SCENARIO-BASED REQUIREMENTS MODELLING

A PROGRESS REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

May 2014

Student: Abdulrahman Abdallah Hussein
Supervisor: Dr. Liping Zhao

SCHOOL OF COMPUTER SCIENCE
TABLE OF CONTENTS

TABLE OF CONTENTS ............................................................................................................. 2
LIST OF FIGURES .................................................................................................................. 4
LIST OF TABLES ..................................................................................................................... 4
ABBREVIATIONS .................................................................................................................. 5
ABSTRACT .............................................................................................................................. 6

1. INTRODUCTION ............................................................................................................. 7
  1.1 Project Background ...................................................................................................... 7
  1.2 Project Aims and Objectives ....................................................................................... 8
  1.3 Project Deliverables .................................................................................................... 8
  1.4 Report Structure ......................................................................................................... 9

2. BACKGROUND ............................................................................................................... 10
  2.1 Requirement Models .................................................................................................. 10
    2.1.1 Scenarios ............................................................................................................. 11
    2.1.2 Graphical Models ............................................................................................... 12
  2.2 PLANT: from scenarios to requirement models ....................................................... 12
  2.3 Business Process Model and Notation ....................................................................... 14
  2.4 UML State Machine Diagram ................................................................................... 15
  2.5 Model Driven Architecture ....................................................................................... 16
  2.6 Model Transformations ............................................................................................. 18
  2.7 Eclipse MDA Support ................................................................................................. 19
    2.7.1 Eclipse Modelling Project .................................................................................... 19
    2.7.2 Eclipse Modelling Framework ............................................................................ 20
    2.7.3 Model Development Tools ................................................................................. 20
    2.7.4 Model to Model Transformation ........................................................................ 20
    2.7.5 Epsilon .............................................................................................................. 21
  2.8 Related Work .............................................................................................................. 21

3. RESEARCH METHODOLOGY ...................................................................................... 23
  3.1 Project Organization ................................................................................................... 23
    3.1.1 Project Plan .......................................................................................................... 23
    3.1.2 Evaluation Plan ................................................................................................... 24
  3.2 Development Guidelines ........................................................................................... 24
    3.2.1 Basic Assumptions ............................................................................................. 24
    3.2.2 Transformation Approach ............................................................................... 25
LIST OF FIGURES

Figure 2-1: Categories of requirement models ................................................................. 11
Figure 2-2: PLANT pattern language and the multi-perspective requirement models ........ 13
Figure 2-3: BPMN 2.0 metamodel, Data Object class diagram ....................................... 14
Figure 2-4: MDA levels of abstraction ................................................................................... 16
Figure 2-5: MOF metamodel levels ...................................................................................... 17
Figure 2-6: Model transformations ....................................................................................... 18
Figure 2-7: The Eclipse Modelling Project .......................................................................... 20
Figure 3-1: Project Plan ........................................................................................................ 24
Figure 3-2: An example of a simple BPMN process model .................................................... 25
Figure 3-3: UML State chart expected as output for the above BPMN ............................... 25
Figure 4-1: Architecture of the transformation prototype system ...................................... 27
Figure A-1: Requirement engineering activities ................................................................... 36
Figure B-1: BPMN 2.0 metamodel, Process class diagram .................................................. 37
Figure B-2: UML 2.4.1 metamodel, State Machine class diagram ........................................ 37

LIST OF TABLES

Table 2-1: Classification of visual requirement models ....................................................... 12
Table 2-2: Model transformations classification .................................................................... 19
Table 2-3: Comparison of related work .............................................................................. 22
Table 4-1: Normalization phase – initialization step ............................................................. 28
Table 4-2: Normalization phase – data object nodes processing step ................................... 29
Table 4-3: Normalization phase – Activity nodes processing step ....................................... 29
Table 4-4: Filtering phase – application of filtering rules .................................................... 30
Table 4-5: Synthesizing phase – initialization step ............................................................... 31
Table 4-6: Synthesizing phase – OR composite states ......................................................... 31
Table 4-7: Synthesizing phase – AND composite states ....................................................... 32
Table 4-8: Synthesizing phase – constructing the hierarchical state chart ........................... 32
<table>
<thead>
<tr>
<th>ABBREVIATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOM</td>
</tr>
<tr>
<td>ATL</td>
</tr>
<tr>
<td>BPMN</td>
</tr>
<tr>
<td>CIM</td>
</tr>
<tr>
<td>EMF</td>
</tr>
<tr>
<td>EOL</td>
</tr>
<tr>
<td>ETL</td>
</tr>
<tr>
<td>GOM</td>
</tr>
<tr>
<td>MDA</td>
</tr>
<tr>
<td>MDD</td>
</tr>
<tr>
<td>MDE</td>
</tr>
<tr>
<td>MOF</td>
</tr>
<tr>
<td>OCL</td>
</tr>
<tr>
<td>OMG</td>
</tr>
<tr>
<td>OTM</td>
</tr>
<tr>
<td>PIM</td>
</tr>
<tr>
<td>POM</td>
</tr>
<tr>
<td>PSM</td>
</tr>
<tr>
<td>QVT</td>
</tr>
<tr>
<td>RML</td>
</tr>
<tr>
<td>UML</td>
</tr>
<tr>
<td>XMI</td>
</tr>
<tr>
<td>XML</td>
</tr>
</tbody>
</table>
ABSTRACT

Software developers inevitably face the challenge of delivering software that meets the user's requirements. Scenarios, usually written in natural language, have been widely used to represent a user-centric perspective of the system. Many practitioners argue that deriving supplementary visual models is an essential practice to understand the problem context from different viewpoints. A process model and an object life cycle model are two such viewpoints which represent system behaviour and control flow, and states of objects and their transitions, respectively. Together, the two models provide a mechanism for requirement's consistency check. In Model Driven Engineering (MDE), the focus is moved towards models, which are considered the most important artefact. MDE approach is based on automatic transformation between models at different abstraction levels as well as code generation. Proponents of MDE argue that it promises to deliver better productivity, portability and quality.

