



Norwegian University of
Science and Technology

Evaluate How the STEP Standard AP 242 Could Enable Knowledge Transfer between CAD and KBE Environments

Evaluer hvordan STEP standard AP 242 kan
muliggjøre konvertering mellom CAD og KBE

Jerome Schätzle

Mechanical Engineering

Submission date: April 2016

Supervisor: Ole Ivar Sivertsen, IPM

Norwegian University of Science and Technology
Department of Engineering Design and Materials

THE NORWEGIAN UNIVERSITY
OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ENGINEERING DESIGN
AND MATERIALS

**MASTER THESIS AUTUMN 2015
FOR
STUD. TECHN. JÉRÔME SCHÄTZLE**

**EVALUATE HOW THE STEP STANDARD AP 242 COULD ENABLE KNOWLEDGE
TRANSFER BETWEEN CAD AND KBE ENVIRONMENTS**

***Evaluer hvordan STEP standard AP 242 kan muliggjøre konvertering mellom CAD og
KBE***

The STEP standard provides a framework for representing and exchanging product data independently from any particular system. In the last years there have been several Application Protocols (AP) released in which different functionalities of the standard have been defined. The aerospace and automotive industry respectively drove the development of specific protocols, namely AP 203 and AP 214. The new AP 242 aims at merging these two parallel protocols and at further extending the STEP standard in order to become the cornerstone standard of the cross-process capabilities for interoperability of core engineering design information. The new specifications mainly address the consistent technical product data representation, which is necessary for an effective exchange and long term archiving of this data.

A KBE system with its object oriented paradigm offers many advantages in comparison to the classical engineering approach with its standalone tools for CAD, FEM and all the other parts of the engineering process. As some of these standalone tools are well established in the industry and are often well adapted to the specific needs of the customers, it would be of great use to integrate some of these tools into the KBE environment. A transfer format able to store all the corresponding product knowledge and a format which is easily interpretable by any tool would significantly lower the initial hurdle to implement a KBE system as this could be done step by step.

This master thesis will investigate if the new AP 242 standard could be such a tool. This will be done by looking into its capabilities to store design relevant knowledge. The focus will thereby lay on the interface between CAD and KBE systems. One of the main capabilities would be the translation of parameterized CAD models into KBE modules or classes. The possibility to do this could be examined with the aid of an example STEP file.

The master assignment includes the following:

1. Conduct a literature review on the STEP standard AP 242 and evaluate its capacity to transfer codified knowledge.
2. Learn how to program in the KBE language AML, including geometry modeling.

3. Investigate what translators are available for transmitting AP242 models from and into CAD systems and who supports these translators.
4. Contact different teams developing and supporting AP242 translators and discuss the possibilities to transfer AP242 models into AML code, especially regarding parameterization.
5. If no translator code is available as basis for translating into the KBE language AML, specify how an AML code could be implemented based on a subset of the features available in the AP 242 standard.
6. As far as time allows, based on point 4 or 5 above, implement AML code to import parameterized geometric models in AML and run test examples.

Formal requirements:

Three weeks after start of the thesis work, an A3 sheet illustrating the work is to be handed in. A template for this presentation is available on the IPM's web site under the menu "Masteroppgave" (<http://www.ntnu.no/ipm/masteroppgave>). This sheet should be updated one week before the master's thesis is submitted.


Risk assessment of experimental activities shall always be performed. Experimental work defined in the problem description shall be planned and risk assessed up-front and within 3 weeks after receiving the problem text. Any specific experimental activities which are not properly covered by the general risk assessment shall be particularly assessed before performing the experimental work. Risk assessments should be signed by the supervisor and copies shall be included in the appendix of the thesis.

The thesis should include the signed problem text, and be written as a research report with summary both in English and Norwegian, conclusion, literature references, table of contents, etc. During preparation of the text, the candidate should make efforts to create a well arranged and well written report. To ease the evaluation of the thesis, it is important to cross-reference text, tables and figures. For evaluation of the work a thorough discussion of results is appreciated.

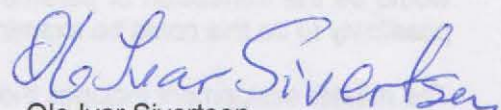
The thesis shall be submitted electronically via DAIM, NTNU's system for Digital Archiving and Submission of Master's theses.

Contact persons:

Ivar Marthinusen, IPM/NTNU
Christos Kalavrytinous, IPM/NTNU



Torgeir Welo
Head of Division



Ole Ivar Sivertsen
Professor/Supervisor

Preface

This master's thesis is a cooperation between the Department of Engineering Design and Materials at the Norwegian University of Science and Technology (NTNU) in Trondheim and the Institute for Engineering Design at the RWTH Aachen University.

I would like to thank you, Professor Ole Ivar Sivertsen for the chance to work in your group on this interesting topic. Thank you for the guidance and the many inspiring discussions about KBE.

Thank you, Ivar and Chris for the nice introduction into KBE, the support and the valuable feedback.

Andrea, thank you for the incredible support over the last year.

I would like to thank Professor Jörg Feldhusen for the opportunity to write my thesis in Norway and Fatmir Sulejmani for the great support despite the distance and for the instructive discussions in the last weeks.

Much appreciated help was provided by Jochen Hänisch and Arne Tøn from *Jotne*¹. Thank you for the software license, the valuable feedback and the hands-on introduction into EDM.

Special thanks goes to Sharon Barber from *Imagecom Inc.*² for the provision of test files and a very instructive Skype call.

I wish to thank *Datakit*³ and *STEP Tools*⁴ for the provided test licences.

Thank you, Lisa for your patience, your support and your incomparable English expertise.

¹<http://www.jotne.com>

²<http://www.aspire3d.com>

³<http://www.datakit.com>

⁴<http://www.steptools.com>

Abstract

For a complementary use of different digital development environments, the exchange of product data between those environments is essential. This thesis evaluates the STandard for the Exchange of Product model data (STEP) in its potential to transfer product data between Computer Aided Design (CAD) and Knowledge Based Engineering (KBE) environments.

The most commonly used STEP protocols mainly provide the structures to represent geometric data. As the International Organization for Standardization (ISO) aims to extend the standard with structures that can represent non-geometric data during all the phases of a product's life cycle, they recently published the STEP AP242 standard, which contains additional data structures that can represent construction history, parameterized and constrained dimensions and features. These elements are used in CAD systems and provide information about the designers' intentions during the construction process.

With the aim to provide a basis for the evaluation of STEP's potential in transferring such data, this thesis extensively reviews the standard and the related literature. In order to support the findings, an example STEP file is generated and transferred into the Adaptive Modeling Language (AML) KBE framework. More specifically, the example file is transferred with the help of parsers both for the EXPRESS structures defined in AP242 and for the STEP file that contains the data model. Finally, the implementation of an appropriate interface in AML enables the successful reconstruction of the example geometry.

Considering the literature review and the insights gained during the implementation of the example geometry, the potential of the STEP standard to provide the data structures to represent knowledge elements, such as construction history, can be approved. However, STEP AP242 is not yet implemented in commercial CAD systems. Moreover, the mapping between the EXPRESS and the AML structures incorporates some challenges. Thus, the STEP standard enables the transfer of such elements between CAD and KBE environments mostly theoretically.

Contents

Assignment	iii
Preface	v
Abstract	vi
List of figures	x
List of tables	xi
Nomenclature	xii
1. Introduction	1
1.1. Background	1
1.2. Motivation	2
1.3. Objectives of the thesis	3
1.4. Research and standardization	4
1.5. Structure	6
2. Theory	8
2.1. Fundamentals	8
2.2. Product data exchange	10
2.2.1. Translators	11
2.2.2. Mapping	12
2.3. Design intent	13
2.4. ISO 10303 STEP standard	16
2.4.1. Structure of STEP	18
2.4.2. Part 11 – The EXPRESS language reference manual	21
2.4.3. Part 21 – Clear text encoding of the exchange structure	24
2.4.4. Part 22 – Standard data access interface	28
2.4.5. Part 55 – Procedural and hybrid representation	28
2.4.6. Part 108 – Parameterization and constraints for explicit geometric product models	30
2.4.7. Part 111 – Elements for the procedural modeling of solid shapes	31
2.4.8. AP 203 – Configuration controlled 3D designs of mechanical parts and assemblies	31

2.4.9. AP 214 – Core data for automotive mechanical design processes	31
2.4.10. AP 242 – Managed model based 3D engineering	32
2.4.11. AP 209 – Multidisciplinary analysis and design	33
2.5. Knowledge Based Engineering	33
2.5.1. Adaptive Modeling Language	33
3. Methodology & Implementation	36
3.1. Concept development	36
3.2. Example STEP file	38
3.2.1. Software tools	38
3.2.2. EDM implementation	40
3.3. EXPRESS structure mapping	44
3.3.1. Software tools	45
3.3.2. Wirth Syntax Notation converter	46
3.3.3. EXPRESS schema parser	48
3.3.4. AML code generation	49
3.4. Data transfer	52
3.4.1. STEP P21 parser	52
3.4.2. AML population	52
4. Results	55
5. Discussion	58
5.1. Knowledge transfer with the STEP standard	58
5.2. Implementation of a STEP translator	61
6. Conclusion	64
7. Future work	66
Bibliography	67
A. File structure	70
B. EDM commands	71
C. Code listings	74
C.1. EDM – procedural query	74
C.2. AML	83
C.2.1. Main	83

C.2.2. Geometric representation classes	94
C.2.3. Feature mapping classes	99
C.2.4. Populate transfer file	105
C.3. Python	109
C.3.1. WSN converter	109
C.3.2. Schema Parser	123
C.3.3. P21 Parser	146
D. WSN rules EXPRESS language	164
E. Risk assessment	213

List of figures

2.1. Data exchange with direct translators.	11
2.2. Data exchange using an intermediate neutral format.	12
2.3. Mapping between units of information. Adapted from Kim et al. (2008).	12
2.4. Usage of 3D data exchange formats as primary format in 2010 (Prawel 2010).	17
2.5. High level structure of the STEP standard. Based on STEP Tools Software (2010) and Loffredo (1999).	19
2.6. Initial and modular STEP architecture. Based on STEP Tools Software (2010) and ISO TC 184/SC4 N1863 (2005).	20
3.1. Implementation overview.	37
3.2. Example geometry, generated in Siemens NX10.	40
3.3. Hierarchical structure <code>Procedural_Shape_Representation_Sequence</code>	42
3.4. Hierarchical structure <code>Bound_Parameter_Environment</code>	43
3.5. Hierarchical structure of an <code>Edge_Curve</code> in the STEP exchange file.	53
4.1. Object tree of the imported model. Geometrical representation in different stages of the construction history.	55
4.2. Example geometry that was imported into AML.	57
A.1. File dependencies of binding pre-processing and post-processing.	70

List of tables

2.1. Correlation of the terminology conventions in OOP, AML and EXPRESS . . .	10
2.2. Infrastructure Parts (IS), Integrated Resources (IR) & Application Protocols (AP) discussed in this work.	21
2.3. Quick reference mapping table (ISO 10303–21:2002).	25
2.4. Syntax of different data types in instance descriptions.	26

Nomenclature

Acronyms

3D	Three-Dimensional
AAM	Application Activity Model
AI	Artificial Intelligence
AIC	Application Interpreted Constructs
AML	Adaptive Modeling Language
AP	Application Protocol
API	Application Programming Interface
AR	Application Resources
ARM	Application Reference Model
B-rep	Boundary Representation
BNF	Backus Normal Form
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Machining
CAX	Computer Aided x
CC	Conformance Class
CHAPS	Construction History and ParametricS
CNC	Computer Numerical Control
DoD	Department of Defense
EBNF	Extended Backus Normal Form
EDM	EXPRESS Data Manager
ENGEN	Enabling Next GENERation design
FBS	Frame Based System

FEM	Finite Element Method
GR	Generic Resources
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDL	Interface Definition Language
IGES	Initial Graphics Exchange Specification
IR	Integrated Resources
ISO	International Organization for Standardization
IT	Information Technology
JT	Jupiter Tessellation
KB	Knowledge Base
KBE	Knowledge Based Engineering
LEAP	Linked Engineering and mAnufacturing Platform
LISP	LISt Processing
LOTAR	LOng Term Archiving and Retrieval project
MS	Mapping Specifications
NIST	National Institute of Standards and Technology
OO	Object Oriented
OOP	Object Oriented Programming
PDF	Portable Document Format
PDM	Product Data Management
PIP	PIP Installs Packages
PLY	Python Lex–Yacc
PMI	Product and Manufacturing Information
RBS	Rule Based System
SCL	STEP Class Library
SDAI	Standard Data Access Interface

STEP	Standard for the Exchange of Product model data
VGL	Virtual Geometry Layer
WSN	Wirth Syntax Notation
XML	Extensible Markup Language

Symbols

\subset	Subset of
ϵ	Epsilon – empty string
\$	US American Dollar
k	kilo – one thousand

1. Introduction

1.1. Background

Over the last years, several digital product development environments evolved on the market. While each of these environments has its individual strong and weak points, taken together they meet the diverse requirements of the different phases of a product's development process. Therefore, the product is usually developed in multiple environments. In order to enable an efficient cooperation between the environments, the product data has to be exchanged via suitable interfaces (Feldhusen et al. 2013).

In the field of Computer Aided Design (CAD) systems, several exchange formats have evolved over the years for this purpose. They are primarily designed to translate geometry models between different systems and describe the surface boundary between the solid and non-solid parts. This so-called Boundary representation (B-rep) is composed of faces, edges and vertices, which are used as reference in the receiving system to recreate the geometry of the object.

This pure geometric information is sufficient for some use cases, such as for generating Computer Aided Machining (CAM) files. For several other uses however, this so-called *dumb geometry* (Pratt et al. 2006) is not sufficient, because the lost information is needed in the receiving system and has to be recreated manually. Especially the exchange history, parameters, constraints and features are important to understand the intent behind the design of the model. For this reason, when it comes to archiving or exchanging geometry without the loss of such information, the native file format of the CAD system in use is often the only option. In other cases however, these limitations are even required, for example if a subcontractor is not allowed get access to this information. Stiteler (2004) states that most companies do not require the exchange of parametric and constraint data with other companies, except in cases of dedicated collaborative design. Nevertheless, a neutral file format that is able to store these elements is of great interest for many companies, especially for internal use (Stiteler 2004).

The international Standard for the Exchange of Product model data (STEP) provides a definition for such a neutral file format. As it is regularly extended with new sub-standards, STEP covers more and more areas of a product's life cycle. Recently, new structures were implemented into the standard that provide the ability to map elements of *design intent*,

such as exchange history, parameters, constraints and features. However, commercial CAD systems do not support most of these new structures yet (Barber et al. 2010).

Another approach to product development is Knowledge Based Engineering (KBE). KBE systems provide methods for capturing knowledge rules and create multidisciplinary models based on these rules. This includes not only the design process, but the entire product and process development cycle. KBE combines approaches from Object Oriented Programming (OOP), Artificial Intelligence (AI) and CAD, facilitating customized or variant design automation solutions (Chapman et al. 1999 and La Rocca 2012). KBE applications are built with so-called KBE platforms. These applications are dedicated programs to solve specific problems, such as the modeling of a hardware product or the manipulation of other types of data (La Rocca 2012). Some KBE frameworks are implemented into CAD systems, such as Knowledge Fusion in *Siemens NX*, and some are standalone systems, such as the Adaptive Modeling Language (AML) by *TechnoSoft*. KBE systems are usually based on programming languages that support the object oriented paradigm. La Rocca (2012) refers to the common roots of Artificial Intelligence (AI) research and KBE to explain that most of the KBE languages are derived from or based on one of the first programming languages Lisp. The programming language used in this thesis is AML, which was originally written in Common Lisp and subsequently recoded in a proprietary language with a similar syntax (La Rocca 2012).

