

A kernel language based exchange framework for behavioural modelling languages

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Ing. Christian Kletzander, BSc.

Matrikelnummer 01125210

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.Doiz. Mag. Dr. Manuel Wimmer

Wien, 31. August 2018

Christian Kletzander

Manuel Wimmer

A kernel language based exchange framework for behavioural modelling languages

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Ing. Christian Kletzander, BSc.

Registration Number 01125210

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.Doiz. Mag. Dr. Manuel Wimmer

Vienna, 31st August, 2018

Christian Kletzander

Manuel Wimmer

Erklärung zur Verfassung der Arbeit

Ing. Christian Kletzander, BSc.
Lagerhausstraße 15, 2265 Drösing

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. August 2018

Christian Kletzander

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn Priv. Doz. Mag. Dr. Manuel Wimmer, der meine Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Bedanken möchte ich mich für die zahlreichen interessanten Debatten und Ideen, die maßgeblich dazu beigetragen haben, dass diese Masterarbeit in dieser Form vorliegt.

Meiner Verlobten Jasmin Bernhart danke ich besonders für den starken emotionalen Rückhalt über die Dauer der Erstellung dieser Masterarbeit.

Abschließend möchte ich mich bei meinen Eltern Evelin und Dietmar Kletzander bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für meine Sorgen hatten.

Christian Kletzander, Drösing, 31.08.2018

Acknowledgements

At this point of the thesis I want to say thank you to all people who supported and motivated me writing this master thesis.

First, thanks to Mr. Priv. Doz. Mag. Dr. Manuel Wimmer who maintained and surveyed this master thesis. He gave me helpful suggestions and constructive criticism during creation of this thesis.

I want to say thank you to all persons who gave me ideas and started interesting controversies with me, which were necessary to create this thesis.

Moreover, lots of thanks to my affianced Jasmin Bernhart who gave me emotional support during the creation of this master thesis.

Last, I want to say thank you to my parents Evelin and Dietmar Kletzander, which had a friendly ear to me and supported me during my whole study.

Christian Kletzander, Drösing, 31.08.2018

Kurzfassung

In der Automatisierungstechnik werden verschiedenste Arten von Verhaltensdiagrammen innerhalb eines Unternehmens eingesetzt. Die Kompatibilität dieser Diagramme zueinander ist begrenzt, weil nur eine spezielle Menge von standardisierten Dateiformaten für den Import und Export unterstützt werden. Derzeit gibt es kein System, welches ein beliebiges Verhaltensdiagramm durch die Nutzung einer generalisierten Kernsprache in ein beliebiges anderes Verhaltensdiagramm transformieren kann. In der Literatur findet man Beispiele, welche einen speziellen Diagrammtyp in einen anderen vordefinierten Diagrammtyp transformieren können, jedoch sind hier die Möglichkeiten auf die vordefinierten Typen beschränkt. In dieser Arbeit möchte ich ein System entwickeln, welches den Intermediate Modelling Layer (IML) des AutomationML Konsortiums als Kernsprache zum Austausch beliebiger Verhaltensdiagramme benutzt. Durch den Einsatz einer Fallstudie möchte ich evaluieren, ob es zu einem Informationsverlust bei der Transformation mit der Kernsprache IML kommt. Zum Einsatz kommen hierbei zwei verschiedene Verhaltensdiagramme, das Aktivitätsknotennetzwerk (AONN) und GANTT. Nach der Transformation von GANTT nach AONN und retour kommt es zu keinem Informationsverlust. Wenn man AONN nach GANTT und retour transformiert, verliert man die Attribute: Verzögerung, spätester Startzeitpunkt, frühester Startzeitpunkt und spätester Endzeitpunkt.

Abstract

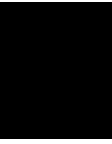
The interoperability for exchanging behavioural models between different tools in automation is only achieved by a small amount, which are supporting standardized import and export formats. There is no transformation framework existing for exchanging different behavioural models through a standardized kernel language. The literature describes several techniques to transform a modelling language into another pre-defined modelling language, but all of these are fixed to specific types of modelling languages and do not support the general exchange between any behavioural modelling language.

In this thesis, I introduce a new technique that allows exchanging a small amount of behavioural modelling languages through a standardized kernel language based exchange framework. I am using the Intermediate Modelling Layer (IML) from the AutomationML consortium as a kernel language for exchanging activity-on-node networks (AONN) into GANTT charts and back. By doing a case study based evaluation the generated input and output models of the different behaviour modelling types are analysed for possible information loss after exchanging them. The round trip transformation from GANTT to AONN and back has no information loss, whereas AONN to GANTT and back loses information attributes like delay, latest start time point, earliest start time point and latest end time point.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Motivation	1
1.2 Problem statement	2
1.3 Aim of the work	2
1.4 Methodological approach	3
1.5 Structure of the thesis	4
2 State of the Art	5
2.1 AutomationML	5
2.2 Information Integration Theory	9
2.3 Metamodeling	9
2.4 Model Transformations	12
2.5 Possible Kernel Languages for Modelling Languages	14
3 Methodology	17
3.1 Design Science Methodology	17
3.2 Used concepts	20
3.3 Methods and Models	21
3.4 Pivot Model	25
3.5 Formal Languages	27
3.6 Design Methods	34
3.7 Data Models	35
4 Implementation	37
4.1 IML-Based Exchange Framework	37
4.2 Behaviour Models	40
4.3 Transformations	47
4.4 Transformation Contracts	51

5 Case Study Based Evaluation	67
5.1 Using IML as a kernel language for exchanging behaviour models	67
5.2 Dealing with information loss	77
6 Related Work	83
7 Conclusion and Future Work	85
7.1 Conclusion	85
7.2 Future Work	86
List of Figures	87
List of Tables	88
Bibliography	91



Introduction

This chapter gives an overview about the problem domain. Moreover, it is important to mention the relevance for research work in this chapter. At the beginning, an introduction indicates problem fields, which are summarized in the problem statement. Furthermore, the goals are defined and the methodological approach is shown for reaching the desired goals. At last, the structure of the thesis is outlined.

1.1 Motivation

Since the first development of automated processes, an underlying automation language was needed. Today, many automation languages exist and the interoperability between these languages is only achieved to a limited extent. Interoperability between languages has been a problem since the second programming language was invented [Chi13].

In today's factories, groups of engineers exist, such as mechanical-, electrical- and software engineers, who require different views of information in their working area. This leads to the problem of having different data representations of information, which makes it difficult to exchange this information between the departments. Therefore it is necessary to combine different automation languages in the industrial area to support the different tools in an automated tool chain [BS82] [SAC⁺15].

A variety of behavioural languages exist, because of the usage of different tools and groups of engineers in industry. This leads to a lower interoperability between processes that are administrated over different departments in a company. The understanding of the various syntaxes and semantics of the used languages is one of today's problems. One approach to improve this situation is to develop a kernel language for behavioural modelling languages to close the gap between these different languages and support Interoperability [BTL⁺13] [BCW17].

1.2 Problem statement

However, with the solution of a kernel language based exchange framework the need for model transformations to and from the kernel language is given. Model transformations are used for passing information over syntactical, structural and semantical varieties of models. When a transformation is applied, a loss of information could happen. This is another open research area, which has to be analysed by doing a case study [HMPR04].

Furthermore, the input metamodels for the transformations have to follow a defined structure to guarantee an output model. Every behaviour metamodel has specific characteristics which make them different to metamodels of the same kind of language. This makes it necessary to describe a way of generalization on metamodels of the same language.

Moreover, behaviour models could be distinguished between task-oriented (GANTT), time-related (AONN) and event-driven (State Charts) approaches. To constitute a kernel language based exchange framework the three approaches must be generalized. This thesis is focused on exchanging the task-oriented and the time-related approach.

1.3 Aim of the work

This thesis will focus on the Intermediate Modelling Language (IML) from the Automation Markup Language (AutomationML) Consortium. It should be implemented as a standardized Ecore metamodel. The transformations should be written in the ATLAS Transformation Language (ATL).

The main goal should be the constitution of IML used in a kernel language exchange framework for behaviour models. The research interest behind this goal is the exchange of different input behavioural modelling languages over the pivot language IML and the evaluation of the resulting output models.

SG1: The first sub-goal is to develop a standardized Ecore metamodel for the IML. The definition of IML is done by the AutomationML Consortium and is currently in progress for a standardization. Moreover, it is important to mention that there could be problems arising due to specific constructs, which are not available in Ecore. In addition, problems in the definition of IML itself could be found and should be reported to the AutomationML Consortium.

SG2: The second sub-goal is to develop transformations from task-oriented charts and time-related charts to IML and back. This is realised by defining a GANTT and AONN metamodel, which is used for our transformations. In addition, this development should lead to the possibility of exchanging different types of approaches through IML.

SG3: The third sub-goal is the evaluation of IML as a standardized kernel language for exchanging behaviour modelling languages. This evaluation is done by doing a two part case study. The first part transforms a task-related behaviour model into a time-related

model over IML and back. The second part transforms a time-related model into a task-related model and back over IML. The result of this case study is an evaluation which analyses the difference between the input and the output model.

1.4 Methodological approach

To reach the expected results the methodological approach comprises the following steps.

In the first phase, it is necessary to analyse the problem domain by finding suitable metamodels for the given behaviour models. Furthermore, the analysis of these languages is important to understand the different semantics and conceptions of them. The major aspects of such behaviour models have to be identified in this phase [Mur08].

One important step in this phase is the study of the different definitions and representations of time in the chosen behavioural models. Every behavioural model uses a specific transition condition technique to switch from one state to another. For this case the different approaches for firing a transition have to be analysed. There is the possibility of having dependencies between states and conditions. Furthermore, some models could be event based. Some states use either their own timeline for their run or the global time.

In the second phase, generalized metamodels for every behavioural language must be constructed. This is important due the existence of different metamodels for a specific behavioural language. Furthermore, this leads to a more stable starting point for our model transformation into the kernel language. The construction of a metamodel follows the definition of a "Design as a Search Process" by searching for a potential solution in the solution space [HMPR04].

In the next step, possible relations and similarities between the different semantics of the formal languages have to be found. It is necessary to especially look on different concepts, e.g. the definition of a state, the dependency of states and the definition of conditional elements in the languages. Based on the findings transformations into the kernel language can be built. Moreover, in this generalization part similarities between the semantics and the resulting kernel language have to be noticed for our model transformations [EWSG89].

In this master thesis, I focus on the language "AutomationML". AutomationML is a registered consortium with the purpose to create a standardized data exchange format for engineering processes of production systems, for ensuring the interoperability between models. The consortium is interested in transforming behaviour models into AutomationML. For this purpose I choose the IML as a possible kernel language for finding associated model transformations for GANTT, AONN and State Charts [BY12] [DLP08].

Based on the kernel language IML, model transformations for transforming one behavioural modelling language into another over the kernel language, will be created. It is important to generate transformations from the generalized metamodels of the behavioural languages into the pivot language to support interoperability in the different behavioural modelling domains.

The third phase concentrates on the evaluation of the resulting kernel language based exchange framework. By doing a Case Study an evaluation of the possible use of a kernel language for model exchange is done by comparing the source and generated target models. To support my evaluation, I use the guidelines for conducting and reporting case study research in software engineering [RH09].

Finally, an iterative and incremental process is used. Therefore the resulting feedback from the previous iteration will be used to improve the results in the next iteration.

1.5 Structure of the thesis

Chapters 2 and 3 are describing the used concepts, methods and models, languages, design methods and data models. Chapter 4 is covering the practical solution concept and the realisation of theoretical concepts. In Chapter 5, the case study set up is described and their evaluation result. At the end, in Chapter 6 and 7 future work is discussed and an overall summary is done.

State of the Art

In this chapter the currently available techniques are described and delimited. These fundamentals are the basis of the implementation and will be used to found a solution for the problem domain.

2.1 AutomationML

AutomationML is a neutral standardized data exchange format for the use in industrial automation systems engineering. It was developed by the companies Daimler, ABB, Siemens, Rockwell, Kuka, Zhlke, netAllied and the universities of Magdeburg and Karlsruhe [DLPH08].

The engineering process and the involved engineering tools have become key factors for engineering quality and effectiveness as well as for the integration of the control system engineering within the overall engineering of modular manufacturing systems [LHK10].

Figure 2.1 is an excerpt of different software tools which are supporting the engineering process of manufacturing systems. The goal of AutomationML is a standardized, open format to become a better interoperability of tools. Normally in practice it depends on the engineering disciplines and project phases which tools are used. The data is stored in different files which cause inefficiency. This leads to an error-prone engineering process because the data has to be manually redrawn [DLPH08].

2.1.1 Intermediate Modeling Layer

IML acts like a layer between the target output format PLCopen XML and various input formats. It is possible to generate another output format than PLCopen XML. The IML shall be used for the description of sequencing and behaviour of different elements. The layer is displayed in Figure 2.2. The IML consists of the following elements [DLPH08]:

2. STATE OF THE ART

Anwendung	Untergruppe	Werkzeuge (Beispiel)	Anwendung	Untergruppe	Werkzeuge (Beispiel)
CAD		<ul style="list-style-type: none"> •CATIA v4, v5 •Autocad •UGS •SolidWorks •PTC ProEngineer •MicroStation •Blender •3d Max •Maya 	Enterprise Resource Planning (ERP)		<ul style="list-style-type: none"> •SAP R3 •Oracle Peoplesoft •MS Navision
			Reporting		<ul style="list-style-type: none"> •Cognos •Crystal Reports •Eclipse BIRT
Simulation	Material Flow Simulation	<ul style="list-style-type: none"> •Simple++/e-MPlant •Witness •Quest •ALB (Automatic Line Builder) Delmia 	Visualization	Mock-up	<ul style="list-style-type: none"> •e-MEngineer •many others
				Plant Visualization	<ul style="list-style-type: none"> •JViz •OpenGT •OpenFlight
	Robot Simulation	<ul style="list-style-type: none"> •Cosimir •IGrip D5 •Catia v5 Robotics •Robcad 		HMI	<ul style="list-style-type: none"> •WinCC / WinCC Flexible •intouch
	Process Simulation	<ul style="list-style-type: none"> •FEM Ansys 	Control Programming	PLC	<ul style="list-style-type: none"> •STEP 7 •RSLogix •RSLinx •CoDeSys •iMap
	Electrical Simulation	<ul style="list-style-type: none"> •PSPICE •Electronics Workbench •Multisim 		Robot Control	<ul style="list-style-type: none"> •ABB Robotstudio •KUKA SIM •Dürr 3D Onsite
Office	Text Processing	<ul style="list-style-type: none"> •MS Word •OpenOffice 	CAE		<ul style="list-style-type: none"> •Ruplan •ePlan •Eagle •Target 3001
	Spreadsheet Analysis	<ul style="list-style-type: none"> •MS Excel •OpenOffice 	Process configuration		<ul style="list-style-type: none"> •3D Onsite •Robotstudio •Robscan Design/Control •Bos 6000
	Presentation	<ul style="list-style-type: none"> •MS Powerpoint •OpenOffice 	Facility Management		<ul style="list-style-type: none"> •Bentley Microstation •Speedycon •Triplan •Autocad Architectural
	Databases	<ul style="list-style-type: none"> •Access •Oracle •MS Sql 	Computerized Maintenance Management System (CMMS)		<ul style="list-style-type: none"> •Maximo •Datastream7i •API Pro
	Communication	<ul style="list-style-type: none"> •Email 	Authoring		<ul style="list-style-type: none"> •Adobe Acrobat •Illustrator •Wiki •Excel •Sharepoint •MacroMedia
Project Management		<ul style="list-style-type: none"> •MS Project •MindManager 	Functional Engineering		<ul style="list-style-type: none"> •AutomationDesigner •Comos •Automation Framework
Product Data Management (PDM)		<ul style="list-style-type: none"> •UGS TeamCenter •Dassault Smartteam •Dassault Enovia 			
Product Lifecycle Management (PLM)		<ul style="list-style-type: none"> •e-MPlanner •Delmia E5 DPE 			

Figure 2.1: Different software tools used in the industrial area [Dra10, p. 5]

- *Header* (A set of all IML Elements belonging to the system)
- *State* (A state describing a stable situation within a system where a dedicated set of properties is valid)
- *State Transition* (A state transition is a change from one state to one or several next states by interpreting state transition conditions)
- *Activity* (An activity describes one or more operations related to certain states)

- *Selection Divergence* (A selection divergence is a logical association between one predecessor state and two or more successor state transitions. The successor state transitions can be regarded as having an XOR relation)
- *Simultaneous divergence* (A simultaneous divergence is a logical association between one predecessor state transition and two or more successor states. The successor state can be regarded as having an AND relation)
- *Selection convergence* (selection convergence is a logical association between two or more predecessor state transitions and one successor state. The predecessor state transitions can be regarded as having an XOR relation)
- *Simultaneous convergence* (A simultaneous convergence is a logical association between two or more predecessor states and one successor state transition. The predecessor states can be regarded as having an AND relation)
- *Event* (An event is fired by an activity to which it is associated. Events are mainly used as triggers for state transitions)
- *Variable* (A variable is a modelling entity that characterizes states and activities. Its value can be changed by activities; it can be used as trigger conditions for state transitions or as system input and output)
- *Comment* (Comment is used for descriptive information)
- *Additional Data* (Additional data allow storing information beyond the definitions)

2.1.2 PLCopenXML

The XML data format PLCopen XML specified by the PLCopen association is standardized and usable with IEC 61131-3 software projects. It can store all relevant programming information used in programming projects. With the usage of PLCopen XML it is possible to extend the standard with an own XML schema. This XML schema could be used to add additional data fields to the standard. The AutomationML uses the programming language Sequential Function Charts (SFC) and stores the information in the PLCopen XML format by using an own XML schema and mapping rules from the IML system to PLCopen XML documents [DLPH08] [PLC].

In the following example a IML state s is mapped to an PLCopen `<step>` element. It is possible to see the additional information for using an own XML Schema:

2. STATE OF THE ART

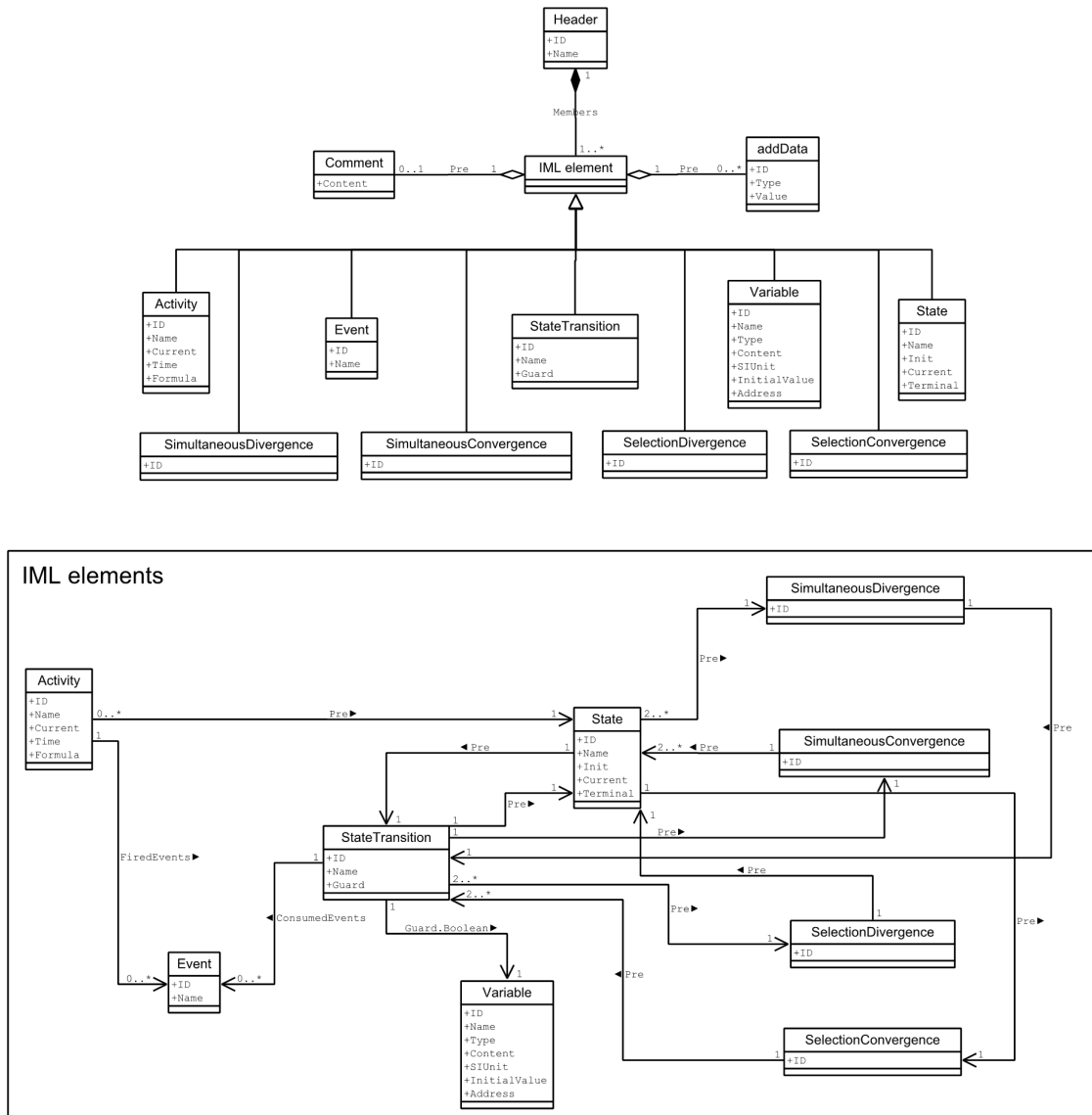


Figure 2.2: Overview of the IML [DLPH08]

```
<step>
  <addData>
    <data name="http://www.automationML.org/AML_addData.xsd">
      <AutomationML>
        <StateStatus> current </StateStatus>
      </AutomationML>
    </data>
  </addData>
</step>
```


2.2 Information Integration Theory

The *Information Integration Theory* was developed by Norman Anderson in 1981. Information Integration Theory is for describing how a person integrates information from a number of sources. Many sources lead to a complex judgement to find out which source the relevant information has [And81]. In order to distinguish between different sources, four interlocking concepts were developed:

- *Stimulus Integration*: Integration is the central concept of the Information Integration Theory. There are multiple stimuli available which could have a dependency to each other. Single stimuli are rare and could help in understanding information and give predictions or not. The integration theory studies the integration of different stimuli. In this case two central questions are arising: How are stimuli integrated to receive a response and what stimuli was more effective than another one?
- *Stimulus Valuation*: Stimuli differ in the valuation part and are classified in two types: physical and psychological stimuli. Physical stimuli are observations from experimental studies. In the Information Integration Theory psychological stimuli were used because they don't need any experimental study to get a result. Psychological stimuli are immediate causes of thought and behaviour. The valuation part has the purpose to change physical stimuli into psychological stimuli by using a validation operation. An example could be the scoring of a specific color. This specific color looks for example different to persons with a color decease.
- *Cognitive Algebra*: Researches have shown that stimuli integration often uses simple algebraic rules in the background. Some of these rules are averaging, subtracting or multiplying of stimuli information.
- *Functional Measurement*: Functional Measurement is the last basic concept of the Information Integration Theory. It uses the cognitive algebra with numerical representations of the stimuli. For this case the measurement of psychological values for the stimuli is important.

The *Functional Measurement Diagram* is shown in Figure 2.3. Stimuli or Sources **S** are processed through a validation operation **V** to receive their psychological values **s**. In the next step the different stimuli are combined by an integration function **I** into a response **r**. At last, the externalized response is processed through the response function **M** to receive the observable response **R**.

2.3 Metamodeling

A metamodel is obviously a model of a model. Therefore the model from which the first one is generated, is called a metamodel. A metamodel is important for defining the frame of the resulting model. It defines rules and constraints to help build up a model from the

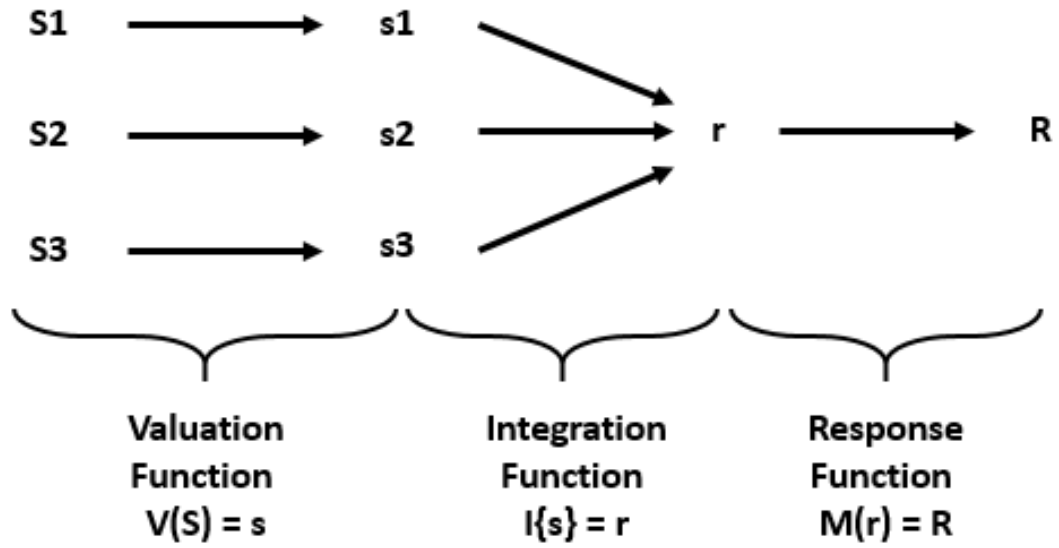


Figure 2.3: Functional Measurement Diagram

metamodel. Additionally, with this frame it is possible to have a validation of the resulting model. Because of this, language concepts and their grammar for the specification of models could be a definition for metamodels [BCW17]. This has many advantages like a normalized format for exchanging, the possibility to check the correctness of models and a small and accurate definition of the language concepts.

