DO178B](image-url)
Software coding is performed by software developer collecting inputs such as software design and low level requirement. The safety guidelines mentioned in Software Development Plan (SDP) and Software Code Standard (SCS) that are developed during the planning phase are used as safety guidance to develop source code and object code.

**Checks**

The system safety assessment process (See Section 2.1) emphasizes more on justifying the integrity of software by facilitating an end-to-end check during the development of software. The transition from one activity to the other is monitored by this safety construct *Check* which assures the transition requirement emphasized by the DO178B standard are satisfied. Information is verifiable if it can be checked for correctness by a person or tool, so augmenting an element called check in SPEM 2.0 will help to perform an end-to-end check. The vital role of check is to maintain the consistency of development process and the configuration of software being developed. The main essence of safety software relies on correctness, completeness and consistency of software being developed. The proposed check element will not produce any workproduct but it act as a milestone (see section 2.4.2) that has to be reached to move to next phase/process.

To this concept, the following icon (graphical concrete syntax) is associated

*Icon associated to the check concept:*

The icon consists of a folder with a tick notation to realize it as checking process that are carried out in the software process.

*Element implication:*

The standard DO178B oblige to provide an intensive check in processes where it enlists a set of check that should be carried out while moving from one phase to the other. To show the usability of check safety modeling construct, the development process mentioned in standard DO178B is considered. The development process in the standard suggests to check the requirement phase with the requirement objectives, whether a. system requirement that are allocated to software are specified in high level requirements, b. the high level requirement conforms the software requirement standard, c. the high level requirements are stated in quantitative terms and tolerances are included before moving to the design process. Similarly, for each transition between the other phases, a check is conducted with the objectives mentioned in the standard as a safety guidance documents. The check element performs these checks before moving from requirement process to design process. The background chapter (Section 2.6.2) clearly explains the planning processes. The following Figure (21) depicts the check element in the development process emphasized in DO178B standard.

*Figure 21 Software Development Process- DO178B*
Safety Report

Safety report is an artifact produced while a safety activity like review or audit is carried out in a process. The safety report consists of the results of review and audit carried out in a process/phase and it also records the deviations in process/phases from DO178B standard objectives if any. The performer use this safety report as a feedback document to correct the deviations and also used as a supporting document to provide assurance that the processes have been carried in compliance to the DO178B standard.

To this concept, the following icon (graphical concrete syntax) is associated

Icon associated to the Safety Report concept:

The proposed review element is portrayed as an existing artifact with a safety hat to show that the artifact consist of safety information.

The element implication of safety report will be explained in the following Section. The Figure (22) and (23) shows the role of safety report in process models.

Review

A software review is a meeting during which a software product is examined by project personnel, managers, users, customer, user representatives or other interested parties for a comment or approval [53]. The software review is considered to be an important transition criteria between processes/phases in DO 178B. The completeness of a process is determined only when the review on particular process are conducted and approved[54]. The compliance of the software processes towards the standard DO178B are ensured by conducting reviews. The DO 178B asserts a combination of review, analysis and test that plays a vital role in verification process. The review also delivers an assessment of accuracy, completeness and verifiability of the software requirements; software architecture and source code. The analyses of the software process are also carried out within the subclass of review. The distinction between the review and analyses is that analyses deliver repeatable evidence of correctness whereas the reviews provide a qualitative assessment of correctness. A software review is performed with the safety guidance such as checklist or similar aid and an analysis examine the functionality, performance, traceability and safety implication of a software component. The proposed review element will provide a completeness of a process and approval to enter the consecutive process thereby scrutinizing the safety of the software.

To this concept, the following icon (graphical concrete syntax) is associated

Icon associated to the Review concept:

The proposed review element is portrayed as a folder with a magnifying glass notation to realize it as keen reviewing process carried out in the software process.

Element implication:

To show the practical usage of review element, the planning process mentioned in DO178B standard is considered. According to DO178B, an assurance review must be conducted in planning process to
ensure that the software plans and the software development standard comply with the safety guidelines framed by DO178B standard. The safety guidance such as lifecycle guidelines, review and assurance guidelines and development standard guidelines are used to review the plans and lifecycle data produced in planning process. As a result of assurance review (safety review), a review report is produced for checking any deviations in the plans and development standard produced in the planning process.

![Software Planning Process- DO178B](image_url)

**Figure 22 Software Planning Process- DO178B**

- **Audit**

An audit is an independent examination of software life cycle processes and their outputs to confirm that it satisfies the required attributes. The software quality assurance process of DO178B proclaims that audit plays a vital role in the quality of software. The main role of audit is verifying the compliance with the standard DO178B and to check for deviations. The deviations found in the process are then recorded, evaluated, tracked and resolved. Such a monitoring of activities in the software life cycle is performed to provide assurance that the activities are completely docile. The results of the audit process play an important role in the certification of the software.

To this concept, the following icon (graphical concrete syntax) is associated

**Icon associated to the Audit concept:**

The proposed audit element is represented with a notepad checklist notation to portray that the vital role of audit is to detect the deviation in software process and resolve it by recording the deviation.

**Element implication:**

The auditing is mainly carried out in software quality assurance process of DO178B. The software quality assurance of the standard emphasize to perform an audit on software development and integral processes during the software lifecycle to assure software plans comply with the standard and to ensure that software development environment has been provided as specified in the software plans. The Figure (23) denotes the role of audit in quality assurance process; where the quality analyst carries out the audit for assure quality. The quality audit ensures that software plan comply with the DO178B...
standard and the software development environment is provided as specified in the software plan. The deviations are recorded in audit report and the corrections are made in compliance with the software configuration management plan.

Figure 23 Software Quality Assurance Process- DO178B

5.4 S-SPEM Extension

5.4.1 S-SPEM Extension- Safety Activity and Safety Workproduct

SPEM 2.0 is extended with the above mentioned safety constructs in this chapter. As Figure (24) shows, the extension of SPEM 2.0 consist of two new safety element class called SafetyActivity and SafetyWorkProduct, and a Safety Level enumeration for the activities. Safety activity and safetyworkproduct constitutes an inventory of process modeling constructs for SPEM 2.0 (Safety Report, Check, Audit, Review). Figure (24) shows the Extended SPEM (S-SPEM) that highlights the augmented extension in gray.