1.2. Motivation

KBE systems provide the flexibility to store geometric, as well as non-geometric product data. With this ideally complete description of a product, all phases of the product's life cycle can be supported with a specific data set based on a central data model. The geometric modeling with a traditional CAD system however can offer some advantages, especially when the focus does not lie on complex system definitions with a lot of non-geometric data, but on the design of a product whose appearance is more important than the technical implementation. The direct interaction with the graphical representation during the modeling process in CAD environments helps to develop the design or the technical implementation of products. In order to describe a geometry based on rules, the engineer first has to visualize it. Fish et al. (1990) state that the mind's ability to visualize an object is greatly supported by visual feedback, such as sketches or digital images.

Opinions are divided on which of these approaches is better, but La Rocca (2012) points out that a discussion about the advantages of the KBE programming approach compared to the

interactive operation of a CAD system is fundamentally misdirected. Instead, the author recommends acknowledging the convenience based on different applications and exploiting the synergy between the two methods.

Since CAD was and still is one of the most commonly used digital tools in product development, the data bases of most companies are full of old digital product designs, which are stored in native file formats. This forces the companies to maintain licensed versions of the appropriate software for the period they want to be able to access the files. Naturally, that is also one of the reasons why CAD vendors are reluctant to implement some of the STEP structures, as they benefit from the customer's dependency on their software. A system independent format that can transfer all information defined in the source system would ensure that the models can be stored and accessed over time without any loss of information. At the same time, companies aiming to introduce a KBE or a new CAD environment would notably benefit from a method to transfer their old designs into the new environment.

The potential financial savings ascribable to such an exchange format are difficult to estimate, as there are many short and long term processes involved. Stiteler (2004) however conducted a business case with several companies to evaluate the potential of a new method of exchanging CAD information that maps construction history, parametric relationships and constraints of the delivering system. Stiteler (2004) was able to translate test models with a success rate of 67%, from which 47% were translated with complete accuracy and 20% required minor rework. Thus, the project verified that it is possible to exchange CAD models which contain at least some parts of the *design intent* using the STEP standard. One of the companies participating in the project claims that a complete implementation of such a method would save over 400k\$ for one of their CAD migration projects alone.

1.3. Objectives of the thesis

In order to enable a synergy between CAD and KBE environments, the main task is to establish the exchange of geometric models together with the elements of *design intent* between those environments. One important method in that context is the transfer of geometry with the help of the STEP exchange format. Most KBE environments already have interfaces for the import of STEP files, but these interfaces only support geometric data, such as points, edges and surfaces. Similarly, the implementation of non-geometric data into the STEP export interfaces of CAD systems are very limited. This is due to the CAD vendors, who need some time to implement the newest STEP data structures, after their release as an extension of the STEP ISO standard. These extensions are packaged in so-called Application Protocols

(AP) and aim to provide data structures for specific fields, like AP203 for the aerospace and AP214 for the automotive industry. These two APs are the most widely supported protocols and were merged into the new AP242. With the second edition of AP203, new structures were introduced that potentially enable the transfer of *design intent*.

Hence, one of the main questions investigated in this thesis is, whether it is possible to transfer elements of *design intent*, such as construction history, parameters, constraints and features with the STEP standard. The objective of the thesis is to provide an overview of the structure of the STEP format and to point out, which of the parts contain the structures necessary to map the elements of *design intent*. With this information, the aim is to import STEP files into the KBE framework AML. Based on the transfer of an example geometry that contains some of the discussed elements of *design intent* the feasibility of such an implementation is evaluated. The scope of this work therefore includes to investigate the level of implementation of the STEP modules into the most common CAD systems, such as SolidWorks, CATIA, Siemens NX or Pro/ENGINEER (Prawel 2010). It is envisaged to generate the example geometry with one of those CAD systems or an external tool that is able to generate STEP files. For this reason, it is planned to contact different vendors of STEP related software to assess the suitability of their tools. Based on the insight gained by investigating these questions, the final aim of the thesis is to evaluate the question, if and how the STEP standard AP242 enables knowledge transfer between CAD and KBE environments.

1.4. State of research and standardization

Over the years, several standards have been developed that exchange various aspects of product model data among heterogeneous CAD systems. Among these, the STEP format has become the most promising standard for representing and exchanging product data, superseding various national and international standards. It has been endorsed by leading organizations in aerospace, automotive and shipbuilding industry as well as the US Department of Defense (DoD), such as *Boeing*, *Lockheed Martin*, *IBM*, *Rockwell* and *NASA* (Stiteler 2004). Moreover, the DoD announced that they consider STEP as their exchange standard of choice. (Reynolds 2002, as cited in Stiteler 2004).

Until recently, the common product data exchange formats mainly focused on the exchange of pure geometrical data. Therefore, several projects over the last years aimed to enhance the exchange methods according to the new requirements described in the previous sections. Some of the projects that aimed or still aim to develop methods to exchange more than pure geometric models are summarized in the following section.

One of the first projects in this field was ENGEN (ENabling Next GENERation design). The project used a representation model based on the STEP methodology and mainly focused on the transfer of geometric constraints in 2D sketches (Kim et al. 2008).

Based on the work of ENGEN, the Construction History and ParametricS (CHAPS) project aimed at providing an initial business case for smart CAD exchange using the at the time emerging second edition of AP203 (ISO 10303–203:2005). For this purpose, they developed their own translators, which provides a new method of exchanging CAD information including construction history, parametric relationships and constraints defined in the source system (Stiteler 2004). The method is based on the Application Programming Interfaces (API) of the sending and the receiving system. Via the API, the model structure is read from the sending system and subsequently mapped to a neutral exchange format, which is based on STEP AP203 Ed2. From this exchange file, the authors rebuild the model in the receiving system as a native file by accessing the system’s API. As mentioned in section 1.2, they succeeded in the attempt to exchange different example models between several CAD systems. The involved organizations all expressed the immense potential of such a transfer method (Stiteler 2004). Originally, the project was devised as a two–phase project, initially targeting construction history only. While parametric and constraint information was added later, the publication of these parts was prevented due to technical issues. The development progress was regularly reported in the CAx Implementer forum¹, which is a joint testing effort between the two organizations *PDES Inc*² and *ProSTEP iViP*³. In general, the objective of the forum is to accelerate CAx translator development. For that purpose, it provides common test activities for the CAD domain by merging similar platforms of the two organizations.

One of the partners in the CHAPS project, *Theorem Solutions*⁴, developed the CHAPS translators. They continued developing commercial versions of these translators (Stiteler 2004) and nowadays provide various data translation products. Their flagship application is the CADverter, which allows the direct translation between major CAD systems as well as standard based formats, such as STEP.

Another approach to exchange procedural model data is to use the journal or trail files created by CAD systems, which contain records of every action executed in the system. A team from *Korea Advanced Institute of Science and Technology* has developed a non–STEP neutral format with a layered ontology that maps the structure of CAD features (Seo et al. 2005). All information is translated from these journal files into a neutral format. Moreover,

¹<https://www.cax-if.org>, last accessed: 2016-03-26

²<https://www.pdesinc.org>, last accessed: 2016-03-26

³<http://www.prostep.org/nc/en.html>, last accessed: 2016-03-26

⁴<http://www.theorem.com>, last accessed:2016-03-26

the team was involved in the development of ISO 10303–112:2006, which allows the exchange of construction history representations of 2D profiles or sketches (Kim et al. 2008).

Pratt et al. (2006), Kim et al. (2008) and Kim et al. (2011) report on prototype translators that are able to use the new STEP resources mentioned in section 1.1. According to them, the standardized exchange of CAD models containing *design intent* information is possible and has been successfully demonstrated. Another conclusion from these sources is that the development of translators for such exchange files is more complex than for previous versions of the standard, since the functions in the CAD systems cannot be mapped to each other one-to-one.

Barber et al. (2010) present an interpretation of the schema definitions of these new STEP structures and show their approach to map construction history and features between CAD systems. The authors describe test files that were generated with their software and conclude that the new data structures in AP203 Ed2 provide a way to store *design intent* using constructional operations and thus facilitate the exchange of feature based CAD models.

The international LOng Term Archiving and Retrieval (LOTAR) project is hosted by *PDES* and *ProSTEP*. Their objective is to provide a solution for long term archiving and retrieval of digital data, such as 3D CAD and other product data. LOTAR uses STEP AP203 and AP214 for this purpose, as it is the most advanced open format in their view.⁵

In the context of KBE, Lützenberger et al. (2012) investigate methods for knowledge acquisition and codification. Their work is part of the *LinkedDesign* project, which aims at developing the Linked Engineering and mAnufacturing Platform (LEAP). LEAP federates all product life cycle information that is relevant to drive engineering and manufacturing processes.⁶ One question Lützenberger et al. (2012) investigate in their work is, whether the STEP standard can be used to codify information in a KBE environment on the one hand and to exchange data between CAx and KBE environments on the other hand. The authors especially refer to the new STEP structures introduced with the newest APs, because of their ability to transfer elements of *design intent*. They see potential in the STEP standard for these use cases and state that the standard should be tested further regarding that question.

1.5. Structure

Chapter 2 contains a summary of the STEP standard and its parts that are related to the capture and exchange of *design intent*. Additionally, the basics of KBE and other methods

⁵<http://www.lotar-international.org/home.html>, last accessed: 2016-04-05

⁶<http://www.linkeddesign.eu>, last accessed: 2016-03-26

used in the thesis are outlined. In chapter 3, the methodology used to implement the different parts of the transfer process from CAD to KBE is described. Moreover, the used software tools and their setups are explained to establish a basis for the description of the implementation itself. The results are summarized in chapter 4 and discussed in chapter 5. The discussion aims at answering the questions posed in chapter 1. The conclusion in chapter 6 summarizes the outcome of the thesis regarding the knowledge transfer between CAD and KBE environments. Chapter 7 contains suggestions on possible next steps to further investigate the possibilities of knowledge transfer between KBE and CAD environments. Finally, long listings of source code and supporting material are collected in the appendix.



2. Theory

This chapter covers the fundamental introductions into different fields, on which the investigations in this thesis are based on. In order to be able to discuss ideas and structures from these different fields in an unambiguous way a common naming convention has to be defined, as each of these fields has developed its own terminology over the years. Therefore, the first section of this chapter introduces various paradigms from different fields and points out the differences in their terminology. The subsequent sections go more into detail in each of the fields.

2.1. Fundamentals

The fields on which this thesis mainly is based on are Object Oriented Programming, Knowledge Based Engineering and more specifically one of its frameworks Adaptive Modeling Language (AML), and the EXPRESS language that is used to define STEP structures. Thus, this section introduces each of these fields together with its naming conventions and concludes with the naming convention used in this thesis.

Object Oriented Programming

Object Oriented Programming is based on the idea to represent real world objects with object constructs that codify the properties of these objects and their relationship between each other. These object types are specified in *class* definitions that serve as a template to create such an object. These *classes* contain so-called *attributes* that describe the properties of a specific type of object. In order to manipulate these *attributes*, *classes* can be associated with *methods* that specify manipulation procedures. A fundamental principle of OOP is *inheritance*, which means that *classes* can *inherit* from each other, namely their *attributes* and their *methods*. This relation is referred to as a *superclass – subclass* relation. In order to materialize these *classes* into actual objects, they are *instantiated*. The resulting *instance* or *object* is a unique version of the corresponding *class* with specific values assigned to its *attributes*. Some *classes* are not intended to be *instantiated*, but rather to provide a common set of *methods* for other *classes* which these can *inherit*. They are referred to as *abstract classes*. However, the definition of these *methods* in the corresponding *subclasses* can be redefined specifically for this *subclass*. The *methods* are therefore *overloaded* with a

new definition and the functionality of such *methods* is context depended. Another important principle of OOP is the so-called *abstraction*. It is used to emphasize what a *class* is or does rather than how it is defined internally. In this way the complexity of large class hierarchies can be managed and the important ideas or properties of a system stand out. *Classes* and therefore their *instances* can be connected to other *objects*. Depending on their characteristics, these connections are referred to as *association*, *aggregation* and *composition*. *Associations* represent a semantically weak relationship between otherwise unrelated objects. The *objects* have their own life time and there is no specific owner. *Aggregations* and *compositions* are specializations of *associations*. *Aggregations* are relations between two or more *objects* in which the *objects* have their own life time, like it is the case in *associations*, but the ownership is specified. *Compositions* in turn are a specialized form and therefore a subset of an aggregation ($composition \subset aggregation \subset association$), where the owned *objects* do not have an own life time, i.e. their existence depends on the *parent object*. As these definitions cover the scope of this work, other principles of OOP are not further discussed here. Next, the methodology Knowledge Based Engineering is introduced. KBE uses OOP principles and therefore provides similar functionalities.

Knowledge Based Engineering (AML)

KBE is a methodology for the capture and re-use of product and process engineering knowledge. The objective is to reduce the time and the costs of product development, which is primarily achieved through automation of repetitive design tasks while capturing, retaining and re-using design knowledge (La Rocca 2012).

The Adaptive Modeling Language (AML) is a framework for implementing KBE systems. It is based on Lisp, which is the first OOP language. Despite the common roots, the terminology differs in some points. As KBE and AML are introduced in more detail in section 2.5, the focus in this section lies on the terminological differences compared to OOP.

Similar to OOP, AML applications are based on the definition of *class* hierarchies. The main difference in the terminology is that *attributes* are called *properties*. Another peculiarity is the definition of a *subobject* list in the class. The list elements can either reference existing *objects* or create new *instances* of *classes* inside the list. These *object* – *subobject* relations are comparable to *aggregations* and *compositions*, which are described in the last section. While *functions* define procedures for the general use in any class, *methods* are specifically defined for a class and are called together with an instance of this class. An example for such a method is an operation that calculates the volume of a specific object.

EXPRESS language

The EXPRESS language is defined in the STEP standard (ISO 10303–11:2004) and is used to specify the information requirements of the other parts of the standard. The definition of the language is covered in detail in section 2.4.2, whereas its terminology is compared to OOP and AML in this section.

EXPRESS can represent *class* hierarchies with the help of *entities*. *Inheritance* is implemented with *supertype* – *subtype* relations in contrast to *superclass* – *subclass* relations in OOP. Not all of the principles of an OOP are implemented in EXPRESS, but the rest of the used terminology is quite similar to the one in OOP and AML.

Terminology

The terminology used in the scope of this work depends on the context. The commonly used notations in each field are used in the same way in this thesis. An overview of the differences that were pointed out in the previous sections is shown in table 2.1.

OOP	AML	EXPRESS
Class	Class	Entity data type
Superclass	Superclass	Supertype
Subclass	Subclass	Subtype
Attribute	Property	Attribute
Method	Method	–
Function	Function	Function

Table 2.1.: Correlation of the terminology conventions in OOP, AML and EXPRESS

The following sections cover the transfer of data between different systems in general and the transfer of product data that contains information about the intention of the designer in particular. Subsequently, the STEP standard is reviewed and evaluated in regard to its abilities to transfer such data. In order to do that, the structure of the standard is explained and the parts that are relevant for the scope of this thesis are summarized. Finally, KBE and AML are introduced more thoroughly to provide the basis for the following chapters.

2.2. Product data exchange

As mentioned in chapter 1, every phase of a product’s life cycle is supported or enabled by different software tools. Especially the engineering tasks rely heavily on digital development

environments. These environments define and describe the product in their own proprietary or standardized data formats. For a data transfer between multiple such environments, they either must provide interfaces to the relevant formats themselves or external tools must provide this functionality.

2.2.1. Translators

There are two main approaches to translate data between multiple systems or formats. The first approach is to define a translator for each pair of systems in both directions. As a result, the number of necessary translators grows quadratically ($n(n - 1)$) with the number of systems n , as shown in Figure 2.1.