The context of this project is set by a recent study which introduced a pattern language to create a set of multi-perspective requirement models from scenarios. Creating these models manually seemed to take time and effort. This project looks into creating an object life cycle model, using UML state machine diagram, from a process model represented using Business Process Model and Notation (BPMN). The process model is assumed to be generated from a given scenario at the early stages of requirements elicitation. The project will make use of the Epsilon platform distributed as part of the Eclipse Modelling Project. So far, a chain of transformations with a set of transformation rules has been identified for the implementation. It is hoped that by automatically creating the target object model, this will help to minimize the efforts required to create requirement models for the purpose of requirements consistency validation or for further software generation.
1. INTRODUCTION

In the requirements acquisition and analysis phase of software development, system requirements are often captured as scenarios that are described within a use case. A scenario describes the interactions between the system and actors in order to achieve a goal for some stakeholders. Scenarios are often written as textual narratives which are then transformed into different kinds of models toward the system design and implementation. This process, usually implemented manually, could be difficult and time-consuming at some times.

Despite the fact that scenarios have an important role in requirement engineering, their informal textual nature can impose some drawbacks. On one hand, scenarios are simple narratives and hence proved to be useful to capture user’s knowledge. On the other hand, they can be inaccurate and have implicitly hidden goals. This can present a difficulty for software developers to accurately map scenarios into design models. Requirements modelling can provide a solution to tackle this problem. However, manually creating the different requirement models from the scenario can still be a challenge and time consuming effort [1]. The motivation of this project is to help in minimizing this effort by automatically generating models using model transformation technologies.

The next section of this chapter sets the context of the project by giving a brief overview of the project background, followed by the project aims and deliverables. The last section walks through the report structure.

1.1 Project Background

This project builds on recent research [1] which developed a pattern language, called PLANT, for transforming scenarios into different requirement models. In the aforementioned paper, a scenario metamodel was created with four major aspects: process or action path, object state transition, agent interaction, and goal. Accordingly, four different patterns were used to convert each aspect into a corresponding target model. The process aspect is transformed into a Process-Oriented Model (POM) which represents the sequence of actions undertaken in the path towards the goal of the scenario. The object aspect is transformed into an Object Transformation Model (OTM) which represents the state transitions of an object that is caused by the process actions. The agent aspect is transformed into an Agent-Oriented Model (AOM) that shows agents communication in the scenario. Finally, the goal aspect is transformed into a Goal-Oriented Model (GOM) that represents the scenario goal and its sub-goals. The approach was assessed by conducting a usability study and the testing results indicated its usefulness and usability. However, in order to increase the efficiency, the paper concluded with a suggestion to seek the automation in some parts of the manual modelling process.
1.2 Project Aims and Objectives

The aim of this project is to automatically transform from the POM into the OTM defined by the PLANT pattern language. A POM model is a process flow model usually represented by a UML activity diagram or a Business Process Modelling Notation (BPMN) diagram, while an OTM is a data flow model often represented as a UML state chart or Object State Transition Network (OSTN). The actions performed by agents in a scenario to achieve its goal may cause a change to the state of some objects. While the object state transitions are usually implicitly included in the process model, explicitly extracting it into an object state model would provide a useful complementary data model.

For this project, the initial POM is assumed to be generated manually from a given scenario and described using BPMN 2.0 process model. This model with its metamodel will act as the source model for the transformation to be implemented and the output target model will be the OTM described using UML state machine model which conforms to its own metamodel. In order to achieve this, the project will involve the following:

- Study the relative modelling standards and specifications.
- Research current available model transformation technologies and their classification. Features available in modelling frameworks and tools such as Eclipse Modelling Project (EMP) will be considered.
- Study the metamodels for the source model and target model and investigate the feasibility of defining transformation rules.
- Design, implement and test the transformation program.

Model transformation languages such as Atlas Transformation Language (ATL) and Epsilon Transformation Language (ETL) are commonly used in model-to-model transformations. However, the key challenge for this project is to identify transformation rules for behaviour models rather than structural models. The project should investigate whether it is possible at all the times to automatically create accurate transformation from BPMN process models into UML state chart models or there are certain conditions under which this is not applicable.

1.3 Project Deliverables

The deliverables of this project are as follows:

- The transformation program developed in Eclipse.
- The test cases used for development and testing with an XMI serialization format for the input and output models.
- The final dissertation report.
1.4 Report Structure

The progress report, in addition to this introductory chapter, follows the following structure:

**Chapter 2: Background**

This chapter gives a review about related literature and background research. This includes requirement models, PLANT pattern language, BPMN and UML state machine models. Model Driven Architecture (MDA) and its standards, model transformations and classifications, and MDA tools support are also covered. The last section presents some related work and the approaches used.

**Chapter 3: Research Methodology**

This chapter starts by explaining the project research strategy and activities given in the project plan and evaluation plan. Afterwards, the assumptions and decisions made for the project scope, approach, development environment and methodology are presented.

**Chapter 4: Progress Report**

This chapter shows the current progress in the project. First, the system architecture is given showing the main components and the phases of implementation. Furthermore, a detailed explanation of the transformation rules and steps that form the core of the system is presented. The last section gives a brief summary and future work to be carried out toward the completion of the project.
2. BACKGROUND

It is widely acknowledged that software requirements play a critical role in the success of software projects. Poorly written requirements can render some systems unusable or useless at all. Software requirements are defined as the needs to be delivered in a software system, and any constraints imposed on the system [2]. Requirements engineering is the discipline in software engineering concerned with all aspects of software requirements. Different activities are carried out in requirements engineering processes depending on the system being developed. These activities generally include developing and managing software requirements during the software development life cycle [3]. A brief overview of these activities is provided in Appendix A.

Modelling is considered an essential aspect in current software development methodologies. A model is an abstraction of the system which represents the system from a certain view or perspective. Providing different models helps in the communication between developers and stakeholders; hence reaching a better understanding of the business needs [4]. In MDD approach, models are the primary artefacts. Models are raised to an abstraction level above the programming language and implementation level, making it easier to use business domain concepts. The promises of MDD are in the flexibility of the implementation platform, and the improved productivity and quality of software, achieved by relying on automatic model transformations and generation of source code [5] [6].

This background chapter provides a literature review on software requirements modelling and model transformations. It begins by giving a brief overview on requirements models focusing on scenarios as a textual representation and other different graphical models. It then goes on to present the PLANT pattern language used to transform scenarios into corresponding requirement models. The BPMN and UML state machine models are also explained. Furthermore, a brief overview of MDA concepts and standards is shown, alongside with an introduction to model transformation techniques and classifications. The following section covers part of the modelling support provided in the Eclipse Modelling Project. The last section presents some related work that tackles similar problems.