In the metamodeling world a 4-layer metamodeling Stack (Figure 2.4) is used. This stack is splitting metamodels in a Language Engineering and a Domain Engineering part. Every layer of this model defines another purpose. The meta-metamodel at the M3-Layer defines the meta language (e.g. MOF, Ecore). The metamodel at the M2-Layer conforms to the meta-metamodel and defines the Language (e.g. UML, ER). The Model at the M1-Layer conforms to the metamodel and represents the System (e.g. a UniversitySystem). The Model Instance at the M0-Layer conforms to the model and represents a System Snapshot (e.g. A UniversitySystem Snapshot).

2.3.1 Meta Object Facility (MOF)

The Meta Object Facility (MOF) is a platform-independent metadata management framework. It was developed by the OMG-Standard with the purpose of defining metamodels. The MOF is for defining language concepts for the modelling of object-oriented structures. There are some generalized concepts for object-oriented languages: objects are described by classes, objects have properties, there are relationships between objects and classes could be grouped by packages [(OM16)].

The MOF is a subset of the Unified Modelling Language (UML) and is similar to the

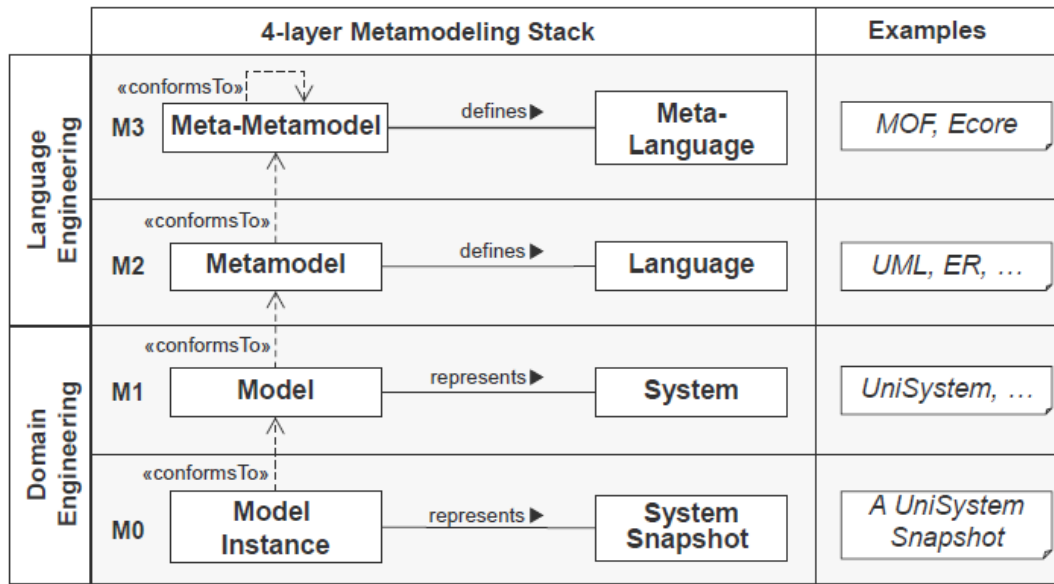


Figure 2.4: 4-layer metamodeling Stack [BCW17, p. 83]

class diagram of UML. Additionally, the metamodel of MOF is generated by limiting the UML metamodel with OCL constraints. Some differences are that no n-ary associations and no association classes are available. Furthermore it is not possible to have overlapping inheritances and interfaces. The major difference between UML and MOF is that MOF is for modelling a metamodel and UML for object-oriented modelling.

The major benefits of the MOF are:

- Simple modelling of metadata, because only important modelling constructs and notations are used from the UML class modelling
- The possibility to map MOF to other technologies like XMI, JMI, ...
- The tool support for modelling a metamodel is extended, because any UML modelling tool could be used for modelling metadata

2.3.2 Eclipse Modeling Framework (EMF)

The Eclipse Modelling Framework (EMF) is a modelling framework for Eclipse [Ste08]. The EMF is not as high complex as other modelling frameworks. The model behind the EMF is called Ecore (Figure 2.5). The Ecore is the meta-metamodel of the EMF. The Ecore meta-metamodel is based on the eMOF (Essential Meta Object Facility) which is a smaller subset of MOF for building metamodels. The aim of Ecore is the mapping of

the eMOF to Java. Because of this it is also called the Java-based implementation of eMOF [BCW17].

The Ecore model (Figure 2.5) is similar to the UML but more simplified. In fact the Ecore model is a simplified subset of the full Unified Modelling Language. The EMF only supports class modelling.

The Eclipse Modelling Framework supports a variety of features including a modelling environment, code generation, various modelling frameworks and im- & export functionality.

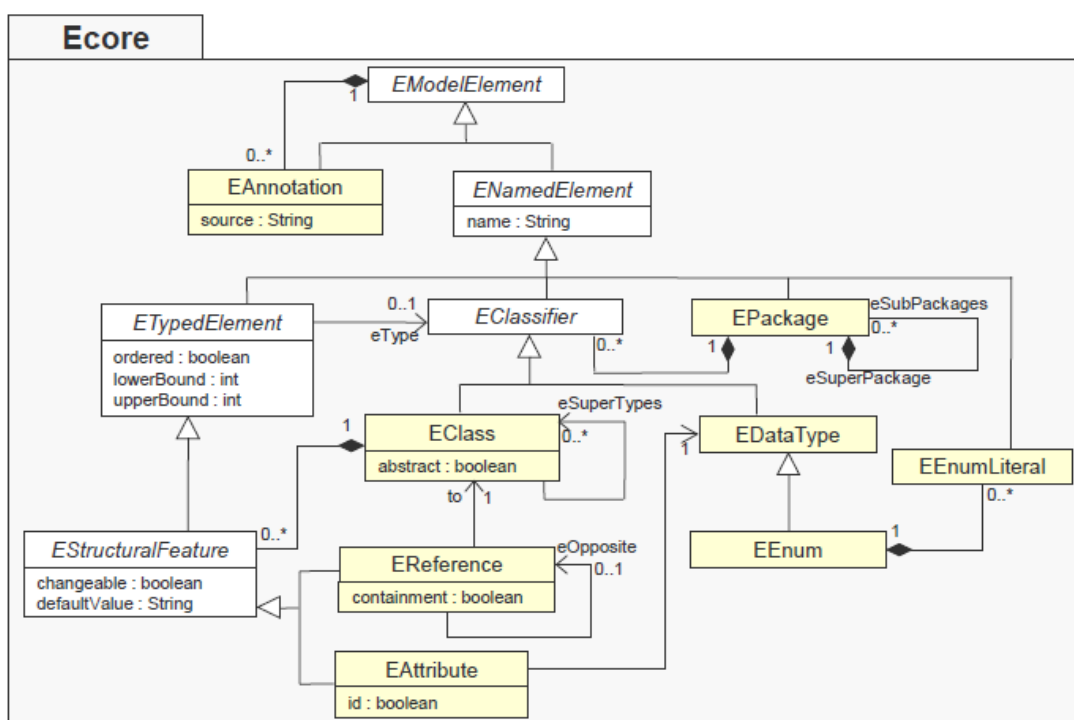


Figure 2.5: Ecore modelling elements at a glance [BCW17, p. 93]

2.4 Model Transformations

There are two types of model transformations: The *Model-to-Model* and the *Model-to-Text* transformation. The Model-to-Model transformation transforms either a Model X into a Model Y or alters a previous Model. On the other hand, the Model-to-Text transformation transforms a Model X into a textual language.

There are five possible methods of Model-to-Model transformations [BCW17]:

- *merged* (to homogenize different versions of a system)

- *aligned* (to create a global representation of the system from different views to reason about consistency)
- *refactored* (to improve their internal structure without changing their observable behaviour)
- *refined* (to detail high-level models)
- *translated* (to other languages/representations)

There are existing two types of approaches for Model-to-Model transformations [BCW17]. The first one is called *out-place* or *endogenous* transformation. This transformation creates a new model from the scratch up whereas *in-place* or *exogenous* transformations are for altering a model. Last-mentioned have the possibility to create, delete or update elements from the input model. Figure 2.6 explains the difference in a graphical representation.

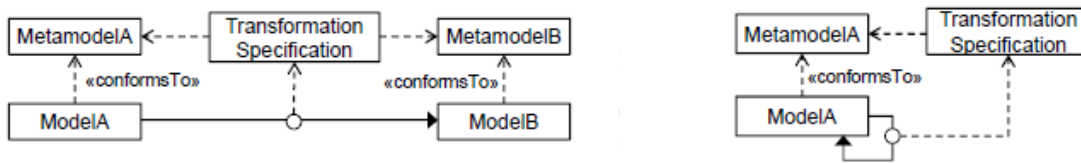


Figure 2.6: Out-place transformation (left), in-place transformation (right) [BCW17]

2.4.1 ATLAS Transformation Language (ATL)

ATL is an Out-Place or endogenous transformation technique. In fact that the source and the target models are distinct, two attributes are very important. A source model is read-only and a target model is write-only [BCW17].

ATL is declarative-imperative hybrid but the recommended style is declarative. In the declarative part you can use so called *Matched Rules* (Figure 2.7) with the support of traceability. This is possible by using the Object Constraint Language (OCL). If you want to use a declarative rule, it is important to use a source pattern to be matched in the source model and a target pattern to be created in the target model for each match during rule application. If needed it is possible to use a optional action block for a sequence of imperative statements.

The imperative part includes Called/Lazy-Rules, Action blocks, Global variables via Helpers. To call an imperative rule it is necessary to give them a name, if needed some

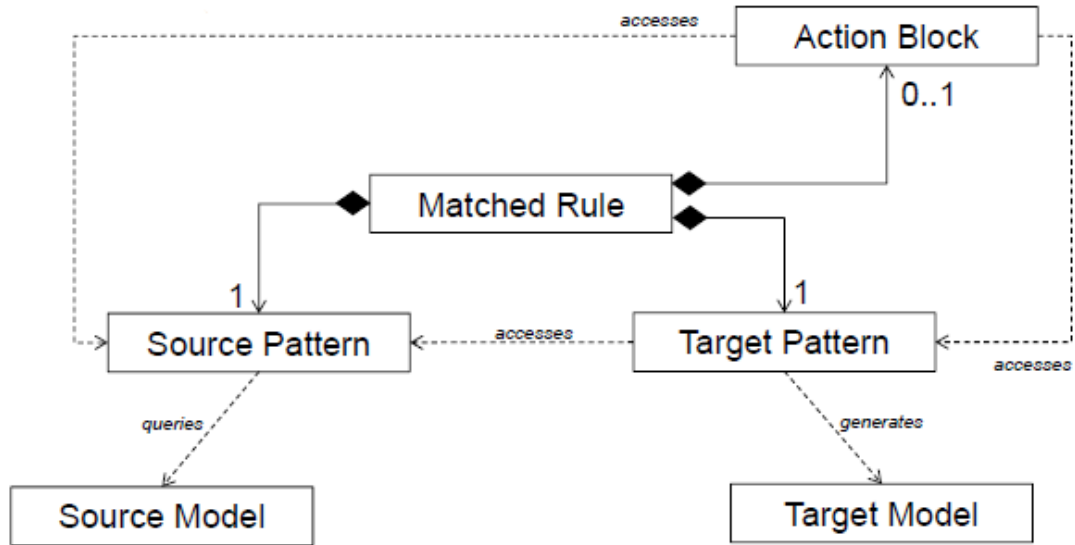


Figure 2.7: Syntax of a Matched-Rule in ATL [BCW17]

arguments and it has to contain a declarative target pattern and/or an optional action block.

An important fact concerning rules in ATL is that Matched-Rules are only applied once for each match and Called/Lazy-Rules are applied as many times they get called from other rules.

One of the biggest advantages of ATL is the so called Implicit Trace Model. This model consists out of the following phases [Ran11]: the module initialization phase, the matching phase and the target initialization phase.

In the *Module initialization phase* the module variables and the trace model gets initialized and Called-Rules for the entry point are executed. After the initialization phase the *Matching phase* is executed. In this phase the source patterns of the Matched-Rules are used to select the needed elements of the source model. Once the source patterns are selected, target elements are created by iterating through all assigned target patterns. In every iterating step a traceability information is stored.

The last phase is the *Target initialization phase*. The created elements get initialized based on the bindings. Finally, the optional imperative code from a do block or possible calls to Called-Rules were executed.

2.5 Possible Kernel Languages for Modelling Languages

Currently some languages exist, which may be classified as kernel languages for behavioural modelling, such as fUML, Kermeta and Petri nets. The focus of this chapter is on an

evaluation of possible kernel languages for modelling languages.

2.5.1 fUML

The Executable Foundational UML (fUML) is a computationally complete and compact subset of the UML metamodel, designed to simplify the creation of executable UML models. Additionally, fUML selects two major systems of UML, the usage of classes and activities for behaviour modelling of such a system. Moreover, fUML is using an own library called *Foundational Model Library* for implementing primitive data types and primitive behaviour operations on these data types. The use of fUML as a kernel language leads to problems, such as supporting only a small subset of UML, which exclude specific chart types of UML for behaviour models. This is caused by the complex semantics used in fUML. Finally, fUML has its focus on executing models and not on exchanging different types of input models. [LLP⁺09] [(OM17)].

2.5.2 Kermeta

Kermeta is known as a kernel language, which integrates concepts from languages like MOF, OCL and QVT. It uses the strategy of having different concepts in one metamodel by using instances of these defined models for our transformations. Kermeta is more useful for transforming databases of different language implementations by using an object-oriented programming language [MFV⁺05].

2.5.3 Petri-Nets

Another form of behaviour modelling is the Petri-Net formalism. The Petri-Net is a modelling technique for sequential and concurrent procedures for distributed systems with the feature of modelling parallel operations. There research on how Petri-Nets have been done could be used to represent sequential and concurrent behaviour models. The result has shown that extending Petri-Nets is possible for several behaviour processes but they could get quickly complex in terms of size [BDK01].

Methodology

In this chapter the methods, patterns, theoretical models and paradigms used for the thesis will be described. Moreover, this chapter discusses the methodologies and tools for reaching the desired goals.

3.1 Design Science Methodology

First of all, I will look on the *Design Science Methodology* which is overall used as the base methodology for this thesis. Furthermore, when using this methodology, we have to speak about a design science project. A design science project, consists of two components:

- *Artefact* (is an object to study in a specific context)
- *Activity* (is for designing and investigating the artefact in a context)

Moreover, a design science project is the study of an artefact interacting with a problem context. In this thesis solutions for the existing problems should be found. In most cases these problems got fixed by improving an existing artefact in a specific context. Additionally, it is important to understand the context itself, to find solutions for the design problem. Sometimes an artefact could handle a specific context without any problem but in some cases the interaction between an artefact and a context have to deal with problems. In contrary, the study of artefacts and contexts alone is not expedient, because in most cases they have to deal with each other. Finally, the design science process helps to find and solve occurring problems, when dealing with an artefact in a specific context [Wie14].

Figure 3.1 shows a graphical representation of the design science process. The artefact interacts in a specific way with a context. Everything known about the artefact is

a possible design for our problem domain. In contrary, the context is an empirical knowledge, because information about the context itself and the interaction with the artefact and the context have to be gained during the process. At last, the knowledge gain and problem resolving mechanism is a living process. By iterating over difficult contexts, problems should be fixed or solved in an iteration with another context.

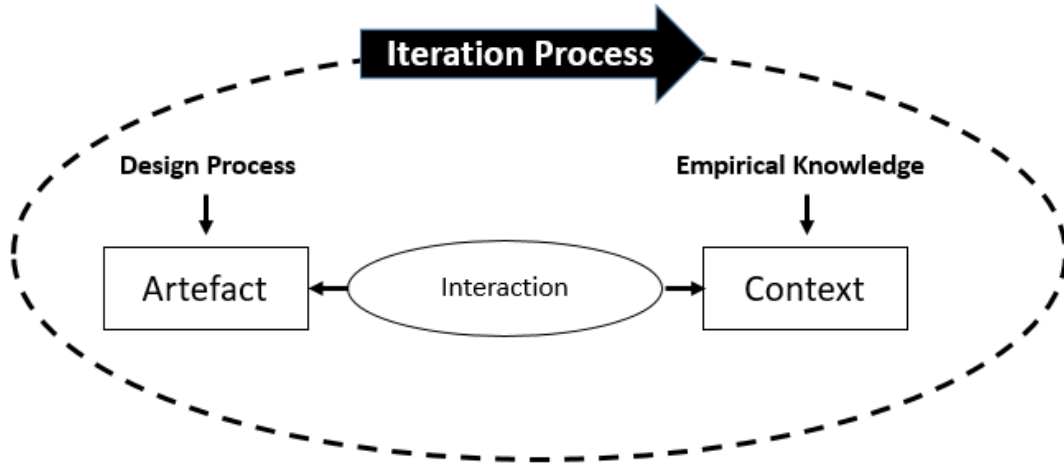


Figure 3.1: Graphical representation of the design science process

In this thesis my artefact is IML, which should be used as a pivot language for exchanging behaviour models. It is important to bring the possible pivot language into different types of contexts, like developing of a metamodel, writing transformations for model exchange and information loss.

The activity component of the design science process is splitting up in the *design activity* and the *investigate activity*. The design activity defines the goals of the researching part for the project. The investigate activity has to build up knowledge about the used context to contribute with it. These two types of activities lead to the following two kinds of research problems in the process, namely the *design problem* and the *knowledge questions*.

Design problems want to change something in the real world. First, an analysis of the designated goals is required. The solution of a design problem is a design and at least there could be more than one solution.

Knowledge questions ask for knowledge about the world as it is and do not want to change something in the real world, like a design problem. In difference to design problems, knowledge questions are assumed to have only one answer. This answer could not be true in all cases and can have a specific uncertainty, because of the possibility to be wrong under some assumptions.

There are two types of knowledge questions. The first one is the *empirical knowledge*

question, which requires data about to world. The second one is the *analytical knowledge question*, which uses mathematics or logic, without the usage of data about the world. In this thesis I focus on the usage of empirical knowledge questions.

Additionally, there are two types of empirical knowledge questions available. The *descriptive questions* focus on the fact of what happened without asking for explanations. It is like using the *Five Ws* to answer these kind of questions [ZWA⁺13]. Another type is the *explanatory question*, which ask why something happened.

The existence of these two kinds of problems lead to the existence of an iterative process in a design science project. Moreover, when a design problem focuses on a change of the real world, knowledge questions could help to achieve this change by asking something about the real world and the resulting information is returned to the design problem. I must also take into account that knowledge questions could generate new design problems. The design activity generates different new designs, which have to be passed back again to the knowledge questions.

In this thesis the global research questions for the design process are the following:

- What concepts constitute a kernel language for behavioural modelling?
- What kind of benefits may be realized by using the kernel language?

When a knowledge question is answered, the so called *knowledge context* is enriched with new information. First, the knowledge context is build up by the knowledge of the author including theories and methods from former studies and could only include information available yet. This knowledge is called *prior knowledge*. Some kind of prior knowledge is:

- *Scientific literature* (scientific theories from existing studies)
- *Technical literature* (specification of other artefacts used in a problem context)
- *Professional literature* (documentations from vendors, magazines including information about the artefact)
- *Oral Communication* (talking with colleagues)

Additionally, when information is not available in prior knowledge, research on the topic has to be done. The resulting knowledge of a design science project is called the *posterior knowledge*.

3.2 Used concepts

3.2.1 Object-oriented concept

The object-oriented concept is a common method to minimize complexity of a software system. Additionally, during the development of the object-oriented concept for software systems, the object-oriented programming approach, in short OOP, was developed. The main goal of this concept is to help understanding complex software systems or complex structures. Object-oriented software development is yet not the only concept available to handle complexity of software projects. The importance of this concept is the existence of many application fields for solving problems. Moreover, it could be combined with many software development approaches [LR11].

The concept supports two main concepts:

- *Classes* (a class defines the data format and available methods of an object)
- *Objects* (an object is an instance of a class)

This leads to the first major feature of OOP named *data encapsulation*. Data encapsulation is separating the data and the method part of a class. Accordingly, the direct access to data of an instantiated class is not allowed. If another object wants to read or write data to our object, it must use a predefined interface. One benefit is the consistency of data is organised by the object itself. Another benefit is the changeability of data definitions in a class. When the underlying data structure of a class is changed, only objects instantiated from the changed class have to be adapted. Objects which use the interfaces of a changed class must not to be adapted because the changed class has to supply the same type of information when accessing a method. This is a good example for reducing complexity of the software system.

Another feature of OOP is *polymorphism*. Polymorphism means to specify a method on a class and every instantiated object of this class or their subclasses has to implement the method. Whenever the method is called on an object of the parent class or a subclass, the function is available and implemented in a specific way. This leads to an abstraction from the class hierarchy.

Thirdly, *inheritance* enables the feature of deriving a class from a base class, whereas the derived class can use methods and attributes from the base class. The derived class can use existing constructs from the base class or overwrite these with an own implementation.

An object consists out of data, which defines the state of an object and the functionality implemented in operations. The operation call of an object is called the *sender-and-receiver* pattern. An object is sending information to another object by calling an operation. Therefore, in some cases the object is waiting for an answer or not. This leads to the existence of having a visibility property set on the data inside an object. This property could have the following values:

- *private*: The operation is only accessible from the object itself or the default value
- *public*: The operation is available for everyone

A class describes the structure of similar objects. The structure of a class consists of properties and operations. Moreover, to define the structure of a class, a classification on related objects have to be done. In this classification process a generalisation of the properties and methods is done.

3.3 Methods and Models

3.3.1 Modelling a metamodel

First, information about a metamodel itself and the process of generating a metamodel is described. A quick overview is given in Section 2.3 about the 4-layer modelling architecture of metamodels.

Metamodels define language concepts and their grammar for the specification of models, as described in Section 3.5. The prefix *meta* is translated from the Greek language and means *about*. In short, a metamodel knows something about other models.

Using a metamodel has one big advantage, the validation of models. Models generated from metamodels have to conform to the definitions of the metamodel. In summary, checking the correctness of models is possible. Another advantage is the precise and concise definition of the language concepts (Section 3.5) in the metamodel.

The next point is the exchange format. Metamodels are exchangeable, because of the serialization syntax. Moreover, this exchangeable format helps to store metamodels in repositories and allows them to get stored in versions. At last, metamodels are extensible. They could be easily extended with profiles.

The *Metamodel Development Process* describes the way of creating a metamodel in an incremental and iterative way.

First, in the *Modelling Domain Analysis* part it is necessary to sketch modelling examples from the same modelling language. The more different examples are existing, the better is the outcome of the metamodel, because more information is given for the identification process. In the identification process the purpose, possible realizations and the different content of the modelling language is analysed. In this process similarities between the different example models should be demonstrated.

In the second part, which is called the *Modelling Language Design*, a formalization process is starting on the results of the previous modelling domain analysis. Moreover, after formalizing the modelling language, the modelling constraints (f. e. OCL) have to be defined. They help to improve the resulting metamodel by validating reference models from the metamodel.

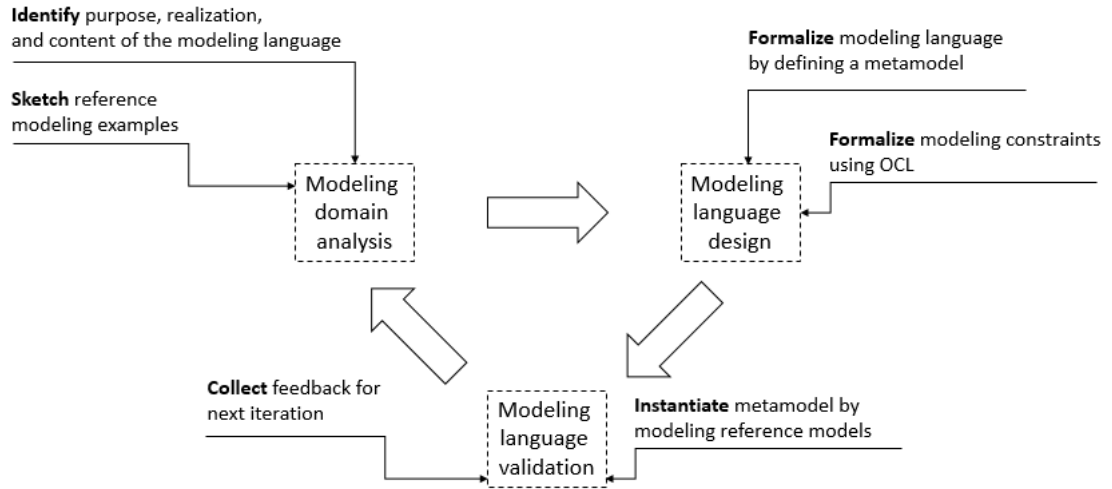


Figure 3.2: Metamodel Development Process

The last part is the *Modelling Language Validation* where the metamodel is used to generate reference models. By creating such models, the validation process of the metamodel will check the integrity of the reference models. In this part the most problems from the formalization process were found. Every problem is collected as a feedback and given back to the first part, the modelling domain analyses. The metamodel development process is an iterative process with a feedback loop to build a metamodel without any lack of information.