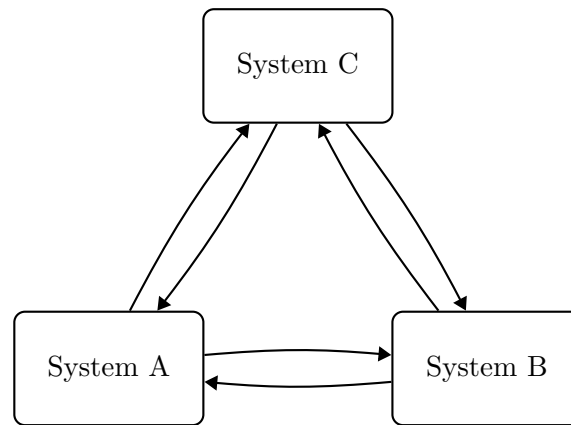


Figure 2.1.: Data exchange with direct translators.

When many systems are involved, it can be favorable to introduce a neutral exchange format, as pictured in Figure 2.2. For each system, one translator for writing the neutral file format and one translator for reading from the file format have to be provided. In this case, the number of necessary translators grows linearly ($2n$) with the number of systems n . Direct translators are very susceptible to changes in the software systems they support. If the functionalities of one system change, all direct translators to the other systems involved have to be adapted. In contrast, when using a neutral file format, only the interface to the neutral file format needs to be adapted when the software is changed. Furthermore, vendors of one company do not have to interact with competitors to develop the direct translators, which usually requires the disclosure of proprietary code. The disadvantage of a neutral file format is the initial effort necessary to develop the definition of the format and the democratic processes involved (Owen 1993). This often leads to retrospective formats – by the time the standard is published, new functionalities are already available in CAE systems for which there is no provision in the standard (Owen 1993).

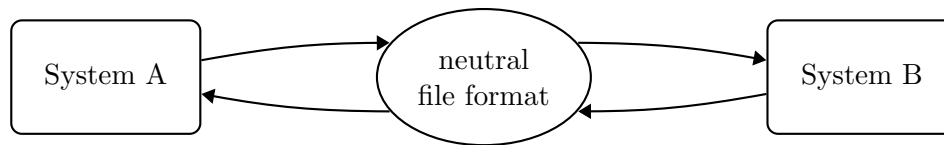


Figure 2.2.: Data exchange using an intermediate neutral format.

In order to translate from a system’s proprietary data structure to an exchange file, the units of information – in the case of CAE mostly units of construction (Kim et al. 2008) – have to be mapped to structures defined in the exchange format.

2.2.2. Mapping

The mapping of data structures can be ambiguous due to semantic differences in the systems. Depending on the granularity of the system’s representation, multiple units of information have to be mapped to a single data structure in the exchange format or vice versa. These cases are illustrated in figure 2.3 and can be referred to as aggregation and decomposition of data elements (Kim et al. 2008). The combination of these types is the complex mapping that is usually necessary in real systems. An example for such units of information are so-called features in CAD systems. They contain a sequence of standardized or frequently used construction commands and encapsulate them into reusable units, such as an extrusion or a pattern. At the same time, these units can represent elements on a lower or a higher level, such as a cartesian point or a whole product.

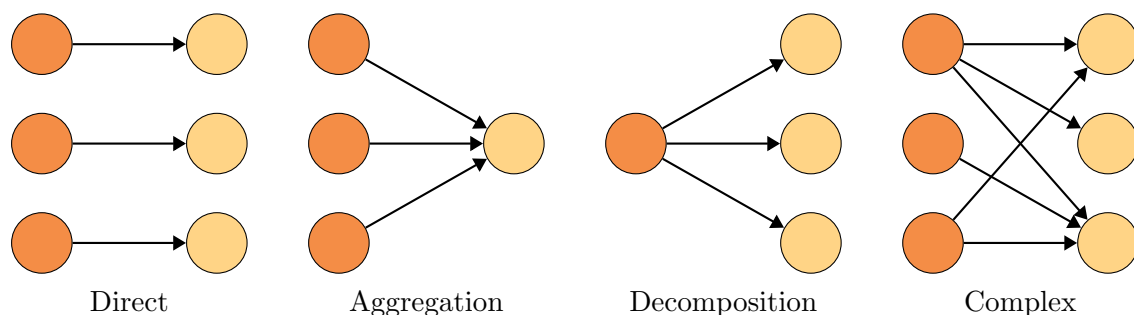


Figure 2.3.: Mapping between units of information. Adapted from Kim et al. (2008).

Different approaches to represent the data structure can pose another problem. CAD systems for instance encapsulate auxiliary parameters, such as explicit constraints in the data structures of the corresponding sketch or of another feature. In contrast to that, the STEP standard defines constraints part-oriented, i.e. the constraints can appear everywhere in

the exchange file and are only linked to the parameter they constrain with the help of an additional construct(Pratt et al. 2006).

Likewise, CAD system features frequently do not map directly to STEP features, because the latter represent compromises that give an approximate compatibility with a wide range of CAD systems (Kim et al. 2008). As a consequence, systems have to provide mechanisms to handle that ambiguity.

In the scope of this work, the units of information from which the intentions of the designer can be derived are of special interest. The next section points out such units and specifies the terminology of *design intent*.

2.3. Design intent

During the design process of a part or product, a designer makes a lot of design decisions depending on different factors, such as functionality, geometric restrictions, cost issues, experience of the designer, appearance or interfaces to other parts. A lot of these decisions are made based on tacit knowledge and are therefore not comprehensible or accessible for other designers. Capturing these decisions during the design process is important for an efficient development process and to be able to profit from previous work. Especially in the domain of KBE the capturing of knowledge is a key factor, as it has to be codified in the form of knowledge elements.

The total of information that enables the reconstruction of the intended design goal is collectively referred to as *design intent*. Pratt et al. (2006) distinguish between *design intent* and *design rationale*. In essence, they define *design intent* as the way the facilities provided by the CAD system are used and *design rationale* as the motivation of the designer to choose a particular methodology. They conclude that the STEP format aims to exchange *design intent* and not *design rationale*.

CAD Systems mainly generate and store data directly related to the geometry of the model. However, more and more functionalities are offered nowadays to store data not only relevant for the design process, but also for other phases of the product life cycle, such as manufacturing or marketing. Nevertheless, some of the data structures defined by the STEP standard, which would enable the transfer of a more complete data model, are not yet implemented in the CAD systems.

In the following sections, the most important elements of *design intent*, such as construction history, parameterized and constraint dimensions and features, and the methods to transfer them with the STEP format are examined.

Construction history

In CAD, there are two approaches to product modeling. One approach is the procedural approach, where the model description is a sequence of instructions which modify the model step by step. This way, the model embodies a history of the construction process and each state can be accessed afterwards. This approach is referred to as implicit approach in the following. The second approach is the explicit approach, where the construction history is not captured, but instead records of explicit geometrical details are stored at discrete moments in time during the design process. One example for this approach is the boundary representation, with which it is possible to restore the exact final geometry and any *snapshot* taken during the process, but this representation does not contain any information about the steps leading to the geometry changes (Pratt 1997). Such models cannot be efficiently edited and the model often has to be constructed again from scratch, when changes have to be performed. Therefore, the construction history is one of the most essential aspects of *design intent* (Pratt et al. 2006). Other important elements of *design intent* are parameters in the model, which represent values that are intended to be varied, and constraints, which define relationships that must be preserved in any change.

However, even if the construction history is transmitted to the receiving system, the interpretation of it may be difficult. One reason for this is that the design history usually lacks information about why the designer decided to use specific design methodologies, i.e. the *design rationale* is missing (Pratt et al. 2006). This is further discussed in section 2.4.5, which covers part 55 of the STEP standard with the title *Procedural and hybrid representation*.

Parameters

Parameters represent dimensions that may be changed in a part model to generate different versions of a part or to define mathematical relations between attributes. They can also refer to other parameters and thereby define relation structures. Parameters without independent existence in the part model are referred to as implicit parameters, such as a parameter that defines the length of an edge, while parameters with their own independent definition in the model are called explicit parameters (Pratt et al. 2006). Explicit parameters can for example be used to specify auxiliary dimensional relations in 2D sketches or features.

Constraints

Constraints provide the possibility to limit the degrees of freedom of certain relations between elements of a model that are required to be maintained if the model is modified. These

conditions can for example specify the spacial relations of parts relative to each or the dimensional relations in a part or a sketch. Examples for common constraints are parallelism or tangency conditions. The remaining degrees of freedom allow a linear modification of the unconstrained dimensions. As soon as the sketch, the feature or the part is fully constraint, it can no longer be modified. Similarly to the classification of parameters, constraints have implicit and explicit forms. Implicit constraints are automatically generated by the system during standard operations, for example during the creation of a rectangular in a sketch, where two of the implicit constraints are the parallelism of the edges and the 90° angle of the corners. Explicit constraints by contrast, are explicitly defined elements that reference to other elements and that constrain them to satisfy specified relationships (Pratt et al. 2006).

Structures for the representation of parameters and constraints are specified in Part 108 of STEP. They are discussed in section 2.4.6.

Features

Design features in CAD systems are basically predefined sequences of CAD-operations for regularly used or standardized geometrical manipulations, such as chamfers, holes and patterns. Part 108 of STEP with the title *Elements for the procedural modeling of solid shapes* provides representations for such design features. The implementation of these structures is described in section 2.4.7.

With these elements of *design intent* in mind, the STEP standard is investigated in the following sections. First, the fundamentals of the STEP standard, such as the definition of the EXPRESS language and the ways to implement the standard, are summarized to allow a well-founded discussion about the specific parts of the standard. Subsequently, the parts that allow a representation of the elements mentioned in this section are summarized. Based on that, the specific implementation of these structures is explained with the help of examples. The aim of the following section therefore is to provide a reference for the implementation of the elements of *design intent* described in section 3.2.2. Additionally, it is the basis for the discussion in chapter 5 about the abilities of the STEP standard to transfer such element. It should be mentioned that the parts of the standard that are discussed in next section are only a small subset of the standard's parts and are selected according to the requirements in the scope of this work.

2.4. ISO 10303 STEP standard

ISO 10303 STEP is an international standard for the computer–interpretable representation and the exchange of product data. It is informally known as STEP (STandard for the Exchange of Product model data). The aim of the project is to develop an engineering product data exchange standard that is capable of describing product data throughout the life cycle of a product, independently from any particular system. This makes the standard suitable not only for neutral file exchange, but also as a basis for product databases and archiving (ISO 10303–1:1994). Hence, the scope is much broader than that of other exchange formats, such as the Initial Graphics Exchange Specification (IGES), which has been one of the most widely used formats for the last 20 years, but was developed primarily for the exchange of pure geometric data. After the initial release of ISO 10303 in 1994, the interest in further developing IGES declined and the last version was published in 1996 (Pratt 2001).

In order to allow regular extensions, STEP is designed as multi–part standard. Many parts are complete and published as standards today, while more are under development. The main design goals of the STEP standard are listed below: (Owen 1993 and ISO 10303–1:1994)

- **Completeness:** The standard should allow a complete representation of a product for exchange and archiving purposes.
- **Extensibility:** STEP should provide a framework to include extensions.
- **Minimal redundancy:** The standard should only provide one way of representing a particular data structure.

According to Pratt (2001), the development of STEP is one of the largest efforts ever undertaken by the International Organization for Standardization (ISO). It is a multi–national project with contributions from industry, academia and governmental institutions. Even though until now only small parts of STEP are implemented and used, the standard is increasingly recognized by industry as an effective means of exchanging product–related data between different CAD systems or between CAD and downstream application systems, such as a Finite Element Method (FEM) application (Pratt 2001).

As described in section 2.2, the product data generated during the different stages of a product’s life cycle is stored in many different systems and usually has to be exchanged inside a company or between different organizations. ISO 10303 aims to cover a wide variety of product types that require specific data structures, such as electronic, electro–mechanical, mechanical, sheet metal or fiber composite products. At the same time, all life cycle stages, such as design, analysis, planning or manufacturing, are intended to be covered (Pratt 2001).

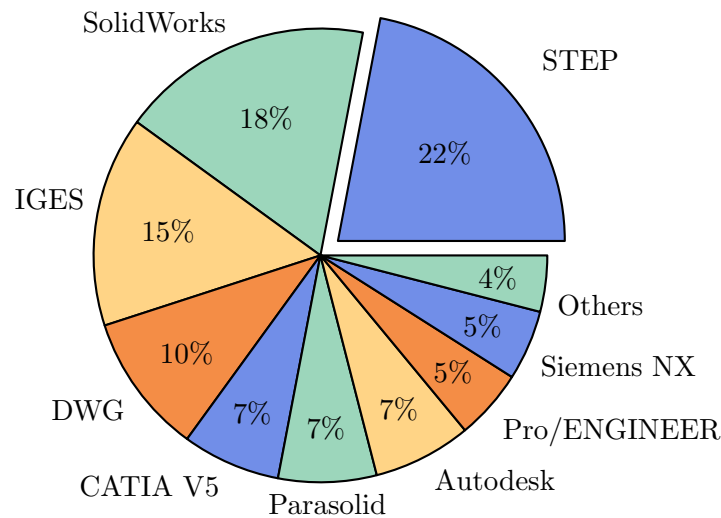


Figure 2.4.: Usage of 3D data exchange formats as primary format in 2010 (Prawel 2010).

This way, a single standard can cover all domains of product data, instead of many national and international standards, which evolved over the years and overlap in their functionalities. In 2010, Prawel (2010) surveyed CAD users from different industries and concludes that STEP is the leading 3D exchange format with 22% of the respondents choosing it as their primary 3D data exchange format. Having conducted similar surveys the preceding years, the author states that this continues a multi-year trend of increasing popularity of STEP and ascribes this to the long-term data archival strategies driven by different groups, such as LOTAR. Another finding of the survey is that more companies are sharing manufacturing data and feature and history data than ever before.

In the following sections, the structure of STEP is described and the parts that are relevant for this work are summarized. These summaries are far from complete and only serve as a general overview of the extensively documented sub-standards of ISO 10303.

2.4.1. Structure of STEP

ISO 10303 is divided into six series of parts with specific functionalities. Each series may consist of one or more parts. The numbering scheme of the series is defined in ISO 10303-1:1994 as follows:

- Description methods #11 – #19
- Implementation methods #21 – #29
- Conformance testing methodology and framework #31 – #39
- Integrated resources
 - Generic resources #41 – #99
 - Application resources #101 – #199
- Application protocols #201 – #1199
- Abstract test suites #1201 – #2199
 - corresponding to the associated application protocols #201 – #1199

These series can be grouped into two main categories, as shown in figure 2.5. The first three series (#11 – #39) form the infrastructure for the file format. They define the storage, exchange and testing of data structures. The remaining series define the actual data structures of the different parts of the standard.

The step architecture is depicted in figure 2.6. The Application Protocols (AP) specify models that satisfy the scope and the information requirements for industry-specific applications. Each AP defines an Application Activity Model (AAM) which describes the activities in the life cycle of a product. The product information requirements for these activities are defined in the Application Reference Model (ARM). The Mapping Specifications (MS) map the ARM into the common set of Integrated Resources (IR), which consist of Application Resources (AR) and Generic Resources (GR). The Generic Resources are independent of applications and can reference each other. The Application Resources can reference the Generic Resources and can add additional resource constructs specifically for a group of similar applications, but cannot reference other Application Resources. (ISO 10303-1:1994) The result of the mapping is an Application Interpreted Model (AIM), consisting of a formal EXPRESS information model that captures everything in the Application Reference Model and ties it to a library of pre-defined resources (Loffredo 1999). Therefore, the APs define specific subsets of the Integrated Resources that vendors can implement into their systems. This prevents compatibility problems that would occur, if every vendor defined its own subset.