2.1 Requirement Models

Software requirements can be represented in different ways from informal narrative using natural language to formal specification using formal mathematical language. Formal specification is recommended for real-time critical systems. However, for many other systems, a well-structured textual document supplemented with graphical models and diagrams is commonly used [7]. Natural language is inherently flexible,
expressive and easy to use, yet it can be ambiguous and difficult to understand. However, it is believed that it will still be widely used to specify requirements [4].

Software requirements and models can be considered from different angles, as shown in Figure 2-1. The following subsections describe scenarios, as a form of textual representation, and a group of other visual models.

![Figure 2-1: Categories of requirement models](image)

### 2.1.1 Scenarios

One effective way to capture user needs for a system is by taking a user-centric approach. A user-centric approach, in contrast to a product-centric way, focuses on what the users do to achieve their goals instead of what the system should do. Common user-centric approaches used in software requirements are use cases and user stories [7].

Use cases are based on identifying the system actors and their goals. An actor is a user role or an external system that communicates with the system. A single use case describes the interaction between the primary actor and the system to achieve his goal [9]. Cockburn considers a use case as a collection of scenarios some of which are successful scenarios leadings to the goal while others are failure scenarios [9]. In his vision, a scenario may include a precondition, the steps taken in the normal flow and alternate flows, and a postcondition. Therefore, scenarios are usually represented in process flow models such as activity diagrams. In fact, scenarios are regarded as the first form of requirements developed; therefore they derive the creation of other models [8].
2.1.2 Graphical Models

“A picture is worth a thousand words”. In software models, there is a debate about to what extent this proverb is true. However, many agree that there is “no single view of the requirements that gives a complete understanding” [7]. Therefore, the objective of requirements modelling is to create different viewpoints of the system at different levels of abstractions. This not only helps in understanding the system, but also in discovering incorrect, inconsistent or missing requirements [7].

A set of models, called Requirement Modelling Languages (RML), has been developed to visually represent requirements in an easy way for both business and technical users [10]. Compared to UML, the latter is created mainly to model system design, and it is considered more complex for business users. The models in RML are classified into four categories: Objectives, People, Systems and Data; as shown in Table 2-1. As a best practice, all four categories are required to make a comprehensive picture of the solution [10].

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Example of Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives</td>
<td>Describe the business value of the system and the priority of the requirements.</td>
<td>Business Objective Model. Feature Tree.</td>
</tr>
<tr>
<td>People</td>
<td>Describe the users of the system and their processes and goals.</td>
<td>Process Flow. Use Case.</td>
</tr>
<tr>
<td>System</td>
<td>Describe the existing systems, their interaction and behaviour, and the user interface.</td>
<td>Ecosystem Map. System Flow.</td>
</tr>
</tbody>
</table>

2.2 PLANT: from scenarios to requirement models

Section 1.1 briefly mentioned a pattern language [1] for transforming scenarios into multi-perspective models; where scenarios are considered as informal narratives. Hence, they suffer from the limitations of natural language addressed before such as ambiguity and difficulty of understanding. The objective is that the generated requirement models, alongside with the scenario, can provide a more comprehensive and accurate view of the system requirements.
The paper defined a scenario metamodel based on the scenario concepts from cognitive science and requirement engineering viewpoints. A scenario is described to have an initial state, a final state and a path that connects to the scenario’s goal. The path is a sequence of actions which involves one or many agents and can affect some object state. The scenario can be a normal scenario where the goal is achieved or an abnormal scenario where the goal is avoided.

There are four major aspects shown by the scenario metamodel. These aspects are the action path, the object state transition, the agent interaction, and the goal. The PLANT pattern language consists of four patterns that produce a corresponding model for each of these aspects. The patterns and the produced models are shown in Figure 2-2 and explained as follows:

- **Establishing the story line**: transforms the scenario sequence of actions into a POM. The POM is a process flow that can be represented by a UML activity diagram or BPMN diagram.
- **Elaborating things that change**: represents the transition of objects states as an OTM. The OTM is a state transition diagram and can be represented by a UML state machine diagram.
- **Identifying agents and their interactions**: shows the interaction between agents using an AOM which can be represented by a UML sequence diagram.
- **Unravelling the goal and its sub-goals**: transforms the scenario’s goal into a GOM which can be represented as i* models.

A pilot usability study was conducted to test the approach. The test was based on rating the usefulness, learnability, efficiency, and satisfaction benchmarks. The results showed an encouraging reception of the approach and its usefulness. However, the study reported an efficiency limitation because building the four models for every scenario was considered time consuming. The paper suggested further research to look into the possibility of generating some of the models automatically.
2.3 Business Process Model and Notation

BPMN is a standard graphical notation adopted by the Object Management Group (OMG) to represent business process activities. The BPMN specification [11] defines the metamodel, graphical notation, and serialization or interchange format. BPMN aims to facilitate communication between business and technical stakeholders by providing an intuitive and easy to use standard model.

As per the specification, BPMN 2.0 supports three types of diagrams: Processes, Choreographies, and Collaborations. Process diagrams represent the business activities performed by an organization. There are five basic categories of BPMN graphical elements defined in the specification:

- **Flow Objects**: the main elements to define the behaviour of the process. Contain three elements: Events, Activities and Gateways.
- **Connecting Objects**: There are four connecting objects that connect elements together: Sequence Flows, Message Flows, Associations and Data Associations.
- **Swim Lanes**: There are two ways to group elements in swim lanes: Pools and Lanes.
- **Artefacts**: There are two artefacts that can be used for providing additional information: Group and Text Annotation.
- **Data**: There are four elements to represent data: Data Objects, Data Inputs, Data Outputs and Data Stores.

Regarding the support of data handling, the BPMN specification states that data and messages flow can be shown as part of the diagrams; nevertheless BPMN is not a data flow language. A business process can show the data items created or manipulated during the process life cycle and their activity’s associations. Data items can be physical or electronic resources; singular or collection; and persistent or volatile.

![Figure 2-3: BPMN 2.0 metamodel, Data Object class diagram [11]](image_url)
Items in BPMN are represented as ItemAwareElement elements which can have an optional state defined by a DataState element. DataObject, similar to DataInput and DataOutput, provides information about what activities require and/or produce. Data objects can be referenced many times in the process using DataObjectReference elements. Data objects are contained in processes or sub-processes only. Figure 2-3 shows part of the BPMN metamodel corresponding to the data package. Appendix B shows the process class diagram.