In short, the metamodel development process determines the modelling concepts, the properties of the modelling concepts and defines an object-oriented design of the language. At next I will focus on an example for building up a metamodel for the AONN.

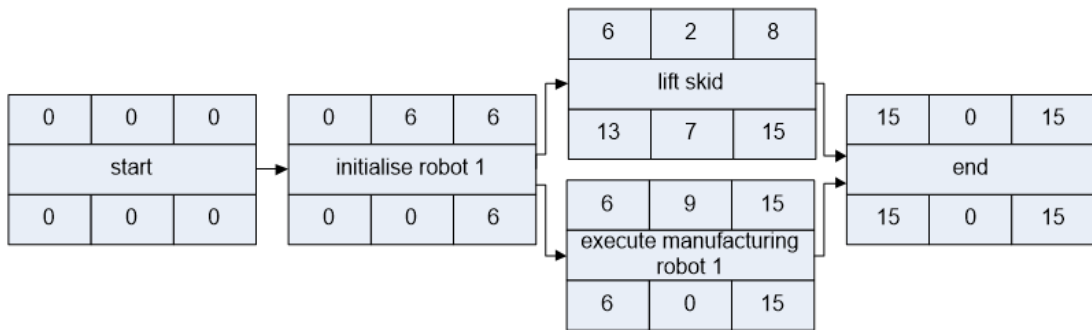


Figure 3.3: Example model of an AONN [DLPH08]

In Figure 3.3 an example model of an AONN is shown. The example includes the following nodes: Project, Activity, ActivityTransition. First, it is necessary to generate a notation table is generated to get an overview of the used concepts. In Table 3.1 I

constructed the notation table of the example model from Figure 3.3. The first used concept is the activity. An activity is some kind of node, storing information about time behaviour of the element and a name. The second concept is the transition between two activities which is displayed as an arrow. At last, the project container itself is defined with a name of the generated model.


Syntax	Concept									
-	Project									
<table><tr><td>Earliest start</td><td>Duration</td><td>Earliest end</td></tr><tr><td colspan="3">Node name</td></tr><tr><td>Latest start</td><td>Buffer</td><td>Latest end</td></tr></table>	Earliest start	Duration	Earliest end	Node name			Latest start	Buffer	Latest end	Activity
Earliest start	Duration	Earliest end								
Node name										
Latest start	Buffer	Latest end								
	Transition									

Table 3.1: Notation table of an AONN

The next step of the metamodel development process is the construction of a *property concept*. This property concept table extracts the information about intrinsic properties and relationships of the modelling concept. The intrinsic properties are describing concepts whereas relationships are needed to find out which concepts are connected and under which constraints. In Table 3.2 the concept properties for our example are shown.

Concept	Intrinsic properties	Relationships
Project	Name	Unlimited number of <i>Activities</i> and <i>Transitions</i>
Activity	Name Earliest Start Duration Earliest End Latest Start Delay / Buffer Latest End	Incoming and outgoing <i>Transition</i>
Transition	-	Source <i>activity</i> and target <i>activity</i> Nodes: Activity

Table 3.2: Concept properties of the AONN example model

In the third step, the object-oriented design of the language is created. In this step the headers of the property concept Table 3.2 are swapped. The concept is turned into a *Class* and the intrinsic properties- and relationships columns are transformed into *Properties*. Moreover, this switch generates the basis of the metamodel. Figure 3.4 shows the generated metamodel.

After defining the metamodel of the AONN, it is now possible to create an abstract syntax

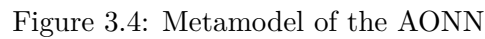
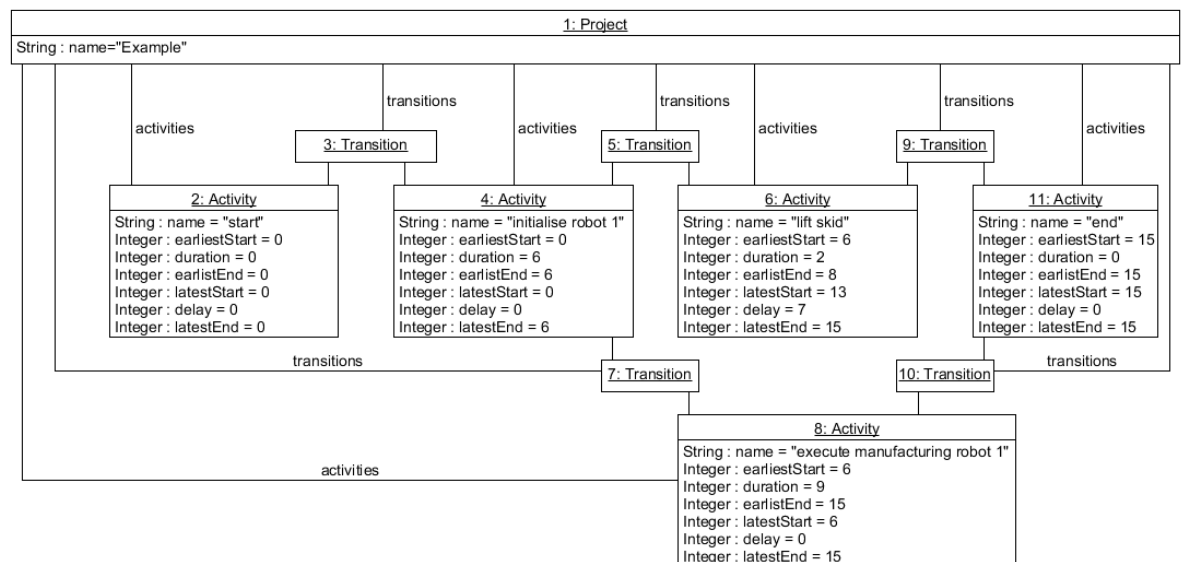


Figure 3.5: Abstract syntax of the example model by using the defined metamodel



In summary, the most important point of the metamodel development process is the iterative way of building up a metamodel out of an existing model. Every phase of the process has its primeness. The next step will get better with the feedback of the previous step and improve the result while it is finished.

3.4 Pivot Model

A language could be transferred into another by writing transformation rules. The existence of many languages, lead to a higher maintainability of these transformations. By changing a simple thing in the structure of a language, all connected transformations must be adapted. An approach to overcome this problem, is the usage of a pivot language or also called an intermediate language [WW07].

A *pivot* is described as an interpreter in simultaneous interpreting, which translates a language A into an intermediate exchange language. The intermediate exchange language forms the basis for transforming languages into another. This pivot model has the ability to transform the language A into any kind of language which supports the pivot language. The pivot language can be seen as an intermediate exchange layer [Mat07].

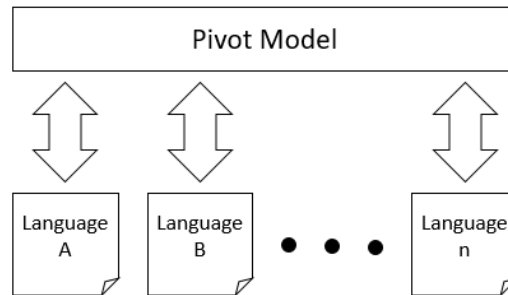


Figure 3.6: Language exchange over a pivot model

This intermediate exchange layer should be developed after a transformation classification. In our case I look especially on behaviour model transformations. For this case a pivot model has to be build up which supports the concepts of behaviour modelling. When building up a pivot language the usage of the set theory is recommended. There are some principles of the set theory which are useful for the concept phase of a kernel language [Vau95].

The *union* is for merging all information into one single model. There are similar parts in all the classified languages which could be generalized in the pivot language. This generalization is for implementing the basic functionality in the pivot model.

Furthermore, important constructs for generalization are the *intersection* and the *difference*. There are existing languages which are including special constructs. These must be modelled in the pivot model for not losing this information during a transformation

process. One option is to model an additional data construct for storing these special information or to integrate it into the pivot model for reusing the concept in other languages.

3.4.1 Model Transformations

A key technology of model-driven engineering is the model transformation. Model transformations are important for models which have to be kept consistent with another model over time. To overcome this problem, bidirectional transformations were used[HTCH16].

A system consists of many parts whereas some parts are in some kind of a relationship to each other. This relationship is because they use the same semantic or match in syntactical ways. These parts are a single piece of the whole system or a part which evolves in combination with another part of the system. The key fact is to hold the system in a consistent state. The little helper used to hold connected parts consistent are called *transformations*. There are two types of transformations, the *unidirectional* or the *bidirectional* transformation.

A unidirectional transformation means the creation of a target model from a given source model. In this case manual modifications on the target model will be ignored and are lost. An automatic system is needed to check consistency on the target side.

A bidirectional transformation has the purpose to hold up consistency between two parts. As an introduction in bidirectional transformations the concept of having a set of *source* artefacts named S and a set of *target* artefacts named T must be introduced. A unidirectional transformation creates a target artefact from a source artefact ($S \rightarrow T$).

The difference between bidirectional and unidirectional transformations is described in an additional operation, the reflection of updates from the target model back to the source model (Figure 3.7). As defined, two artefacts $s \in S$ and $t \in T$ are consistent if $t = get\ s$. The reflection of updates from the target model is defined as $put\ S \times T \rightarrow S$, which takes the original source $s \in S$ with the updated target t' to update the source $s' = put(s, t')$.

After defining a transformation and the existing transformation types between a source and a target model, I will explain the concept behind transformations.

As seen in Figure 3.8 the *transformation definition* has to be declared before a transformation could be done. The transformation definition is written in a *transformation language* (f.e. ATL, QVT, F-Alloy,...). The transformation definition declares rules for transforming a source metamodel into a target metamodel in strict accordance with the underlying relationship of the source and target metamodel [CH06].

After creating the transformation definition, an instance of the source metamodel, the *source model*, can be transformed into a *target model*. The resulting target model must be an instance of the target metamodel. Every instance of a metamodel has to strictly satisfies the requirements of the corresponding metamodels.

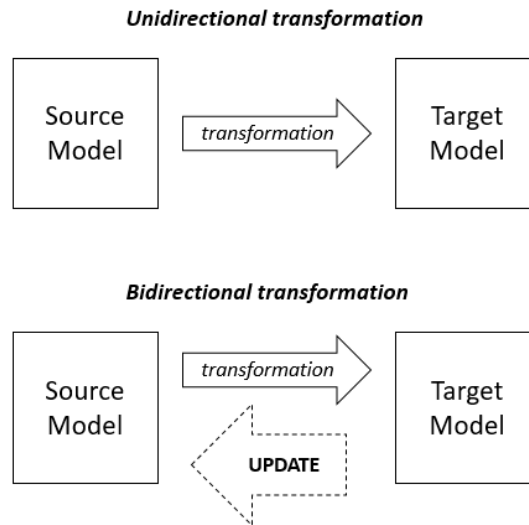


Figure 3.7: Difference between unidirectional and bidirectional transformations

3.4.2 EMF Profiles

Domain-Specific Modelling Languages (DSMLs) help to reduce complexity in software development by preparing modelling languages for designers to their application domain. A big problem with DSMLs is the difficult adaptability to fit into the domain. Moreover, the abstract syntax, the concrete syntax and all related artefacts of the modelling environment re-created or adapted. To overcome these problems, EMF Profiles were introduced in the Eclipse environment [LWWC11].

EMF Profiles use the strategy of lifting the metalevel by inheritance. This allows to directly instantiate profiles by inheriting instantiation capabilities. Moreover, the stereotype «*inheritsFrom*» must be applied to the metamodel profile. As seen in Figure 3.9 the metaclass *Stereotype* in *Profile MM* is a subclass of the meta-metaclass *EClass*. This is required due to EMF. EMF allows only instances of the meta-metaclass *EClass*, residing on M3 to be instantiated on M1.

3.5 Formal Languages

First of all, the anatomy of formal languages is introduced. They are consisting of four components [BCW17]:

- *Abstract syntax*: The abstract syntax is for defining the language concepts and the combination of these concepts. The language concepts are described as data types. The abstract syntax could be seen as the grammar of a specific language. It does not define the notation or meaning of the concepts.

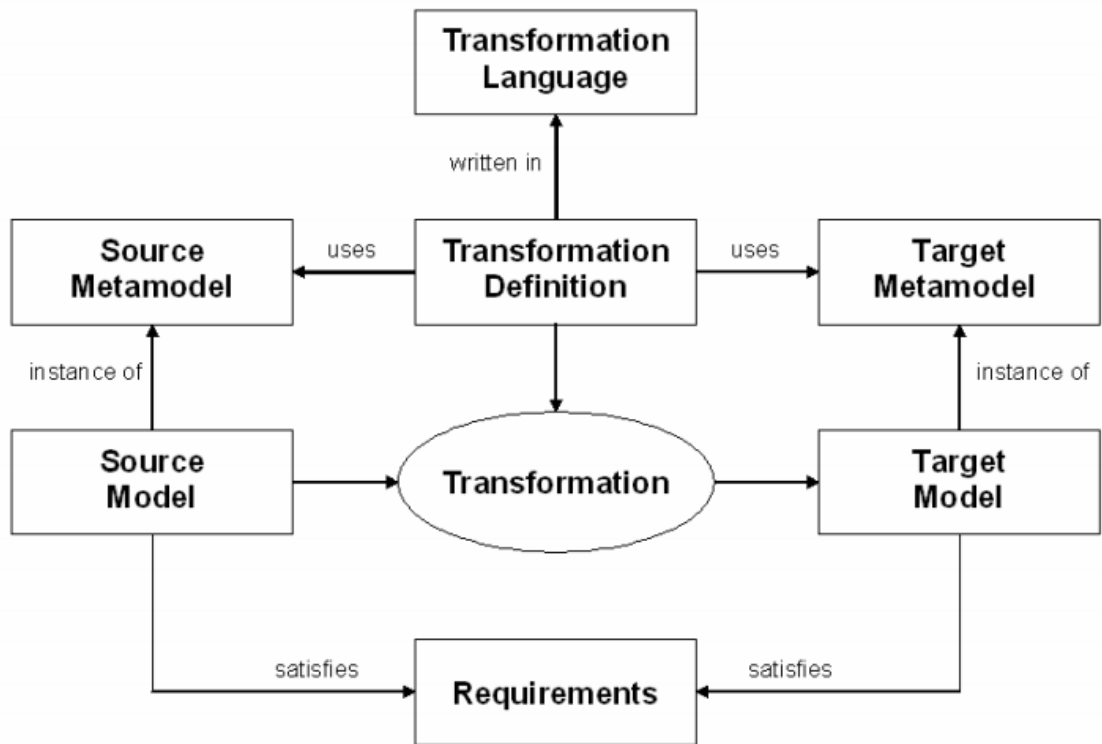


Figure 3.8: The abstract concept of transformations [CH06]

- *Concrete syntax*: The concrete syntax defines a notation for illustrating the language concepts in two ways, textual or graphical.
- *Semantics*: The semantics defines the meaning of the language concepts. Moreover, it defines how these language concepts are interpreted.
- *Serialization syntax*: The serialization syntax is for model exchange between tools and also for persistent storage (f. e. XML, proprietary format, ...).

In the anatomy of formal languages, the main components could be extended with additional components, which are:

- *Extension*: Extension of a language by new language concepts (f. e. UML Profiles).
- *Mapping*: Mapping is acting as a translational semantic definition to add a mapping to other languages or domains (f. e. UML2Java).

3.5.1 Object Management Group (OMG)

The Object Management Group is a non-profit organisation for the IT-Industry. Every kind of organisation can participate in the OMG. The Board of directors is a council

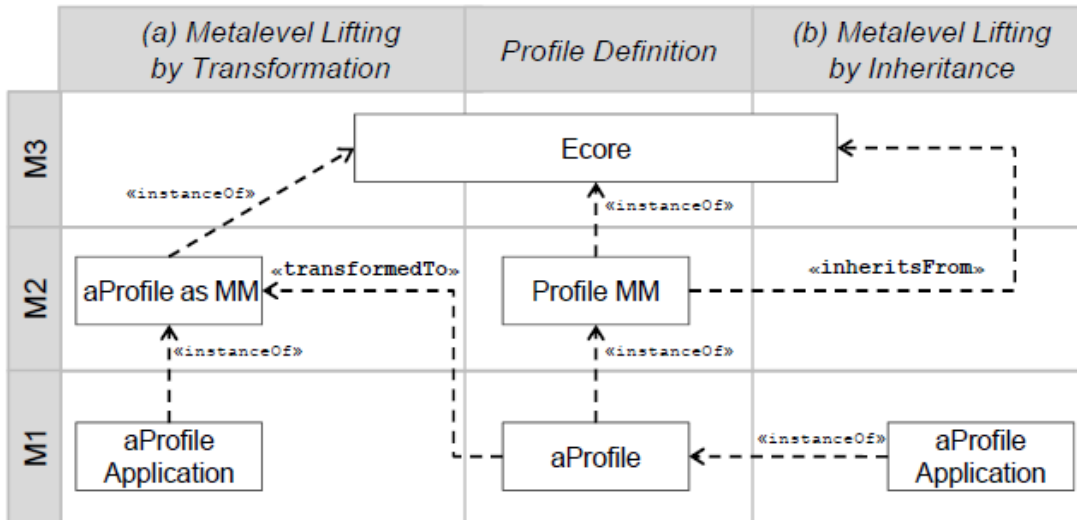


Figure 3.9: EMF Profile Architecture [LWWC11, p. 5]

consisting of the worlds best company's like IBM, Apple or Sun. Since 1989 the OMG developed many standards in the IT like UML, CORBA or MDA. The development of such standards is done by a Task Force. In the Task Force every organisation has the same voting power no matter what member size it is. Every suggested standard is available for the public and many of the suggested standards get ISO certified [(OMa)].

3.5.2 Model-driven architecture (MDA)

The Model-driven architecture includes many software design approaches for developing software systems. The MDA was launched by the OMG in 2001 and is standardized. In the MDA the construction of a software system is split up into four layers (Figure 3.10 [(OMb)]).

The first layer is the *Computation Independent Model (CIM)*. In this type of model the different types of activities for reaching the desired goals are constructed including functional and non-functional requirements. This layer should fit the interests of the developers and the users. It is very important to choose a modelling language both can read. At most the Use-Case or the Activity diagram from the UML standard is used for modelling this task. CIM includes the domain-specific language (DSM).

The second layer is the *Platform Independent Model (PIM)*. This layer describes the needed technical specifications without using a specific technology, programming language or middleware. It is important to construct models which are platform and company independent. The PIM is operating on a higher level of abstraction to model specific techniques more generally. For this purpose UML Profiles help to use stereotypes in modelling specific techniques.

The third layer is the *Platform Specific Model (PSM)*. The purpose of this layer is to describe the used technology by using the PIM. For example, a persistence layer with hibernat could be generated, which defines specific UML Profiles for the defined stereotypes of the PIM.

At last, the code layer is for generating the source code of the goal platform by using the defined techniques in the PSM.

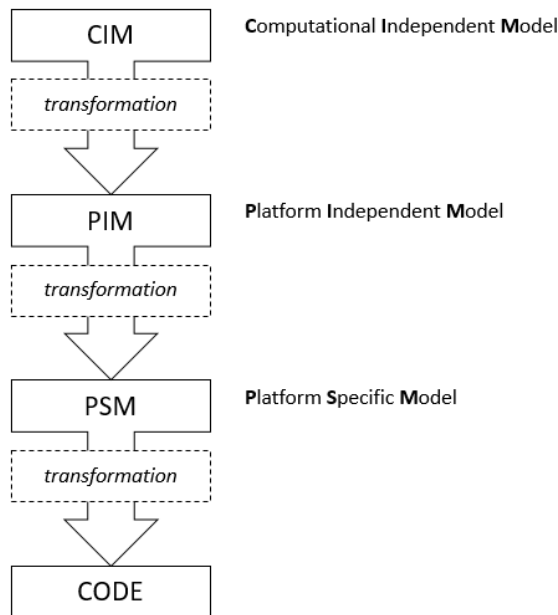


Figure 3.10: Graphical representation of the MDA layers

Moreover, the separation of concerns leads to a platform-, language- and system-independent model. To sum up, the MDA defines the structure, semantics and notations of models using industry standards. These models are called *MDA Models* and are used to create specifications for the acquisition, system and technology. In addition, they produce the documentation part of such a system and makes it executable. The main goal of MDA is to lower the complexity of models and to simplify the interdependency between other systems with the requirement of not using any values from the source model. In Addition, the MDA distinguishes between two types of model transformations:

- Transformations from a model into another Model used by MOF or QVT
- Transformations from a model into a programming language

A transformation uses the elements of an input model and generates the elements of an output model by using a specific set of rules.

3.5.3 Meta Object Facility (MOF)

To sum up the information from Section 2.3, MOF is a standard to generate metamodels by using concepts from the UML class diagram. It uses classes, attributes and associations. In Figure 3.11 the UML metamodel of the class diagram is shown [(OM16)].

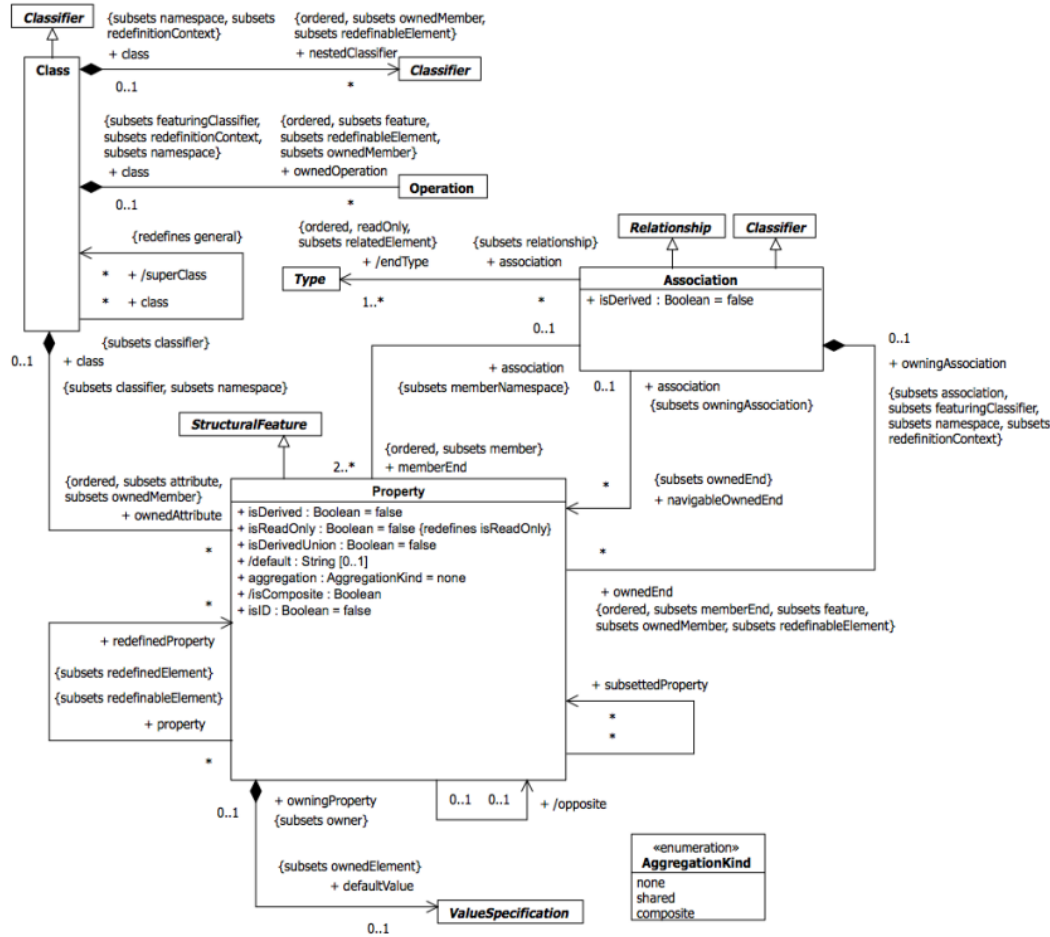


Figure 3.11: Key concrete classes from the UML2 kernel package [(OM16, p. 42)]

The MOF standard states four layers of metamodeling starting at the M3-layer until the M0-layer. Every layer is an instance of its above layer except for the M3-layer. The M3-layer is an instance of the M3-layer itself, which means that MOF uses its own metamodel to describe its possibilities. Figure 2.4 shows the four layer model.

- The *M3-layer* of this model describes MOF itself. Moreover, the MOF standard constructs are used to define metamodels for many application areas.
- The *M2-layer* describes metamodels by using the provided MOF constructs. Figure 3.11 shows the UML class diagram by using MOF.

- The *M1-layer* describes the models which conforms to the metamodel defined in the M2-layer.
- The *M0-layer* uses the concrete instances of classes defined in the M1-layer.

3.5.4 Ecore

The metamodel of EMF (Subsection 2.3.2) is the so called *Ecore*. The Ecore metamodel could be seen as a simplification of the UML class diagram [BCW17]. Figure 3.12 shows the hierarchy of the components in Ecore whereas abstract classes or interfaces are deposited with a lightgrey color. In the following part, the main elements of the Ecore metamodel would be described [Foua].