Safety level

SPEM 2.0 meta-model is extended with safety level enumeration. The system safety assessment factor determines the software level appropriate to the software system and hence the software level has to be inherited in software process while developing safety software for airworthiness. The safety level depicts the measure of rigor that has to be carried out while performing the activity. The safety level and its corresponding safety concerns are shown as follows,

Software Level 4 (SL 4): highest level of rigor is demanded and it is difficult to achieve. Failure of software produced from this process would lead to catastrophe and hence extreme care is taken if the process is coined SL4.
**Software Level 3 (SL3):** less arduous than SL4 but still demands a sophisticated practices and a strict guidance in the processes.

**Software Level 2 (SL2):** requires reasonable amount of checks to assure that the processes follows the safety guidelines.

**Software Level 1 (SL 1):** requires minimum level of rigor while carrying out activities but still demands a good check over the processes.

**Software Level 0 (SL 0):** the process with Software Level 0 is considered to be non-safety and hence no need of guidelines that has to be followed if the process is coined with SL0.

![Extended SPEM (S-SPEM Metamodel)](image)

**Figure 24** Extended SPEM (S-SPEM Metamodel)

**Safety Activity**

Figure (25) depicts the structure of safety activities that are used to facilitate the extension of metamodel. Safety activity can be defined as a special activity or a phase which indicates/insist safety at different levels of the process. The safety activity comprises of four kinds that are widely employed in the software development process to verify the safety concerns emphasized in DO178B. The role of each safety activity kind was explained in Section 5.3 and their practical implementation in software development process will be explained as a case study in chapter 7.
Augmenting a meta-class called SafetyActivity to the existing activity of traditional SPEM 2.0 is due to two main reasons. The activity element of traditional SPEM 2.0 is the centre of SPEM based processes as they can relate to each other to define sequences as well as specify the roles responsible for carrying out the process and the Workproducts (artifacts) that are manipulated. Hence choosing activity for implementing safety constructs would deliver the intended safety information in the software development processes. The second reason is that most of the safety concerns emphasized in DO178B can be easily incorporated in the practice if the importance is focused on activities of traditional SPEM 2.0.

Safety Work-Product

The important factor in process representation includes workproduct dependencies. The SPEM specification provides three way of representing the dependency relationship among the workproducts; composition, aggregation and dependency [2]. The composition indicates that the workproduct is part of another; aggregation indicates that the workproduct uses another workproduct and dependency indicates that the workproduct depends upon another workproduct. The WorkProductUse of the SPEM2.0 is extended with SafetyWorkProduct to model the process with safety report that are related to SafetyWorkProduct with composition. The safety activities make use of safety workproducts to carry out the intended task.

The safetyworkproduct kind is composed of a safety report. A process engineer can model a process not only by defining activities in workbreakdown structure but also by including workproducts in breakdown elements. The role of Safety Report in a process model is explained extensively in Section5.3. The Figure (26) depicts the elements of safety work-product.
5.4.2 S-SPEM Extension- Safety Guidance

The Managed Content package of SPEM is extended with safety guidance element on top of existing guidance which is a describable element. Section 2.4.1 explains briefly about the Managed Content package. The safety guidance elements inherits the existing guidance kind and exploits the guidance kind to make use of the needed modeling construct to emphasize more on safety in process models. The Figure (27) shows the extension of guidance element.

![Figure 26 Elements of Safety Work-Product](image)

It should be noted that the guidance element provided by SPEM 2.0 can also be used to model the normal guidelines, checklist and supporting material. However, to provide a visibility between the normal guidance and the guidance concerning safety, the metamodel is extended with safety guidance.

![Figure 27 S-SPEM Extension (guidance)](image)
Safety Guidance

Figure (28) illustrates the types of safety guidance. The safety guidance takes into account only the needed guidance type from the existing guidance element (see section 2.4.3) and used to express the safety concerns in the process models. The guidance types included in the safety guidance includes Checklist, Guidelines and Supporting Material. The checklist indicates the series of items that needed to be completed or verified and it serves to express the objectives and activities mentioned in each processes of DO178B standard.

![Figure 28 Elements of Safety Guidance](image)

The safety standard DO 178B also emphasizes more about the additional details on how to perform a particular task or grouping of tasks, rules and recommendations on work product and their properties and hence the safety guideline is used to represent the above mentioned additional details concerning safety. All other safety guidance (e.g. User/Organization defined safety rules) is included in the process model using supporting material modeling element of safety guidance.
6. S-SPEM to XPDL Transformation

This chapter explains the transformation of proposed S-SPEM to XPDL metamodel. The XPDL metamodel is briefly explained in the background chapter 2.4 which is used as a foundation to transform S-SPEM to workflow model to validate the process models developed using S-SPEM. This chapter also provides algorithm for the S-SPEM to XPDL metamodel transformation.

A transformation is development of a target model (workflow model) from the source model (S-SPEM) according to a set of mapping called as transformation rules. The source and target models are compliant to their respective metamodels and hence the mappings in terms of metamodels are elucidated in this thesis. The XPDL standard defines workflow terminology, to satisfy interoperability and connectivity between workflow products. This transformation is carried out because the process definition tool of XPDL is used to define the workflow, while the enactment service can execute the workflow. XPDL also provide options to execute the workflow models in many XPDL compliant engines (e.g. Shark). So the validation of a model developed from the proposed S-SPEM can be carried out when transformed into a XPDL workflow process model. The intuitive notion of expressiveness of both these metamodels is considered before carrying out the transformation. Figure (29) explains the transformation of S-SPEM model to workflow model.