Tasks and CallableElements can specify their data input and output requirements. Throw events can have only data inputs while Catch events can have only data outputs. DataInputAssociation and DataOutputAssociation elements are used to associate data as input and output, respectively.

It is worth mentioning that OMG defined two artefacts for the BPMN 2.0 metamodel. One format is based on Meta-Object Facility (MOF), and the other is based on an XML Schema Document (XSD). Therefore, there are two serialization formats supported; an XMI-based and an XML (non-XMI) format.

2.4 UML State Machine Diagram

State machine diagrams are one of the UML behaviour models used to describe object’s states and transitions in a system. The UML 2.x specification [12] defines the metamodel and notation for the state machine diagram. The state machine class diagram is shown in Appendix B.

State machine diagrams represent the life cycle of an object by showing the different states an object can exist and the events or triggers that cause the transition between the states. Transitions can be constrained by a Boolean condition called a guard. The diagram can also show the actions taken within a transition or a state.

State machines can have simple states or composite states. A simple state is an atomic state that does not have any sub-states or regions. A composite state is a state which contains one or more sub-states. Super-states are used for shared transitions from multiple sub-states. State machine diagrams support concurrent states by defining the concept of regions. A region is defined as an orthogonal part of either a state machine or a composite state which contains states and transitions. The notion of pseudostates is defined in the specification to represent different types of abstract and complex vertices. These include initial, fork, join, junction, and choice pseudostates. Fork and join pseudostates are used to represent splitting and merging from parallel states in a region. Choice and junction pseudostates show a decision branching point where only one condition is satisfied.
2.5 Model Driven Architecture

MDA architectural framework is OMG’s vision of the model-driven software development approach. MDA main idea is based on “separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform” [13]. Therefore, the MDA specification [13] defines three viewpoints on a system as shown in Figure 2-4 and described below:

- **Computation Independent Model (CIM):** focus on the domain under consideration and the system requirements while ignoring the technical details.
- **Platform Independent Model (PIM):** focus on the design of the system from a “technology-neutral” point of view without binding to a specific platform.
- **Platform Specific Model (PSM):** include additional details of a specific platform by adding the technical services provided by the platform to the PIM.

MDA is built on a foundation of standards defined by the OMG. The following standards are essential to the role of MDA:

- **UML [12]:** the de-facto standard used for modelling software solutions. UML is considered as a General Purpose Language (GPL) aimed to help software developers in the analysis, design and implementation of systems. The UML specification defines the abstract syntax using an MOF-based meta-model, the graphical notation, and the interchange format.
• **MOF** [15]: a universal standard used to describe different modelling languages. MOF defines an abstract syntax, called metamodel, which defines the syntactical constructs of the model and their semantics. A metamodel can be considered as a model of another model describing its elements and their relationships. MOF specification defines four meta-levels as shown in Figure 2-5. A model in each level is considered an instance of its above model (metamodel) if it conforms to it. The metamodel at level M3 is called a metametamodel because it describes metamodels at level M2. MOF M3 model is recognized as self-describing as it conforms to itself.

![Figure 2-5: MOF metamodel levels [14]](image)

• **OCL**: a formal language which can be used to add more constraints to the model semantics that cannot be expressed in the metamodel. OCL expressions allow the definition of preconditions, postconditions, and guards for model elements where all instances of the model must conform to the metamodel and ensures that the constraints defined using OCL are also true. OCL is widely used in model transformation languages to query and access source and target models [14].

• **XMI**: an MOF-XML mapping standard used as the model exchange mechanism between different modelling tools. It specifies how to translate the model’s abstract syntax into an XML-based concrete syntax [14].
2.6 Model Transformations

Model transformation is at the heart of model driven software development [16]. A transformation takes one or more source models as input and automatically generates one or more target models as output according to a transformation definition. A transformation definition is a set of transformation rules written in a transformation language. The transformation rules define how source model elements are mapped into target model elements and how the created target elements are initialized. A transformation tool is used to execute the transformation [6].

Figure 2-6 shows the principles behind model transformations. A transformation \( M_t \) transforms source model \( M_a \), an instance of \( M_Ma \), into a target model \( M_b \), an instance of \( M_Mb \). The transformation itself is considered as a model which conforms to its own metamodel. The transformation definition is written in a transformation language and handles source and target elements at the metamodel level. The transformation execution is a running instance of the transformation definition and handles model instances.

![Figure 2-6: Model transformations [14]](image)

Models can be transformed within the same abstraction level or between different abstraction levels such as from PIM to PSM. A detailed feature-based classification of model transformation approaches was implemented in [17]. Mens and Gorp [18] also proposed a similar taxonomy which includes aspects of tools and techniques available. A recent survey [19] carried out by Biehl, based heavily on the previous two papers, unified and represented the characteristics of the classification. Table 2-2 shows a list of some of the main classification criteria used in the study.
### Table 2-2: Model transformations classification, data taken from [19]

<table>
<thead>
<tr>
<th>Classification Criteria</th>
<th>Features</th>
</tr>
</thead>
</table>
| **Change of Abstraction** | *Vertical*: change (increase or decrease) the level of abstraction.  
*Horizontal*: change the representation but not the level of abstraction. |
| **Change of Metamodel** | *Exogenous*: source and target metamodels are the same.  
*Endogenous*: source and target metamodels are different. |
| **Number of Models** | *In-place*: source and target models are the same (one model).  
*Out-place*: source and target models are distinct. |
| **Target Type** | *Model-to-Model*: target model has a defined metamodel and elements.  
*Model-to-Text*: target model is an arbitrary text. |
| **Language Paradigm** | *Imperative/Operational*: explicit control flow (like in Java/C languages).  
*Declarative/Relational*: no explicit control flow.  
*Hybrid*: support both imperative and declarative constructs.  
*Graph*: models handled as graphs.  
*Template-based*: used for M2T.  
*Direct Manipulation*: use a general purpose programming language. |
| **Rule Application Control** | *Implicit*: no order control.  
*Explicit*: order specified within rules.  
*External*: order specified outside of rules. |
| **Rule Scheduling** | *Rule Selection*: control rule application (explicit)  
*Rule Iteration*: use recursion, looping.  
*Phasing*: specify the phases where rules can be executed. |

### 2.7 Eclipse MDA Support

The MDA specification defined by OMG has been implemented by different commercial and open source groups. The Eclipse Modelling Project [20] is an example of a widely used open source MDA implementation. This section gives a brief overview of the Eclipse Modelling Project and its main subprojects.