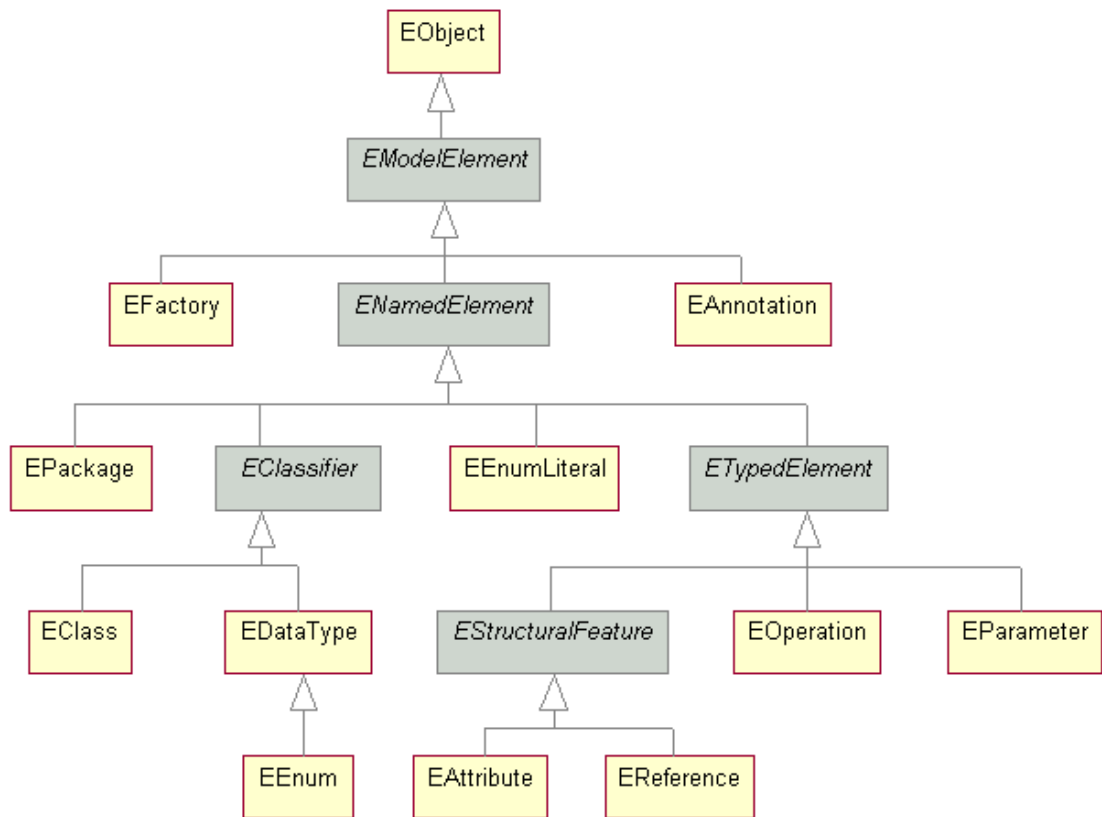


Figure 3.12: The Ecore components relation hierarchy [Foua]

The *EObject* is similar to the `java.lang.Object` from Java. Every object in EMF is importing this interface to have the main functionality available like reflections, notifications and access to the metaobject *EClass*, the container and to the available resources.

The *EFactory*-elements of a package are generated by their responsible *EFactory*. The reference to an *EPackage* is the only reference found in a *Factory* done by the *ES-*

tructuralFeature. A EFactory consists of the methods create, createFromString and convertToString for generating the necessary Objects.

Every *EModelElement* in Ecore, which means completely every element, has the possibility to store additional information by using the *EAnnotation*.

The *EClassifier* is an abstract class for referencing an EReference with an EClass or an EAttribute with an EDataType. The keywords instanceClassName and instanceClass returns the implemented class. Another keyword is the isInstance operation for checking if an object given as a parameter is the same EClassifier.

The *EPackage* is similar to packages in Java. It is possible to group elements. Moreover, packages could be build up in a hierarchical manner. To make this possible eSubPackages where created. Every EPackage has the following attributes named nsURI and nsPrefix to identify a package as unique. The attributes are stored in the EClassifiers and could be read by the reference eClassifiers.

The *EClass* is used to model classes. Every EClass has a name attribute available in ENamedElement and can have many references and attributes which could be seen in eAllAttributes, eAllReferences or eAllStructuralFeatures. One important paradigm is the availability of multi-inheritance. An EClass can have many superclasses available in eSuperTypes whereas other metamodels only support one inheritance between classes. There is the method isSuperTypeOf to check if a delivered class is inherited. The attributes interface and abstract are similar to the Java implementation. They could be read by using the getEStructuralFeature.

The *EDataType* is used to model simple data types. These data types could be similar to existing Java classes or build up with own classes to fit complex structures. The properties instanceClassName and instanceClass are for using the existing Java classes.

The datatype *EEnum* can have every value of a predefined list. The values of an EEnum are called *EEnumLiteral*.

An EClass could be extended to model behaviour of a class by so called *EOperations*. Operations are callable with the command eAllOperations and could have none ore many *EParameters*. Moreover, eExceptions could be implemented on classes like in modern programming languages.

The next concept is an *EAttribute* to use attributes with a specific kind of EDataType. Attributes have a name and could only exist in connection with a class. In the class concept the keyword eAttributes will list all attributes which are connected to the specified class.

EReferences are for connecting two classes with each other. A reference is only referencing in one direction. If someone wants to model a bidirectional connection, two references have to be created and the opposite direction in eOpposite must be defined. On the two ends of a reference, EClasses must be defined. For defining the specific type of these EClasses, the keyword eReferenceType is used. With the usage of the EStructuralFeature multiplicities could be modelled. At last, a reference could be a containment.

The abstract class *EStructuralFeature* describes the characteristics of references (EReference) and attributes (EAttribute). The keywords changeable, transient, unique, unsettable and volatile describe values from attributes. In addition, multiplicities like lowerBound, upperBound, required and many are for describing references. The last possible keyword is the defaultValue to set the standard value.

3.5.5 Transformation languages

As described in Section 2.4, I will focus on the model-to-model transformations and especially on Out-Place transformations. I use the method of Out-Place transformations for transforming a behaviour language into a pivot language and backwards. In every transformation case, new target models would be generated.

3.6 Design Methods

3.6.1 The Iterative and Incremental development

As first introduced in the Section 3.1, this thesis focuses on the Design Science Methodology. This methodology is using an iterative, incremental and test-driven process for answering the desired research questions.

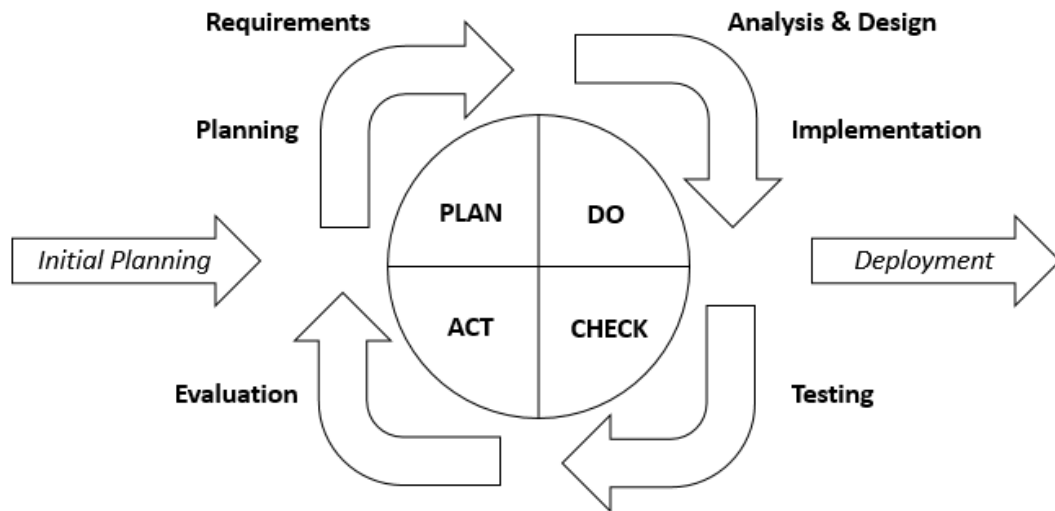


Figure 3.13: Iterative and Incremental development process

The first part of the iterative and incremental development is the *Initial Planning*. In this part the overall objectives of the project have to be defined into smaller sub goals. All the sub goals must define a metric with a reachable and specific goal value which is checked in the end of the incremental process to finish a sub goal [BCW17].

The next part is based on the Plan-Do-Check-Act (PDCA) circle. The first section of the PDCA circle is the *Plan* section. In this section the planning and requirements for the cycle iteration are defined and the possible steps to achieve a positive result. This planning is based on the requirements of the initial planning phase [She31].

In the next section the plan is executed and the artefact is implemented. This section is called the *Do* section. For the next section it is important to collect data for analysis purposes.

Third, the *Check* section is studying the actual results of the implementation. These results are collected during the *Do* section and compared against the expected results of the sub goals from the planning phase. If there are any differences between the planned sub goals, information about the differences must be collected for the next iteration of the cycle.

Finally, the *Act* section is reached. If there are differences between the planned sub goals and the realised implementation, studied in the *Check* section, the current status is defined as the new baseline. The baseline should be improved by learning from the gained information from the previous sections to complete more defined sub goals in the next incremental iteration. This process is done as long as all planned sub goals are successfully resolved. This could be done after one iteration but in most cases after an endless number of iterations.

After finishing the PDCA cycle with a positive *Check* section, the deployment is done. The resulting artefact is functional and agreed with the defined goals.

3.7 Data Models

3.7.1 Intermediate Modelling Layer

IML is introduced in Subsection 2.1.1. It is used as the underlying data model for this thesis and should be studied to be a kernel language for exchanging behaviour models.

Implementation

In this chapter implementations and their suggested solution under the usage of methodologies from Chapter 3 are described. At the beginning, the updated Ecore Intermediate Modelling Layer is shown. Next the major types of behaviour models get introduced. Moreover, transformation rules for transforming specific types of behaviour models into the IML are stated. At last, the graphical editor SIRIUS is adapted to construct graphical representations of IML models.

4.1 IML-Based Exchange Framework

4.1.1 IML-Ecore Metamodel

First, IML must be transformed into an Ecore metamodel. There is currently an IML Ecore metamodel available developed by the *Institute of Software Technology and Interactive Systems* from the Vienna University of Technology. This metamodel is based on an old definition of IML. Currently the IML should be standardized by the IEC and is available in version 0.89. In this thesis i will focus on the version 0.89. Accordingly, an update of the current Ecore metamodel implementation of IML with the updated standardisation of version 0.89 has to be done.

Every element described in Subsection 2.1.1 is transformed into an Ecore *Class* element. Additionally some new elements were introduced:

- *Element* is an abstract class. It is a generalization of all elements which specify a global Ecore class element. The *Header* class has a relationship with the element class. If there is no Header, then all elements can't exist because of a composition.
- *IdentifiableElement* is a differentiation of an *Element*. The counterpart is the *Event*. They distinguish from each other by their attributes. The *IdentifiableElement* has an *ID* attribute, whereas the *Event* has a *name* attribute.

- *ConnectionPoint* is an abstract class which has the purpose of generating connections. Every Class which inherits from the Connection Class has the ability to make a Connection.
- A *Connection* consists of a *source* and a *target* ConnectionPoint.
- The *Jump* class is for jumping from a specific State to another State.
- The *Time* class is for defining time points on activities. A time point stores information about the earliest or latest start or end of an activity. Moreover the delay and duration of an activity is stored. The Time class describes a time object for designated behaviour models.

The current implemented Ecore metamodel is shown in Figure 4.1. It displays the development state of the Intermediate Modelling Layer with the current version 0.89.

4.1.2 SIRIUS Editor for IML

The *Sirius* modelling environment is an open-source software developed by the Eclipse Foundation. Furthermore, SIRIUS creates an own modelling workbench including the Eclipse modelling techniques EMF and GMF. It distinguishes between two kinds of environments, the *specification environment* and the *runtime environment* [Foub].

The specification environment is mainly used by the *specifier* or the *developer*. It is possible to define custom multi-view workbenches including diagrams, trees and tables. Additionally, the instant feedback feature helps to model faster. At last, SIRIUS is highly customizable by supporting native tooling, Java or extension points.

The runtime environment is used by the *end-user* and executes the specification defined by the specification environment. There is no code generation for the end-user. Moreover, the environment is viewpoint-based so it could be adapted to the needs of the users.

There is currently a SIRIUS project for IML available developed by the *Institute of Software Technology and Interactive Systems* from the Vienna University of Technology. This project must be adapted to fit to the needs of IML 0.89. The difference to the currently available implementation is the *Time* class of the Ecore metamodel. This must be added to the current SIRIUS project.

The gradient representation of the ActivityNode has to be updated to display the time point information instead of displaying only the name of the activity represented. An update of the label expression is needed by using ACCELEO for creating an expression. If the *DA_-Activity* is generated, the value of the *earliest start time point* should be display. If the *TA_-Activity* is generated, the value of the *duration* should be displayed.

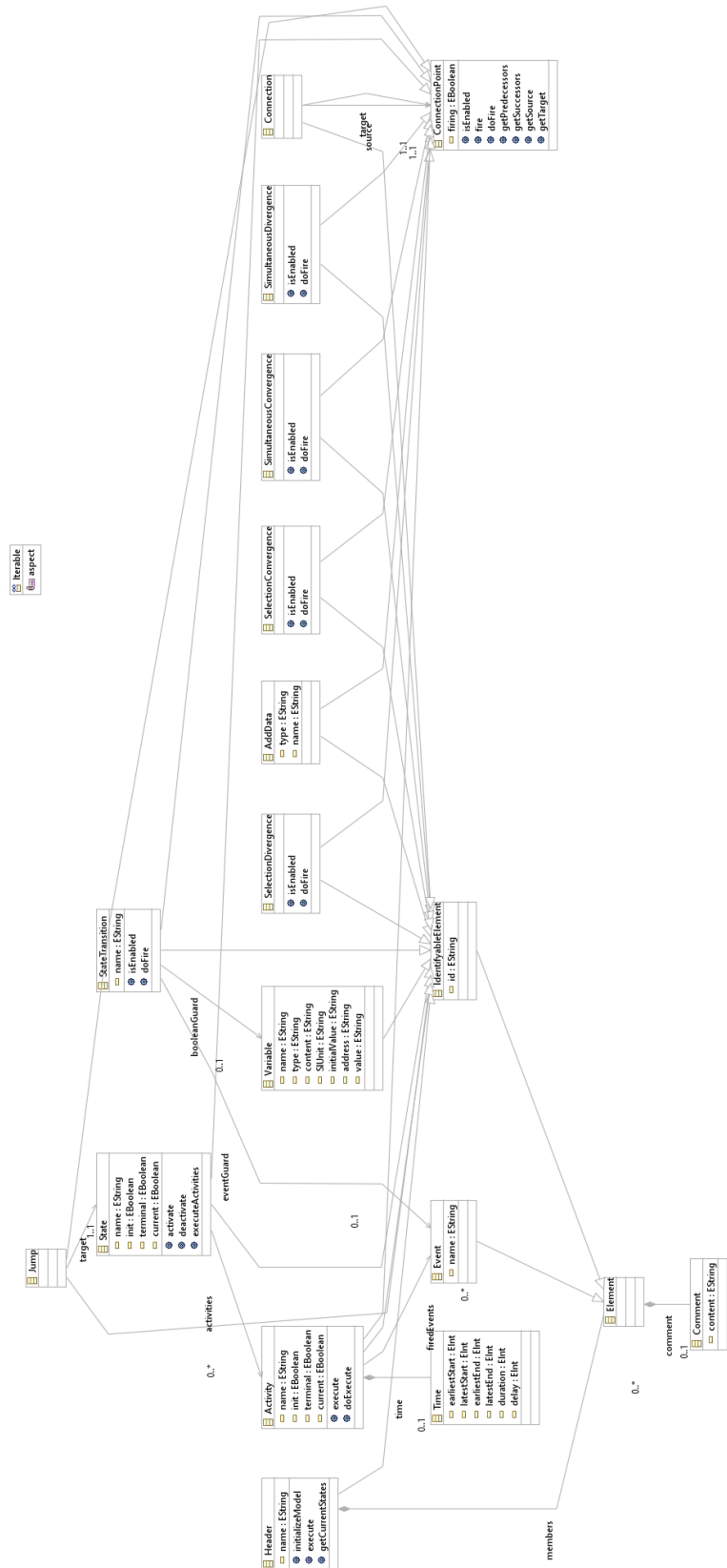


Figure 4.1: IML Ecore metamodel

The used expression is:

```
[if self.name.startsWith('DA') then 'D '  
.concat(self.time.earliestStart.toString())  
.concat(' | ').concat(self.name)  
else 'SD ' .concat(self.time.duration.toString())  
.concat(' | ').concat(self.name) endif/]
```

Table 4.1 shows the notation table for the graphical representation of the IML metamodel generated by the SIRIUS editor.


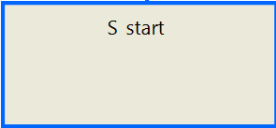
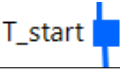
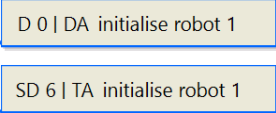

Graphic	Concept
	Initial state
	State
	State transition
	Activity
	Simultaneous con- & divergence

Table 4.1: Notation table for the IML definition used in SIRIUS

4.2 Behaviour Models

The first point I would like to mention is, for constructing a kernel language for behaviour models must be understood the different types of behaviour models. I have looked up different types of behaviour models like GANTT, AONN, Impulse diagrams, State charts and Sequence Function charts. After analysing these different diagrams, three different approaches of behaviour models are depicted: Event-driven, Task-oriented and Time-related. In the upcoming subsections the different approaches are described.

4.2.1 Event-driven approach - State machines

A good example for describing the *event-driven* approach is the *State Machine (SM)*. A state machine consists out of states and events. To change from one state to another state a specific event has to occur. There is no dependence to any kind of duration on how long the state machine is in a specific state. On the other hand a state can not change to another state without any event. The state machine must be some kind of event-driven approach.

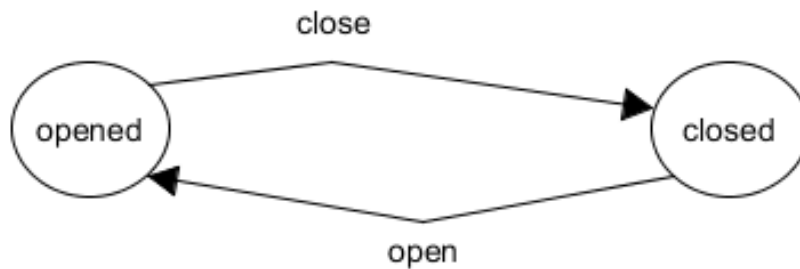


Figure 4.2: Example of a door modelled with a state machine

In Figure 4.2 a small example of a door with a state machine diagram is modelled. In this example the door has two states: *opened* and *closed*. Additionally, two events could occur on this door: *close* and *open*. The resulting transitions lead to the following possible changes on the door:

- If the door is in the state *opened* and the event or transition *close* is happening, the door changes to the state *closed*.
- If the door is in the state *closed* and the event or transition *open* is happening, the door changes to the state *opened*.

These transitions could be displayed in a *State transition table* as shown in Table 4.2.

		State (next)	
		Opened	Closed
State (current)	Opened	-	close
	Closed	open	-

Table 4.2: State transition table for the example shown in Figure 4.2

In this thesis I will focus on the two approaches *task-oriented* and *time-related*. A distinction to the event-driven approach must be made, because the task-oriented and

time-related approaches have many similarities. Both approaches are not bound to events to trigger state transitions. They are connected over the completion or expiration of tasks whereas event-driven approaches have to wait for occurring events. First, it is more significant to look at modelling techniques with the same background. Moreover the evaluation of the results are more significant and easier to make as comparing an event-driven approach with one of the others. At last, state machines are not the main modelling technique used in industrial automation processes whereas AONN and GANTT charts are more appreciable.

4.2.2 Task-oriented approach - GANTT

The *task-oriented* approach is commonly used in behavioural modelling when a behaviour between tasks should be modelled. GANTT charts are using this approach to model dependencies between tasks. There are typically two options when a task could be fired in a GANTT chart.

The first option is that a task must wait with the processing until all depended tasks are finished. Furthermore, a task could have a dependency to one or multiple predecessor tasks. In addition to that, tasks are existing which have no dependency to any task. These tasks are *time-related* tasks. They are start processing when a specific time is reached independent of any other task.

This leads to the existence of two possible modelling techniques for GANTT charts. They could be modelled with a *global* time, which means that every task starts counting the past time from the beginning, whereas the *local* time is only for the current task. The local start time of a task is zero and when the duration has past the end is reached. The global time calculation takes every predecessor task in count until they reach the first task of the chain.

Additionally, IML supports the global and local time approach. At least the type of implementation differs between the two time approaches. The global time is used, when existing tasks are not depending on each other. This example could be seen in Figure 4.3. The usage of tasks without dependencies results in using the global time scale to align the tasks and results in the corresponding sequence.

After transforming Figure 4.3 into IML, the global time scale is used, which defines when the different states have to start by using the delay attribute. This attribute could be seen on the *DA_-Activity* connected to every state and marked with a *D* in Figure 4.4. The whole chart has to be splitted up by using a *Simultaneous Divergence* element and merged together at the end by a *Simultaneous Convergence* element.

Besides the local time approach is used when a task get in a dependency relationship with each other. As an example I want to use the adapted Figure 4.5, which is Figure 4.3 with added dependencies between tasks.

After transforming Figure 4.5 into IML, the local time scale is used. Every state is in dependency to the predecessor state and also the time scale starts from beginning after a

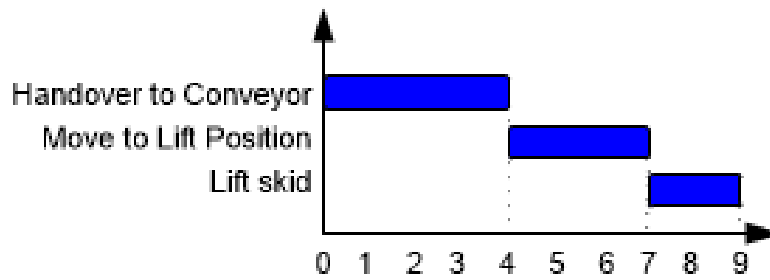


Figure 4.3: Example of a GANTT chart with no dependencies resulting in using the global time scale

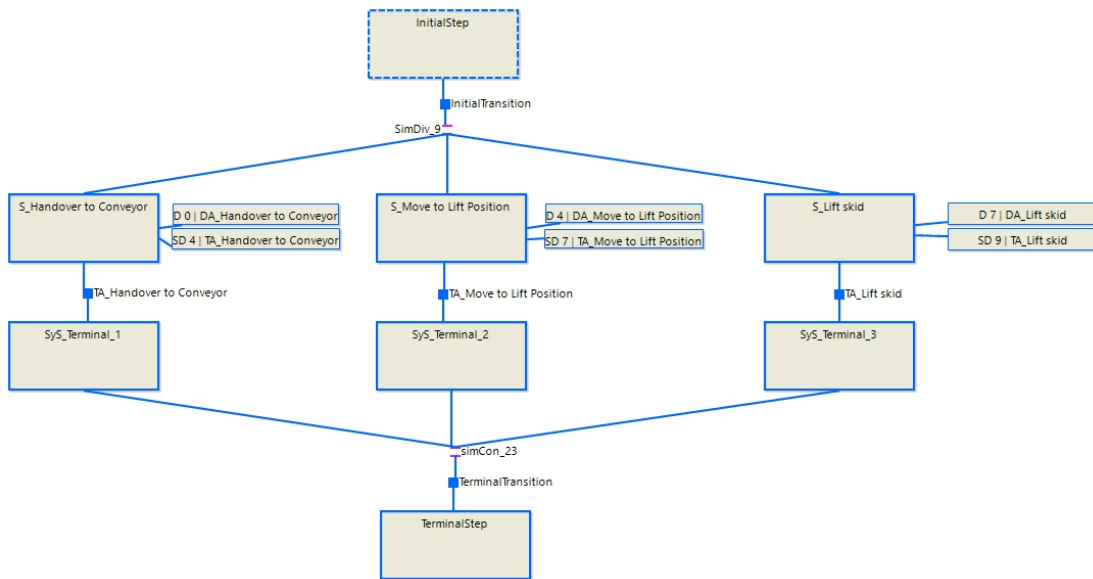


Figure 4.4: Example of a non task dependent GANTT chart transformed into the IML

state. The resulting IML chart from Figure 4.6 is now a sequence of states which runs from the top to the bottom.

A very important point in Figure 4.6 are the activities of the states. All *DA_*-Activities are set with a zero, because they start quickly after the predecessor state finishes without any delay after the predecessor state, whereas in Figure 4.4 the *DA_*-Activities have a delay to wait on each predecessor state. The *TA_*-Activities are now storing the duration how long a state is running, whereas in Figure 4.4 the *TA_*-Activities store the global end time of a state.