![Figure 29 Transformation of S-SPEM to XPDL](image)

The S-SPEM acts as a source metamodel and the XPDL metamodel acts as a destination metamodel that are used to frame the transformation rules. The S-SPEM process models and the workflow models are compliant to their corresponding metamodels. The transformation rules are used by the transformation engine to manipulate the process model developed from S-SPEM to form a workflow model. Without the ability to perform model transformation, the S-SPEM models and the workflow model must be developed and understood separately and this often requires as much as effort as studying the model specification again re-creating the models from scratch in another modeling language. This thesis aims at reducing such an effort by providing the mapping of modeling constructs so that manual transformation of models is made easy.
The following properties mentioned by M.K.Jochen [55] has to be checked during model transformation,

- **Syntactic Correctness**: The syntactic correctness of model transformation has to be ensured. The model obtained from the transformation (target model) has to be compliant with the rules of syntax.

- **Semantic Preservation**: The confirmation that the semantics are preserved during the model transformation. This guarantees that the target model is semantically equivalent to the source model and important properties are preserved during the transformation.

- **Structural Properties**: The structure and the sequence of modeling elements in the source model have to be preserved and well reflected in the target model. This ensures that the target model is provided with proper association between the modeling elements with respect to the source model.

The transformation rules and the algorithm are explained in the following Sections.

### 6.1 Transformation Rules

The transformation rules contain all necessary information to map the source and target model. The transformation is framed according to the ontology of both the metamodels. The transformation engine uses these rules to create the target model out of the source model and hence the transformation rules play a vital role in transformation of models. This Section explains the mapping between the metamodels and provides motivation for the same.

#### Role

A role may represent a single person or a set of people of appropriate skills or responsibility, a piece of software, a machine automata or something else. A role is an entity that is responsible for carrying out a particular activity. The S-SPEM categorizes role into two main entities based upon their use in static and dynamic process models. The RoleUse are used in dynamic process models whereas the Role definition is used to define a set of related skills or responsibilities in a static process models. Table 5 depicts the transformation mapping of role.

<table>
<thead>
<tr>
<th>SPEM Element</th>
<th>XPDL Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role Definition</td>
<td>Participant Declaration</td>
<td>Participant type of participant is set to Role.</td>
</tr>
<tr>
<td>Role Use</td>
<td>Swimlanes</td>
<td>RoleUse of the SPEM element is mapped to Swimlanes</td>
</tr>
</tbody>
</table>

*Table 5 Mapping of Role to its corresponding XPDL modeling element*
The Participant declaration of XPDL matches the definition of Role Definition in S-SPEM and hence there is no loss of semantics while mapping Role definition to participant declaration. The role S-SPEM entity of dynamic process model RoleUse is mapped to Swimlanes of XPDL because the swimlanes designate the participant and performer information to the lanes the accommodates the activities that are carried out by the respective performer or participant.

Activity

The Activity of XPDL is used to describe several activities by the pre-defined customized activity implementation type. All these activities share a common attributes, but the usage of the other attributes, particularly participant and application assignment and the use of relevant data field may be customized to the required activity implementation type. Table 6 shows the mapping between phase, iteration and process of SPEM elements to Activity type of XPDL elements.

The SPEM element Phase denotes a significant period of the project. It is defined as a special activity in SPEM 2.0, because of its significance in determining the breakdowns and setting a timeframe for carrying out a piece of work. The best corresponding mapping for Phase in XPDL is an activity with the implementation type event activity. An event in XPDL is defined as something that happens during the course of a project. The implementation type event of a XPDL entity activity defines the start, intermediate and end event to sequence the control and data flow though the activity. By customizing the attribute of intermediate event to none and specifying the timer to a default value, the start and end activity can comfortably represent the SPEM element phase in XPDL.

<table>
<thead>
<tr>
<th>SPEM Element</th>
<th>XPDL Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>Activity</td>
<td><em>Phase is transformed to Activity. Mapped to a corresponding XPDL activity type “Event Activity”</em></td>
</tr>
<tr>
<td>Iteration</td>
<td>Activity</td>
<td><em>Iteration is transformed to Activity. Mapped to a corresponding XPDL activity type “SubFlow”</em></td>
</tr>
<tr>
<td>Process</td>
<td>Activity</td>
<td><em>Process is transformed to generic Activity.</em></td>
</tr>
<tr>
<td>Task</td>
<td>Activity</td>
<td><em>Task is transformed to Activity. The corresponding XPDL entity is an alternative implementation type “Atomic Activity”</em></td>
</tr>
</tbody>
</table>

Table 6 Mapping of Activity element to its corresponding XPDL modeling element
The SPEM entity Iteration is a set of nested activities that are repeated more than once. It represents an important structuring element to organize work in repetitive cycles. The appropriate mapping for iteration in XPDL is an alternative implementation type of activity called subflow activity type. In subflow activity type, the transition restrictions are specialized in such a way that the sub-process call/return within the activity along with the normal transition. Since the sub-process call/return within the activity, the activity can be iterated. Hence the subflow implementation type of XPDL is mapped to iteration entity of S-SPEM. Figure (30) explains the subflow type activity of XPDL. All the incoming split transitions flow out the activity through outgoing transitions during normal transition restrictions. The call and return of sub-process is carried out in a repeated manner in the alternative implementation type subflow. Hence the normal transition restriction provides incoming and outgoing transition along with the subflow activity transition that calls the sub-process more than once.

The S-SPEM entity Process is mapped to generic activity of XPDL element without any alternative implementation type. The activity definition of XPDL denotes the a set of partially ordered work definitions intended to reach the development milestone and hence mapping the activity element to the process entity of S-SPEM is more appropriate.

![Figure 30 Subflow implementation type of XPDL taken from [3]](image)

The task is an atomic activity that is included within a process. A task is used to depict the work in the process that is broken down into a finer level of process model detail. The S-SPEM entity Task is mapped to an alternative implementation type of XPDL element activity called atomic activity. The atomic activity is used as an element for defining the process.