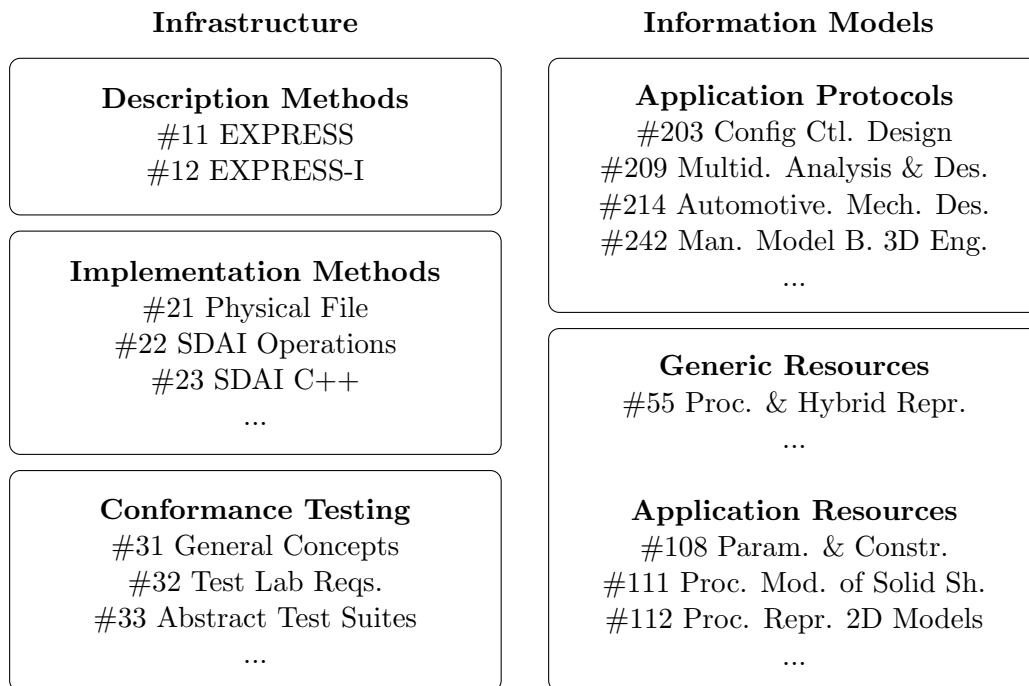


Figure 2.5.: High level structure of the STEP standard. Based on STEP Tools Software (2010) and Loffredo (1999).

In the initial ISO architecture, the Application Interpreted Constructs (AIC) play a major role in terms of the reuse of defined resources. The AICs define collections of common definitions that can be shared between different APs. This reduces the number of pages in the AP documents and assures consistency among APs that refer to the same AICs. Examples for AICs are ISO 10303–501:2000 *Edge-based wireframe* or ISO 10303–514:1999 *Advanced boundary representation*.

Soon, the architecture was redefined to simplify the extension of the standard. This new modular STEP architecture is also pictured in figure 2.6. The modular approach extends the AIC concept of the initial ISO 10303 architecture by including the relevant portions of the AP's Application Reference Model (ARM) into the Application Modules (AM) (SCRA 2006). This well-documented grouping of requirements into reusable modules significantly simplifies the development of APs.

Many of the APs have reached international standard status and are ready to be implemented into software systems. The CAD/CAM vendors however only implement some subsets of the APs due to the high costs for implementation. For every AP, associated Conformance Classes (CC) define subsets that can be implemented within the same application domain without the need to implement all aspects of the AP. The most widely implemented APs in CAD software are AP203 and AP214, which overlap in some of their CCs. (SCRA 2006)

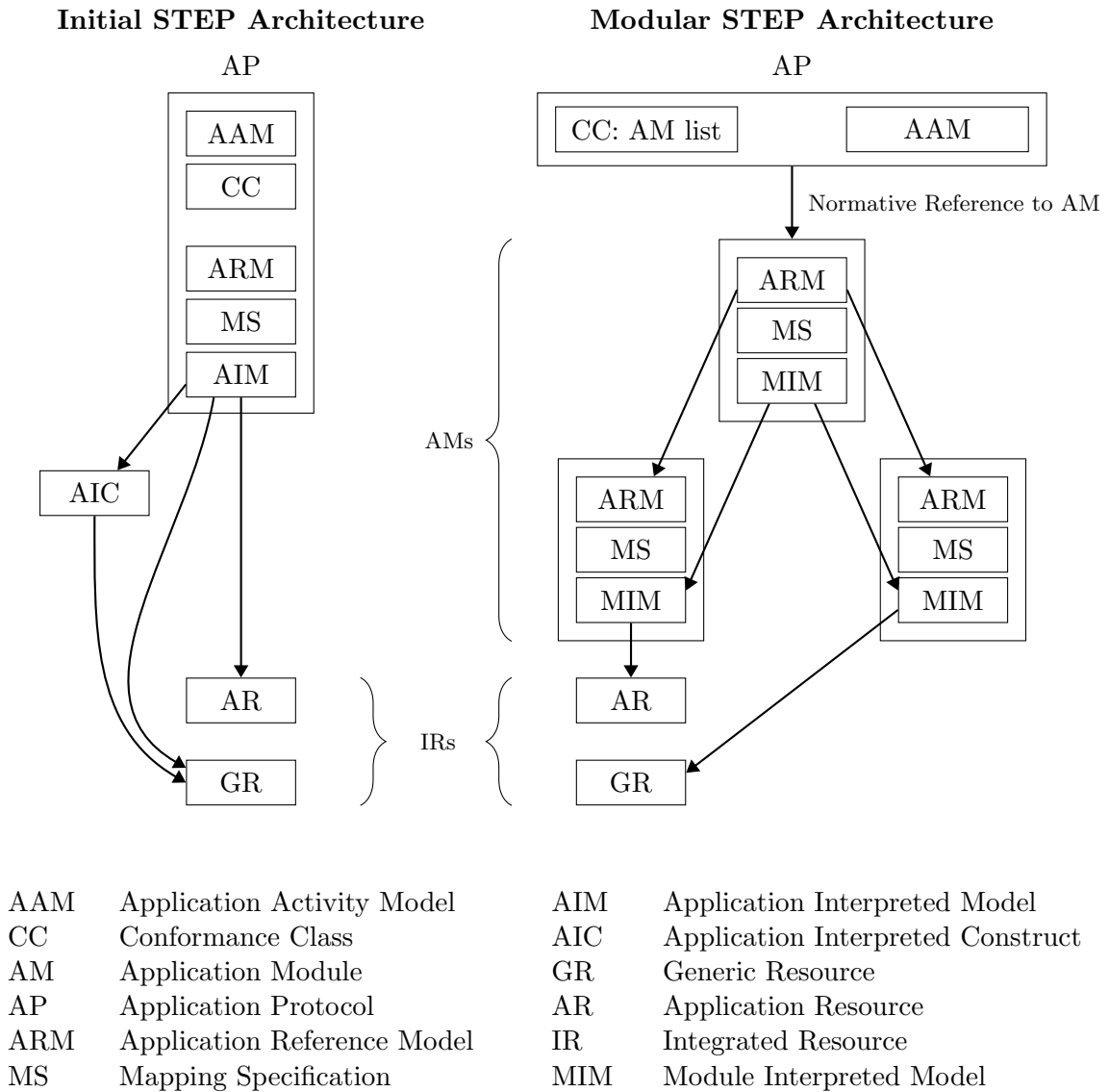


Figure 2.6.: Initial and modular STEP architecture. Based on STEP Tools Software (2010) and ISO TC 184/SC4 N1863 (2005).

Table 2.2 shows an overview of the STEP parts relevant for the transfer of *design intent*, which are further specified in the following sections.

	Part	Year	Title
IS	Part 1	1994	Overview and fundamental principles
	Part 11	1994	The EXPRESS language reference manual
	Part 21	1994	Clear text encoding of the exchange structure
	Part 22	1998	Standard data access interface
IR	Part 55	2005	Procedural and hybrid representation
	Part 108	2005	Parameterization and constraints for explicit geometric product models
	Part 111	2007	Elements for the procedural modeling of solid shapes
AP	AP 203	1994	Configuration controlled 3D designs
	–Edition 2	2011	
	AP 214	2001	Core data for automotive mechanical design processes
	–Edition 3	2010	
	AP 242	2014	Managed model based 3D engineering
	–Edition 2	planned	
	AP 209	2001	Composite and metallic structural analysis
	–Edition 2	2014	Multidisciplinary analysis and design (renamed)

Table 2.2.: Infrastructure Parts (IS), Integrated Resources (IR) & Application Protocols (AP) discussed in this work.

2.4.2. Part 11 – The EXPRESS language reference manual

In part 11 of the STEP standard, the EXPRESS language is defined. EXPRESS is used to specify the information requirements of other parts of the standard. The main requirements for the language is that it is readable by humans and parsable by computers. The language consists of elements that allow an unambiguous data definition and a specification of constraints on the defined data (ISO 10303–11:2004). These elements are called entities and are defined with the help of attributes, which can be a data type or a reference to another entity.

The language syntax is defined in ISO 10303–11:2004 with notation rules. The notation is called Wirth Syntax Notation (WSN) and can be defined by using its own notations rules, as shown in listing 2.1.

As shown below for some of the rules defined in ISO 10303–11:2004, WSN notation rules can be visualized with so-called railroad diagrams. The rule for explicit attributes `explicit_attr` for example contains one repetition and one option, which are visualized with different kinds of loops in the railroad diagram. This visual representation significantly simplifies the comprehension of the syntax, especially for complex and deeply nested rules.¹ The complete set of WSN rules defined in ISO 10303–11:2004 is visualized with these railroad diagrams in appendix D. With these rules, the EXPRESS schemas can be parsed and interpreted in

¹Additionally, the elements of a rule are linked to their own defining rule in the PDF, in order to be able to jump from one rule to another. This is a great help during the development of a parser for this grammar.

```

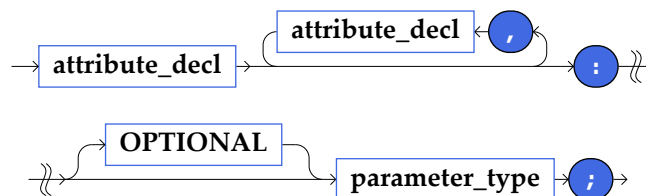
syntax          = { production } .
production     = identifier '=' expression '.' .
expression     = term { '|' term } .
term           = factor { factor } .
factor        = identifier | literal | group | option | repetition .
identifier     = character { character } .
literal       = ''' character { character } ''' .
group         = '(' expression ')' .
option        = '[' expression ']' .
repetition    = '{' expression '}' .

```

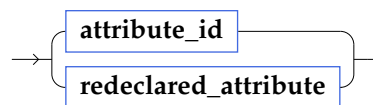
Listing 2.1: Notational conventions and WSN defined in itself (ISO 10303–11:2004).

different programming languages. The programming language Python offers some well documented parser libraries, which is why the parsers in the scope of this work are implemented in Python, as described in section 3.3.1.

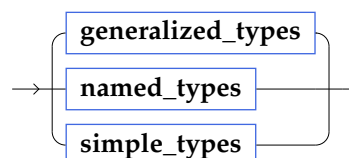
215 `explicit_attr = attribute_decl { ',' attribute_decl } ':' [OPTIONAL] parameter_type ';' .`



177 `attribute_decl = attribute_id | redeclared_attribute .`



266 `parameter_type = generalized_types | named_types | simple_types .`



EXPRESS uses some keywords, such as ENTITY, END_ENTITY or SUPERTYPE, which are referred to as literals. They are written in capital letters to simplify their interpretation for parser programs. The full set of reserved keywords and operator words, such as OR and ANDOR are defined in the grammar rules 1 to 122 listed in appendix D.

In general, an EXPRESS schema consists of a header and the definitions of CONSTANTS, TYPES, ENTITIES, RULES and FUNCTIONS. The entities define elements with attributes and possible inheritance relations to other entities. The attributes in turn can reference another entity or a specific data type. Some of these attributes are internally derived from other attributes. The expression used to compute these derived attributes is either defined directly in the entity declaration or in a function declarations. EXPRESS has several predefined data types, which are listed below:

- NUMBER
 - REAL
 - INTEGER
 - BINARY
- LOGICAL
- BOOLEAN
- STRING
- AGGREGATION
 - ARRAY
 - LIST
 - BAG
 - SET
- SELECT
- ENUMERATION

Those datatypes can be used to define custom types in an EXPRESS schema. A new data type could for example be a LIST of SELECT-groups of specific entities. Additionally, the entities can use the functions, which are defined in the schema.

As part of ISO 10303–11, EXPRESS–G is defined as the graphical representation of the EXPRESS lexical model. It can visualize EXPRESS rules and is intended for human communication.

The next two sections cover two of the implementation methods of STEP. First, the physical STEP file and second, the SDAI. The information models described in the subsequent sections are defined using the EXPRESS language. They are stored in the form of so-called EXPRESS schemas, which can be used to populate data models according to the schema's definitions.

2.4.3. Part 21 – Clear text encoding of the exchange structure

As one of the possible implementation methods, part 21 of ISO 10303 defines the physical exchange file in which the data models can be stored. The files are often called p21-File or physical file and have the file ending `.stp` or `.step`. They are ASCII encoded with typically one instance per line, which makes them easily readable. Since the objects in the p21-File are defined according to a specific EXPRESS schema, they can only be interpreted together with the according AP schema (SCRA 2006).

STEP files have a header section and one or more data sections. The header section contains general information about the file, the author and the EXPRESS schemas used in the data sections. The data sections contain the instances of the entities that are to be transferred.

Mapping of EXPRESS entity data types

For each instance of an entity in the STEP file, the attributes and other information defined in the STEP schema must be mapped to the STEP file. Table 2.3 shows a quick reference of the mapping rules from an EXPRESS schema to a physical file. Some of the EXPRESS elements are not directly mapped to the STEP file, but have to be implemented in the target system instead, such as the derived attributes. The rules on how they are derived are defined in the entity definition and do not have to be transferred with the STEP file. Table 2.4 shows how the different data types are mapped to the STEP file. A string for example is indicated by a starting and an ending apostrophe.

When an EXPRESS entity data type is instantiated in the exchange structure, it is mapped as a so-called ENTITY_INSTANCE. As defined in ISO 10303-11:2004, these instances are subdivided into SIMPLE_ENTITY_INSTANCES on the one hand and COMPLEX_ENTITY_INSTANCES on the other hand. While the entities that are not defined as subtypes of other entities are always mapped as SIMPLE_ENTITY_INSTANCE in the exchange structure, the entities that are defined as subtypes of other entities are only mapped as SIMPLE_ENTITY_INSTANCE if they are leaf entities. This means that they have no subtype entities or their subtype entities are not in their evaluated set (see ISO 10303-11:2004). Conversely, when these conditions are not met, the entities are externally mapped as COMPLEX_ENTITY_INSTANCE. The internal mapping of the SIMPLE_ENTITY_INSTANCES follows a set of rules to determine the order in which the inherited and explicit attributes appear in the exchange structure. All inherited attributes appear sequentially prior to the own explicit attributes of an entity. The inherited attributes of one supertype entity are ordered according to their appearance in the supertype entity itself. If this supertype entity in turn inherits attributes

EXPRESS element	Mapped onto:
ARRAY	list
BAG	list
BOOLEAN	boolean
BINARY	binary
CONSTANT	NO INSTANTIATION
DERIVED ATTRIBUTE	NO INSTANTIATION
ENTITY	entity instance
ENTITY AS ATTRIBUTE	entity instance name
ENUMERATION	enumeration
FUNCTION	NO INSTANTIATION
INTEGER	integer
INVERSE	NO INSTANTIATION
LIST	list
LOGICAL	enumeration
NUMBER	real
PROCEDURE	NO INSTANTIATION
REAL	real
REMARKS	NO INSTANTIATION
RULE	NO INSTANTIATION
SCHEMA	NO INSTANTIATION
SELECT	See ISO 10303–21:2002, 11.1.8
SET	list
STRING	string
TYPE	See ISO 10303–21:2002, 11.1.6
UNIQUE RULES	NO INSTANTIATION
WHERE RULES	NO INSTANTIATION

Table 2.3.: Quick reference mapping table (ISO 10303–21:2002).

from its supertype entities, they appear prior to the ones from the original supertype entity. If multiple supertype entities are defined, the order of their attributes is defined by the order in which they are called in the SUBTYPE OF expression of the initial entity. If superotypes are referenced multiple times, all but the first one are ignored. (ISO 10303–21:2002)

The instance description of a simple entity starts with the instance id, followed by an equal sign and the entity name. The subsequent list is an enumeration of the attribute values in the correct order. In order to illustrate how these attribute values can be mapped onto the appropriate attributes defined in the entity definition, the mapping from a STEP schema to a STEP file is shown in the following example.