At next I would like to introduce the GANTT Ecore metamodel. This metamodel is downloaded from <http://www.ganttproject.biz/>. It is open-source and available for using in any kind of project [Sot]. In Figure 4.7 the fully non adapted GANTT Ecore

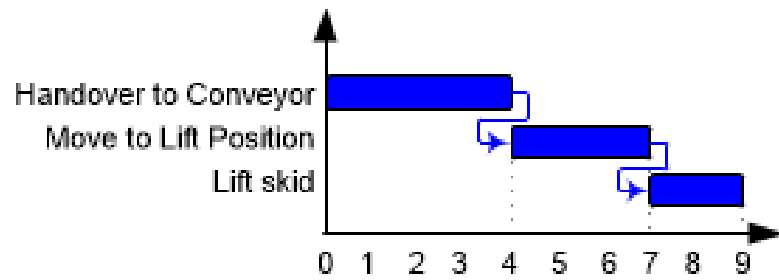


Figure 4.5: Example of a GANTT chart with dependencies between tasks resulting in using the local time scale

metamodel is shown.

In addition, I will describe the main parts of the metamodel used:

- The *Project* class describes the whole GANTT chart and is mandatory to create, because all other classes are depending on this class.
- A project has one mandatory global task to reach, which is called *Tasks* in the metamodel. Every smaller unit task is in a relation with the *Tasks* class, which describes the overall reaching goal.
- A *Task* itself describes a smaller unit off the overall task. This class has the main information about how long the duration is and when the task would be finished. As described, the metamodel does not store information when a task is starting, it rolls up the time calculation from behind starting at the point of completion.
- A task could have none or multiple dependencies described through the *Depend* class. Every depend class links to only one previous task. If someone wants to build up a multiple dependency, more depend classes linking to each single task of dependency must be created.

4.2.3 Time-related approach - AONN

The last approach is the *Time-related* approach. There is a type of behaviour model which depends on time relations. This behaviour model is called the *activity-on-node network* (AONN). The former name of this model type was *Program Evaluation Review Technique* (PERT). Figure 4.8 describes the main structure of a node.

A node consists of the following attributes:

- *Node name*: The name of the given node
- *Earliest start*: The earliest start time point of the node

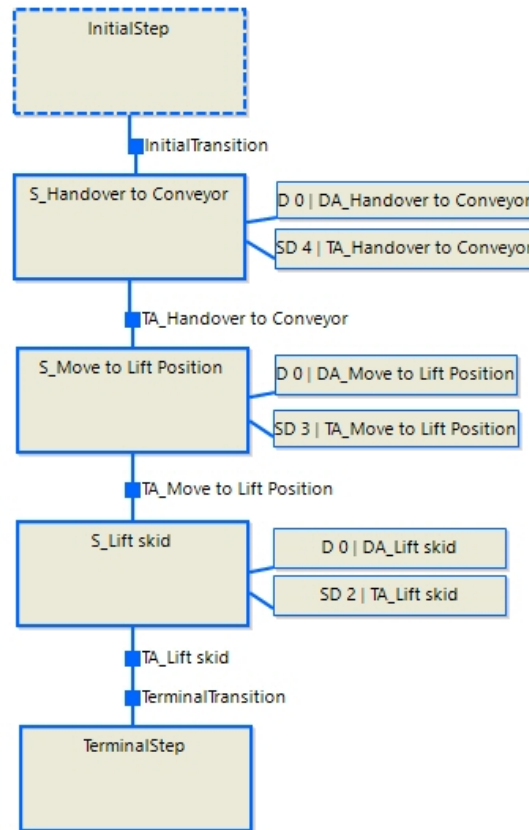


Figure 4.6: Example of a task dependent GANTT chart transformed into the IML

- *Duration*: The fixed duration time of the node
- *Earliest end*: The earliest end time point of the node is the summing up of the earliest start and the duration attribute
- *Buffer*: The buffer attribute defines the maximal waiting time until the node begins with the processing
- *Latest start*: The latest start time of the node is the summing up of the earliest start and the buffer attribute
- *Latest end*: The latest end is the summing up of the latest start with the duration attribute
- *Predecessor*: A node can have a relation to none or multiple predecessor nodes
- *Successor*: A node can have a relation to none or multiple successor nodes

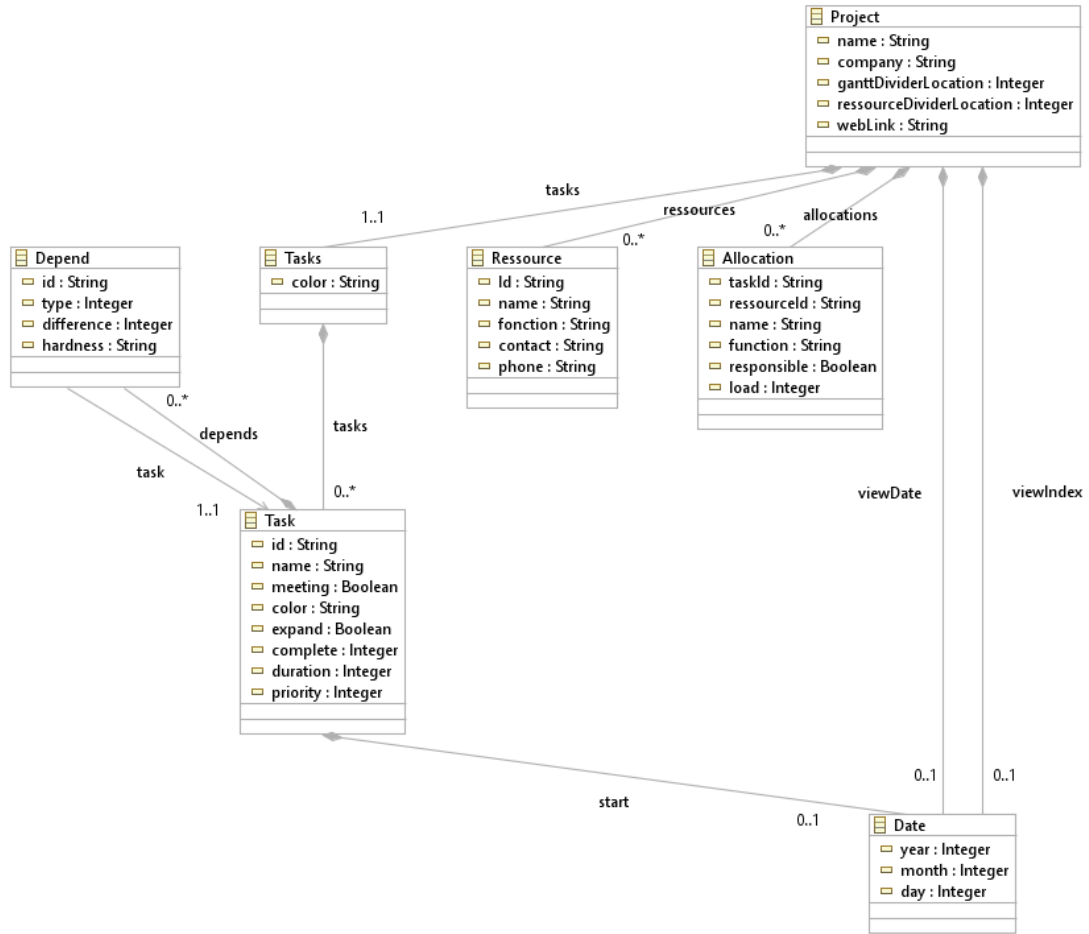


Figure 4.7: Ecore metamodel of the GANTT chart [Sot]

An AONN is using the time-related approach. Furthermore, using the earliest and latest start time points help avoiding processing failures by using a waiting time. This type of behaviour model is not depending on any events or not waiting on when a task is finished, because a task has to be finished between the earliest and latest end time point.

An AONN is using the global time scope on all nodes of a chart. It is not possible to use the local time scope in this case, because of the major relationship between a node and the fixed time points. Figure 4.9 shows an example of an AONN.

After transforming the activity-node-network from Figure 4.9 the time points are stored in activities at the different states. The *DA_*-Activities are storing the time point of the earliest start, whereas the *TA_*-Activities are storing the duration of a state.

At next I would like to introduce the AONN Ecore metamodel. This metamodel is developed on my own by using the metamodel developing process. The development

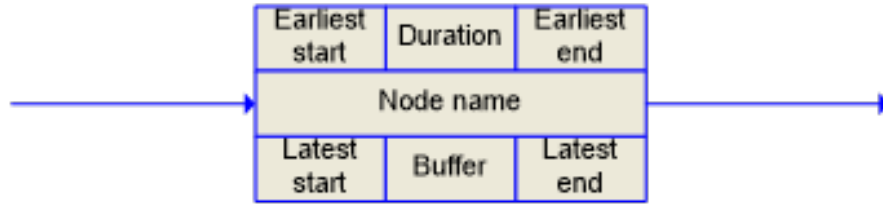


Figure 4.8: Basic structure of an AONN node

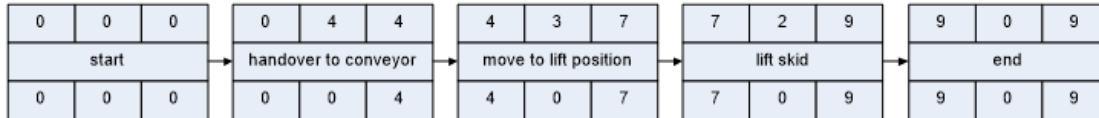


Figure 4.9: Example of an AONN

of the AONN Ecore metamodel is described in Subsection 3.3.1. The resulting Ecore metamodel is shown in Figure 4.11.

In addition, I will describe the parts of the metamodel:

- The *Project* class describes the whole AONN and is mandatory to exist, because all other classes are depending to this class.
- The *Node* class is the main class of the chart. First, it describes all the important attributes like earliest or latest start or end time point, the duration and the delay. Moreover, it saves the name of the node. At last, information about predecessors and successors is saved in relations to other nodes. A node could have no or multiple pre- or successors.

4.3 Transformations

This section describes the developed transformations for this thesis. The scope of this thesis is on the development of ATL transformations from GANTT charts or AONNs to the IML and back.

First, I would like to describe the transformations from GANTT or AONN to IML because they are very similar to each other.

4.3.1 GANTT to IML

The transformation from GANTT to IML should generate an IML chart for every used GANTT chart. Additionally, the generated IML should have at least one initial state followed by a state transition and one termination state. Every GANTT bar should be transformed into an IML state followed by two associated activities and a state transition.

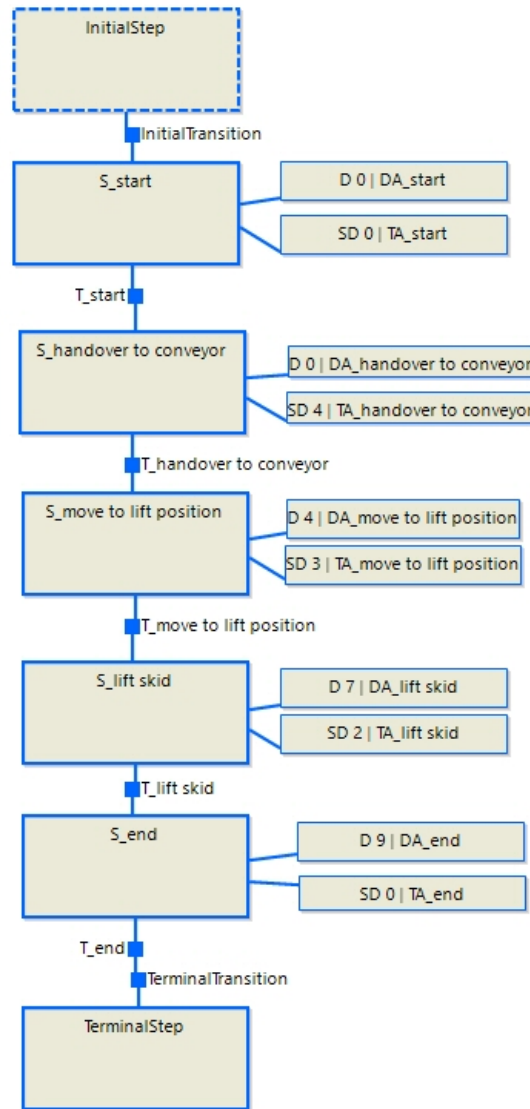


Figure 4.10: IML chart of the transformed AONN from Figure 4.9

The first activity of the generated state should represent the delay in execution of the GANTT bar. The other activity is for defining the duration of the state.

Another important point for the transformation are successor and predecessor relationships of GANTT bars. If a GANTT bar has two or more predecessors, an IML simultaneous divergence element must be generated. If a GANTT bar has two or more predecessors, a synchronous IML elements and additionally an IML simultaneous convergence element must be generated.

The ATL transformation for transforming a GANTT chart into an IML chart has to

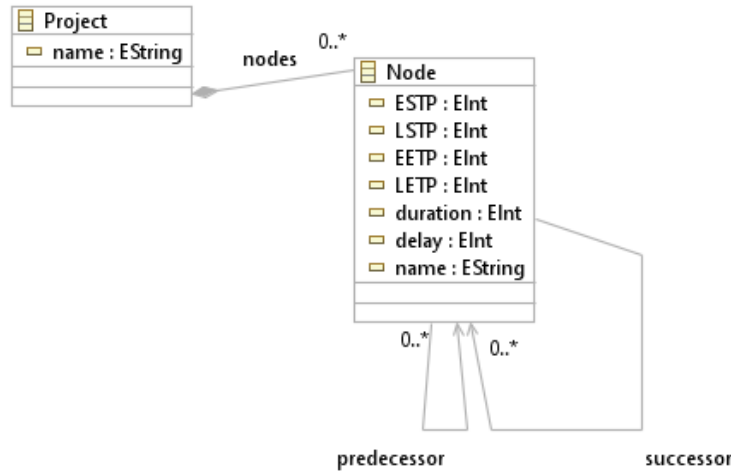


Figure 4.11: Ecore metamodel of the AONN

deal with caching during the transformation phase. Many elements are generated after knowing all pre- or successor of a GANTT bar. Caching was implemented to handle these special cases of not running a strict transformation from the first bar to the last bar. There are existing many cases because of the possible pre- and successor connections of a bar. In addition an abstract rule hierarchy was build up.

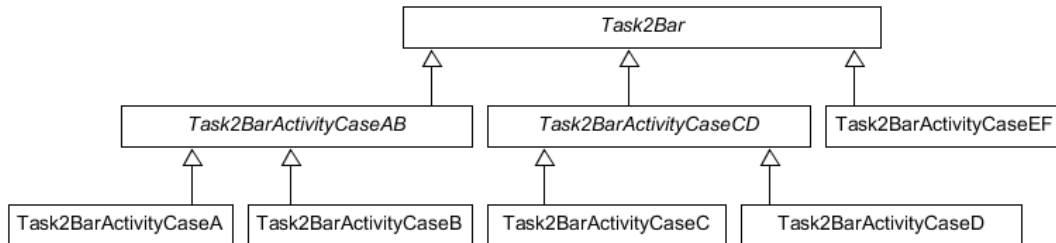


Figure 4.12: ATL rule hierarchy for transforming a GANTT bar to an IML state

The abstract rule *Task2Bar* is for generating the generalized main parts of a bar to state transformation. It generates the state, the two activities and the state transition.

The abstract rule *Task2BarActivityCaseAB* is a more specific rule than *Task2Bar* because it only takes bars into account, which have no predecessor bar. The *Task2BarActivityCaseA* rule is matched, when the bar is the only bar in the diagram with no predecessor bars, whereas *Task2BarActivityCaseB* is matched, when the bar has no predecessor and at least one other bar in the GANTT diagram has no predecessor.

The abstract rule *Task2BarActivityCaseCD* is a particularized rule of *Task2Bar* and

takes only bars into account, which have only one predecessor bar. The matched rule *Task2BarActivityCaseC* matches on bars with only one predecessor bar and no other bar in the GANTT chart has the same predecessor bar, whereas *Task2BarActivityCaseD* is matched on bars with only one predecessor and the predecessor has more than one successor bar.

The last matched rule *Task2BarActivityCaseEF* has one similarity for Case E and Case F. Both cases have more than one predecessor node. The predecessor nodes of Case E have only one successor node, whereas the predecessor nodes of Case F have more than one successor node.

In conclusion, for creating all these cases it is important to create helpers for especially calculating the maximum time of a sequence of depending elements, calculate the number of end bars in the chart, calculate the number of depending bars to a specific one and calculate the dependencies other bars have on a specific bar.

4.3.2 IML to GANTT

First, the *Header* of the IML chart is transformed into a GANTT project. All generated tasks from the IML chart must be in a relationship with the project class.

The transformation from IML states to GANTT tasks is based on recursions. There is a simple way to overcome the problem of defining GANTT tasks with a dependency (local time) or without a dependency (global time) by iterating backwards from every IML state. To distinguish between the two possible time points, two conditions are existing.

The first condition is needed to calculate the global start time of a specific state. When a *simultaneous convergence* is reached, the maximum time of all paths have to be calculated. This is done by summarizing all *TA_-Activity* duration values for every path of the divergence. The maximum duration time is then used in the recursion for finding the global start time of a task.

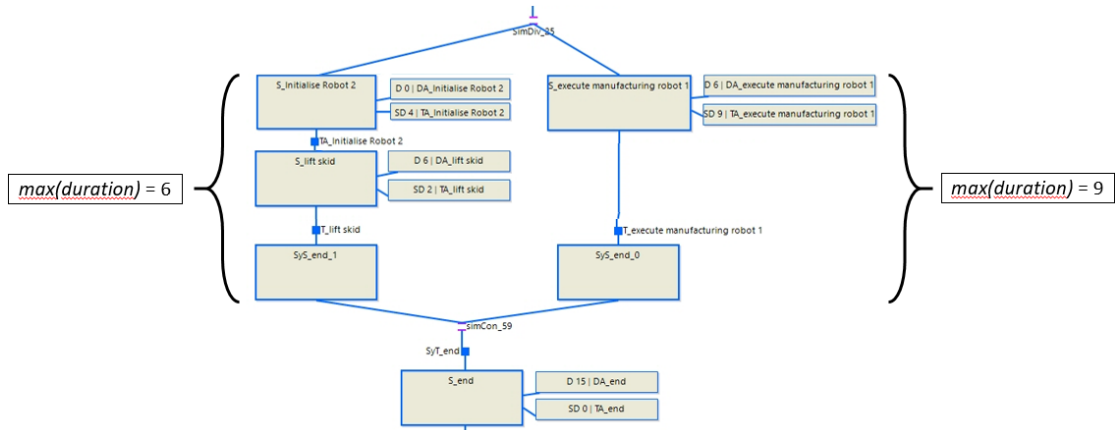


Figure 4.13: Example of a path calculation

In Figure 4.13 an example for calculating the duration of splitted paths is shown. The global time for S_end should be calculated and a simultaneous convergence / divergence split is on the way to the initial state. To calculate the maximum time of the left path, all duration values from the $TA_$ -Activities are summed up, which leads to the calculation of $4 + 2$. The right path has only one state with a value of 9. At last, the maximum-function is used to find the highest value from all paths, which is in our case the right path.

When the global time is calculated, the duration of the current state is added to define the time when the task is complete. The duration of the generated task is the duration of the transformed state subtracted by the delay of the transformed state.

The second condition is the generation process for *Depend* classes. If the current transformed state has no predecessor states in IML, no depend classes were generated but if it has predecessors then for every predecessor a depend class is generated.

4.3.3 AONN to IML

The transformation from AONN to IML is realized the same way as used in Subsection 4.3.1 and instead of GANTT tasks, AONN nodes are used. The only difference between the transformation from GANTT to IML and AONN to IML is the activities of a state. The first activity of the generated state should represent the earliest start time in execution of the AONN node. The other activity is for defining the duration of the state.

4.3.4 IML to AONN

The transformation from IML to AONN works a little bit different then the transformation from IML to GANTT. First, the IML *Header* element is transformed into an AONN *Project* element.

Second, every *State* element having two activities must be transformed into an AONN *Node* element. The attributes are copied from the state and the timing attributes of a node are copied from the concurrent *Activity* element. The $DA_$ -Activity has the information stored about the earliest start time point and the latest start time point. The $TA_$ -Activity stores the information about the duration, delay (buffer) and the earliest end time point and the latest end time point.

Moreover, there is a need for caching states during the ATL transformation because of the existence of predecessor and successor relations. When a predecessor or successor node is not transformed yet, then the state and node relation is cached. When a new state is transformed to a node, the cache is checked if the node is ready for setting up a relation to the current node.

4.4 Transformation Contracts

After defining the different cases for the transformation into IML, the test process during the implementation phase is described. Furthermore, to test the defined transformations

different input sample models have to be generated to match all different cases for the transformation.

First, a sample model in the start language is generated by using the eclipse model builder. The sample model has to fit to specific defined cases to test them. When the model is finished the transformation into the IML is started. The resulting output model is then checked through the definition of IML. When an error is emerged, the transformation is reviewed and adapted to fit to the needs. Whereas an error free output model is generated into a graphical model representation to make a last check on the generated model.

Furthermore, the check process of the output model is supported by the usage of a check-list. This check-list is basically written for every transformation type. At next the different transformations and their check-lists are described in short. The needed elements and their attributes of every case is shown in the IML standardization.

4.4.1 GANTT to IML

First, the check-list for transforming GANTT charts to IML is described.

1. Mapping of the start of a GANTT chart to IML elements
 - Case A: The start has only one successor bar
 - Case B: The start has more than one successor bar
2. Mapping of GANTT chart bars to IML elements
3. Mapping of a GANTT chart bar start point to IML elements
 - Case A: The bar has no predecessor bar
 - Case B: The bar has exactly one predecessor bar
 - Case C: The bar has more than one predecessor bar
4. Mapping of a GANTT chart bar end point to IML elements
 - Case A: The bar has no predecessor bar
 - Case B: The bar has exactly one predecessor bar
 - Case C: The bar has more than one predecessor bar
5. Mapping of GANTT chart successor bars to IML elements
 - Case A: The bar has no or exactly one successor bar
 - Case B: The bar has more than one successor bar
6. Mapping of GANTT chart predecessor bars to IML Elements
 - Case A: The bar is the only bar in the diagram with no predecessor bars

Case B: The bar has no predecessor bar and at least one other bar in the GANTT chart exists with no predecessor bar

Case C: The bar has only one predecessor bar and no other bar has the same predecessor bar

Case D: The bar has only one predecessor bar and this predecessor bar has more than one successor bar

Case E: The bar has more than one predecessor bar, the predecessor bar has only one successor bar

Case F: The bar has more than one predecessor bar, the predecessor bar has more than one successor bar

7. Mapping of the end of a GANTT chart to IML elements

Case A: There exists only one predecessor state transition for the terminal state

Case B: There exists one simultaneous convergence for all synchronisation states

Example 1 - E1

In the first example (E1) a GANTT chart without any pre- & successor relations is transformed into IML (Figure 4.14). The important fact in E1 is the timing relation of the different bars. The resulting IML output model is shown in 4.15.

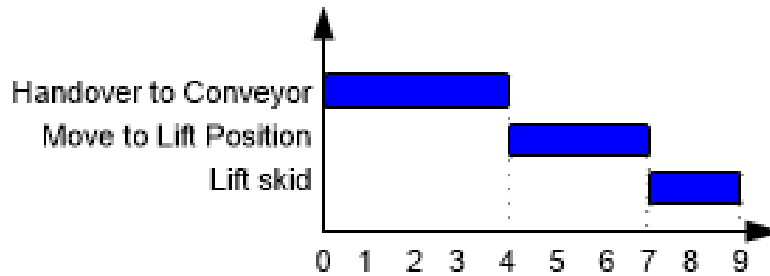


Figure 4.14: GANTT chart - Example E1

In this example the following cases from Table 4.3 are tested.

Example 2 - E2

In the second example (E2) a GANTT chart with pre- & successor relations is transformed into IML (Figure 4.16). The important fact in E2 is the timing relation of the different bars in relation to the predecessor bars. The resulting IML output model is shown in 4.17.

In this example the following cases from Table 4.4 are tested.

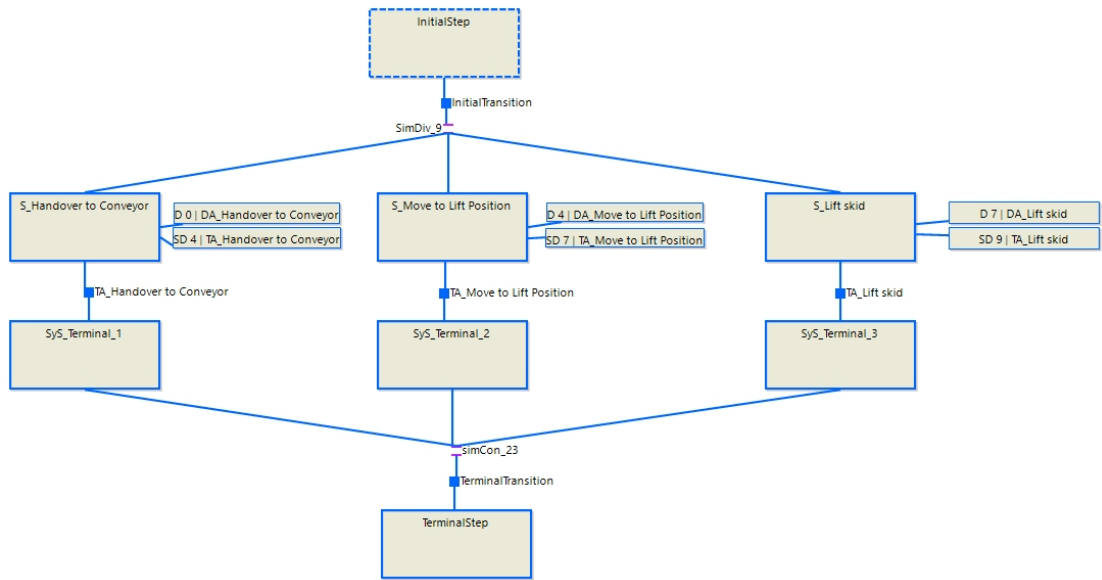


Figure 4.15: Example 1 of a non task dependent GANTT chart transformed into the IML

		Case					
		A	B	C	D	E	F
Step	1		X				
	2	X					
	3	X					
	4	X					
	5	X					
	6		X				
	7		X				

Table 4.3: Used cases in GANTT chart - Example E1

Example 3 - E3

In the third example (E3) a GANTT chart with pre- & successor relations is transformed into IML (Figure 4.18). A case is tested where a bar has more than one successor relation and the successor bars of this bar are time shifted. The resulting IML output model is shown in 4.19.