The XPDL allows the users to extend the functionality any XPDL entity to meet their individual needs. It is done by extending the attribute of an XPDL element which closely matches with the source element (in our case, S-SPEM element). The XPDL allows two ways to use the extended attributes, they are anonymous extended attribute and Namespace qualified extensions. We used Namespace qualified extension to extend the XPDL elements by adding attributes.
### Table 7 Mapping of Safety Activity Element to its corresponding XPDL modeling element

In order to deploy the namespace qualified extensions, we first extend the XPDL schema to add the extensions in the places in which XPDL schema allows the tool to add extensions. The resulting schema can be used in addition to the XPDL schema to validate the extension. Since some of safety elements of S-SPEM cannot be directly mapped to a corresponding XPDL element, we use the extended attribute of an activity to map the safety elements in XPDL. The extension places are marked in the XPDL schema by `[namespace = "##other" processContents= "lax"]`. The mapping of S-SPEM safety elements Review, Audit and Check to an extended attribute of an activity is shown in Table 7.

<table>
<thead>
<tr>
<th>SPEM Element</th>
<th>XPDL Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review</td>
<td>Activity and Extended Attribute</td>
<td>Transformed to <em>Activity</em>; Transformed to extended attribute of corresponding <em>Activity</em>.</td>
</tr>
<tr>
<td>Audit</td>
<td>Activity and Extended Attribute</td>
<td>Transformed to <em>Activity</em>; Transformed to extended attribute of corresponding <em>Activity</em>.</td>
</tr>
<tr>
<td>Check</td>
<td>Activity and Extended Attribute</td>
<td>Transformed to <em>Activity</em>; Transformed to extended attribute of corresponding <em>Activity</em>.</td>
</tr>
</tbody>
</table>

### Artifact

An artifact is a graphical object that provides supporting information about the process or elements within the process and it does not directly affect the flow of process. The XPDL provides modelers the capability of displaying additional information about the process that doesn’t affect the sequence flow of the process by using an element called artifact. This feature of XPDL is exploited to directly transform the artifact of SPEM element into an artifact of XPDL element. The mapping of artifacts are tabulated in Table 8.

<table>
<thead>
<tr>
<th>SPEM Element</th>
<th>XPDL Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact</td>
<td>Artifact</td>
<td>Transformed to <em>Artifact</em>. The corresponding XPDL entity is one of the standard artifact type <em>“Data Object”</em></td>
</tr>
<tr>
<td>Deliverable</td>
<td>Artifact</td>
<td>Transformed to <em>Artifact</em>. The corresponding XPDL entity is one of the standard artifact type <em>“Data Object”</em></td>
</tr>
<tr>
<td>Guidance</td>
<td>Artifact</td>
<td>Transformed to Artifact. The corresponding XPDL entity is one of the standard artifact type “Data Object”</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Safety Guidance</td>
<td>Artifact</td>
<td>Transformed to Artifact. The corresponding XPDL entity is one of the standard artifact type “Annotation”</td>
</tr>
<tr>
<td>Safety Report</td>
<td>Artifact</td>
<td>Transformed to Artifact. The corresponding XPDL entity is one of the standard artifact type “Data Object”</td>
</tr>
</tbody>
</table>

Table 8 Mapping of Artifact to its corresponding XPDL modeling element

The artifact of XPDL element contains three standard elements- a Data object, a group and an annotation. The Data Object can be an electronic document or a physical document that provides information regarding the process, used by the process or an output of a process. These abilities of Data Objects are used to transform the Guidance, Deliverable, Artifact and Safety report of S-SPEM element. Artifact contains the information used by the process, guidance contains the information regarding the process. Safety report contains the information of safety activities and the deliverable contains the output of a process. Hence the transformation of artifact, guidance and the deliverable into a Data object would be more reasonable and apt. Safety guidance is a new proposed element of S-SPEM element which is transformed into an annotation. The annotation of XPDL element is one of a standard entity of artifact that acts as a little note in between the process which emphasizes on the checks that has to be carried out. Safety guidance can also be mapped to data object, but since the safety guidance is a vital safety entity it has been mapped to annotation to show the difference from other data objects.

6.2 Transformation Algorithm

The transformation algorithm provides a clear understanding of steps that has to be carried out to transform the source model (S-SPEM) to the target model (XPDL). The transformation of the S-SPEM to XPDL is developed based on ontological information of both the metamodel explained in the previous sections. The main idea of this algorithm is to follow the transformation rules and sequence the transformation so that loss of semantics can be eliminated.

As indicated in the algorithm, an empty XPDL model is created. The Package header and the Redefinable header are initialized to the empty XPDL model that is created. The packager header contains information like XPDL version, Vendor details and the date. The redefinable header contains information like publication status and author information. Once the header files are initialized the mapping table has to be framed. The mapping table contains the tabulated information of transformation rules that act as a backbone for developing an efficient transformation without losing semantics. The first transformation of S-SPEM element to XPDL element starts at defining the participant involved in carrying out the workflow process. The Role and RoleUse are transformed into participants where the participant declaration is made for each Role element and a Swimlane is
assigned for each RoleUse element of the S-SPEM model. Once the roles are transformed to XPDL, an empty workflow process is created to illustrate the activities that have to be performed by the roles.

The transformation of activities and artifacts has to be made once the empty workflow process is created, so that the workflow process can accommodate the transformed activities and artifacts. The Phase, Iteration, Process, Task, Review, Audit and check are transformed to an Activity of the XPDL model. During the transformation of S-SPEM element to an Activity, Some S-SPEM elements cannot be directly mapped to the Activity element of XPDL. For such cases, the XPDL provides options for transforming the S-SPEM (that are not mapped directly to activity) to an alternative implementation type of activity element. Alternative implementation type of activity can be a Block activity, Subflow activity, Event activity, and Atomic activity. The transformation rule that contains the close ontological matching of S-SPEM elements to XPDL elements are used to transform the S-SPEM elements that are not directly mapped to activity (For.eg : Iteration is transformed to an alternative implementation type Subflow). The artifacts of S-SPEM elements are mapped to Artifacts of XPDL element. Similar to activity mapping, the artifact element of S-SPEM that cannot be directly mapped to artifact element of XPDL are mapped to an alternative implementation of artifact