#### 2.7.1 Eclipse Modelling Project

The Eclipse Modelling Project[^1] is one of the top level projects in Eclipse that provides a set of modelling frameworks, tools and standard implementations. The project contains a number of specialized subprojects such as Abstract Syntax Development, Concrete Syntax Development, Model Development Tools, and Model Transformations. Figure 2-7 shows an overview of the different capabilities and key standards available from the Eclipse Modelling Project.

[^1]: http://www.eclipse.org/modeling/
2.7.2 Eclipse Modelling Framework

Eclipse Modelling Framework (EMF) is central to the Eclipse Modelling project. EMF includes a core framework that provides abstract model definition and XMI serialization features. It also includes components for building editors and Java classes for EMF models. Ecore is the abstract syntax component in EMF that is “closely aligned” to OMG’s Essential MOF.

2.7.3 Model Development Tools

The focus in Model Development Tools (MDT) is to provide implementation of industry standard metamodels. BPMN 2.0 and UML 2.0 specifications are implemented as part of this subproject.

The BPMN MDT\(^2\) component provides a simple graphical interface which allows designing and modelling of BPMN process diagrams. It supports the two OMG metamodel formats, MOF and XSD. Therefore, the serialization of the models generated can be either XMI-based or XML-based. The default format used is the XML-based [22]. UML2 component provides the implementation of OMG UML 2.x specifications with an XMI schema for models interchange. Papyrus\(^3\) provides an environment for building EMF models, particularly UML models.

2.7.4 Model to Model Transformation

Model to Model Transformation (MMT) subproject provides a framework for model transformation languages and engines. Three main languages are defined as part of it: Atlas Transformation Language (ATL), (Query/View/Transformation) QVT Operational, and QVT Declarative.

\(^2\) http://wiki.eclipse.org/MDT-BPMN2
\(^3\) http://www.eclipse.org/papyrus/
ATL is a hybrid unidirectional transformation language. ATL supports two types of transformation rules: matched rules and called rules. Matched rules define patterns for source model elements to be matched and how target model elements are created and initialized. Called rules must be explicitly called from ATL imperative blocks. [14].

QVT Operational and Declarative languages provide a partial implementation of OMG’s MOF QVT specification. QVT Declarative is similar to the declarative approach in ATL with the support of bi-directional transformation. QVT Operational is an imperative unidirectional transformation approach [21].

2.7.5 Epsilon

Epsilon is a model management platform consisting of a set of tools and languages. Epsilon Object Language (EOL), an imperative language, is the base language in the platform. It supports OCL syntax and is used to handle EMF models. The list of languages also includes Epsilon Transformation Language (ETL), Epsilon Merging Language (EML), and Epsilon Flock. ETL, similar to ATL, is a hybrid rule-based model-to-model transformation language. The main difference is that it supports querying and modifying both source and target models. Epsilon Flock is a rule-based language used to update models with regard to changes in their metamodels [23].

2.8 Related Work

A number of studies have examined the area of model transformations in general. Few were found related to the transformation between process and data models. To my knowledge, none of them has addressed the transformation from BPMN 2.0 process models to UML state machine diagrams. Table 2-3 gives a short list of related papers showing the source model, target model, and a description of the approach used in the study.

The most related work to this project is a study conducted by Eshuis and Van Gorp [24]. In this study, the authors presented an approach that takes a UML activity diagram and automatically constructs a hierarchical state machine diagram. GrGen⁵, a graph transformation tool, was used for the implementation. The approach is divided into two main phases: the first phase filters the activity diagram by eliminating irrelevant nodes, and the second phase creates the state chart from the filtered activity diagram. One important point highlighted in the study that the transformation may need extended rules to handle special situations, such as if the process model includes a cross-synchronization, which is not supported in UML state chart diagrams.

⁴ http://www.eclipse.org/epsilon/
⁵ http://www.info.uni-karlsruhe.de/software/grgen/
Table 2-3: Comparison of related work

<table>
<thead>
<tr>
<th>Source Model</th>
<th>Target Model</th>
<th>Transformation Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML activity diagram [24]</td>
<td>UML state machine diagram</td>
<td>Automatic transformation using GrGen graph transformation engine</td>
</tr>
<tr>
<td>BPMN 2.0 process model [25]</td>
<td>Data model (class diagram)</td>
<td>Defined a set of group rules</td>
</tr>
<tr>
<td>BPMN 2.0 process model [26]</td>
<td>UML Class model</td>
<td>Semi-automatic chain of transformations using QVT Relations and MOFScript</td>
</tr>
<tr>
<td>BPMN 1.0 process model [27]</td>
<td>UML activity diagram</td>
<td>Automatic transformation using ATL</td>
</tr>
<tr>
<td>BPMN process model [28]</td>
<td>BPMN process model</td>
<td>Explicit addition of data objects.</td>
</tr>
</tbody>
</table>

In [25], the authors studied the generation of data models, similar to class diagrams, from BPMN model. The approach is based on three groups of rules: the first group identifies the data model entities; the second group defines the relationship between the entities, and the third group adds attributes to the entities. The data entities generated are captured from data stores, participants, and pools represented in the process model. The study did not provide a specific implementation technique.

In [26], a UML class model is also generated from BPMN 2.0 model before creating a PSM based on JavaEE profile, and generating component source code. Similar to [25], the class model entities are generated from process, lanes, participants and data stores defined in the BPMN model. However, the associations and attributes need to be added manually afterwards.

Cibran described an ATL transformation from BPMN 1.0 to UML activity diagrams [27]. The study defined a conceptual one-to-one mapping for some similar elements, one-to-many mapping for rich BPMN elements (such as events), and mapping for overloaded BPMN elements (such as gateways). However, data flow elements were not addressed in the transformation.

Meyer and Weske introduced an approach to extract data objects and their states from general process models, giving a usage example with BPMN [28]. The process models use only activities, gateways (xor, and), and control flow elements. The approach is based on using a language parser to parse activity’s labels based on a set of assumptions. The process model with extracted data objects can be used further for object model generation.
3. RESEARCH METHODOLOGY

This chapter presents the research methodology adopted to gain some of the background knowledge necessary for the project. The following sections show the project organisation and plan, followed by the project development guidelines.