In this example the following cases from Table 4.5 are tested.

Example 4 - E4

In the fourth example (E4) a GANTT chart with two pre- & successor relations is transformed into IML (Figure 4.20). A case is tested where two start bars are existing

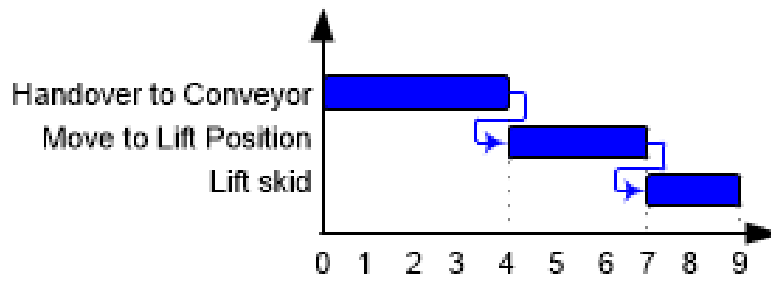


Figure 4.16: GANTT chart - Example E2

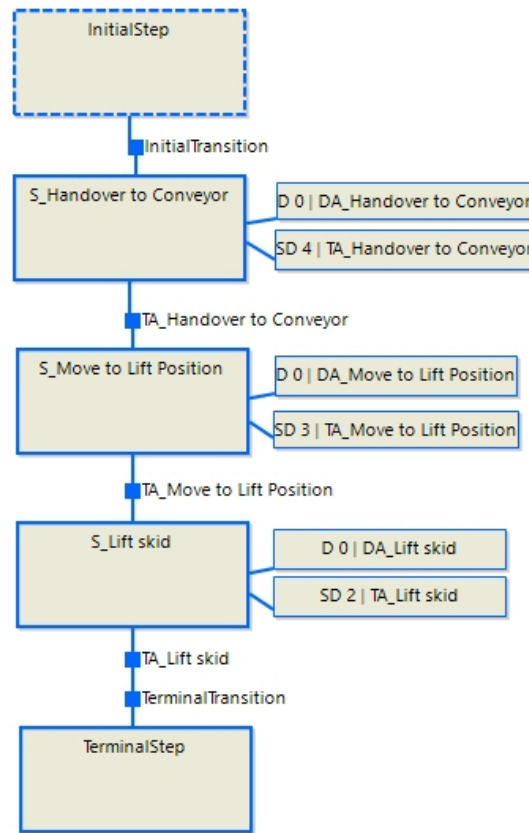


Figure 4.17: Example 2 of a task dependent GANTT chart transformed into the IML

but one is time shifted and a bar has more than one predecessor relations. The resulting IML output model is shown in 4.21.

In this example the following cases from Table 4.6 are tested.

		Case					
		A	B	C	D	E	F
Step	1	X					
	2	X					
	3		X				
	4		X				
	5	X					
	6	X		X			
	7	X					

Table 4.4: Used cases in GANTT chart - Example E2

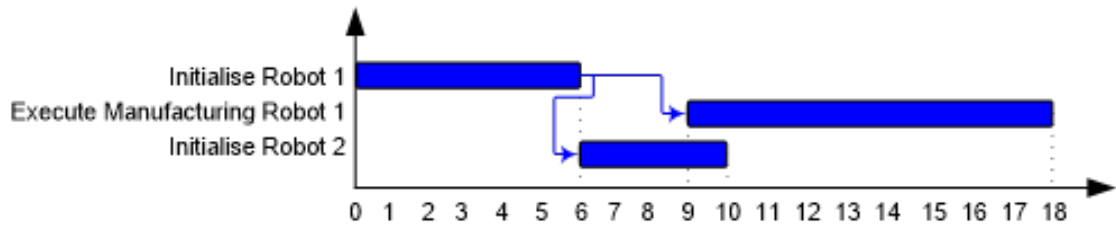


Figure 4.18: GANTT chart - Example E3

		Case					
		A	B	C	D	E	F
Step	1	X					
	2	X					
	3	X	X				
	4	X	X				
	5	X	X				
	6				X		
	7		X				

Table 4.5: Used cases in GANTT chart - Example E3

Example 5 - E5

In the fifth example (E5) a GANTT chart with a bar having more than one predecessor and the predecessor has more than one successor is transformed into IML (Figure 4.22). The resulting IML output model is shown in 4.23.

In this example the following cases from Table 4.7 are tested.

4.4.2 AONN to IML

At next the check-list for transforming AONN charts to IML is described.

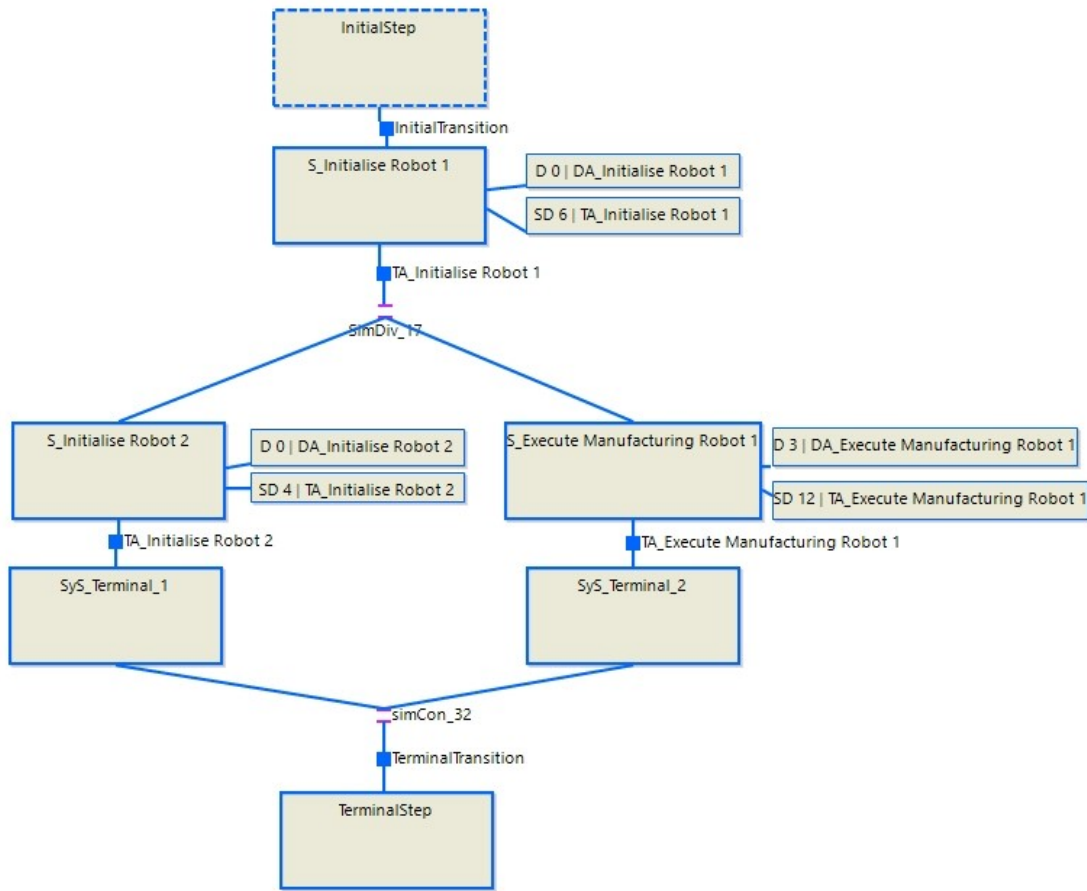


Figure 4.19: Example 3 of a task dependent GANTT chart transformed into the IML

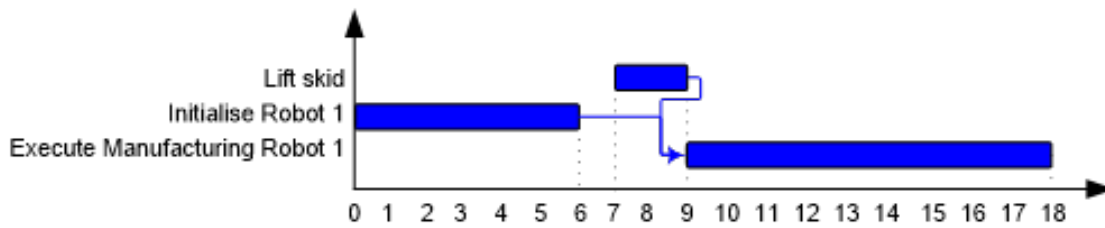


Figure 4.20: GANTT chart - Example E4

1. Mapping of the start of an AONN to IML elements

Case A: The start has only one node

Case B: The start has more than one node

2. Mapping of AONN nodes to IML elements

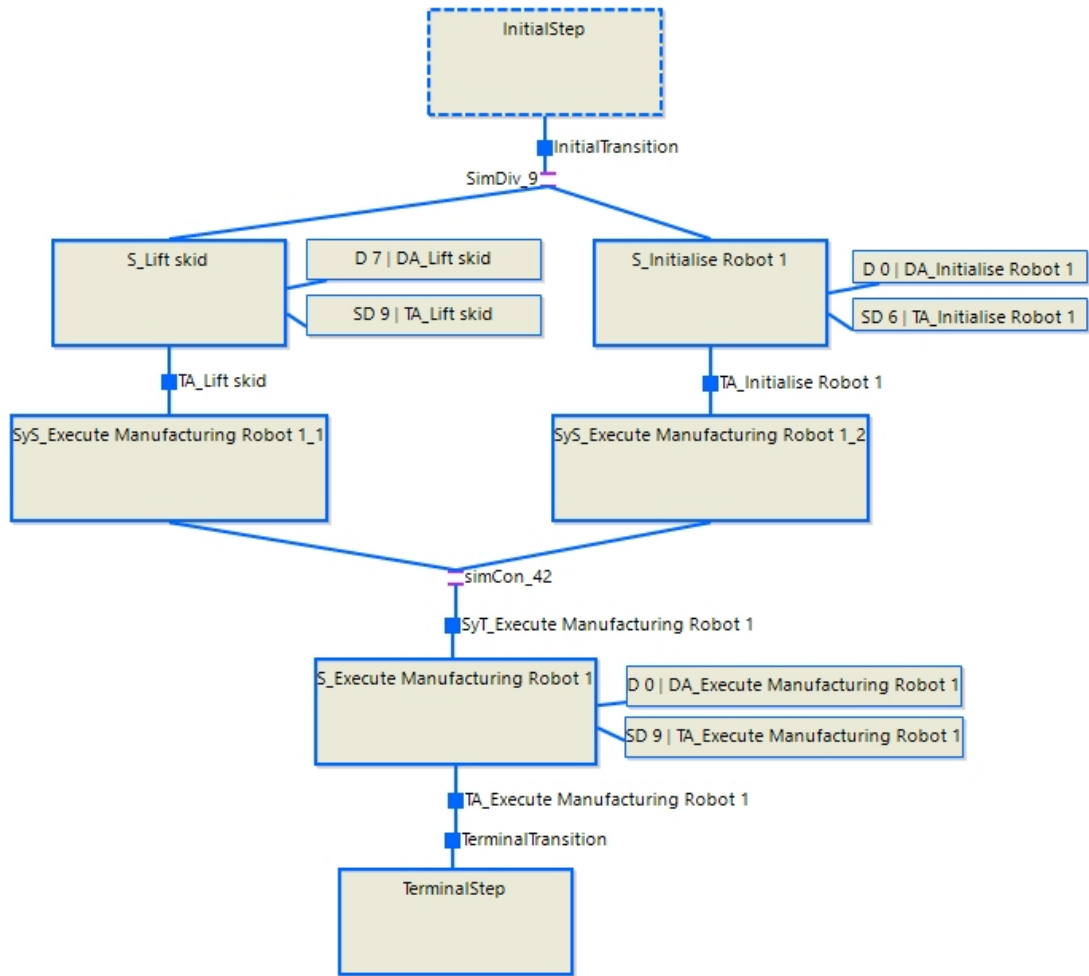


Figure 4.21: Example 4 of a task dependent GANTT chart transformed into the IML

3. Mapping of AONN node start points to IML elements
4. Mapping of AONN node end points to IML elements
5. Mapping of AONN successor nodes to IML elements
 - Case A: The node has no or exactly one successor node
 - Case B: The node has more than one successor node
6. Mapping of AONN predecessor nodes to IML Elements
 - Case A: The node is the only node in the diagram with no predecessor node
 - Case B: The node has no predecessor node and at least one other node in the network exists with no predecessor node

		Case					
		A	B	C	D	E	F
Step	1		X				
	2	X					
	3	X		X			
	4	X		X			
	5	X					
	6		X			X	
	7	X					

Table 4.6: Used cases in GANTT chart - Example E4

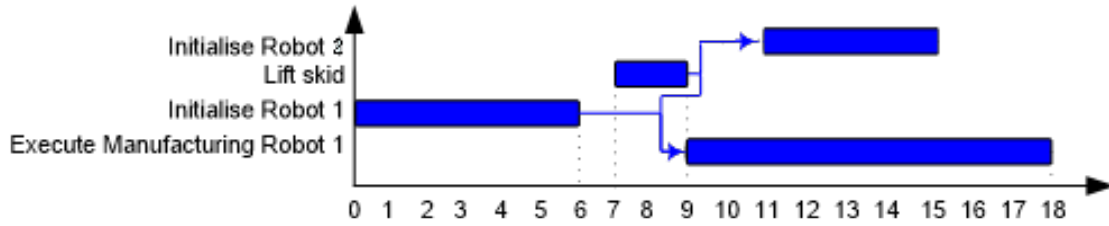


Figure 4.22: GANTT chart - Example E5

		Case					
		A	B	C	D	E	F
Step	1		X				
	2	X					
	3	X		X			
	4	X		X			
	5		X				
	6		X				X
	7		X				

Table 4.7: Used cases in GANTT chart - Example E5

Case C: The node has only one predecessor node and no other node has the same predecessor node

Case D: The node has only one predecessor node and this predecessor node has more than one successor node

Case E: The node has more than one predecessor node, the predecessor node has only one successor node

Case F: The node has more than one predecessor node, the predecessor node has more than one successor node

7. Mapping of the end of an AONN to IML elements

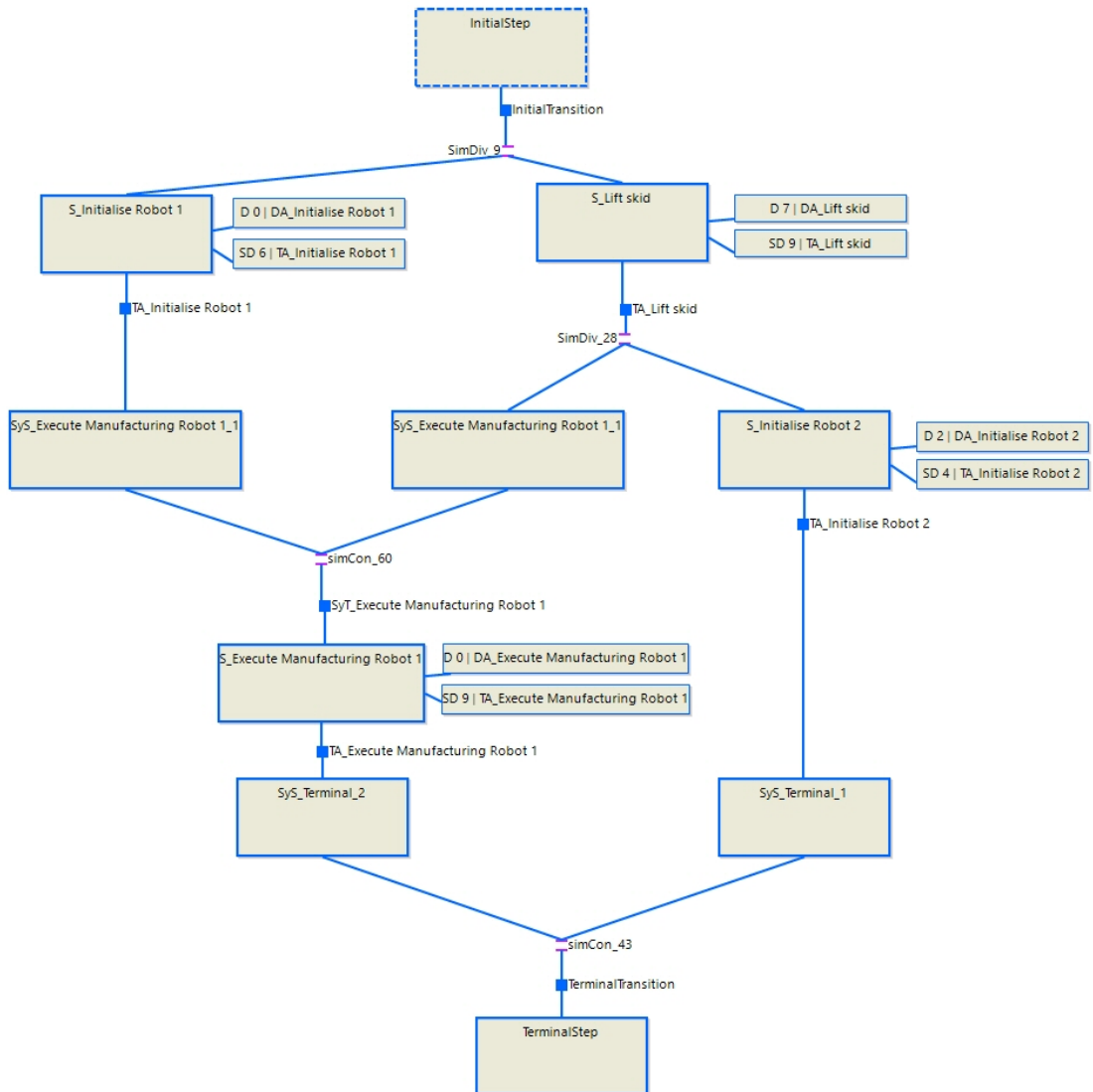


Figure 4.23: Example 5 of a task dependent GANTT chart transformed into the IML

Case A: There exists only one predecessor node transition for the terminal state

Case B: There exists one simultaneous convergence for all synchronisation states

Example 1 - E1

In the first example (E1) an AONN with single pre- & successor relations is transformed into IML (Figure 4.24). The resulting IML output model is shown in 4.25.

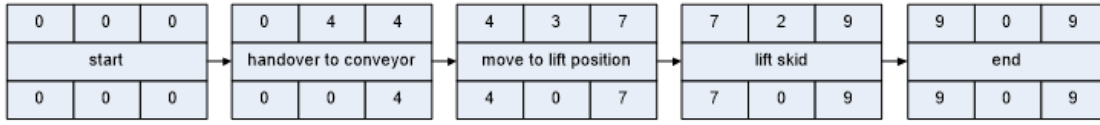


Figure 4.24: AONN - Example E1

In this example the following cases from Table 4.8 are tested.

		Case					
		A	B	C	D	E	F
Step	1	X					
	2	X					
	3	X					
	4	X					
	5	X					
	6	X		X			
	7	X					

Table 4.8: Used cases in AONN - Example E1

Example 2 - E2

In the second example (E2) an AONN with pre- & successor relations is transformed into IML (Figure 4.26). Important is the testing of nodes having the same pre- & successor node. The resulting IML output model is shown in 4.27.

In this example the following cases from Table 4.9 are tested.

		Case					
		A	B	C	D	E	F
Step	1	X					
	2	X					
	3	X					
	4	X					
	5	X	X				
	6	X		X	X	X	
	7	X					

Table 4.9: Used cases in AONN - Example E2

Example 3 - E3

In the third example (E3) an AONN with pre- & successor relations is transformed into IML (Figure 4.28). In this network the existence of multiple start- & endpoints is tested.

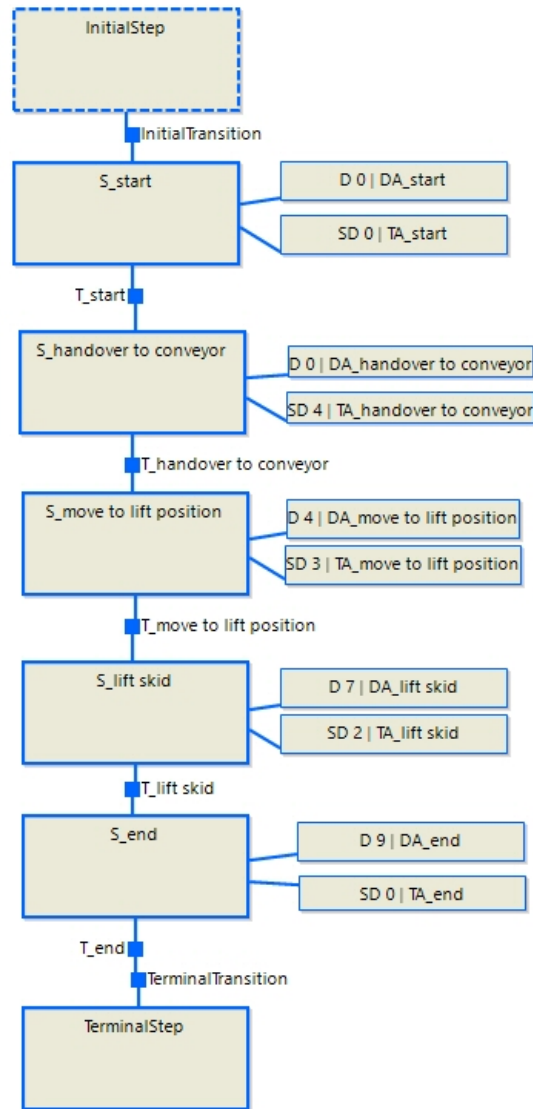


Figure 4.25: Example 1 of an AONN transformed into the IML

Additionally, the Case 6F is especially tested in this example. The resulting IML output model is shown in 4.29.

In this example the following cases from Table 4.10 are tested.