The example EXPRESS schema in listing 2.2 is taken from ISO 10303–21:2002 and shows subtype/supertype relations between some example entities. In listing 2.3, these entities are populated in the data section of a p21–File. Entity aa and bb are abstract superotypes and

Attribute value		Example
#□	Entity reference	#1, #01, #123
.□.	Enumeration value	.T., .REAL.
(□)	Structured data type (list)	(1.0, 2.0), ('str1', 'str2')
\$	Empty optional attribute	(#1, \$)
'□'	String	'str1'
*	Redeclared, derived attribute, value is processed internally	(#1, *)

Table 2.4.: Syntax of different data types in instance descriptions.

therefore do not map to the exchange structure ((A) & (C)). The entity `xx` inherits from `bb`, which in turn inherits from `aa`. Thus, the instance of `xx` in the data section has four attributes. The order of the attributes is determined as described before: first the inherited attributes and then the attributes defined in the entity itself. This means that the attribute `attrib_a` maps to the data section in the first slot of the instance of `xx` as an inherited attribute. It refers to an instance of `zz` – in this case the instance #1 ((B)). The attributes of `bb` `attrib_b1` and `attrib_b2` also map to instance #4 in the second and third slot. They refer to the two instances of `yy` #2 and #3 ((D) & (E)). Attribute `attrib_x` refers to its value 4.0 ((F)).


```

1 ENTITY aa ABSTRACT SUPERTYPE OF (ONEOF(bb,cc)); -----> (A)
2   attrib_a : zz; -----> (B)
3 END_ENTITY;
4
5 ENTITY bb SUBTYPE OF (aa)
6   ABSTRACT SUPERTYPE OF (ONEOF(xx)); -----> (C)
7   attrib_b1 : yy; -----> (D)
8   attrib_b2 : yy; -----> (E)
9 END_ENTITY;
10
11 ENTITY cc SUBTYPE OF (aa);
12   attrib_c : REAL;
13 END_ENTITY;
14
15 ENTITY xx SUBTYPE OF (bb);
16   attrib_x: REAL; -----> (F)
17 END_ENTITY;
18
19 ENTITY zz;
20   attrib_z : STRING;
21 END_ENTITY;
22
23 ENTITY yy;
24   attrib_1 : REAL;
25   attrib_2 : REAL;
26   attrib_3 : REAL;
27 END_ENTITY

```

Listing 2.2: Example of a simple subtype/supertype relationship. Entity definition in EXPRESS. (ISO 10303-21:2002)

```

1 #1 = ZZ('ZATTR');
2 #2 = YY(1.0, 2.0, 0.0);
3 #3 = YY(2.0, 2.0, 0.0);
4 #4 = XX(#1, #2, #3, 4.0);
5
6     ↑   ↑   ↑   ↑
7     (B) (D) (E) (F)

```

Listing 2.3: Sample entity instance of the entity data type xx in the data section. (ISO 10303-21:2002)

2.4.4. Part 22 – Standard data access interface

As an alternative to the physical STEP file, the Standard Data Access Interface (SDAI) is an implementation method for data structures defined in EXPRESS. It specifies the operations available to an application for the purpose of acquiring and manipulating the data structures. The definition in ISO 10303-22:1998 is independent of any computing language or system. The SDAI specifies the requirements of a programming interface for the generation and manipulation of instances of EXPRESS entities. Together with EXPRESS, the SDAI specifies a data access interface that is independent of the underlying storage technology (ISO 10303-22:1998).

The definitions for particular languages are called bindings and are defined in Part 23, 24 and 27 of the standard for C, C++ and Java. SDAI bindings are either early-bound or late-bound. An early binding defines access functions for a specific EXPRESS schema. These access functions are generated once by an EXPRESS compiler. For each of the declarations in the EXPRESS schema a corresponding class is generated in the target language. One major advantage of early bindings is that the compiler can do extensive type checking on the application and can detect conflicts at compile time (Loffredo 1999). Late bindings on the other hand have a fixed set of functions that do not change with the schema. A late binding generally uses compiled representation of the EXPRESS schema, called the data dictionary. The data dictionary is accessed and manipulated with queries. The advantage of late bindings compared to early bindings is higher simplicity due to less initial work. One disadvantages compared to late bindings is the lack of compile-time type checking (Loffredo 1999).

As discussed before, the following sections contain Integrated Resources, which define sets of constructs to represent different kinds of data. In this way, the Integrated Resources provide a common set of resource constructs which can be used by the APs, as described in the subsequent sections.

2.4.5. Part 55 – Procedural and hybrid representation

ISO 10303-55:2005 with the title *Procedural and hybrid representation* provides mechanisms for the representation of the history of operations used to generate the model. The constructional operations themselves are represented by entity data types defined in other parts of ISO 10303, for example the entity `extruded_face_solid` in ISO 10303-42. Procedural models store information on how the model will behave when edited in the target system (ISO 10303-55:2005). As discussed in section 2.3, the history of constructional operations

embodies an important part of *design intent* information, especially in combination with parameters.

The common STEP files nowadays mostly transfer the geometry model as an explicit representation of the shape without any procedural information. For the implementation of a procedural representation, Pratt et al. (2006) describe a hybrid shape representation. The procedural representation is the primary representation containing all the operations needed to reconstruct the model, while the secondary representation holds the explicit geometrical information for the final stage of the model. The explicit data structure is usually built from low level elements which must be transmitted together with the model in order to specify it completely. The secondary model can be used in the receiving system to test the validity of the reconstruction, or to resolve ambiguities in the case of several valid solutions (Pratt et al. 2006). The operations defined in the primary procedural model all map to instances of `geometric_representation_item` or `topological_representation_item`. The secondary explicit model will be one of the explicit forms of `shape_representation`. Explicit elements are distinguished from procedural elements simply by the fact that they do not participate in instances of a `procedural_shape_representation_sequence` (ISO 10303-55:2005).

```

73 #63= EXTRUDED_FACE_SOLID('Extruded Cube', #56, #58, 40.);
82 #76= SOLID_WITH_FLAT_BOTTOM_ROUND_HOLE('Through Hole
    ↪ ', ' ', #63, #75, *, 1, (10.), (40.), 0.);
97 #97= SOLID_WITH_SINGLE_OFFSET_CHAMFER('Chamfer', $, #76, (#92), 5.);
98 #99= PROCEDURAL_SHAPE_REPRESENTATION_SEQUENCE($, (#63, #76, #97), $, $);
99 #101= PROCEDURAL_SHAPE_REPRESENTATION($, (#99), $);
100 #103= SHAPE_DEFINITION_REPRESENTATION($, #101);

```

Listing 2.4: Definition of a `procedural_shape_representation_sequence`. (File: `example_ap242.stp`)

In listing 2.4, the `procedural_shape_representation_sequence` includes three procedural models, which represent steps in the construction history of the model. The first step is an extrusion, followed by a hole feature and a chamfer. This enables the receiving system to recreate these steps, in the order they appear in the `procedural_shape_representation_sequence`, to construct the geometric model.

2.4.6. Part 108 – Parameterization and constraints for explicit geometric product models

Part 108 of the standard specifies constructs to represent model parameters and constraints. This is done by defining additional entities that assign these parameter or constraint values to other transferred geometric elements. This enables the receiving system to reconstruct some of the behavior of the model that was defined in the source system. One example for this added *design intent* is the variant management in a model family whose members vary in one or more parameterized dimensions. Ideally, the complete implementation in both systems enables the target system to edit the model just like it would be possible in the source system (ISO 10303–108:2005).

Variables can be represented by instances of the entity `variational_parameter` which consists of the `bound_variational_parameter` and the `unbound_variational_parameter` entity.² A bound parameter is associated with an attribute of another entity instance, whose value represents the current value of the parameter. This association is defined with the help of the entity `instance_attribute_reference`, which has the name of the parameter and the instance, to which the parameter belongs, as attributes. By contrast, an unbound model parameter is not directly associated with any other attribute, but could for example be useful for the definition of mathematical relations (ISO 10303–108:2005).

```

1 #290 = AXIS2_PLACEMENT_3D(...);
2 #300 = BLOCK('BLOCK1', #290, 4.0, 6.0, 8.0);
3 #310 = INSTANCE_ATTRIBUTE_REFERENCE ('GEOMETRIC_MODEL_SCHEMA.BLOCK.X',
   ↪ #300);
4 #320 = FINITE_REAL_INTERVAL(2.0, .CLOSED., 10.0, .CLOSED.);
5 #330 = BOUND_MODEL_PARAMETER ('XPARAM', #320, *, 'BLOCK X-DIMENSION',
   ↪ *);
6 #340 = BOUND_PARAMETER_ENVIRONMENT(#310, #330);

```

Listing 2.5: Parameter binding to an instance attribute. (example from ISO 10303–108:2005)

In Listing 2.5, the definition of a `bound_variational_parameter` is shown exemplarily. The entity #340 provides the link between the specified instance attribute #310 and the parameter #330, which is bound to it. #320 defines the domain of the parameter. The entity `finite_real_interval` is defined in ISO 10303–50.

²`variational` was formerly named `model`, but was redefined in ISO 10303–108:2005 due to name conflicts.

2.4.7. Part 111 – Elements for the procedural modeling of solid shapes

Part 111 of the standard specifies elements for the procedural modeling of solid shapes. These elements provide the capability to exchange feature-based CAD models on the basis of the `procedural_shape_representation` discussed before. The procedural model can also include entities defined in ISO 10303-42, especially for boolean operations, but the entities defined in this part of the standard are specifically defined in a way to facilitate the exchange of models with a representation of their constructional history (ISO 10303-111:2007).

ISO 10303-111 defines a single schema, the `solid_shape_element_schema`, which defines a set of geometric elements, e.g. extrusions, chamfers or blendings.

The following four sections cover the different APs that are relevant in the scope of this work. The first two, AP203 and AP214 are the basis for AP242 which in turn is extended by AP209. As described in section 2.4.1, the APs are specific subsets of the Integrated Resources.

2.4.8. AP 203 – Configuration controlled 3D designs of mechanical parts and assemblies

This part of ISO 10303 specifies the configuration controlled 3D design of mechanical parts and assemblies and was mainly driven by the aerospace and defense industry. It provides structures to exchange wireframe, surface and boundary representation (B-rep) solid models, together with the administrative data for the whole design life cycle. Until 2011 it did not provide any definitions for all the other life cycle phases, but in the second version of the Application Protocol, the data structures for the exchange of construction history were implemented. It is the most widely used part of ISO 10303 (Kim et al. 2008).

2.4.9. AP 214 – Core data for automotive mechanical design processes

The aim of AP214 is to provide the structure to exchange information between applications which support the development process of a vehicle. It was mainly driven by the automotive industry. Most of the AP214 translators have only implemented Conformance Classes `cc1` and `cc2` (out of 20 Conformance Classes) that are very similar to the AP203 geometry and topology definitions (SCRA 2006).

As these two standards overlap in many of their contents, it was decided in 2009, to merge them into AP 242 (Feeney et al. 2015).

2.4.10. AP 242 – Managed model based 3D engineering

AP242 is a major new Application Protocol, as it combines and replaces the following APs while being upward compatible:

- AP 201: Explicit draughting. (2D drawing geometry related to a product)
- AP 202: Associative draughting. (2D/3D drawing with association, but no product structure)
- AP 203: Configuration controlled 3D designs of mechanical parts and assemblies.
- AP 204: Mechanical design using boundary representation.
- AP 214: Core data for automotive mechanical design processes.

STEP AP 242 Ed1 therefore provides all the functionalities covered by the most commonly implemented and used APs 203 Ed2 and 214 Ed3 (Feeney et al. 2015). It additionally defines new structures for:

- 3D parametric & geometric constraints design
- Geometric dimensioning and tolerancing
- Business Object Model
- Tessellation
- Kinematics

Regarding the ability to transfer design intent, AP242 therefore includes the structures for the representation of parameters and constraints in addition to the structures for construction history that were introduced in AP203 (ISO 10303 Whitepaper Ed1 2009). The business object model for AP242 is specified in ISO 10303-3001 and consists of Business Objects (BO) representing major concepts and information requirements of Managed model-based 3D engineering (ISO 10303-242:2014). The BO model is defined in an EXPRESS schema and alternatively in a XML (Extensible Markup Language) schema. According to Vettermann (2015), the XML version of the BO model is the designated backbone for data exchange in the manufacturing industry, where the capability to exchange PDM data in the XML format is a key factor. The Jupiter Tessellation (JT) format would cover the graphical visualization in this scenario, as these two formats complement each other (Vettermann 2015). The XML BO model also enables the advanced external referencing structures for kinematics in assemblies, composite parts and tessellated data (Fischer 2015). AP242 has only one Conformance Class

with the title *managed_model_based_3d_engineering_cc1*, thus it only can be implemented with all the functionalities at once.

The development of AP242 is hosted by *PDES* and *ProSTEP* and tested on the CAx Implementer forum, see section 1.4 for more information about these organizations. AP242 Ed2 will incorporate more tolerancing standards, support for electrical wire harness, additive processes and some more extensions (Feeney et al. 2014).

2.4.11. AP 209 – Multidisciplinary analysis and design

The modular edition of AP209 is an explicit extension of AP242. AP209 Ed2 thus provides the same functionalities as AP242 and additional structures to express engineering analysis and simulation data.

2.5. Knowledge Based Engineering

La Rocca (2012) describes two different types of KB systems, the Rule Based Systems (RBS) and the Frame Based Systems (FBS). In RBSs the knowledge is expressed as set of IF–THEN rules, while FBSs are based on classes and offer Object Oriented (OO) features, such as abstraction and inheritance. La Rocca (2012) distinguishes between class–frames and instance–frames in FBS. The class–frames contain the class definitions with so–called property slots, which can point to other frames. Classes can be related to superclasses, from which they inherit the property slots and other elements. Instance–frames in turn are unique specifications of class–frames, defined through the assignment of a specific value set to the slots of the class–frame.

KBE systems are specialized KB systems that are able to handle the specific needs of the engineering design domain. These mainly are the manipulation of geometry and the data processing for analysis applications (La Rocca 2012).

As mentioned in chapter 1, one of these KBE frameworks is AML. In the next section the basic structure of a KBE project in AML is described.

2.5.1. Adaptive Modeling Language

As AML is based on the concept of OOP, applications are defined with the help of class structures. These classes are organized in one or more source files, which can be compiled by

the system. Based on this compiled system, the user can create models with instances of the defined classes.