S-SPEM to XPDL Algorithm

Input: SPEM model  
Output: XPDL model

1. Start;
2. Create an empty XPDL Model;
3. Create Process Header and Redefinable header;
4. Create Mapping table;
5. Transform Role and Role Use of SPEM to Participant of XPDL;
   \{Role, RoleUse\} \rightarrow \{Participant\}
6. Create Empty Workflow Process;
7. Transform the workflow process elements;
   7.1. Transform all activities;
       7.1.1. Transform \{Phase, Iteration, process, task, review, audit, check\} \rightarrow \{Activity\};
       7.1.1.1. If transformation includes alternative implementation, define the parameters for the corresponding alternative implementation activity type in XPDL model;
   7.2. Transform all Artifacts;
     7.2.1. Transform \{Artifact, Deliverable, Guidance, Safety Guidance, Safety Report\} \rightarrow \{Artifact\}
     7.2.1.1. If transformation includes alternative implementation, define the parameters for the corresponding alternative implementation artifact type in XPDL model;
8. For extended attributes
   8.1. Set extended attribute of activity to the corresponding activity in XPDL\{ Review, Audit\};
   8.2. Set extended attribute of Artifact to corresponding Artifact in XPDL \{ Check, Safety Guidance, Safety Report\};
9. Create association between entries;
   9.1. Transition information,
       9.1.1. Determine the association information of SPEM elements from SPEM model
       9.1.2. Map the corresponding association information to its respective XPDL elements.
   9.1.3. Determine the sequence of elements in the XPDL model using the specific ID provided to the XPDL elements
9.2. Transition relationship,
   9.2.1. Create relationship between same level entries;
       Activity- Activity relationship;
   9.2.2. Create relationship between different entities;
       Artifact- Activity relationship;
       Participant- Activity relationship;
   9.3. Other relationship,
       9.3.1. Create relationship for other extended attributes;
10. Process validity checking;
11. End;
Once the activity and artifacts are transformed, the extended attributes are defined to those elements that are not directly mapped or mapped to an alternative implementation type of an XPDL element. The transformation of all the elements ends at the step 8 and the sequence of elements in the XPDL model that corresponds to the S-SPEM model starts at step 9.

The association between the transformed XPDL elements in the XPDL model is carried out in the Step9. The association information of the each S-SPEM elements in the S-SPEM model is determined and the information is passed to the respective XPDL element. The sequencing of the elements in the XPDL model is carried out by the determining specific ID defined for each XPDL element in the XPDL model. Once the association information and the sequencing information is known the transition relationship starts at step 9.2. The transition relationship is vital in providing a XPDL model with a better clarity in workflow. The relationships between the same level entries (activities) are initially established so that the participant and artifact that rely on the activities can be related appropriately. Once the activity-activity relationship are established, the artifact produced/used by the activities are related to the corresponding activity and the participant that carryout the activity is related to the corresponding activity. The relationship between the elements with extended attributes to the XPDL elements are established finally.

Once the transformation of elements and the sequencing of the XPDL elements are completed, the process validity check has to be conducted to assess the error in the model. This process validity checks on semantics and completeness of the transformed XPDL model. The examples of S-SPEM model to XPDL model transformation are provided in Chapter 6 and its corresponding XPDL is provided in Appendix.
7. Case Study

This chapter explains the usefulness of S-SPEM with two case studies. The S-SPEM models introduced in section 3.4 are transformed into its XPDL models. The case study 1 explains the transformation of quality assurance S-SPEM process model into its XPDL workflow model and the case study 2 explains the transformation of Software development S-SPEM process model into its XPDL workflow model.

The success of the model transformation depends on systematic approaches in designing and implementing the model transformation. This case study motivates the model transformation by transforming the models manually using the algorithm and transformation rules designed in chapter 6 for the model transformation. By manually transformation of the models, the assurances of reaching the transformation rules are obtained (i.e.) the transformation of each modeling element of source model into its appropriate match element in the target can be checked. The S-SPEM process models are validated by transforming it into a XPDL workflow model in this case study.

The models transformation is carried out by rules based approach and the properties mentioned above are checked by implementing the transformation in this case study. The validation of the S-SPEM process models are carried out by the transformation in one direction: given a source (S-SPEM) model to generate the target (XPDL) model. At this level, the correctness of the model transformation is also verified by checking whether the model transformation produces unique results for different source models. For determining the uniqueness in the target model, quality assurance process models and the development process mentioned in DO 178B are modeled in S-SPEM and are used as an input for the transformation process in this case study.

7.1 Case Study 1:

Transformation of Quality Assurance S-SPEM process model to XPDL workflow model

The Quality Assurance process provides acceptance that the software life cycle processes produces software that conforms to its requirements by assuring that these processes are performed in compliance with the approved software plans and standards. Validation of this quality assurance process will maximize the acceptance level that the software produced is compliant with the approved plans and standards.

The input model for the transformation is S-SPEM Quality Assurance process model that is shown in Figure (31). The explanation of Figure (31) is provided in Section 5.3. The transformed Quality Assurance XPDL workflow model is shown in Figure (32). The XPDL code for the transformation is provided in the appendix.
Input: Quality Assurance S-SPEM process model

![Quality Assurance Process model in S-SPEM](image)

**Figure 31 Quality Assurance Process model in S-SPEM**

**Implementation**

The first step of transformation includes defining the package header and redefinable header. The package header and redefinable header for the transformation of quality assurance process model are defined as given below.

```xml
<xpd1:PackageHeader>
  <xpd1:XPDLVersion>2.1</xpd1:XPDLVersion>
  <xpd1:Vendor>(c) Mälardalen Hogskola</xpd1:Vendor>
  <xpd1:Created>2012-12-24 00:36:04</xpd1:Created>
</xpd1:PackageHeader>

<xpd1:RedefinableHeader PublicationStatus="UNDER_TEST">
  <xpd1:Author>Karthik</xpd1:Author>
  <xpd1:Version>1.0</xpd1:Version>
</xpd1:RedefinableHeader>
```