3.1 Project Organization

3.1.1 Project Plan

The project has been organized into the following four main phases, as shown in Figure 3-1:

The Literature review phase includes identifying the scope and general context of the project by studying the PLANT research paper and its related work to understand the motivation behind the project, and skimming through requirements engineering and modelling topics to set the wider context. It also involves taking an overview about MDA in order to become familiar with the topic as it was new for me. The milestone at the end of this phase is the initial report.

The Detailed Background study phase involves studying in detail the MDA base standards and specifically the BPMN and UML state machine specifications. It also includes examining the model transformation classifications and technologies available, and considering the tools and languages supported. This is achieved by studying related work found to understand what approaches and tools have been taken in similar projects and to make an informative decision about a suitable methodology that can be applied for this project. This phase also consists of learning and running the proposed tools to become familiar with the development environment. The outcome of this phase is a justified implementation approach with the system architecture and design explained in the progress report (this document).

The implementation and testing phase consists of building the system components by implementing the transformation rules using an incremental and iterative approach. The implementation will consist of two main phases each of which is divided into smaller iterations. The first phase will implement the filtering component and the second phase will implement the synthesizing component. Each iteration involves testing of the developed part using different case studies. The implementation phase ends with an evaluation of the approach. The milestone of this phase is the transformation program with the test cases used.

The Dissertation Write-up phase is about writing up the draft dissertation and reviewing feedback and changes before submitting the final dissertation.
3.1.2 Evaluation Plan

To evaluate the project approach used the prototype system will be run against different artificial examples and real-life test cases. The output of the system will be inspected for correctness against the expected output, and the time taken for the transformation will be measured to reflect the efficiency of the approach. The results can be compared against the ones obtained in the original study [24]. The report will also outline the situations under which the transformation might fail.

3.2 Development Guidelines

Based on the background research conducted and the analysis of related work, the following guidelines have been outlined for the project:

3.2.1 Basic Assumptions

BPMN 2.0 provides a rich set of notations and semantics on hand for the user. For this project, only a small, yet commonly used [29], subset of the elements will be considered. These include general tasks, events, exclusive and parallel gateways. The process model is also assumed to have one data object with multiple data states.
Figure 3-2 shows an example of a simple BPMN process model with a single data object *Purchase Request* passing into four different states. The UML state chart expected as output of the transformation system is shown in Figure 3-3, showing the same states and transitions for the data object.

![Figure 3-2: An example of a simple BPMN process model](image)

![Figure 3-3: UML State chart expected as output for the above BPMN](image)

### 3.2.2 Transformation Approach

As explained in section 2.8, the most related work found is [24] in which a hierarchical state chart is constructed from a process model. The study used activity diagrams as the input process model. Compared to BPMN, the two standards share some basic semantics while they differ in their notations and expressivity. The approach presented in [24] has been studied and will be applied for this project. The implementation of the system is influenced by the phases and rules defined in the approach. An additional initialization step is required because BPMN 2.0 allows using
activities with multiple sequence flows as input or output. In contrast, activities in activity diagrams as used in the mentioned study can have only one input and output control flow. The phases and rules are explained in detail in the next chapter. The paper also defined corner cases where the transformation might fail, and an extended rule to handle such cases. For a formal definition of the approach, the reader is invited to refer to the study.

3.2.3 Tools and Language

It can be shown from the model transformation classifications outlined in Table 2-2 in the previous chapter that a number of suitable model transformation tools and languages exist for different transformation problems. The tools and language choice is based on the problem and features of the language. Initially, ATL was considered for implementation of the system because of the language features and its popularity. ATL is a hybrid language widely-used for out-place exogenous transformations. It also supports in-place refinement of models. The rule-based declarative syntax makes the transformation definition concise and understandable. After studying the transformation approach in [24], it appeared that ATL might not be suitable. This is due to the fact that the transformation requires a chain of in-place refinements of the BPMN model some of which generates an output that does not conform to BPMN metamodel (mixed control and data flow). The transformation chain includes copying of the input model, removing some elements from the model, and creating new elements in the output model. By referring to the authors of [24], it turned out that other two suitable options are GrGen, a graph transformation language used by the authors in the study, and Epsilon, briefly introduced in section 2.7.5. For the time being, the latter is considered for the implementation as it provides a set of modelling languages that can update a model according to changes in its metamodel. Epsilon provides a comprehensive modelling platform on top of Eclipse IDE, which makes it a preferable option.

3.2.4 Development Methodology

The development methodology proposed for the implementation is an iterative and incremental approach. The transformation method that will be used consists of a sequence of steps; therefore, it suits well for an iterative development. Each iteration will include a design, implementation, and testing phase; and the implemented part will be demonstrated to my supervisor by the end of the iteration.
4. PROGRESS REPORT

This chapter shows the work done so far in terms of the system architecture and design, and future work planned for implementation in the next phase.

4.1 Current Progress

The main progress achieved so far is the identification of the appropriate approach that can be applied to the transformation problem addressed in this project. A substantial amount of time was dedicated for learning about the available tools in the Eclipse Modelling Project. The following subsections present the high level components of the prototype system, phases of implementation, and transformation rules.

4.1.1 System Architecture

Following the same architecture defined in [24], the transformation prototype system consists of two main modules as shown in Figure 4-1. In the first phase, the Normalization and Filtering module takes as input the BPMN 2.0 process model serialized in XMI or XML format and generates an intermediate filtered model in normal form. In the second phase, the Synthesizing module takes the filtered model and creates a hierarchical state machine model that conforms to UML 2.x metamodel. Each phase is divided into steps corresponding to the approach presented in [24]. These steps are explained in the next section.
4.1.2 Phases and Rules

The transformation method is based on two main phases: Filtering and Synthesizing.

- Filtering Phase:

In this phase, a BPMN 2.0 process model with one data object and multiple states is taken as input. The phase is divided into two sub-phases: normalization and filtering. *Normalization* consists of three steps performed in order: *initialization*, *data objects processing*, and *activities processing*. In contrast to activity nodes in UML activity diagrams, BPMN specification permits activities with multiple incoming or outgoing sequence flows (splitting and merging). Hence, a new *initialization* rule shown in Table 4-1, which is not needed in [24], is added in which the multiple (outgoing/incoming) sequence flows are replaced with an equivalent gateway (fork/merge).