On the basis of the example class definition shown in listing 2.6, a short overview of AML is given in this section. This is just a very brief introduction, for more information, refer to TechnoSoft (2010). As AML is based on Lisp, all definitions are in the form of nested lists starting with an opening parenthesis (and ending with a closing parenthesis). The first element of a list is the operator, which calls for example a method, such as `define-class` or a mathematical operation, such as `+`. The following list elements are the operands, which are fed as arguments to the method or operation. `(+ 1 2)` for example generates the output 3.

```

1 (in-package :aml)
2
3 (define-class example-class-1
4   :inherit-from (
5     object
6   )
7   :properties (
8     ;; user defined slots
9     property-1 1.0
10    ;; computed slots
11    property-2 (* 0.5 ^property-1)
12  )
13  :subobjects (
14    (subobject-1 :class 'example-class-2
15                property-1 (list 42.0 1.0 0.0)
16    )
17  )
18 )

```

Listing 2.6: Class definition in AML.

The `(in-package :aml)` at the beginning tells the compiler to use the `:aml` package, which contains predefined classes, functions and methods. The `:inherit-from` statement in the class definition defines the superclasses from which the class inherits the property and method definitions. The `object` class is the highest level pre-defined class that should be instantiated by the user. In the `:properties` statement, the properties of the class are defined with a property name followed by the formula defining the value of this property. These property slots can be divided into user defined and computed slot, depending on if they are defined by the user or processed internally, depending on the values of other properties. The `:subobjects` statement defines the subobjects of which the object is composed of. A table for example could be defined with four legs and a table top as subobjects. Words

with a colon `:` at the beginning symbolize keywords. Keywords are followed by arguments that can be passed to the functions or methods. They have a default value, so they do not necessarily have to be specified in the function or method call. Functions and methods define constructs that take arguments as input and process them internally to generate an output. Methods are defined for specific classes while functions can be called from all classes in the same namespace (in this case `:aml`). An example function definition is shown in listing 2.7. The function call `(quadratic-formula 1 3 2)` would return `(-1.0 -2.0)`. The `let` environment offers the possibility to declare local variables.

```

1 (defun quadratic-formula (a b c)
2   (let (
3     (radical (- (expt b 2) (* 4 a c)))
4     (denominator (* 2 a))
5     (numerator-plus (+ (- b) (sqrt radical)))
6     (numerator-minus (- (- b) (sqrt radical)))
7   )
8   (list
9     (/ numerator-plus denominator)
10    (/ numerator-minus denominator))
11 )
12 )

```

Listing 2.7: Function definition in AML.

The Integrated Development Environment (IDE) for AML is a customized version of XEmacs,³ which is a fork of the Emacs editor. These editors are highly extensible far beyond the scope of other text editors. These extensions are programmed in the Lisp dialect Emacs Lisp and provide the user with the possibility to customize the editor with own commands or to automate repetitive work. The AML XEmacs editor for example provides the functions to start the compiling process or to automatically adjust the indents of large amounts of code.

The geometrical output of the models is presented in the AML main modeling form, as can be seen in chapter 4. The content of the form itself is also programmed in AML. It is therefore possible to define a specific user interface that displays the necessary data and provides an interface to interact with the data model.

The theoretical background of this chapter is the basis for the next chapter that covers the methodologies used to implement the objectives that are defined in chapter 1. Especially the review of the STEP standard serves as a guideline for the implementation, as it provides the methods for the main implementation steps in a condensed manner.

³<http://www.xemacs.org>, last accessed: 2016-04-05

3. Methodology & Implementation

This chapter covers the concept development on how to implement the objectives defined in chapter 1. Additionally, it includes short descriptions of the software tools used for this implementation. Their basic setup is described to ensure replicability. Some tools were evaluated but not used for the implementation. They are mentioned nevertheless for overview purposes.

3.1. Concept development

As presented in chapter 1, the goal of this thesis is to evaluate if and how it is possible to transfer *design intent* between CAD and KBE systems with the help of the STEP standard. In order to be able to evaluate this question, the aim is to transfer an example model. For this purpose three steps have to be accomplished either with the help of existing software solutions or with proprietary developments.

1. It has to be determined which CAD software or translator tool already supports the new STEP structures in question. With the help of this software or tool an appropriate example geometry has to be defined and exported into a STEP exchange file. This geometry has to include at least some of the representation structures that contain *design intent*.
2. The structures defined in the AP242 schema have to be mapped to AML structures. This enables the AML environment to build a data model according to the relations defined in the STEP schema.
3. The data stored in the physical STEP file has to be populated into the AML application. Each entity instance taht is listed in the STEP file correlates to an instantiation of a class defined in step two.

The final implementation of these three steps is depicted in figure 3.1. The according sections however first cover the different software tools that potentially provide the required functionality and then finish with the actual implementation. The file structure together with a visualization of the dependencies is shown in figure A.1.

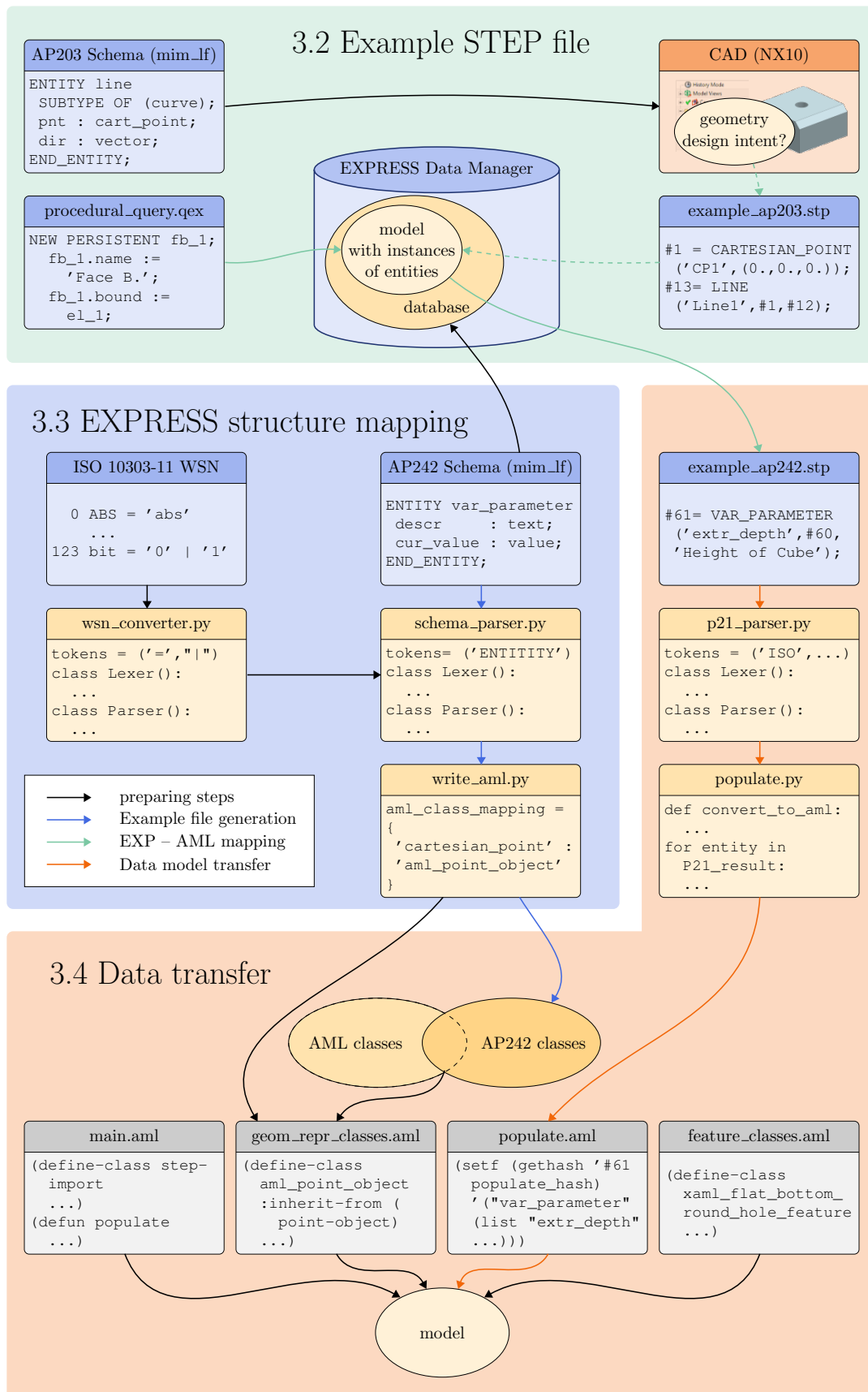


Figure 3.1.: Implementation overview.

3.2. Example STEP file

In the course of the research about the implementation status of AP242 specific features into commercial products, it became clear that the relevant features for the exchange of *design intent* are not yet included into released CAD software. Some CAD vendors claim to support some AP242 functionalities, but those mainly are the ones already defined in AP203 and AP214, such as boundary representation, geometric validation properties (e.g. volume and surface area) or supplemental geometry (e.g. points, axes and planes) (Coronado et al. 2014). There are some sources which summarize the different implementation levels of the specific AP242 parts, but they concentrate on the structures for 3D Tessellated Geometry, XML representation and Product and Manufacturing Information (PMI) (AFNet 2015). The CAx Implementor Forum¹ lists the implementation coverage of ten different applications for over 100 sub-features of AP242, although not including the features for the representation of the elements of *design intent*. Similarly, the CAD vendors themselves offer some information about the implementation level of the respective AP, but these information are mostly for advertising purposes and therefore not very detailed. This information nonetheless confirms the impression described above. The structures for the representation of construction history, parameters, constraints and features are not implemented into today's CAD environments. Likewise, it could not be determined if and when CAD vendors plan to include these structures.

This only leaves lower level applications with EXPRESS parsers as option for the generation of the needed example files. As a result of their ability to map structures that are defined in EXPRESS schemas to corresponding structures in their database, these applications are not limited to specific APs.

3.2.1. Software tools

STEP Class Library – STEPcode

The STEP Class Library (SCL) was developed by the *National Institute of Standards and Technology* (NIST) in the early days of the STEP standard. It was renamed in 2012 to STEPcode². The project is a collection of open source libraries and tools around the technologies of ISO 10303. This collection of tools includes EXPRESS schema parsers with bindings provided in C, C++ and Python and a library that allows STEP Part 21 files to be read and written.

¹https://www.cax-if.org/vendor_info.php?file_id=8, last accessed: 2016-03-25

²<http://stepcode.org>, last accessed: 2016-03-20

The source files are written in C++ and the `ap203min` example provided in the project files³ was built with `cmake` in order to test the tool.

The population of entities is realized with an implementation of the SDAI. Due to the relatively low level access to the data model, even the definition of a single cartesian point needs a lot of additional code, as seen in the `ap203min` example. Thus, other tools with a higher level of abstraction were examined to generate the custom test files.

The Python binding⁴ is not yet complete, but parts of the code serve as basis for the implementation of the STEP Part 21 parser described in section 3.4.1.

JSDAI

JSDAI⁵ is an open source implementation of the SDAI in Java. The tool can parse EXPRESS schemas and map the entities to JAVA classes. For the same reason as mentioned in the previous section, the tool was not further examined.

STEP Tools

*STEP Tools, Inc.*⁶ provides several libraries and tools for the development of STEP related applications. These tools are packaged into the ST-Developer software that supports STEP AP203, AP214 and AP242. The software provides a common library of these APs in the form of C++ classes. Thus, the tool provides all means to generate custom AP242 files, but the process of populating even easy geometries is rather elaborate. The example C++ project, which populates the data model with different boxes, was successfully built in Visual Studio 2015.⁷ The library of C++ classes is called ROSE (Rensselaer Object System for Engineering) library and contains, besides the classes that hold the EXPRESS-defined data, support classes that help to index the data or to read and write it to the storage model.

Express Data Manager

The EXPRESS Data Manager (EDM) is a software solution from the company *Jotne EPM Technology*⁸ that provides a framework to connect different software applications with the

³<https://github.com/stepcode/stepcode>, last accessed: 2016-03-20

⁴<https://github.com/stepcode/stepcode/wiki/python-generator>, last accessed: 2016-03-20

⁵<http://www.jsdai.net>, last accessed: 2016-03-20

⁶<http://www.steptools.com>, last accessed: 2016-03-20

⁷http://www.steptools.com/support/stdev_docs/stpcad/demos/geometry.html, last accessed: 2016-03-20

⁸<http://www.epmtech.jotne.com>, last accessed: 2016-03-25

help of a shared database environment. Furthermore, this is achieved by providing interfaces to different programming languages and file formats, such as the STEP format. EDM's proprietary storage technology is called EDMdatabase and is designed to store and process all information defined in EXPRESS schemas and populations thereof.

Applications can be connected to the database with the help of the EDMinterface. It provides a SDAI C binding as well as a proprietary C binding. However, the EDM can also be controlled with a Graphical User Interface (GUI), which is the way the database manipulations were performed in the scope of this work. Modifications, such as the population with new entities or the change of attribute values, are defined in so-called query schemas. These schemas are written in an EDM specific language, whose syntax is similar to the EXPRESS syntax. In the following section, an example of a query function in such a schema is shown in order to illustrate the implementation of the example geometry in EDM.

This approach allows to focus on the population of the data structure, as EDM takes care of the SDAI calls in the background. Therefore, the query functions are very abstract and require only a small amount of code to populate the entities.

3.2.2. EDM implementation

As described in section 2.4.5, the envisaged way to implement an implicit procedural representation is to combine it with the explicit representation of the geometry. In the scope of this minimal implementation however, only the procedural representation is transferred.

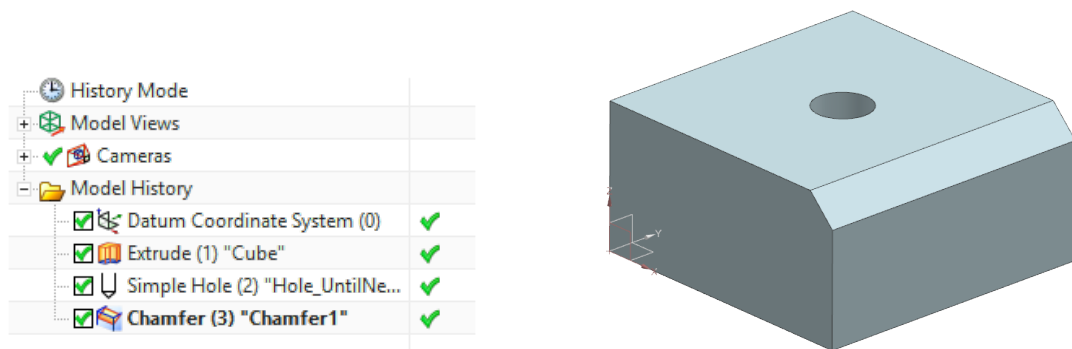


Figure 3.2.: Example geometry, generated in Siemens NX10.

The chosen example geometry shown in figure 3.2 consists of an extrusion with a through hole and a chamfer. It is based on one of the geometries used in Barber et al. (2010). With this part, the transfer of construction history, features and potentially parameters and constraints can be demonstrated. As described in the previous section, the population of the data model with entities is accomplished with the definition of an EDM query schema.

An example of such a query is shown in Listing 3.1. In the LOCAL environment, the local variables that are used in the query function are defined, in this case `cp_1` from the type `CARTESIAN_POINT`. The `NEW PERSISTENT` command writes the `cp_1` instance to the data model and the following commands assign values to the corresponding attribute slots. The values of the attributes can either be another entity instance or a generic data type, such as `string`. The complete query code is given in listing C.1.

```
1 QUERY_FUNCTION procedural_repr : STRING;
2
3 LOCAL
4   cp_1 : CARTESIAN_POINT;
5 END_LOCAL;
6
7   NEW PERSISTENT cp_1;
8   cp_1.name := 'Cartesian Point 1';
9   cp_1.coordinates := [0,0,0];
10
11 END_QUERY_FUNCTION;
```

Listing 3.1: Population of a `CARTESIAN_POINT`.