Once the headers are defined the first transformation of model elements start from transformation of Role in S-SPEM model to a Swimlane in the XPDL workflow process model. The Role in the quality assurance process model is a Quality Analyst which is transformed into a swimlane by defining the participant ID and Name in XPDL. The performer is defined as QA1. The pseudo code for the transformation is given below.

```xml
<xpd1:Participants>
  <xpd1:Participant Id="QA1" Name="Quality Analyst">
    ............................................................
    ............................................................
  </xpd1:Participant>
</xpd1:Participants>
```
The activities are transformed once the participants are defined in XPDL. The safety activity Audit of the S-SPEM Quality Assurance process model is transformed into an extended attribute of an activity as mentioned in the transformation rules. The StartEvent and EndEvent signify the time period of the activity and it is employed to represent a phase. The pseudo code for the transformation of audit element is provided below.

```xml
<xpd1:Activities>
    <xpdl:Event>
        <xpdl:StartEvent Trigger="None"/>
    </xpdl:Event>
    <xpdl:Event>
        <xpdl:EndEvent Result="None"/>
    </xpdl:Event>
    <xpdl:Activity Id="Q.Assurance_act1" Name="Quality Audit">
        <xpdl:Implementation>
            <xpdl:No/>
        </xpdl:Implementation>
        <xpdl:ExtendedAttributes>
            .............................................
            .............................................
        </xpdl:ExtendedAttributes>
    </xpdl:Activity>
</xpdl:Activities>
```

The artifacts are transformed as following and associations of artifacts to the activities are established in later stages. The data object of the XPDL is used to depict the artifact of S-SPEM in a XPDL model. The artifacts in Quality Assurance Process are Software Configuration Management plan, Software plan and Software Development Environment plan that are transformed into data objects in XPDL workflow model. The pseudocode for transformation is provided below.

```xml
<xpd1:Artifacts>
    <xpdl:Artifact>
        <xpdl:Artifact ArtifactType="DataObject" Id="SW.CM.plan" Name="Software Configuration Management plan">
            .............................................
            .............................................
        </xpdl:Artifact>
    <xpdl:Artifact>
        <xpdl:Artifact ArtifactType="DataObject" Id="SW.Plan" Name="Software Plan">
            .............................................
            .............................................
        </xpdl:Artifact>
</xpdl:Artifacts>
```
The safety artifact (audit report) is transformed to a data object in XPDL workflow model. The transformation is carried out by defining the DataObject as given below.

Once all the modeling elements of S-SPEM are transformed to the modeling elements of XPDL workflow process model, the associations between the modeling elements are established. The association of all the modeling elements brings out the structure and the sequence of modeling elements in the XPDL workflow process model. The association is done by the ID’s defined for each modeling elements.
The final step of this transformation is adding the start and end event activity that completes the XPDL workflow process model. The output shows the corresponding target (XPDL) model of quality assurance process in Figure (32).

**Output:** Quality Assurance XPDL workflow model

![Figure 32 Quality Assurance Process model in XPDL workflow](image)

The quality assurance model in XPDL depicts that the quality assurance S-SPEM model is transformed manually. The transformation also validates the usability of S-SPEM quality assurance process model.

### 7.2 Case Study 2:

**Transformation of Development Process S-SPEM process model to XPDL workflow model**

In case study 1, the quality assurance process in S-SPEM that contained modeling elements like quality audit, roles and work products were transformed to its respective XPDL modeling elements. This case study aims at changing the scenario by transforming the development process that contains S-SPEM elements like Iterations and Check element. By providing a transformation of S-SPEM development process model into its corresponding XPDL workflow model, the uniqueness of transformation can be verified.

The Figure (33) shows the development process in S-SPEM and the explanation of development process in S-SPEM is provided in Section 5.3. The XPDL code for transformation is provided in the Appendix.

**Input:** S-SPEM Development Process model

![Figure 33 Development Process in S-SPEM](image)
Implementation

The first step of transformation is defining the package header and redefinable header and it is similar to that elucidated in case study 1. Once the header files are defined, the roles involved in the S-SPEM process models are transformed into participants of XPDL model as explained in the transformation rules. The participant declaration is also similar to that mentioned in case study 1. Since the participant for development process are not included in the S-SPEM for this case study, the next modeling construct (activities) mentioned in algorithm are transformed.

The S-SPEM activity iterations are initially transformed into an alternative implementation type of XPDL activity element. The alternative implementation type of XPDL activity for the corresponding iteration element in SPEM 2.0 is Subflow element. The iteration elements in S-SPEM development process model are Requirement phase, Design phase, Coding phase, Integration phase. The transformation is done by defining the subflow activity ID and its Name as represented in the pseudo code.

```xml
<xpd1:Activities>
  <xpd1:Activity>
    <xpd1:Activity Id="newpkg1_wp1_act1" Name="Requirement_Phase">
      <xpd1:Implementation>
        <xpd1:SubFlow Id="newpkg1_wp1"/>
      </xpd1:Implementation>
    </xpd1:Activity>
  </xpd1:Activity>
</xpd1:Activities>
```

Once the iteration activity is transformed, the safety activity check elements of S-SPEM are transformed to its corresponding XPDL modeling element as elucidated by transformation mapping. The check elements of the S-SPEM development process are Requirement (Req) check, Design check, Coding check and integration check. The corresponding XPDL element for Check element is activity with an extended attribute. Each phase of the development process has to be checked before the start of next phase. The requirement objectives hold the checking information that adds safety to the software development process. The NodeThe Activity definition in a XPDL is defined as shown below.

```xml
<xpd1:Activity>
  <xpd1:Activity Id="newpkg1_wp1_act7" Name="Requirement Check">
    <xpd1:Implementation>
      <xpd1:No/>
    </xpd1:Implementation>
    <xpd1:Performers>
      <xpd1:Performer/>
    </xpd1:Performers>
    <xpd1:ExtendedAttributes>
      <xpd1:ExtendedAttribute Name="Safety"/>
    </xpd1:ExtendedAttributes>
  </xpd1:Activity>
</xpd1:Activity>
```
The workproducts in S-SPEM are transformed into its corresponding XPDL element. In this case study, the safety guidance is transformed into corresponding annotations. NodeGraphicsInfos is used to specify the visualization of the modeling element. The safety guidance in this case study is requirement objective, design objective, coding objective and integration objective. The pseudo code for transformation is given below.

```
<xpd1:Artifact>
<xpd1:Artifact ArtifactType="Annotation" Id="newpkg1_art1"
Name= "Req Check">
    <xpd1:NodeGraphicsInfos>
        ..........................................................
        ..........................................................
    </xpd1:NodeGraphicsInfos>
</xpd1:Artifact>
```

The association between the activities and artifacts are established after transforming all the S-SPEM modeling constructs to XPDL constructs. The activities Requirement Phase, Design Phase, Coding Phase, Integration Phase and the artifacts Requirement Check, Design Check, Coding Check, Integration check are transformed to its corresponding XPDL constructs. The ConnectorGraphicsInfos are used to specify the visualization of the associations between the modeling elements.