<table>
<thead>
<tr>
<th>Table 4-1: Normalization phase – initialization step</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normalization: Step 1: initialization</strong></td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>BPMN process model with one data object and multiple data states. Each data object has at least one incoming or outgoing data association/flow.</td>
</tr>
<tr>
<td><strong>1:</strong> for each activity with multiple uncontrolled outgoing sequence flows do:</td>
</tr>
<tr>
<td>2: add a parallel gateway (fork) and change the source of the sequence flows to the gateway.</td>
</tr>
<tr>
<td>3: add a sequence flow from the activity to the fork gateway.</td>
</tr>
<tr>
<td><strong>4:</strong> end for</td>
</tr>
<tr>
<td><strong>5:</strong> for each activity with multiple uncontrolled incoming sequence flows do:</td>
</tr>
<tr>
<td>6: add an exclusive gateway (merge) and change the target of the sequence flows to the gateway.</td>
</tr>
<tr>
<td>7: add a sequence flow from the merge gateway to the activity.</td>
</tr>
<tr>
<td><strong>8:</strong> end for</td>
</tr>
<tr>
<td><strong>Postconditions:</strong> Each activity has one incoming and outgoing sequence flow.</td>
</tr>
</tbody>
</table>

The second step ensures that each data object has at least one incoming and outgoing data flow. This might fail, as described in Table 4-2, if the activity connected to the data object is not unique. In this case, the user has to manually correct the model. A similar rule, not shown here, is applied for data objects with no outgoing data associations. The source of the control flow from the unique activity where the data object is output is changed to the data object; otherwise the step also fails.
### Table 4-2: Normalization phase – data object nodes processing step

**Normalization: Step 2: objects processing**

**Preconditions:**
BPMN process model with one data object and multiple data states.
Each data Object has at least one incoming or outgoing data association.
Each activity has one incoming and outgoing sequence flow.

1: **for each** data object with number of incoming data associations = 0 do:
2: if activity where data object is input is unique (#=1) then:
3: if the activity has multiple input or output data objects then:
4: fail.
5: else:
6: change the target of the control flow entering the activity to the data object.
7: end if
8: else:
9: fail.
10: end if
11: end for

**Postconditions:** Each data object has at least one incoming and outgoing data flow.

### Table 4-3: Normalization phase – Activity nodes processing step

**Normalization: Step 3: Activities processing**

**Preconditions:**
BPMN process model with one data object and multiple data states.
Each activity has one incoming and outgoing sequence flow.
Each data object has at least one incoming and outgoing data flow.

1: **for each** activity with number of incoming data associations > 0 do:
2: add a parallel gateway (join) and change the target of the sequence/data flows to the gateway.
3: add a sequence flow from the join gateway to the activity.
4: end for.
5: **for each** activity with number of outgoing data associations > 0 do:
6: add a parallel gateway (fork) and change the source of the sequence/data flows to the gateway.
7: add a sequence flow from the activity to the fork gateway.
8: end for.

**Postconditions:** Each activity has one incoming and one outgoing edge.
The third step ensures that each activity node has exactly one incoming and outgoing edge. After applying the initialization step, each activity will have one incoming and outgoing sequence flow. By applying the rule shown in Table 4-3, a process model in the normal form (each activity node has exactly one incoming and outgoing edge) is generated.

The final filtering step applies a set of filtering rules to the normalized process model iteratively and in an arbitrary order, as summarized in Table 4-4. The filtering rules remove irrelevant activity and control nodes that do not affect the object’s states. By the end of this phase, a filtered process model with only data objects and control nodes is produced.

Table 4-4: Filtering phase – application of filtering rules

<table>
<thead>
<tr>
<th>Filtering phase:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preconditions:</strong> BPMN process model in the normal form.</td>
<td></td>
</tr>
<tr>
<td><strong>The following filtering rules R1-R10 are applied iteratively in arbitrary order:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>R1:</strong> remove all nodes (except data objects) with exactly one incoming and outgoing edge.</td>
<td></td>
</tr>
<tr>
<td><strong>R2:</strong> merge pairs of exclusive gateways (decisions) where the successor has only one incoming edge from the predecessor. Guard conditions are preserved.</td>
<td></td>
</tr>
<tr>
<td><strong>R3:</strong> merge pairs of exclusive gateways (merges) where the predecessor has only one outgoing edge to the successor.</td>
<td></td>
</tr>
<tr>
<td><strong>R4:</strong> remove a self-loop.</td>
<td></td>
</tr>
<tr>
<td><strong>R5:</strong> remove a redundant edge connecting a parallel gateway (fork) to a parallel gateway (join).</td>
<td></td>
</tr>
<tr>
<td><strong>R6:</strong> remove an end event following a parallel gateway (fork).</td>
<td></td>
</tr>
<tr>
<td><strong>R7:</strong> merge pairs of parallel gateways (forks) where the successor has only one incoming edge from the predecessor.</td>
<td></td>
</tr>
<tr>
<td><strong>R8:</strong> merge pairs of parallel gateways (joins) where the predecessor has only one outgoing edge to the successor.</td>
<td></td>
</tr>
<tr>
<td><strong>R9:</strong> remove a redundant exclusive gateway (decision) which is in parallel to a data object (has the same fork gateway as predecessor and same join gateways as successor). Guard conditions are moved to the data object edges.</td>
<td></td>
</tr>
<tr>
<td><strong>R10:</strong> remove a redundant exclusive gateway (merge) which is in parallel to a data object (has the same fork gateways as predecessor and same join gateway as successor).</td>
<td></td>
</tr>
<tr>
<td><strong>Postconditions:</strong> Process model with only data object and control (gateway) nodes (filtered BPMN process model).</td>
<td></td>
</tr>
</tbody>
</table>

- **Synthesizing phase:**

The synthesizing phase is divided into two sub-phases: building the state hierarchy and constructing the state chart. The initialization step in the first sub-phase, Table 4-5, takes the filtered process model produced from the filtering phase and
introduces basic or leaf states for each data object in the model. This is followed by an addition of OR states as parent of the basic states and control gateways that are not parallel control nodes. The second step merges each pair of OR nodes that match the rule defined in Table 4-6. Edges between the basic states are removed and raised to the new parent OR node. Edges between states are needed for the final state chart as they represent transitions, and they are created in the second sub-phase. The third step in this sub-phase, explained in Table 4-7, introduces AND states as parent of a set of OR states having the same predecessor and successor, and parallel gateway as input or output. A new OR state is added as the parent of the AND state and the edges are raised to this OR state.