The implementation of structures that contain construction history and parameters are described in the following sections. As discussed in section 2.4.5, these structures are specifically designed to represent similar structures in CAD systems. The construction history for instance is represented by an instance of the `Procedural_Shape_Representation_Sequence`. All entity instances that are referenced by this sequence are part of the primary (also referred to as procedural or implicit) model, in contrast to the instances that define the secondary (explicit) model. In order to minimize the development effort, only the procedural model is used in this case. The main part of the additional effort for the implementation of the explicit model would not occur in the population of the data model but in the thereby necessary implementation of complex entities in the EXPRESS to AML mapping, described in section 3.3. Subsequently to the description of the population of the construction history, the implementation of the parameters is described. For the intended parameterization of one of the dimensions in the example geometry, the `Bound_Parameter_Environment` provides the necessary structures. The parameter is bound, as it is not independently present in the data model like an unbound parameter, but directly related to an attribute.

Construction history

The construction history is implemented with the help of the `Procedural_Shape_Representation_Sequence` described in section 2.4.5. The resulting hierarchy is shown in figure 3.3. The first construction step is the extrusion represented by the entity `Extruded_Face_Solid` with a branch of sub entities that define it. Next, the hole feature refers to the extruded solid as base object and adds the entities that define the hole. Finally, the chamfer is applied on the solid that already contains the hole. This way, each stage of the construction history is stored and accessible.

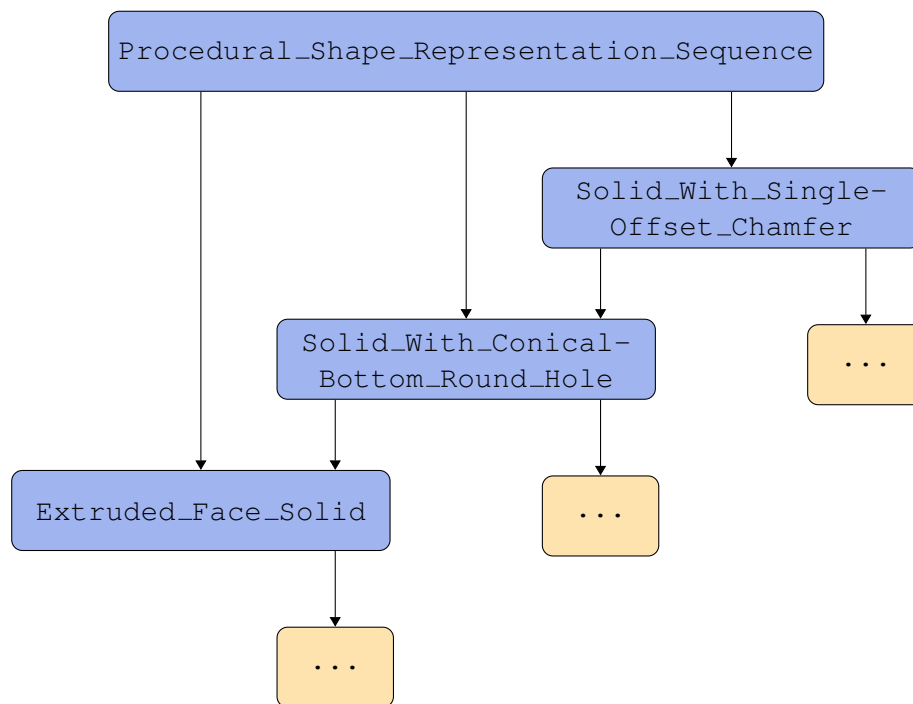


Figure 3.3.: Hierarchical structure of a `Procedural_Shape_Representation_Sequence`.

Parameters

As described in section 2.4.6, parameters are not directly stored in the instance whose attribute is parameterized, but as an additional instance of a `Bound_Model_Parameter`. This parameter and the targeted object instance are connected with the help of a `Bound_Parameter_Environment`, as seen in figure 3.4. In the case of this minimal example, the height of the extrusion is parameterized. However, this parameterization does not only influence the extrusion feature, but also the `Cartesian_Point` that defines the starting point for the hole feature or the position of the chamfered edge. Ideally, this dependency is

apparent from the way the elements are defined. The starting point of the hole for example could be defined relative to the plane in which the upper surface of the extruded cube lies in. In CAD systems, this reference usually is established with the help of one or more auxiliary planes or coordinate systems. Even though the same relations could be modeled with the help of the structures provided by AP242, the hole and the chamfer in the example geometry are not defined relative to the extruded part in order to keep the definition as simple as possible. In this special case, it is beneficial that the parameterized dimension is not directly stored in the attribute of the extrusion instance, as the parameter now can be connected to any attribute with the help of the mentioned `Bound_Parameter_Environment`. The supposedly easier solution would be to reference the parameter directly from within the attribute. One reason why it is not implemented like this in the standard, could be the compatibility to interfaces that do not support the interpretation of the parameter. With the `Bound_Parameter_Environment`, the model can be processed without the parameters and is still valid, while the parameters can be added later if they are supported. Thus, for all the depended attributes in the example geometry, a `Bound_Parameter_Environment` is defined that assigns the parameter with the name `Extrusion_Height` to each of these attributes. The `Instance_Attribute_Reference` refers to the instance that owns the corresponding attribute. This reference contains the schema that defines the corresponding entity, the entity name itself and the attribute name. In the case of the extrusion, that adds up to `GEOMETRIC_MODEL_SCHEMA.EXTRUDED_FACE_SOLID.DEPTH`. If the attribute has multiple dimensions, the index of the dimension is appended in square brackets.

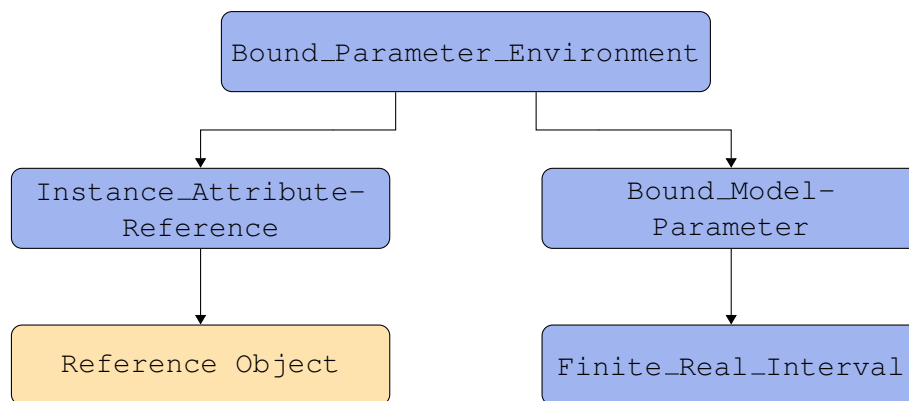
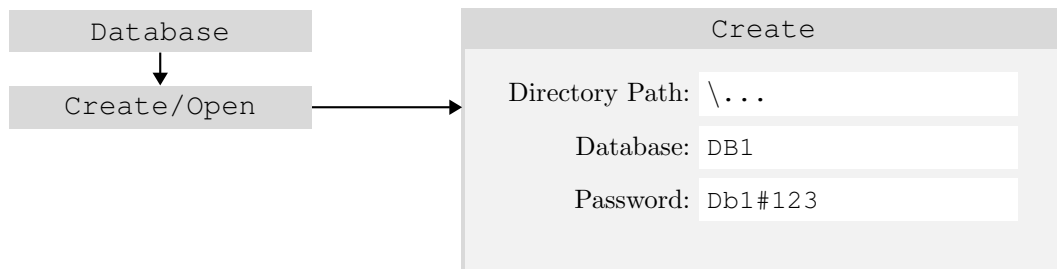


Figure 3.4.: Hierarchical structure of a `Bound_Parameter_Environment`.

Data population

The first step in the EDM is to create a database with the following commands in the GUI. The other commands described in this section are depicted with the corresponding settings in appendix B).



Next, the EXPRESS schema that describes the AP242 Module Interpreted Model in its long form (`mim_1f.exp`) is loaded into the database. The database is now able to store data structures according to the AP242 definitions.

After creating a data model in the database, this data model can be populated with the query schema. The model can either be empty or already populated with entity instances from a STEP import for example. As the entity `bound_variational_parameter` has two attributes with the name `label` inherited from different supertype entities, the value of one of the attributes is unaffected by the query command and is edited manually in the GUI. Finally, the populated data model can be exported as STEP or XML file.

3.3. EXPRESS structure mapping

The representation structures defined in the EXPRESS schema of specific AP have to be mapped to the appropriate structures in the receiving system in order to enable the import of data structures that are defined according to this EXPRESS schema. In the case of AML, this representation is accomplished by defining a class for each entity in the schema. This class ideally maps all the information available in the entity definition, such as super-/subtype relations or attributes. As described in section 2.4.2, it is possible to store much more information in EXPRESS schemas, such as functions, types and constants, but these are omitted in the scope of this work, since this first minimal implementation only aims to provide a proof-of-concept application.

The generation of the AML classes is automated with a parser and is described in section 3.3.3. Since the STEP file parser, which is described in section 3.4.1, is based on a Python parser

from the STEPtools project, the parser for the AML class mapping is implemented with a Python parser as well.

3.3.1. Software tools

Python Lex–Yacc Compiler

The Python tool PLY (Python Lex–Yacc) is used for several tasks in this work.^{9,10} PLY is a Python implementation of the C tools Lex and Yacc. Both tools together form a so-called compiler. A compiler is a computer program that transforms source code from one programming language into another. A common use case is the translation from human readable high-level programming languages into executable machine code. Another application is the translation between different language families. Compilers are often separated into two parts. The first part analyses the structure of the source program and the second part transfers this structure into the target language. The analyzer part itself can be separated in a lexical analyzer, which is called Lex in the case of PLY, and a parser, which is called Yacc. The lexical analyzer splits the source code into discrete pieces, according to so-called token rules. These token rules are defined with the help of a sequence of characters that define a search pattern, which in turn are referred to as *regular expressions*. If this search pattern applies to a sequence of characters, a token is generated and the sequence of characters is stored in it as token value. The result is a sequence of tokens that is fed into the parser. The parser tries to match them to predefined grammatical rules, which are called productions. After this grammatical analysis, the syntax can be translated into another language.

For the tasks in this thesis, PLY is rather used as a converter than as a compiler since it is not used to compile source code from one language to another but to generate code from different data structures.

Adaptive Modeling Language

Multiple AML source files can be compiled into a so-called AML system by defining a `system.def` file in the system's folder that contains a list of these source files. The system setup used for in the scope of this thesis is shown in listing 3.2.

The path of the system and other path variables can be defined in the `logical.pth` file in the installation folder of AML in order to be able to compile the system with a relative path call. The path definitions for this project are shown in listing 3.3.

⁹<https://www.python.org/>, used python version: 2.7

¹⁰PLY can be installed via pip or directly with the containing `setup.py` (run `$ python setup.py install`)

```

1 (define-system :main-system
2   :require-systems ()
3   :require-libs ()
4   :files ' (
5     "geometric_representation_classes.aml"
6     "feature_classes.aml"
7     "populate.aml"
8     "main.aml"
9   )
10 )

```

Listing 3.2: Definition of the :main-system.

```

1 :thesis          C:\...\thesis\
2 :main-system    :thesis 01_binding\out\systems\main-system\
3 :class-path     :thesis 01_binding\out\classes\

```

Listing 3.3: Path variables defined in logical.pth.

The class hierarchy in the AP242 schema is highly branched and contains over 1700 entities. As the corresponding classes in AML have to be compiled in the right order, the classes load their necessary parent classes on the fly, so that these are compiled prior to themselves. Another advantage of this approach is that only the classes that are used are compiled. This is accomplished by creating one file per class and by adding the following function call for each parent class at the beginning of the class files: `(load_class "parent_class_name")`. The `load_class` function is defined in listing 3.4.

```

21 (defun load_class (class_name)
22   (if (subclassp (make-symbol class_name) 'object)
23     "Class already loaded"
24     (load (logical-path :class-path (concatenate class_name ".aml"))))
25   )
26 )

```

Listing 3.4: Function to load classes on demand. (File: main.aml)

3.3.2. Wirth Syntax Notation converter

The lexical and grammatical rules for EXPRESS schemas are defined in ISO 10303-11:2004. These 345 rules are defined in the Wirth Syntax Notation and are displayed in appendix D as railroad syntax diagrams. WSN is a notation technique for context-free grammars that can be used to describe the syntax of a computer language. As the used Python parser PLY

cannot interpret WSN notations, they have to be transformed into the Backus Normal Form (BNF), which is a related notation form, with the following transformation rules.

- Convert every repetition $\{C\}$ to a new production X

$$A = B \{C\} \quad \rightarrow \quad A = B X$$

$$X = \epsilon \mid X C$$
- Convert every option $[C]$ to a new production X

$$A = B [C] \quad \rightarrow \quad A = B X$$

$$X = \epsilon \mid C$$
- Convert every group (C) to a new production X

$$A = B (C) \quad \rightarrow \quad A = B X$$

$$X = C$$

The WSN converter therefore parses the WSN rules, subsequently transforms them into BNF rules and finally formats them into a new parser code (`schema_parser.py`). Some of the WSN rules are shown exemplarily in listing 3.5. The generated grammar is directly compilable, but the definition of the data that should be transferred to the next rule must be done manually. Some of the rules are split into multiple separate rules and the output for the rules is defined for the different occurring cases, as discussed in section 3.3.3.

```

3   0 ABS = 'abs' .
4   1 ABSTRACT = 'abstract' .
5   2 ACOS = 'acos' .
126 123 bit = '0' | '1' .
127 124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
128 125 digits = digit { digit } .
129 126 encoded_character = octet octet octet octet .
130 127 hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .
131 128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j'
      ↪ | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u'
      ↪ ' | 'v' | 'w' | 'x' | 'y' | 'z' .
132 129 lparen_then_not_lparen_star = '(' { '(' } not_lparen_star {
      ↪ not_lparen_star } .
218 215 explicit_attr = attribute_decl { ',' attribute_decl } ':' [
      ↪ OPTIONAL ] parameter_type ';' .
299 296 schema_decl = SCHEMA schema_id [ schema_version_id ] ';'
      ↪ schema_body END_SCHEMA ';' .
327 324 syntax = schema_decl { schema_decl } .

```

Listing 3.5: WSN rules defined in ISO 10303-11:2004. (File: iso-10303-11-2004.bnf)

3.3.3. EXPRESS schema parser

The EXPRESS schema parser converts the entities that are defined in the AP242 EXPRESS schema into AML classes. The lexing and parsing rules were generated in the preparation step described in the previous section. ISO 10303–11:2004 defines that an implementation of an EXPRESS language parser shall be able to parse any formal specification written in EXPRESS and shall be said to conform to a particular checking level if it can apply all checks required by the level (and any level below that) to a formal specification written in EXPRESS. The implemented parser was not checked for any conformance level, but as the parser is derived from the rules defined in the standard itself, the lexical and grammatical analysis part should conform to the standard. The transfer to AML code however is incomplete, as only some of the defined structures were taken into account. The relevant data is mapped to a class model defined in the parser, as seen in listing 3.6.

```

85 # p_schema_body : p_stack_45 p_constant_decl p_stack_46 # p_stack_45:
      ↪ p_interface_specification # p_stack_46: declarations
86 class Schema_body:
87     def __init__(self, interface_specification , declarations):
88         self.interface_specification = interface_specification
89         self.declarations = declarations
90
91 #p_schema_decl : SCHEMA p_schema_id p_schema_version_id ';'
      ↪ p_schema_body END_SCHEMA
92 class Schema_decl:
93     def __init__(self, schema_id, schema_version_id, schema_body):
94         self.schema_id = schema_id
95         self.schema_version_id = schema_version_id
96         self.schema_body = schema_body
97
98 #p_syntax : p_schema_decl
99 class Syntax:
100     def __init__(self, schema_decl):
101         self.schema_decl = schema_decl
102 # ...