The association among the activities and the artifact type annotations are established as follows.

```
</xpd1:Activities>
<xpd1:Transitions>
    <xpd1:Transition From="newpkg1_wp1_act2" Id="newpkg1_wp1_tra2"
To="newpkg1_art1">
        <xpd1:ConnectorGraphicsInfos>
            ..........................................................
            ..........................................................
        </xpd1:ConnectorGraphicsInfos>
    </xpd1:Transition>
</xpd1:Transitions>
```

The final step of the transformation is including the start and end activity that completes the XPDL workflow process model.

The Figure (34) shows the output of transformation; XPDL development process model. The transformation of S-SPEM Development process model to its XPDL workflow model shows that the modeling elements are transformed without any loss of information. The transformation thus validates the S-SPEM development process model.

**Output:** XPDL Development Process model

![Figure 34 Development Process in XPDL workflow](image)
7.3 Discussion

The case studies explained in this chapter shows that the S-SPEM process models are transformed to their respective XPDL workflow model without much loss of information. Given the time limit of the thesis work, only two S-SPEM process models were transformed in this thesis. From the analysis, it was observed that the uniqueness of the transformation was maintained and hence the transformation of other S-SPEM process models can also be transformed to its XPDL workflow model successfully. The association of modeling constructs in the XPDL model plays a vital role in the preserving the structure of S-SPEM model and hence the association information of each S-SPEM modeling elements in S-SPEM models are determined and the transition between the respective XPDL elements are established.

This transformation of S-SPEM process models to its corresponding XPDL workflow model validates the S-SPEM process models and the usability of the S-SPEM is shown by this approach. Furthermore, the uniqueness of the output ensures that S-SPEM to XPDL model transformation can be carried out without losing the safety and structural properties of the models.

An extensive summary of this thesis work is explained in Section 8.1. From the evaluation of S-SPEM process models and the case studies, few limitations has been recognized and are briefly explained in Section 8.2. The potential leads for possible future work has also been discussed in the following chapter.
8. Conclusion

This chapter presents a brief summary and the outcome of the thesis work. This chapter also recommends few potential leads for future work.

8.1 Summary

This thesis has investigated and established a relationship between traditional SPEM 2.0 and software safety in development process. The de-facto safety standard in aerospace industry RTCA DO 178B is widely considered for developing a modeling language for modeling safety oriented software process. Furthermore, the validation of the process model developed by proposed metamodel is carried out by transforming the S-SPEM process models to XPDL workflow model. The validation is explained and motivated in the thesis work with an appropriate and clear case study.

In this thesis, the traditional SPEM 2.0 metamodel is extended by adding few essential safety elements thus developing an adequate modeling language (S-SPEM) for modeling safety in software development process. This thesis has investigated the RTCA DO-178B safety standard to capture the safety elements needed to develop S-SPEM. The proposed S-SPEM encompasses the important safety elements like Check, Audit, Review, Safety guidance and Safety report to model a safety oriented software development process. Each of these safety constructs are expressed with a visual representation and the work of each safety constructs are explained extensively. These proposed safety elements are promising in providing an end-to-end safety in software development process.

S-SPEM models are validated by transforming the models to XPDL. The reason for transforming the S-SPEM process models to a XPDL workflow model is because the XPDL contains elements to hold graphical information as well as executable aspects which would be used to run a process, thereby developing an extensive modeling language. The transformation rules that comprise a clear mapping of modeling constructs are main source for transformation process. The transformation rules in this thesis were sketched by analyzing the expressiveness of both the metamodel. The algorithm for the transformation explains the step by step approach for carrying out the manual transformation process.

A working case study has been presented in this thesis that explains the transformation of Quality Assurance S-SPEM process model and the Development process of DO-178B modeled in S-SPEM to their respective XPDL workflow process models. The pseudo codes for the process transformation are also elucidated extensively. The XPDL code for the transformation of S-SPEM process models are presented in appendix. From the evaluation of the S-SPEM process models and case studies, this proposed modeling language for safety oriented process is handy for developing a process model that is compliant with DO-178B standard.

Limitations

1. The safety modeling construct proposed in this thesis are introduced in the process model manually. A tool support with all the safety modeling construct must be implemented to model a safety oriented software development process.
2. The transformation of S-SPEM process models to XPDL workflow models are performed manually. An automated transformation must be implemented to optimize the transformation process.

3. Only the activities and the work products have been extended in SPEM 2.0 to introduce safety concerns in process models developed by S-SPEM. The roles can also be transformed to increase the safety in development process.

8.2 Future work

This thesis recommends few potential leads for future work. They are,

- To automate the transformation of process models modeled in S-SPEM to XPDL workflow process, a transformation engine can be developed. The transformation rules and algorithm can further investigated to optimize the loss of semantics during transformation processes.

- A tool support can be implemented with all the modeling constructs of S-SPEM to model the safety oriented processes. The safety elements and the notations mentioned in this thesis can be used to develop a modeling environment for domain specific software development process to provide a tool support that can be used to model safety oriented process.

- Though S-SPEM is designed for avionics domain by considering the airworthiness standard (DO 178B), it can be further extended and customized for meeting the safety demands in various other domains.