**Table 4-5: Synthesizing phase – initialization step**

<table>
<thead>
<tr>
<th>Preconditions:</th>
<th>Process model with only data object and control (gateways) nodes (filtered BPMN process model).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for each data object do:</td>
<td>replace the data object with a basic state with the name of the object’s data state.</td>
</tr>
<tr>
<td>2: end for.</td>
<td></td>
</tr>
<tr>
<td>4: for each basic state and non-parallel control gateway do:</td>
<td>create an OR state as a parent of the node.</td>
</tr>
<tr>
<td>5: end for.</td>
<td></td>
</tr>
<tr>
<td>7: for each non-parallel control node do:</td>
<td>change all edge source and target nodes to the OR parent state.</td>
</tr>
<tr>
<td>8: end for.</td>
<td></td>
</tr>
<tr>
<td>Postconditions:</td>
<td>Model with OR states introduced for all data objects and non-parallel control nodes.</td>
</tr>
</tbody>
</table>

**Table 4-6: Synthesizing phase – OR composite states**

<table>
<thead>
<tr>
<th>Preconditions:</th>
<th>Model with OR states introduced for all data objects and non-parallel control nodes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for each edge connecting two OR nodes do:</td>
<td>if # of outgoing edges from source node = # of incoming edges to target node = 1 then</td>
</tr>
<tr>
<td>2: end if</td>
<td>remove the edge.</td>
</tr>
<tr>
<td>4: merge the two OR states into a new OR state</td>
<td></td>
</tr>
<tr>
<td>5: end if</td>
<td></td>
</tr>
<tr>
<td>6: end for.</td>
<td></td>
</tr>
<tr>
<td>Postconditions:</td>
<td>Model with composite OR states introduced for matched nodes.</td>
</tr>
</tbody>
</table>
The second sub-phase takes the composite AND/OR states generated in the last step, with the filtered process model generated in the first filtering phase to produce a hierarchical state chart. Each fork, join, and exclusive decision gateway is replaced with a fork, join, and choice pseudo state, respectively. Each edge connecting two nodes in the filtered process model is created to connect the corresponding nodes in the output state chart. This rule is explained in Table 4-8. A final post-processing step can be applied, if needed, to adjust transitions to the parent states.

**Table 4-7: Synthesizing phase – AND composite states**

<table>
<thead>
<tr>
<th>Synthesizing: Step 3: AND states</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>Model with OR states introduced for all data objects and non-parallel control nodes.</td>
</tr>
<tr>
<td><strong>1: for each</strong> fork (join) gateway with a set of OR nodes as output (input) do:</td>
</tr>
<tr>
<td><strong>2:</strong> if all the OR nodes has the same successor and predecessor nodes then</td>
</tr>
<tr>
<td><strong>3:</strong> create an AND state as parent of the OR states.</td>
</tr>
<tr>
<td><strong>4:</strong> create an OR state as parent of the AND state.</td>
</tr>
<tr>
<td><strong>5:</strong> replace the source/target of the child OR nodes with the new parent OR node.</td>
</tr>
<tr>
<td><strong>6:</strong> end if</td>
</tr>
<tr>
<td><strong>7:</strong> end for.</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
</tr>
<tr>
<td>Model with composite AND states introduced for matched nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synthesizing: Step 3: Constructing state chart</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>Model with OR/AND states introduced for all data objects and non-parallel control nodes.</td>
</tr>
<tr>
<td>Filtered process model.</td>
</tr>
<tr>
<td><strong>1: for each</strong> fork (join) gateway do:</td>
</tr>
<tr>
<td><strong>2:</strong> create a corresponding fork (join) pseudo state.</td>
</tr>
<tr>
<td><strong>3:</strong> end for</td>
</tr>
<tr>
<td><strong>4: for each</strong> edge in the filtered model connecting two nodes do:</td>
</tr>
<tr>
<td><strong>5:</strong> create an edge connecting the counterpart nodes in the hierarchical state model.</td>
</tr>
<tr>
<td><strong>6:</strong> end for</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
</tr>
<tr>
<td>Hierarchical state chart model.</td>
</tr>
</tbody>
</table>
4.2 Summary and Future Work

This research has been set to investigate the question posed at the beginning of the study; whether it is feasible to automatically generate a UML state chart model from a BPMN model that implicitly embodies an object life cycle. So far, the findings show that this is a non-trivial task indicated by the scarcity of related work addressing the same issue. An approach which targets activity diagrams was examined and found suitable for this project. By this time, the mapping and transformation rules are the main outcomes, and the implementation will be done in the next phase. An important finding is that the approach has defined situations where the transformation requires an extended rule; and others where it may fail due to the semantic differences of the two models.

For future work, the prototype system will be implemented based on the system architecture and transformation rules outlined in the previous section. The implementation of the transformation chains defined by the phases and steps will be done incrementally. The special situations where the transformation might fail will be outlined in the report. The final system will be evaluated and tested against a number of example and real case studies.
REFERENCES


A. APPENDIX A: Requirements Engineering

Requirements engineering refers to the field in software engineering concerned with the processes and activities of deriving and managing software requirements. These activities vary depending on the environment and domain of the system, however the following activities are commonly included as described in both [4][8]:

1. **Requirements elicitation**: communicating with stakeholders in order to understand the business domain and environment. Some of the techniques used to discover and extract user goals are interviews, workshops, prototypes and scenarios.
2. **Requirements analysis**: examining the requirements in detail in order to clearly understand the system. This often includes developing different conceptual models.
3. **Requirements specification**: writing the requirements in well-organized requirement document format to make sure there is no ambiguity.
4. **Requirements validation**: confirming the consistency of specified requirements and models, and mapping them to customer needs.
5. **Requirements management**: keeping track of any requirement’s changes and status update throughout the development.

In [7], these activities are split into two higher levels: requirements development and requirements management, as shown in Figure A-1. It should be noticed that these activities are often interleaved and implemented iteratively.

![Figure A-1: Requirement engineering activities](image-url)
B. APPENDIX B: Part of BPMN and UML metamodels

Figure B-1: BPMN 2.0 metamodel, Process class diagram [11]

Figure B-2: UML 2.4.1 metamodel, State Machine class diagram [12]