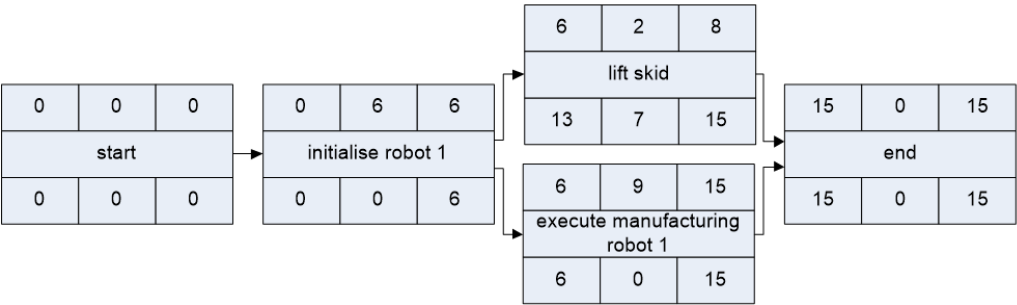


Figure 4.26: AONN - Example E2

		Case					
		A	B	C	D	E	F
Step	1		X				
	2	X					
	3	X					
	4	X					
	5	X	X				
	6		X	X	X	X	X
	7		X				

Table 4.10: Used cases in AONN - Example E3

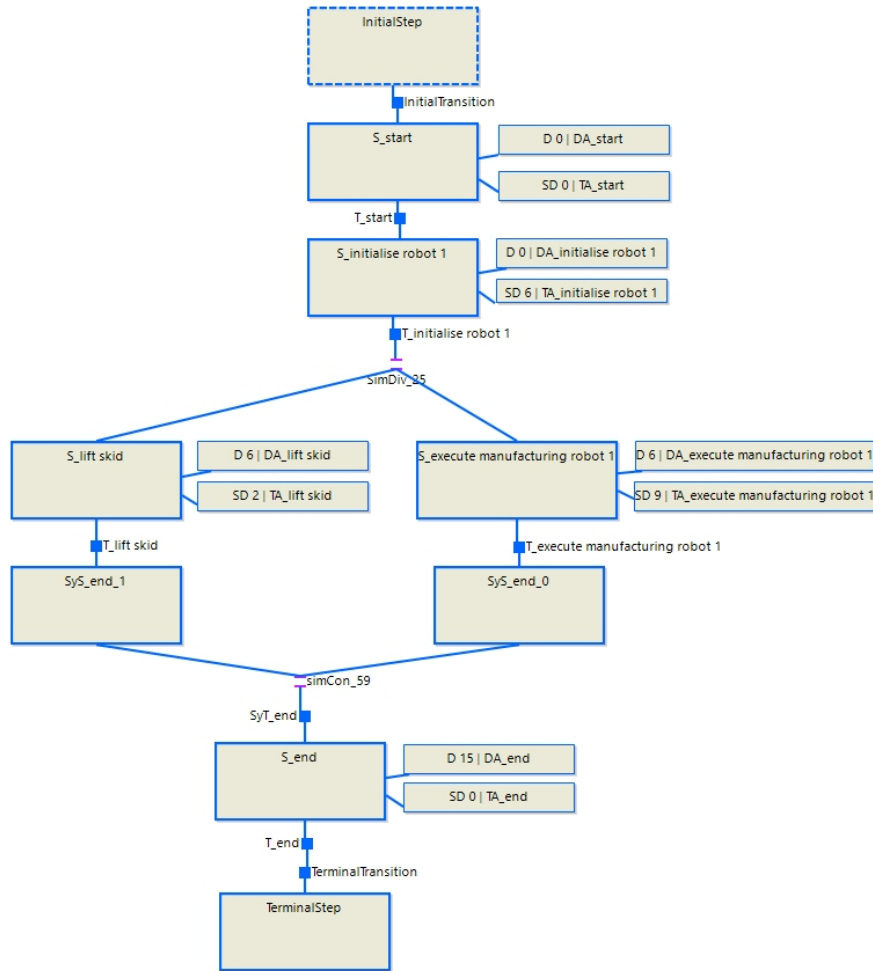


Figure 4.27: Example 2 of an AONN transformed into the IML

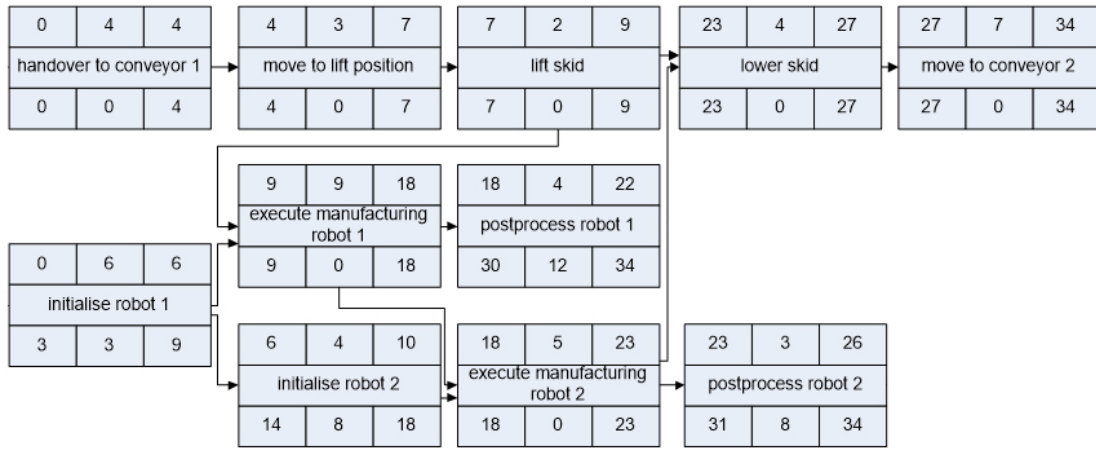


Figure 4.28: AONN - Example E3

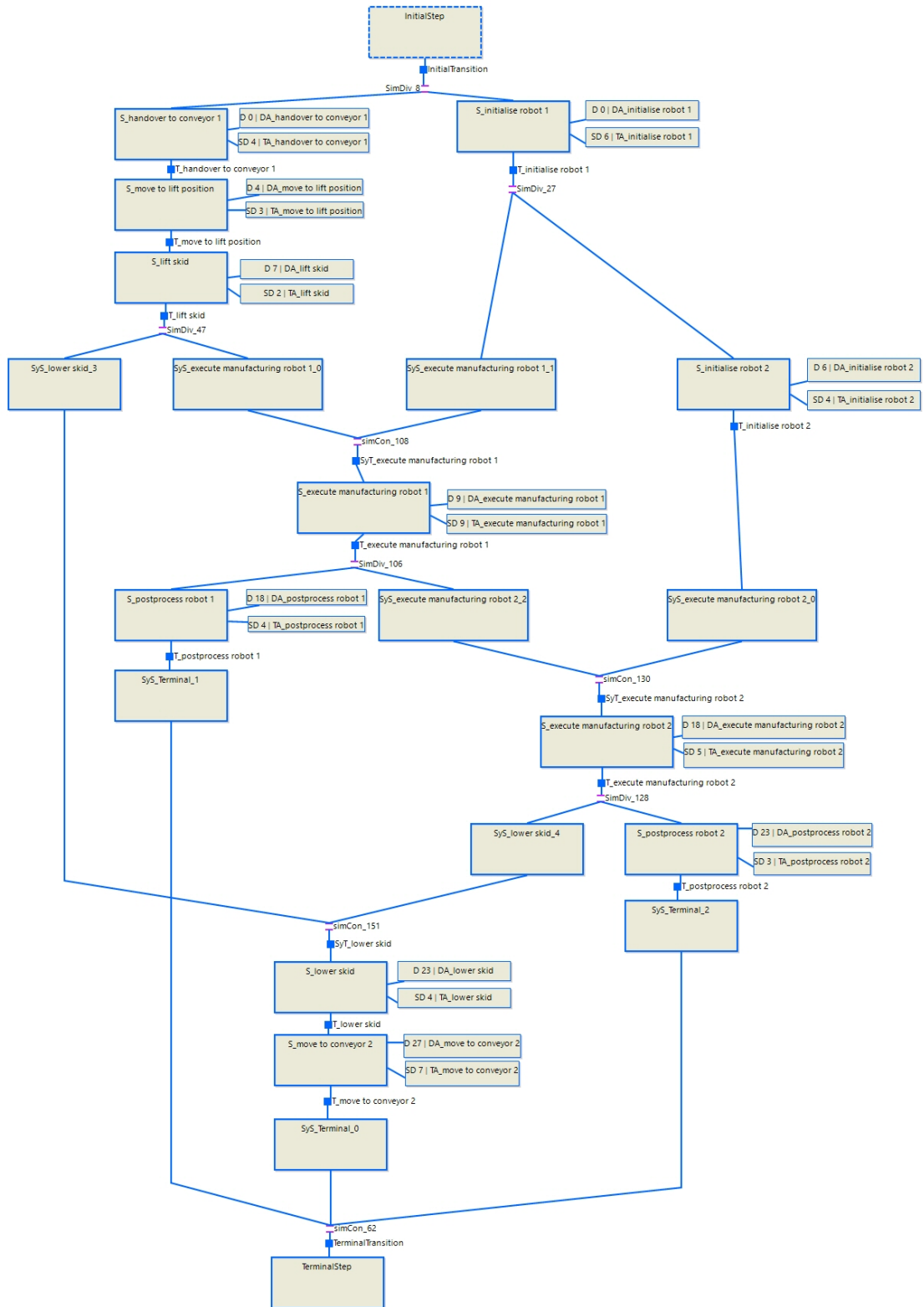


Figure 4.29: Example 3 of an AONN transformed into the IML

Case Study Based Evaluation

5.1 Using IML as a kernel language for exchanging behaviour models

In this section, I perform a case study on IML. The main goal is to evaluate IML to aim as a pivot language for model exchange between different modelling languages. Additionally, the possible loss of information during model exchange has to be tested and evaluated by the comparison of sample input models with generated output models. In my thesis I focus on the two approaches time-related and task-oriented because of the similarities of these approaches.

5.1.1 Research questions

I perform this study to evaluate the capability of using IML as a pivot language for model exchange. Moreover, I specifically want to answer the following research questions (RQ):

1. *RQ1-Completeness*: Is there an information loss when doing model exchanges via the pivot language IML and where do they happen?
2. *RQ2-Correctness*: After transforming a time-related approach into a task-oriented approach, is the resulting model valid?

5.1.2 Case Study Setup

Case and subjects selection

The main objective of this case study is the *successful* transformation of an input sentence of behaviour modelling language X into an output sentence of behaviour modelling language Y over the pivot language IML. In this case successful means that the output

sentence Y is complete and correct. Furthermore, correctness is given when the syntax validation of the output meta-metamodel returns true, whereas completeness could only be reached by having an empty set returned when looking at the delta sentence between the input and output sentence.

A studied case of this case study is the performance of the pivot language. Is it possible to transform any behaviour modelling language into another over the pivot language? Is the pivot language operating differently on different inputs? If I look deeper into these questions, the current transformation from a sentence in language A to a sentence in language B is following the given formula:

$$\begin{aligned}
S_A & \dots \text{Sentence in Language A} \\
S_B & \dots \text{Sentence in Language B} \\
S_{Pivot} & \dots \text{Sentence in Language Pivot} \\
T_{AP} & \dots \text{Transformation from } S_A \text{ to } S_{Pivot} \\
T_{PB} & \dots \text{Transformation from } S_{Pivot} \text{ to } S_B
\end{aligned} \tag{5.1}$$

$$\begin{aligned}
T_{AP}(S_A) &= S_{Pivot} \\
T_{PB}(T_{AP}(S_A)) &= S_B
\end{aligned}$$

As shown in Equation 5.1, to translate a sentence in language A into B over the pivot language, two *different* transformations are needed. There must be a case, which controls the resulting sentence in language B on correctness and completeness. Is there some kind of information loss? Is the needed behaviour available again?

After controlling the sentence in language B another case has to be checked: Is it possible to transform the sentence in language B back to the sentence in language A over the pivot language? Additionally, I introduce two new types of transformations for transforming a sentence in language B into the pivot language and the transformation from the pivot language to the sentence in language A. This is shown in Equation 5.2.

$$\begin{aligned}
T_{BP} & \dots \text{Transformation from } S_B \text{ to } S_{Pivot} \\
T_{PA} & \dots \text{Transformation from } S_{Pivot} \text{ to } S_A
\end{aligned} \tag{5.2}$$

$$\begin{aligned}
T_{BP}(S_B) &= S_{Pivot} \\
T_{PA}(T_{BP}(S_B)) &= S_A
\end{aligned}$$

These Equations 5.1 and 5.2 are showing important facts regarding the language A and the pivot language. Moreover, there are existing two cases, which have to be fulfilled. The first case regards the pivot language. As shown in Equation 5.3 the sentence in the pivot language created after transforming a sentence of language A to a sentence in the

pivot language or transforming B, which must be generated from Equation 5.1, should create the same sentence in the pivot language.

$$\begin{aligned} S_{Pivot} &= S_{Pivot} \\ T_{AP}(S_A) &= T_{BP}(S_B) \end{aligned} \tag{5.3}$$

The other case takes a deeper look on the backward transformation and the expected outcome after transforming a sentence in language A into a sentence of language B and backwards. The goal from exchanging over the pivot language should be that the input sentence A is the same as the resulting output sentence after usage of the four transformations. If the outcome is the same as the input sentence, it is a transformation over the pivot language without any information loss. In Equation 5.4 the principle is shown by formula.

$$S_A = T_{PA}(T_{BP}(T_{PB}(T_{AP}(S_A)))) \tag{5.4}$$

Requirements

First, to start this case study, a system is required which can handle metamodeling for creating input models, write model transformations to generate output models and has the opportunity to execute these transformation rules. The EMF is a package which has every requirement at start and is freely accessible. Additionally, the definition of the pivot language is needed for defining transformation rules.

Setup

Second, the metamodels have to be defined. I decided to use an existing Ecore metamodel for defining GANTT charts [Sot]. The AONN metamodel is created by using the metamodel development process as mentioned in Subsection 3.3.1. The last metamodel is the IML. This metamodel is build up from the standardization document [DLP08]. It follows the defined standards and uses all concepts from the AutomationML standard.

Third, the transformations have to be written in ATL. In the AutomationML standard the transformations from a GANTT or an AONN into IML are defined. These have to be translated into the ATL and especially checked in this case study. Moreover, the backward transformations from IML to GANTT charts or AONNs have to be constructed. There is no standardization for these kinds of transformations and a more intensive testing must be done.

Fourth, sample models for the two kind of charts have to be created. These models must fulfill the requirements of integrating all kind of cases which are existing in the transformation definitions. The used models for this case study could be seen in Section 4.4. These kinds of models are integrating all kinds of cases and the resulting IML outputs are controlled and checked against the standardization.

At last, the input and resulting output sentence of the models must be compared. The goal is a resulting difference between these sentences which has to be an empty set. This has to be checked by using data analysis and define validity procedures.

5.1.3 Data collection procedures

In the data collection part I focus on two examples. Both examples transform the respective model language into the concurrent other language over the pivot language and vice versa. The first example focus on the part of transforming a time-related behaviour modelling language (AONN) into a task-oriented modelling language (GANTT) over the pivot language (IML) and back. As a prerequisite it is important to define measurable metrics in the two modelling languages. These metrics could be read out from the generated xmi-files after a successful transformation.

In the GANTT chart, I focus on the following classes:

- Project
- Task
- Depend

At next a list of measurable metrics is build up defined per class. This metrics must be fulfilled after transforming back into the input language.

Project:

- The number of *Project* instances must be the same.
- The attribute *name* must be set with the input value.

Task:

- The number of *Task* instances must be the same.
- The attribute *id* must be set with the input value.
- The attribute *name* must be set with the input value.
- The attribute *complete* must be set with the input value.
- The attribute *duration* must be set with the input value.

Depend:

- The number of *Depend* instances must be the same.

- The number of relations named *depends* must be the same.
- The attributes defined in every relation *depends* must be the same.

In the AONN, I focus on the following classes:

- Project
- Node

At next a list of measurable metrics is build up defined per class. This metrics must be fulfilled after transforming back into the input language.

Project:

- The number of *Project* instances must be the same.
- The attribute *name* must be set with the input value.

Node:

- The number of *Node* instances must be the same.
- The attribute *name* must be set with the input value.
- The attribute *delay* must be set with the input value.
- The attribute *duration* must be set with the input value.
- The attribute *ESTP* must be set with the input value.
- The attribute *LSTP* could be set with the input value. It is not necessary because it could be calculated by summing up the values of ESTP and delay.
- The attribute *EETP* could be set with the input value. It is not necessary because it could be calculated by summing up the values of ESTP and duration.
- The attribute *LETP* could be set with the input value. It is not necessary because it could be calculated by summing up the values of ESTP, duration and delay.
- The number of relations named *successor* must be the same.
- The relationships *successor* between a node and another node must be as defined in the input model.
- The number of relations named *predecessor* must be the same.
- The relationships *predecessor* between a node and another node must be as defined in the input model.

5.1.4 Analysis procedures

In this thesis, I focus on qualitative data analysis. After defining the important measurable metrics in the previous section, conclusions from the generated data have to be derived.

Time-Related approach (AONN)

First, the input model AONN is chosen. I decided to use the example from Section 4.4.2 as S_A defined in Equation 5.1. Additionally, GANTT is defined as the variable S_B as sentence in language B. As declared in Equation 5.4, I transform the sentence of language A (S_A) over four different transformation files named T_{PA} , T_{BP} , T_{PB} and T_{AP} . The resulting information of Equation 5.4 is stored in $S_{A'}$.

Sentence of language A (S_A) - Input Model

The following values are defined in the example from Section 4.4.2 and shown in Table 5.1.

ID	Node Name	Delay	Duration	EETP	ESTP	LETP	LSTP
1	start	0	0	0	0	0	0
2	handover to conveyor 1	0	4	4	0	4	0
3	initialise robot 1	3	6	6	0	9	3
4	move to lift position	0	3	7	4	7	4
5	lift skid	0	2	9	7	9	7
6	execute manufacturing robot 1	0	9	18	9	18	9
7	postprocess robot 1	12	4	22	18	34	30
8	initialise robot 2	8	4	10	6	18	14
9	execute manufacturing robot 2	0	5	23	18	23	18
10	postprocess robot 2	8	3	26	23	34	31
11	lower skid	0	4	27	23	27	23
12	move to conveyor 2	0	7	34	27	34	27
13	end	0	0	34	34	34	34

Table 5.1: Input metrics for AONN sentence A (S_A)

Additionally, the relationship of each node is presented in Table 5.2. Predecessor relations are marked with a P and successor relations with a S. On the x-axis and y-axis the representative id of the nodes, described in Table 5.1, are shown. The table could be read by finding the node on the y-axis and looking along the x-axis to find the concurrent relationships.

Sentence of language A' ($S_{A'}$) - Output Model

After transforming the input sentence (S_A) via Equation 5.4 into the output sentence ($S_{A'}$), the output metric values are plotted into Table 5.3.

At next the relationship table of the output sentence $S_{A'}$ is shown in Table 5.4.

ΔA - Difference Sentence

ID	1	2	3	4	5	6	7	8	9	10	11	12	13
1		S	S										
2	P			S									
3	P					S		S					
4		P			S								
5				P		S					S		
6			P		P		S		S				
7						P							S
8			P						S				
9						P		P		S	S		
10									P				S
11					P					P		S	
12											P		S
13							P			P		P	

Table 5.2: Relationship table of the AONN input sentence A

ID	Node Name	Delay	Duration	EETP	ESTP	LETP	LSTP
1	start	0	0	0	0	0	0
2	handover to conveyor 1	0	4	0	0	0	0
3	initialise robot 1	0	6	0	0	0	0
4	move to lift position	0	3	0	4	0	0
5	lift skid	0	2	0	7	0	0
6	execute manufacturing robot 1	0	9	0	9	0	0
7	postprocess robot 1	0	4	0	18	0	0
8	initialise robot 2	0	4	0	6	0	0
9	execute manufacturing robot 2	0	5	0	18	0	0
10	postprocess robot 2	0	3	0	23	0	0
11	lower skid	0	4	0	23	0	0
12	move to conveyor 2	0	7	0	27	0	0
13	end	0	0	0	34	0	0

Table 5.3: Output metrics for GANTT sentence A' ($S_{A'}$)

The ΔA difference sentence is defined as the difference between sentence A and sentence A' and is measured by comparing the attributes of all nodes in the sentence. With this comparison technique it is possible to quantify the information loss between the input and the generated output sentence. If ΔA is an empty set, then no information is lost. On the other hand a non-empty set deals with a loss of information.

In my case the result of comparing Table 5.1 and Table 5.3 is a non-empty set, because attribute information from nodes are lost. The following attributes are missing: delay, LSTP, EETP, LETP. The attribute LSTP could be calculated by summing up the

ID	1	2	3	4	5	6	7	8	9	10	11	12	13
1		S	S										
2	P			S									
3	P					S		S					
4		P			S								
5				P		S					S		
6			P		P		S		S				
7						P							S
8			P						S				
9						P		P		S	S		
10									P				S
11					P					P		S	
12											P		S
13							P			P		P	

Table 5.4: Relationship table of the GANTT output sentence A'

values of ESTP and of the delay of a node. The attribute EETP could be calculated by summing up the values of ESTP and duration. The attribute LETP could be calculated by summing up the values of ESTP, duration and delay. In this case the attribute delay is necessary to complete the full list of attributes defined for a node. The difference ΔA is a non-empty set, because the attribute delay is lost.

Last, the difference between Table 5.2 and Table 5.4 is an empty set. All relationships between the nodes are obtained and in the right order. No information on pre- and successor relations get lost after transformation.

Task-Oriented approach (GANTT)

At next I change the input sentence to the GANTT language. I decided to use a new example as S_A defined in Equation 5.1. Moreover, AONN is defined as the variable S_B as sentence in language B. As declared in Equation 5.4, I transform the sentence of language A (S_A) over four different transformation files named T_{PA} , T_{BP} , T_{PB} and T_{AP} . The resulting information of Equation 5.4 is stored in $S_{A'}$.

Sentence of language A (S_A) - Input Model

The following values from the input GANTT sentence example are defined to underlie specific cases from the GANTT check-list. The example is shown in Table 5.1.

Additionally, the dependency of each task is presented in Table 5.6. Dependencies between the tasks are marked with a D. On the x-axis and y-axis the representative id of the nodes, described in Table 5.1, are shown. The table could be read by finding the task on the y-axis and looking along the x-axis to find the dependent tasks.

Sentence of language A' ($S_{A'}$) - Output Model

ID	Node Name	Duration	Complete
1	begin procurement	2	2
2	needs assessment	6	8
3	market survey	4	6
4	vendor selection	4	12
5	site preparation	8	16
6	network installation	8	20
7	special hardware	6	18
8	network available	2	22

Table 5.5: Input metrics for GANTT sentence A (S_A)

ID	1	2	3	4	5	6	7	8
1								
2	D							
3	D							
4		D						
5		D						
6				D				
7				D				
8						D	D	

Table 5.6: Relationship table of the GANTT input sentence A

After transforming the input sentence (S_A) via Equation 5.4 into the output sentence ($S_{A'}$), the output metric values are plotted into Table 5.7.

ID	Node Name	Duration	Complete
1	begin procurement	2	2
2	needs assessment	6	8
3	market survey	4	6
4	vendor selection	4	12
5	site preparation	8	16
6	network installation	8	20
7	special hardware	6	18
8	network available	2	22

Table 5.7: Output metrics for AONN sentence A' ($S_{A'}$)

At next the dependency table of the output sentence $S_{A'}$ is shown in Table 5.8.

ΔA - Difference Sentence

The ΔA difference sentence is defined as the difference between sentence A and sentence

ID	1	2	3	4	5	6	7	8
1								
2	D							
3	D							
4		D						
5		D						
6				D				
7				D				
8						D	D	

Table 5.8: Dependency table of the AONN output sentence A'

A' and is measured by comparing the attributes of all tasks in the sentence. With this comparison technique it is possible to quantify the information loss between the input and the generated output sentence. If ΔA is an empty set, then no information is lost. On the other hand a non-empty set deals with a loss of information.

In this case the result of comparing Table 5.5 and Table 5.7 is an empty set, because all attribute information from every task is compared the same. No attribute is missing or falsely transformed.

Moreover, the difference between Table 5.6 and Table 5.8 is again an empty set. All dependency relations between the tasks are the same. No information about any dependency is lost after transformation.

5.1.5 Validity procedures

I declare that, I have checked all examples from Subsection 4.4.1 and Subsection 4.4.2 against the Equation 5.4. In all appropriate cases the result was the same as mentioned in Subsection 5.1.4. Most of the examples are from the AutomationML group and mentioned in the standardization document [DLPH08]. Additionally, all the cases are representative, because in Subsection 4.4.1 and Subsection 4.4.2 all possible cases are tested.

At next the metric calculation is double checked by summing up the defined values of each task or node and control it with the representative summed up values. Moreover, the metrics are also checked on the counterpart side in the IML. Every metric has a counterpart metric in IML for controlling the definition.

At last, the resulting sentences from applying transformations on different input sentences are not manipulated anyway. The output of the different transformations are renamed for using the generated xmi-file as an input for the next transformation. All transformations are programmed for a generic use. It is defined to use a specific input metamodel and generate a specific output metamodel.

5.1.6 Interpretation of the Results

Answering RQ1. It is important to distinguish between two different cases. A transformation from a modelling language A into IML and vice versa into the same modelling language A has no information loss. The main purpose of the IML is the exchange between different modelling languages and in this case it is possible that information is lost after a transformation like in the example used in Section 5.1.4.

Answering RQ2. Every approach has specific metrics which are needed to calculate other metrics or are needed for the correctness of the modelling language. In my case the metric *delay* is lost after transforming a time-related approach into a task-oriented approach. Moreover, a transformation back into a time-related approach from a task-oriented approach would lead to a lack of information. The metrics *latest start time point*, *latest end time point* and *delay* are lost after transformation.

5.1.7 Threats to validity

First, in the current realization I only focus on two behaviour approaches, the time-related and the task-oriented approach. I do not consider the event-driven approach (e.g. state machines) which may have no information loss after transformation. Second, the standardization of IML itself is currently in a development state for defining a new standard. In some cases my transformation outputs did not match with the current standardization. To overcome this problem another person has taken a look on the standardization document and my realization and we decided to write an error document to the AutomationML group for correcting the false parts of the IML development. Third, the reliability of the written transformations is checked with the given examples. It is possible, that a specific kind of input model can lead to an error output, but only when the defined elements of the input metamodel are not used correctly. Last, the implementations are tested with fixed metamodels. There should be a way of transforming specific realizations of metamodels of a modelling language into one general metamodel and this metamodel is the general input model for the realized transformation.

5.2 Dealing with information loss

As mentioned in Subsection 5.1.6 information loss is happening during transformation of an AONN into a GANTT chart over the pivot language IML. There is a method available for extending an existing metamodel called EMF Profiles and is mentioned in Subsection 3.4.2. The hypothesis is, that an extension of the GANTT metamodel could prevent information loss during transformation.

5.2.1 Definition of the profile

During transformation from an AONN to a GANTT chart, the delay information is lost. For this case a solution for storing this information in the GANTT chart must be applied. Therefore the *Task* class, which stores timing information about a single task, must be

extended. A new *stereotype* called *AONN* is created and linked with the class *Task*. The stereotype has one attribute available called *delay* and is for storing the delay information from the AONN. A graphical representation of the extension is shown in Figure 5.1.