- There is a scope for widening the transformation ability of the proposed S-SPEM (like SPEM2BPMN). The usability of the S-SPEM can be increased by enhancing the interoperability of the metamodel. The transformation can be further developed to transform the S-SPEM process models into BPMN subprocesses by Query/View/Transformation (QVT).
References


Appendix

XPDL for CASE STUDY 1:

Transformation of Quality Assurance S-SPEM process model to XPDL workflow model

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xpdl:Package xmlns:xpdl="http://www.wfmc.org/2008/XPDL2.1"
xmlns="http://www.wfmc.org/2008/XPDL2.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Id="newpkg1" Name="newpkg1"
xsi:schemaLocation="http://www.wfmc.org/2008/XPDL2.1
http://www.wfmc.org/standards/docs/bpmnxpdl_31.xsd">

<xpdl:PackageHeader>
  <xpdl:XPDLVersion>2.1</xpdl:XPDLVersion>
  <xpdl:Vendor>(c) Together Teamsolutions Co., Ltd.</xpdl:Vendor>
  <xpdl:Created>2012-12-24 00:36:04</xpdl:Created>
</xpdl:PackageHeader>

<xpdl:RedefinableHeader PublicationStatus="UNDER_TEST">
  <xpdl:Author>Karthik</xpdl:Author>
  <xpdl:Version>1.0</xpdl:Version>
</xpdl:RedefinableHeader>

<xpdl:Participants>
  <xpdl:Participant Id="QA1" Name="Quality Analyst">
    <xpdl:ParticipantType Type="HUMAN"/>
    <xpdl:Description>Verifies the quality of the safety software by carrying out Quality
audit</xpdl:Description>
  </xpdl:Participant>
</xpdl:Participants>

<xpdl:Pools>
  <xpdl:Pool BoundaryVisible="true" Id="newpkg1_pool1" MainPool="true" Name="Quality
Assurance" Orientation="HORIZONTAL" Process="Q.Assurance">
    <xpdl:Lanes>
      <xpdl:Lane Id="newpkg1_pool1_lan2" Name="Quality Analyst">
        <xpdl:NodeGraphicsInfos>
          <xpdl:NodeGraphicsInfo BorderColor="0,0,0" FillColor="240,240,240"
IsVisible="true" ToolId="JaWE"/>
        </xpdl:NodeGraphicsInfos>
      </xpdl:Lane>
    </xpdl:Lanes>
  </xpdl:Pool>
</xpdl:Pools>
```

73
**XPDL for CASE STUDY 2**

*Transformation of Development Process S-SPEM process model to XPDL workflow model*

```xml
<xpdl:WorkflowProcess Id="newpkg1_wp1" Name="Development_Pro">
  <xpdl:ProcessHeader>
    <xpdl:Created>2012-12-31 16:05:49</xpdl:Created>
  </xpdl:ProcessHeader>

  <xpdl:RedefinableHeader PublicationStatus="RELEASED">
    <xpdl:Author>karthik</xpdl:Author>
    <xpdl:Version>1.0</xpdl:Version>
  </xpdl:RedefinableHeader>

  <xpdl:Activities>
    <xpdl:Activity Id="newpkg1_wp1_act1" Name="Requirement_Phase">
      <xpdl:Implementation>
        <xpdl:SubFlow Id="newpkg1_wp1"/>
      </xpdl:Implementation>
      <xpdl:NodeGraphicsInfos>
        <xpdl:NodeGraphicsInfo BorderColor="0,0,0" FillColor="255,106,106" Height="60" IsVisible="true" LaneId="newpkg1_pool1_lan1" ToolId="JaWE" Width="90">
          <xpdl:Coordinates XCoordinate="108" YCoordinate="32"/>
        </xpdl:NodeGraphicsInfo>
      </xpdl:NodeGraphicsInfos>
    </xpdl:Activity>

    <xpdl:Activity Id="newpkg1_wp1_act2" Name="Design Phase">
      <xpdl:Implementation>
        <xpdl:SubFlow Id="newpkg1_wp1"/>
      </xpdl:Implementation>
      <xpdl:NodeGraphicsInfos>
        <xpdl:NodeGraphicsInfo BorderColor="0,0,0" FillColor="255,106,106" Height="60" IsVisible="true" LaneId="newpkg1_pool1_lan1" ToolId="JaWE" Width="90">
          <xpdl:Coordinates XCoordinate="287" YCoordinate="32"/>
        </xpdl:NodeGraphicsInfo>
      </xpdl:NodeGraphicsInfos>
    </xpdl:Activity>

    <xpdl:Activity Id="newpkg1_wp1_act3" Name="Coding Phase">
      <xpdl:Implementation>
        <xpdl:SubFlow Id="newpkg1_wp1"/>
      </xpdl:Implementation>
      <xpdl:NodeGraphicsInfos>
        <xpdl:NodeGraphicsInfo BorderColor="0,0,0" FillColor="255,106,106" Height="60" IsVisible="true" LaneId="newpkg1_pool1_lan1" ToolId="JaWE" Width="90">
          <xpdl:Coordinates XCoordinate="486" YCoordinate="34"/>
        </xpdl:NodeGraphicsInfo>
      </xpdl:NodeGraphicsInfos>
    </xpdl:Activity>
  </xpdl:Activities>
</xpdl:WorkflowProcess>
```
<xpdl:Transition From="newpkg1_wp1_act4" Id="newpkg1_wp1_tra5" To="newpkg1_wp1_act6">
  <xpdl:ConnectorGraphicsInfos>
    <xpdl:ConnectorGraphicsInfo FillColor="0,0,0" IsVisible="true"
Style="NO_ROUTING_SPLINE" ToolId="JaWE"/>
  </xpdl:ConnectorGraphicsInfos>
</xpdl:Transition>

<xpdl:Transition From="newpkg1_wp1_act1" Id="newpkg1_wp1_tra1" To="newpkg1_wp1_act2">
  <xpdl:ConnectorGraphicsInfos>
    <xpdl:ConnectorGraphicsInfo FillColor="0,0,0" IsVisible="true"
Style="NO_ROUTING_SPLINE" ToolId="JaWE"/>
  </xpdl:ConnectorGraphicsInfos>
</xpdl:Transition>