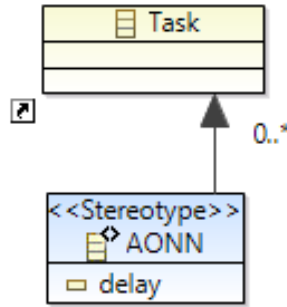


Figure 5.1: Profile for extending a GANTT chart with AONN information

5.2.2 Adjustments of the transformations

At next the extension must be added to my transformations. It is important to adjust the transformations from IML to GANTT and vice versa GANTT to IML. In the first transformation the storing of the delay into the newly generated variable must be ensured. On the other hand, the second transformation must be adapted to read out the delay value and write it into the right class attribute in the IML.

IML to GANTT transformation

First, I will take a look on the adaptations of the transformation IML to GANTT. The basic changes must be done in the Eclipse launcher of the ATL transformation. The adapted configuration is shown in Figure 5.2. The metamodel part is extended with the *Profile* information from EMF profiles. Additionally, the *AonnAPP* declares the profile extension. Moreover, the transformation needs now a second input file and generates a second output file for the extension. In this case the second input file uses the definition of the profile extension. The second output file (xmi) conforms to the profile extension and includes the transformed data from the profile extension.

There have to be done three small adaptations on the existing transformation to include all needed information.

Step 1: The launcher information must be available in the global variable declaration of the ATL.

```
create OUT: GANTT, OUT2: AonnAPP from IN: IML, IN2: Profile;
```

Step 2: Everytime an IML state is transformed into a GANTT task, a new instance of the AONN extension should be created and linked with the task.

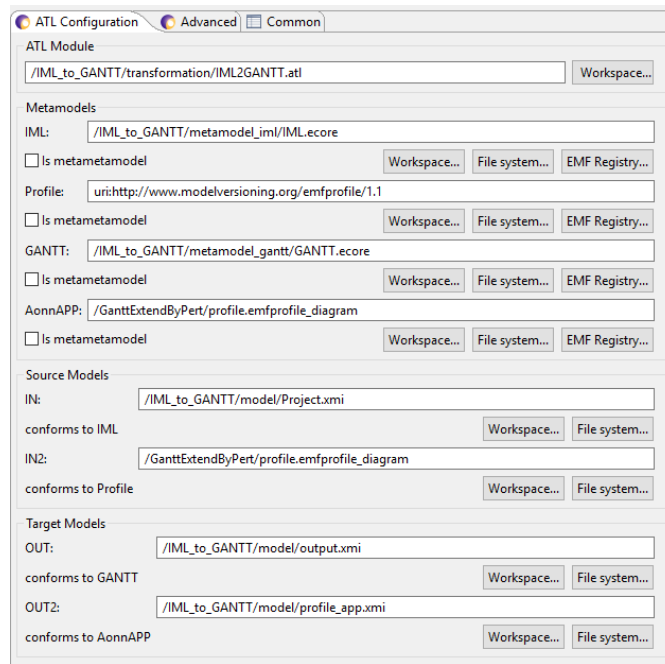


Figure 5.2: Adapted launcher configuration for the transformation IML to GANTT

```
task_delay : AonnAPP!AONN (
    appliedTo <- task,
    extension <- Profile!Stereotype.allInstancesFrom('IN2') ->
        any(x|x.name = 'AONN').extensions -> first()
)
```

Step 3: The attribute *delay* must be filled with the concurrent information.

```
task_delay.delay <- thisModule.getPureDelay(state);
```

GANTT to IML transformation

Second, I will take a look on the adaptations of the transformation GANTT to IML. The basic changes must be done in the Eclipse launcher of the ATL transformation. The adapted configuration is shown in Figure 5.2. The metamodel part is extended with the *Profile* information from EMF profiles. Additionally, the *AonnAPP* declares the profile extension. Moreover, the transformation needs now a second input file and generates a second output file for the extension. In this case the second input file is the created output file from the transformation IML to GANTT, which stores the information about the delay attribute and the associated task. The second output file could declare information for future transformations but is not used in my case.

There must be three small adaptations on the existing transformation to include all needed information.

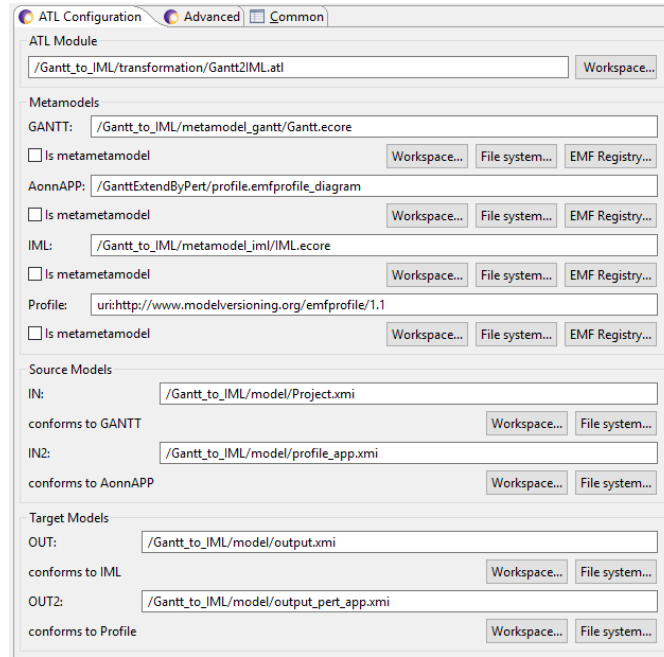


Figure 5.3: Adapted launcher configuration for the transformation GANTT to IML

Step 1: The launcher information must be available in the global variable declaration of the ATL.

```
create OUT: IML, OUT2: Profile from IN: GANTT, IN2: AonnAPP;
```

Step 2: Everytime a GANTT task is transformed into an IML state, a corresponding instance of the AONN extension should be looked up.

```
aonn_app : AonnAPP!AONN (aonn_app.appliedTo = task)
```

Step 3: The attribute *delay* must be set on the right place in the IML state on the corresponding time element.

```
delay <- aonn_app.delay
```

5.2.3 Extended Example

At next I want to show an example which is extended with the EMF Profiles technique. For this case, I want to start with the example from Section 5.1.4. I adapted the transformations from IML to GANTT and GANTT to IML as described in the subsection above and extended the GANTT metamodel with the EMF Profile for AONN.

Sentence of language A'' ($S_{A''}$) - Output Model

The output sentence $S_{A''}$ is now storing the information about the delay. The output table with all necessary metrics are shown in Table 5.9

ID	Node Name	Delay	Duration	EETP	ESTP	LETP	LSTP
1	start	0	0	0	0	0	0
2	handover to conveyor 1	0	4	0	0	0	0
3	initialise robot 1	3	6	0	0	0	0
4	move to lift position	0	3	0	4	0	0
5	lift skid	0	2	0	7	0	0
6	execute manufacturing robot 1	0	9	0	9	0	0
7	postprocess robot 1	12	4	0	18	0	0
8	initialise robot 2	8	4	0	6	0	0
9	execute manufacturing robot 2	0	5	0	18	0	0
10	postprocess robot 2	8	3	0	23	0	0
11	lower skid	0	4	0	23	0	0
12	move to conveyor 2	0	7	0	27	0	0
13	end	0	0	0	34	0	0

Table 5.9: Output metrics for sentence A'' ($L_{A''}$)

Moreover, the relationship table of the output sentence $S_{A''}$ is shown in Table 5.10.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13
1		S	S										
2	P			S									
3	P					S		S					
4		P			S								
5				P		S					S		
6			P		P		S		S				
7						P							S
8			P						S				
9						P		P		S	S		
10									P				S
11					P					P		S	
12											P		S
13							P			P		P	

Table 5.10: Relationship table of the output sentence A''

ΔA - Difference Sentence

The extended ΔA difference sentence for this extended example is defined in the difference between sentence A and sentence A''. A deeper look on the differences between Table 5.3 and Table 5.9 leads to the missing of the metrics EETP, LETP and LSTP. The named metrics could be calculated as mentioned in the sections above.

The ΔA difference between Table 5.4 and Table 5.10 leads to an empty set.

Summarized, with the usage of EMF Profiles the information loss could be reduced against zero. The lost metrics could be easily calculated by using other metrics. This means the information loss from Section 5.1.4 is compensated.

Related Work

Best to my knowledge the kernel language based exchange framework for behavioural modelling languages is a new approach to exchange different types of behaviour models. The framework creates a temporary kernel language for defining 1:1 transformation rules between a common behaviour model and the kernel language. Therefore the complexity is low because of only having a fixed transformation from a specific language type to the kernel language.

Nevertheless a lot of related work is done in the section of model transformation in the industrial area. The *Model Morphing Approach* from Marion Murzek [Mur08] is defining a specific set of morphing rules per modelling language type, which have to be executed to generate an integrated model from the input model. The integrated model could then be exchanged 1:1 with a modelling transformation technique to generate an output model. This approach uses an incremental way of changing an input model with generic rules to create a transformable integrated model.

Another approach for doing model transformations is defined by Andy Schürr [SVV12] [Tra05] [Sch94] [SK08]. In his paper *Applications of Graph Transformations with Industrial Relevance* the Triple Graph Grammar (TGG) is used for transforming a *Modal Sequence Diagram* (MSD) into a *Timed Game Automata* (TGA). The approach is using metamodels for the left-hand side (MSD), the right-hand side (TGA) and the correspondence model, which defines the rules for copying or extending source elements into the target model. The correspondence model must be built up between every specific source and target model. This could be very complex and lead to strong dependencies between the models. This results in a generated correspondence model for every transformation target language from a specific model. Moreover, a general export file from a generalized correspondence model could not be generated and imported into other modelling language tools. At last, the TGG approach is an imperative process, while ATL additionally supports declarative constructs.

The *model synchronisation* approach is based on TGG and extended with forward- and backward update functionality between two models to obtain completeness and correctness. The defined TGG correspondence rules between the source and target model is extended by synchronization rules to update the source model when something is changed on the target model and vice versa. This approach is good for synchronising two models which are very similar between each other except some attributes or classes. The synchronisation of two completely different modelling languages leads to a complex system of synchronisation rules, because of the many attributes of each class which must be copied and saved for backward compatibility. The additional generated attributes need to be integrated in the existing data structure for storing the information. This leads to adoptions on the metamodel level and is creating different metamodels of the same modelling language [EEGH15] [LLR17] [GW09].

Industrial transformations for validation purposes like PLC to Petri-Net, is one area which is important for industry [BCC⁺08]. The validation process of PLC diagrams could be very complex. To overcome this problem transformations into another modelling language like Petri-Nets are existing. In particular these could be solved by validating the kernel language IML. When transforming PLC into IML, a standardized validation process could be set-up for validating every modelling language, which could be transformed into IML. Furthermore, validators for different modelling languages could be generalized in a single point.

Summarized, the kernel language based exchange framework has the ability to use the ATL to transform an input model directly into a model instance of the IML. The IML has the ability to produce an export format, which could be directly imported into a modelling tool or transformed into another modelling language. The exchange layer has the ability to perform validation tasks on the resulting IML. I think the kernel language based exchange framework could help to raise the interoperability between different modelling languages and tools to join features and functionality of each language together.

Conclusion and Future Work

7.1 Conclusion

In this master thesis I focused on IML from the AutomationML consortium as an exchange layer between behaviour modelling languages like GANTT and AONN. The big difference between these two modelling languages is the usage of a task-driven and a time-related approach. I challenged IML as a pivot language to find out if an information exchange without any kind of loss between different behaviour languages is possible. Therefore, an Ecore metamodel for IML has been created to take use of the ATL technique for transforming a sentence A into a sentence B over the pivot language IML. Additionally, the transformations for transforming a sentence of language A into IML, IML into a sentence of language B and both vice versa has been created and tested by using respective examples, which are testing a large amount of cases. Finally, a case study was done to demonstrate the strength of IML as a pivot language.

The first example regarding the transformation from GANTT to AONN and back over the pivot language had no information loss. The important metrics are transformed correctly and did not change after using all transformations. The second example started with the AONN and transformed it into the GANTT and back using the pivot language. An information loss concerning one important metric, the delay, occurs. To overcome this problem, the usage of EMF Profiles has been investigated. The GANTT metamodel has been extended and the information loss could be prevented.

Although the pivot language IML is losing information, but it could be used as a pivot language by using boundaries. The exchange between GANTT and AONN and vice versa has no loss of information when the delay is zero. Moreover, the extension of GANTT could help prevent the information loss when the delay is greater than zero.

7.2 Future Work

7.2.1 State Machine Transformations

As next steps the focusing on exchanging event-driven approaches like the state machine should be considered by using a case study. First, an Ecore metamodel should be created by using the UML 2 standardisation. Second, two kinds of transformations must be prepared, the transformation from a state machine into IML and backward. Third, representative examples must be created to test the transformations and the pivot language definition. Last, exchanges between all types of modelling languages must be done to find out which transformations are loosing information and extensions for these kind of Ecore metamodels must be defined to prevent this loss.

7.2.2 Petri Net Transformations

Another important future work is the transformation from IML to Petri nets. Petri nets are used in many kinds of software projects to visualize the behaviour of a system. There are many standards available for exporting and importing these kind of models.

7.2.3 Model Co-Evolution

A solution for model co-evolution should be implemented. This problem happens when a sentence of language A is transformed into a sentence of language B and a change on sentence B is happening afterwards. The transformation from sentence B' into sentence A' creates an updated sentence of A. A technique must be found to update sentence A to create sentence A' .

List of Figures

2.1	Different software tools used in the industrial area [Dra10, p. 5]	6
2.2	Overview of the IML [DLPH08]	8
2.3	Functional Measurement Diagram	10
2.4	4-layer metamodeling Stack [BCW17, p. 83]	11
2.5	Ecore modelling elements at a glance [BCW17, p. 93]	12
2.6	Out-place transformation (left), in-place transformation (right) [BCW17]	13
2.7	Syntax of a Matched-Rule in ATL [BCW17]	14
3.1	Graphical representation of the design science process	18
3.2	Metamodel Development Process	22
3.3	Example model of an AONN [DLPH08]	22
3.4	Metamodel of the AONN	24
3.5	Abstract syntax of the example model by using the defined metamodel	24
3.6	Language exchange over a pivot model	25
3.7	Difference between unidirectional and bidirectional transformations	27
3.8	The abstract concept of transformations [CH06]	28
3.9	EMF Profile Architecture [LWWC11, p. 5]	29
3.10	Graphical representation of the MDA layers	30
3.11	Key concrete classes from the UML2 kernel package [(OM16, p. 42]	31
3.12	The Ecore components relation hierarchy [Foua]	32
3.13	Iterative and Incremental development process	34
4.1	IML Ecore metamodel	39
4.2	Example of a door modelled with a state machine	41
4.3	Example of a GANTT chart with no dependencies resulting in using the global time scale	43
4.4	Example of a non task dependent GANTT chart transformed into the IML	43
4.5	Example of a GANTT chart with dependencies between tasks resulting in using the local time scale	44
4.6	Example of a task dependent GANTT chart transformed into the IML	45
4.7	Ecore metamodel of the GANTT chart [Sot]	46
4.8	Basic structure of an AONN node	47
4.9	Example of an AONN	47

4.10	IML chart of the transformed AONN from Figure 4.9	48
4.11	Ecore metamodel of the AONN	49
4.12	ATL rule hierarchy for transforming a GANTT bar to an IML state	49
4.13	Example of a path calculation	50
4.14	GANTT chart - Example E1	53
4.15	Example 1 of a non task dependent GANTT chart transformed into the IML	54
4.16	GANTT chart - Example E2	55
4.17	Example 2 of a task dependent GANTT chart transformed into the IML	55
4.18	GANTT chart - Example E3	56
4.19	Example 3 of a task dependent GANTT chart transformed into the IML	57
4.20	GANTT chart - Example E4	57
4.21	Example 4 of a task dependent GANTT chart transformed into the IML	58
4.22	GANTT chart - Example E5	59
4.23	Example 5 of a task dependent GANTT chart transformed into the IML	60
4.24	AONN - Example E1	61
4.25	Example 1 of an AONN transformed into the IML	62
4.26	AONN - Example E2	63
4.27	Example 2 of an AONN transformed into the IML	64
4.28	AONN - Example E3	65
4.29	Example 3 of an AONN transformed into the IML	66
5.1	Profile for extending a GANTT chart with AONN information	78
5.2	Adapted launcher configuration for the transformation IML to GANTT	79
5.3	Adapted launcher configuration for the transformation GANTT to IML	80

List of Tables

3.1	Notation table of an AONN	23
3.2	Concept properties of the AONN example model	23
4.1	Notation table for the IML definition used in SIRIUS	40
4.2	State transition table for the example shown in Figure 4.2	41
4.3	Used cases in GANTT chart - Example E1	54
4.4	Used cases in GANTT chart - Example E2	56
4.5	Used cases in GANTT chart - Example E3	56
4.6	Used cases in GANTT chart - Example E4	59
4.7	Used cases in GANTT chart - Example E5	59

4.8	Used cases in AONN - Example E1	61
4.9	Used cases in AONN - Example E2	61
4.10	Used cases in AONN - Example E3	63
5.1	Input metrics for AONN sentence A (S_A)	72
5.2	Relationship table of the AONN input sentence A	73
5.3	Output metrics for GANTT sentence A' ($S_{A'}$)	73
5.4	Relationship table of the GANTT output sentence A'	74
5.5	Input metrics for GANTT sentence A (S_A)	75
5.6	Relationship table of the GANTT input sentence A	75
5.7	Output metrics for AONN sentence A' ($S_{A'}$)	75
5.8	Dependency table of the AONN output sentence A'	76
5.9	Output metrics for sentence A'' ($L_{A''}$)	81
5.10	Relationship table of the output sentence A''	81

Bibliography

- [And81] Norman H. Anderson. *Foundations of information integration theory*. Acad. Press, 1981.
- [BCC⁺08] Darlam Fabio Bender, Benoît Combemale, Xavier Crégut, Jean-Marie Farines, Bernard Berthomieu, and François Vernadat. Ladder metamodeling and PLC program validation through time petri nets. In *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, pages 121–136, 2008.
- [BCW17] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-driven software engineering in practice*. Morgan & Claypool, 2017.
- [BDK01] Eike Best, Raymond R. Devillers, and Maciej Koutny. *Petri net algebra*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2001.
- [BS82] Susan Bonner and Kang G. Shin. A comparative study of robot languages. *IEEE Computer*, 15(12):82–96, 1982.
- [BTL⁺13] Tegawendé F. Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillère. Popularity, interoperability, and impact of programming languages in 100, 000 open source projects. In *37th Annual IEEE Computer Software and Applications Conference, COMPSAC 2013, Kyoto, Japan, July 22-26, 2013*, pages 303–312, 2013.
- [BY12] Zhang Bo and Li Yafen. Research of relational model transformation based on atl. In *IEEE International Conference on Computer Science and Automation Engineering*, pages 115–118, 2012.
- [CH06] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646, 2006.
- [Chi13] David Chisnall. The challenge of cross-language interoperability. *Commun. ACM*, 56(12):50–56, 2013.

- [DLPH08] Rainer Drath, Arndt Lüder, Jörn Peschke, and Lorenz Hundt. Automationml - the glue for seamless automation engineering. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*, pages 616–623, 2008.
- [Dra10] Rainer Drath. *Datenaustausch in der Anlagenplanung mit AutomationML*. Springer-Verlag Berlin Heidelberg, 2010.
- [EEGH15] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation - General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2015.
- [EWSG89] Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, and David A. Gebala. Organizing the tasks in complex design projects. In *Computer-Aided Cooperative Product Development, MIT-JSME Workshop, MIT, Cambridge, USA, November 20/21, 1989, Proceedings*, pages 229–252, 1989.
- [Foua] The Eclipse Foundation. Eclipse modeling framework (EMF) (<http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html>). Last Accessed: 06.07.2017.
- [Foub] The Eclipse Foundation. Sirius (<http://www.eclipse.org/sirius/>). Last Accessed: 16.10.2017.
- [GW09] Holger Giese and Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software and System Modeling*, 8(1):21–43, 2009.
- [HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.
- [HTCH16] Soichiro Hidaka, Massimo Tisi, Jordi Cabot, and Zhenjiang Hu. Feature-based classification of bidirectional transformation approaches. *Software and System Modeling*, 15(3):907–928, 2016.
- [LHK10] Arndt Lüder, Lorenz Hundt, and Andreas Keibel. Description of manufacturing processes using automationml. In *Proceedings of 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010, September 13-16, 2010, Bilbao, Spain*, pages 1–8, 2010.
- [LLP⁺09] Codrut-Lucian Lazar, Ioan Lazar, Bazil Pârv, Simona Motogna, and István Gergely Czibula. Using a fuml action language to construct UML models. In *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2009, Timisoara, Romania, September 26-29, 2009*, pages 93–101, 2009.

- [LLR17] Anthony Legendre, Agnes Lanusse, and Antoine Rauzy. Toward model synchronization between safety analysis and system architecture design in industrial contexts. In *Model-Based Safety and Assessment - 5th International Symposium, IMBSA 2017, Trento, Italy, September 11-13, 2017, Proceedings*, pages 35–49, 2017.
- [LR11] Bernhard Lahres and Gregor Raýman. *Objektorientierte Programmierung; das umfassende Handbuch ; [objektorientierte Programmierung verständlich erklärt ; von den Prinzipien über den Entwurf bis zur Umsetzung ; Praxisbeispiele in UML, Java, C#, C++, JavaScript, Ruby, Python und PHP]*. Galileo Press, 2011.
- [LWWC11] Philip Langer, Konrad Wieland, Manuel Wimmer, and Jordi Cabot. From UML profiles to EMF profiles and beyond. In *Objects, Models, Components, Patterns - 49th International Conference, TOOLS 2011, Zurich, Switzerland, June 28-30, 2011. Proceedings*, pages 52–67, 2011.
- [Mat07] Bräuer Matthias. Design and prototypical implementation of a pivot model as exchange format for models and metamodels in a qvt/ocl development environment, 2007.
- [MFV⁺05] Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, Frédéric Fondement, Philippe Studer, and Jean-Marc Jézéquel. On executable meta-languages applied to model transformations. In *Model Transformations In Practice Workshop*, 2005.
- [Mur08] Marion Murzek. *The model morphing approach; horizontal transformation of business process models*. PhD thesis, Univ. u. Techn. Univ., 2008.
- [(OMa)] Object Management Group (OMG). <http://www.omg.org/>. Last Accessed: 04.07.2017.
- [(OMb)] Object Management Group (OMG). Model driven architecture (MDA) specification, version 2.0. <http://www.omg.org/mda/specs.htm>. Last Accessed: 06.07.2017.
- [(OM16)] Object Management Group (OMG). Meta-object facility (MOF) specification, version 2.5.1. OMG Document Number formal/2016-11-01 (<http://www.omg.org/spec/MOF/2.5.1/>), 2016.
- [(OM17)] Object Management Group (OMG). Semantics of a foundational subset for executable uml models (FUML), version 1.3. OMG Document Number ptc/17-02-01 (<http://www.omg.org/spec/FUML/1.3>), 2017.
- [PLC] PLCOpen. Plcopen technical committee 6: Xml formats for iec 61131-3 version 2.01. http://www.plcopen.org/pages/tc6_xml/index.htm. Accessed: 2017-06-17.

- [Ran11] Andrea Randak. Atl4pros: introducing native uml profile support into the atlas transformation language. Master's thesis, Univ. u. Techn. Univ., 2011.
- [RH09] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.
- [SAC⁺15] Debalina Sengupta, John P. Abraham, Manuel Ceja, Michael A. Gonzalez, Wesley W. Ingwersen, Gerardo J. Ruiz-Mercado, and Raymond L. Smith. Industrial process system assessment: bridging process engineering and life cycle assessment through multiscale modeling. *Journal of Cleaner Production*, 90:142–152, 2015.
- [Sch94] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16-18, 1994, Proceedings*, pages 151–163, 1994.
- [She31] Walter A. Shewhart. *Economic control of quality of manufactured product*. Van Nostrand, 1931.
- [SK08] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In *Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings*, pages 411–425, 2008.
- [Sot] Jean-Sébastien Sottet. Ganttproject (<http://www.ganttproject.biz/>). Last Accessed: 15.08.2017.
- [Ste08] Dave Steinberg. *EMF: Eclipse Modeling Framework, Second Edition*. Addison-Wesley Professional, 2008.
- [SVV12] Andy Schürr, Dániel Varró, and Gergely Varró, editors. *Applications of Graph Transformations with Industrial Relevance - 4th International Symposium, AGTIVE 2011, Budapest, Hungary, October 4-7, 2011, Revised Selected and Invited Papers*, volume 7233 of *Lecture Notes in Computer Science*. Springer, 2012.
- [Tra05] Laurence Tratt. Model transformations and tool integration. *Software and System Modeling*, 4(2):112–122, 2005.
- [Vau95] Robert L. Vaught. *Set theory - an introduction (2. ed.)*. Birkhäuser, 1995.
- [Wie14] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.
- [WW07] Hua Wu and Haifeng Wang. Pivot language approach for phrase-based statistical machine translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007.

- [ZWA⁺13] Zhiyuan Zhang, Bing Wang, Faisal Ahmed, I. V. Ramakrishnan, Rong Zhao, Asa Viccellio, and Klaus Mueller. The five ws for information visualization with application to healthcare informatics. *IEEE Trans. Vis. Comput. Graph.*, 19(11):1895–1910, 2013.