```

Listing 3.6: Extract of the class model representing the EXPRESS Structure. (File: schema_parser.py)

The class model simplifies the handling of the data in the next step, which is the AML class generation defined in the file `write_aml.py`. The classes can be instantiated by the production rules with the corresponding attributes, like seen in listing 3.7.

```

188     def p_syntax(self, p):
189         """ p_syntax : p_schema_decl
190             | p_syntax p_schema_decl """
191         if len(p) == 3:
192             p[0] = p[1] + p[2]
193         elif len(p) == 2:
194             p[0] = Syntax(p[1])
195         else:
196             print 'Indexerror in p_syntax with len(p):'
197             print len(p)

```

Listing 3.7: Root production of the EXPRESS grammar. (File: schema_parser.py)

The `syntax` production is the root production of the grammar and its content is returned by the parser as result. The content of the tokens that are matched by the other productions can be processed with specific rules and is subsequently passed to the next highest production until it reaches the root production. The grammar however has some unused branches, namely the productions related to remarks (`tail_remark`, `embedded_remark`, etc.). As they have no connection to the root rule, they are not reachable. This leads to a warning during the compilation of the parser but it does not effect the other productions.

3.3.4. AML code generation

The output from the parser described in the previous section is an AML file for each class corresponding to a specific entity. To simplify the use of these classes, the inherited attributes and the information about where they come from are included as a comment. As mentioned before, the necessary parent classes are loaded at the beginning of each class file.

```

23 (in-package :aml)
24
25 (load_class "ap242_swept_face_solid")
26
27 (define-class ap242_extruded_face_solid
28   :inherit-from (
29     ap242_swept_face_solid
30     aml_extrusion_object
31   )
32   :properties (
33     index (list
34           'name

```

```

35         'swept_face
36         'extruded_direction
37         'depth
38     )
48     extruded_direction 'direction
49     depth 'positive_length_measure
50 )
51 :subobjects (
52
53 )
54 )

```

Listing 3.8: Translated extruded_face_solid class. (File: ap242_extruded_face_solid.aml)

The class names are modified with a prefix to clarify that they are derived from AP242 entities. To simplify the geometric representation in AML, the classes are mapped to similar native AML classes from which they inherit the Virtual Geometry Layer functions (TechnoSoft 2010). This leads to some problems, because the class structure in STEP is much more subdivided than in AML. Therefore, a mapping class for each of these connections is defined to clarify where the class can get the attribute values to provide the generic AML classes with the necessary data for the geometrical representation. One example for this different class structure is the `extrusion_object` shown in listing 3.8, where the necessary data has to be collected from different child entities. The mapping class `aml_extrusion_object` provides the necessary connections and is implemented as parent class into the corresponding ap242 class. As shown in listing 3.9, this is done automatically by listing the classes that are to be mapped in the Python code that generates the AML classes. The mapping classes however have to be defined manually. The disadvantages of this approach and possible alternatives are discussed in chapter 5.

```

18 aml_entity_suffix = ''
19 aml_type_suffix = ''
20 aml_entity_prefix = 'ap242_'
21 aml_type_prefix = 'ap242_'
22 system_name = 'main-system' # defined in logical.pth in AML path
23 class_path = 'class-path' # defined in logical.pth in AML path
24
25 aml_class_mapping = {
26     'cartesian_point' : 'aml_point_object',
27     #'vector' : 'vector-class',
28     #'line' : 'line-object',

```



```

29     'edge_curve'           : 'aml_line_object',
30     'edge_loop'          : 'aml_sewn_object',
31     'face_bound'         : 'aml_bounded_object',
32     'extruded_face_solid' : 'aml_extrusion_object',
33     'solid_with_flat_bottom_round_hole' : '
        ↪ xaml_flat_bottom_round_hole_feature',
34     'axis2_placement_3d'  : 'aml_coordinate_system_class',
35     'solid_with_single_offset_chamfer' : 'xaml_single_offset_feature',
36     'bound_parameter_environment' : 'xaml_parameter'
37 }

```

Listing 3.9: Mapping between AP242 classes and AML geometric objects. (File: schema_write_aml.py)

The mapping is only implemented for the classes that are used in the example geometry. That means that other entities can be populated into the AML object tree, but without geometric representation. The implemented mapping classes are defined in the `geometric_representation_classes.aml` file. Similarly, the representation of the CAD features, namely the extrusion, hole and chamfer feature are implemented with specific AML classes defined in `feature_classes.aml`. Some of the features are straight forward to implement, as AML provides similar feature classes, such as for the extrusion or the hole, but some are more complex such as the chamfer. According to ISO 10303-111:2007, chamfers are created with the help of two curves with an offset from the original edge along the two adjacent surfaces. A ruled surface is then created between these curves and is used as new chamfer surface. However, these adjacent surfaces are not specified in the exchange file, as they can be the result of another feature such as an extrusion. Thus, the AML class for the chamfer feature has to find these surfaces and their spatial orientation in order to define the offset curves. This is possible with the low-level `:vgl` (Virtual Geometry Layer) functions (TechnoSoft 2010, Appendix C: VGL Functions), which are not documented very extensively. For the chamfer for example, the faces of the object can be returned with the function `(vgl::k-sub-geoms (the geom) 2)`. Subsequently, the faces that include the corresponding edge have to be determined. Next, the edge has to be displaced along these surfaces and a new surface has to be created. Due to difficulties with these functions, the chamfer feature is only partially implemented, which means that the import does not work with every possible chamfer.

3.4. Data transfer

The last step is to transfer the data in the physical STEP file into the AML data model. Since the software used in this step is the same as in the second step, their setup is not repeated here.

3.4.1. STEP P21 parser

Once again, the interpretation of the data is accomplished with a PLY parser. This parser however is partly based on the STEP P21 parser provided by the SCL/STEPcode project mentioned in section 3.2.1. The parser reads the STEP file and generates a transfer file that subsequently is implemented into the AML environment. This transfer file contains a Lisp data structure with the instances that are defined in the STEP file together with their attribute slot values.

3.4.2. AML population

The interpretation of the transfer file is accomplished with a set of AML classes and functions defined in `main.aml`. Each instance that is defined in the transfer file is processed and populated into the data model. Their attribute slots are assigned to the right property with the help of the index property defined in the class. The first slot is assigned to the first member of this index list and so forth. The rules applied to determine the order of the list members are described in section 2.4.3.

The entity instances defined in the STEP file form a hierarchical structure with possibly multiple root instances. In the case of the example geometry, the multiple roots result from the `Bound_Parameter_Environments` described in section 2.4.6, as they have no parent object. Child objects are populated as subobjects into their parent object, they hence have to be populated prior to their parent classes. But at the same time, the inheritance relations are defined in the top-down direction in the STEP file, so the population has to start at the roots. In order to resolve this contradiction, a recursive function is defined. This population function starts at the roots and recursively calls itself for the population of possible child instances until it reaches a leaf instance. This leaf instance is then populated and the function calls can finish one after the other and climb back to the root entity.

The data structure in STEP files can not only fork in the top-down direction, but also merge, i.e. instances can be referred to from multiple parent instances. These relations are mapped in AML as subobject relations. AML however does not support the assignment of the same

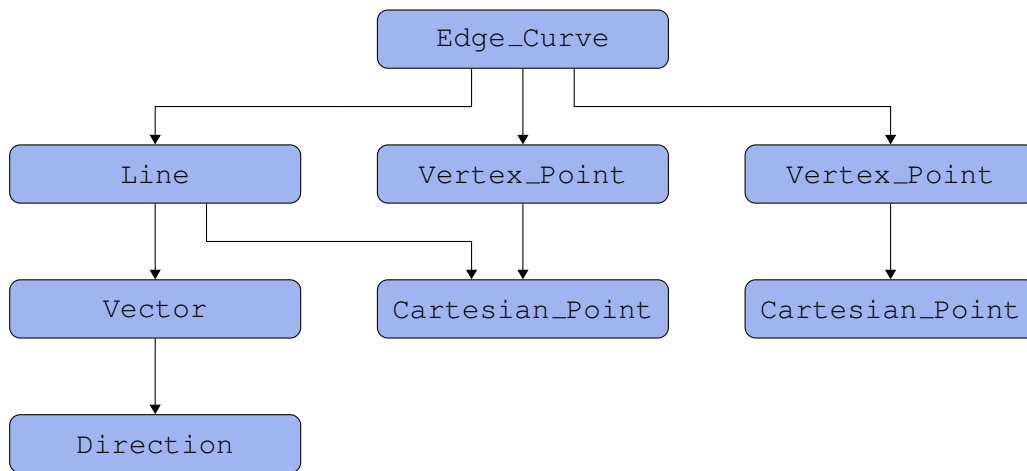


Figure 3.5.: Hierarchical structure of an Edge_Curve in the STEP exchange file.

subobject to multiple parent objects. This means that each time the class tree merges, the following branch is duplicated. An example for this merging in the top-down direction is shown in figure 3.5. An Edge_Curve is built from two Vertex_Points and a connecting Line in between them. The Vertex_Points each refer to a Cartesian_Point and the Line is defined by one of these Cartesian_Points and a Vector. Therefore, the Cartesian_Point is used by two other objects and is duplicated in AML in order to be represented in both parent objects. In the case of the Procedural_Shape_Representation described in section 3.2.2, whole branches are duplicated to represent the different stages of construction history.

After the population of the data structure, the parameterization has to be implemented. The information about which attributes have to be parameterized is stored in the Bound_Parameter_Environments. These objects are therefore processed one after another in order to change the formula of the corresponding properties. As some of the objects are duplicated, as mentioned before, they all have to be changed accordingly. One of the functions enabling this process is shown in listing 3.10. The split-reference function splits an Instance-Attribute_Reference, such as GEOMETRIC_MODEL_SCHEMA.CARTESIAN_POINT.COORDINATES [2] into the schema, entity name, property and a possible index. These are then used to access the specific attribute in order to implement the dependency to the parameter. The other functions and classes enabling the parameterization are listed in appendix C.2.1.

```

139 ;; example input: GEOMETRIC_MODEL_SCHEMA.CARTESIAN_POINT.COORDINATES [2]
140 (defun split-reference (string)
141   (setf sub_list '())
142   (setf sub_string string)
143   (setf rest_string string)

```

```

144 (loop for point_sep from 1 to (count #\. string)
145   do (let (
146         (substr (subseq rest_string 0 (position #\. rest_string)))
147         (reststr (subseq rest_string (+ (position #\. rest_string)
148           ↪ 1) (length rest_string)))
149       )
150     (progn
151       (setf sub_string substr)
152       (setf rest_string reststr)
153       (setf sub_list (append sub_list (list sub_string)))
154     )
155   )
156 ;; Check for index -- e.g. coordinates[1]
157 (if (= 1 (count #\[ rest_string))
158   (progn
159     (print rest_string)
160     (setf sub_string (subseq rest_string 0 (position #\[
161       ↪ rest_string)))
162     (setf rest_string (subseq rest_string (+ (position #\[
163       ↪ rest_string) 1) (- (length rest_string) 1)))
164     (setf sub_list (append sub_list (list sub_string)))
165   )
166   (setf sub_list (append sub_list (list rest_string)))
167   (print sub_list)
168 )

```

Listing 3.10: Function to split a reference path into schema, entity, property and index.
(File: main.aml)

4. Results

The purpose of transferring the example geometry defined in section 3.2 was to evaluate, whether it is possible to transfer knowledge in the form of *design intent* between CAD and KBE environments with the help of the AP242 STEP format. In section 3.1, three steps were defined that are needed to accomplish this exemplary transfer. The first step – the acquisition or generation of an appropriate example file – was implemented with the help of the EDM software from *Jotne*. For the second step – the mapping of the EXPRESS structures defined in AP242 to AML classes – a converter was developed with the compiler tool PLY. The mapping is not complete yet, the 1727 entities together with their inheritance structure and attributes were however successfully converted into AML classes. For the third step – the actual data transfer into AML – some additional classes and functions had to be developed in AML. Especially the geometrical representation had to be implemented manually, as it is system specific and neither defined in the EXPRESS schema nor in the STEP exchange file. This manual implementation was performed with the help of native AML classes that contain similar geometries. As parts of the geometrical representation, the features were also implemented with a mapping to native AML classes.

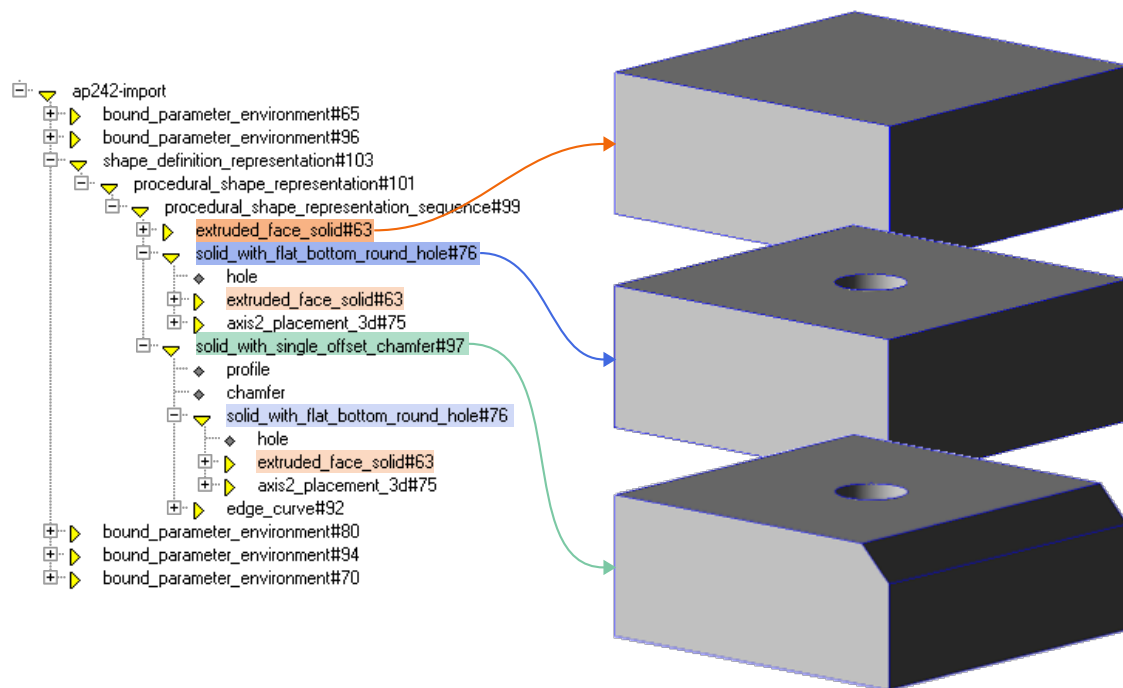


Figure 4.1.: Object tree of the imported model. Geometrical representation in different stages of the construction history.

The object tree of the resulting model is depicted in figure 4.1, where the graphical representations on the right show the different stages of the construction history. Figure 4.2 shows the parameterization of the `Extrusion_Depth`. The parameter value is accessible in the root object `ap242-import` and controls all connected attributes.

In conclusion, the transfer of elements of *design intent* was successful. Furthermore, a possible approach to implement the import of STEP data structures, which extends the current scope of purely geometrical information, was shown. Nevertheless, during the process of implementing a transfer chain from the geometry model to the STEP exchange file and into AML, certain limitations emerge. These limitations are mostly related to the CAD and AML environments or have a political or economic origin. The STEP standard itself provides the structures to represent most of the data that is produced during a product's life cycle. Still, it is not easy to implement interfaces to these structures in existing applications. These and other aspects are further discussed in chapter 5.

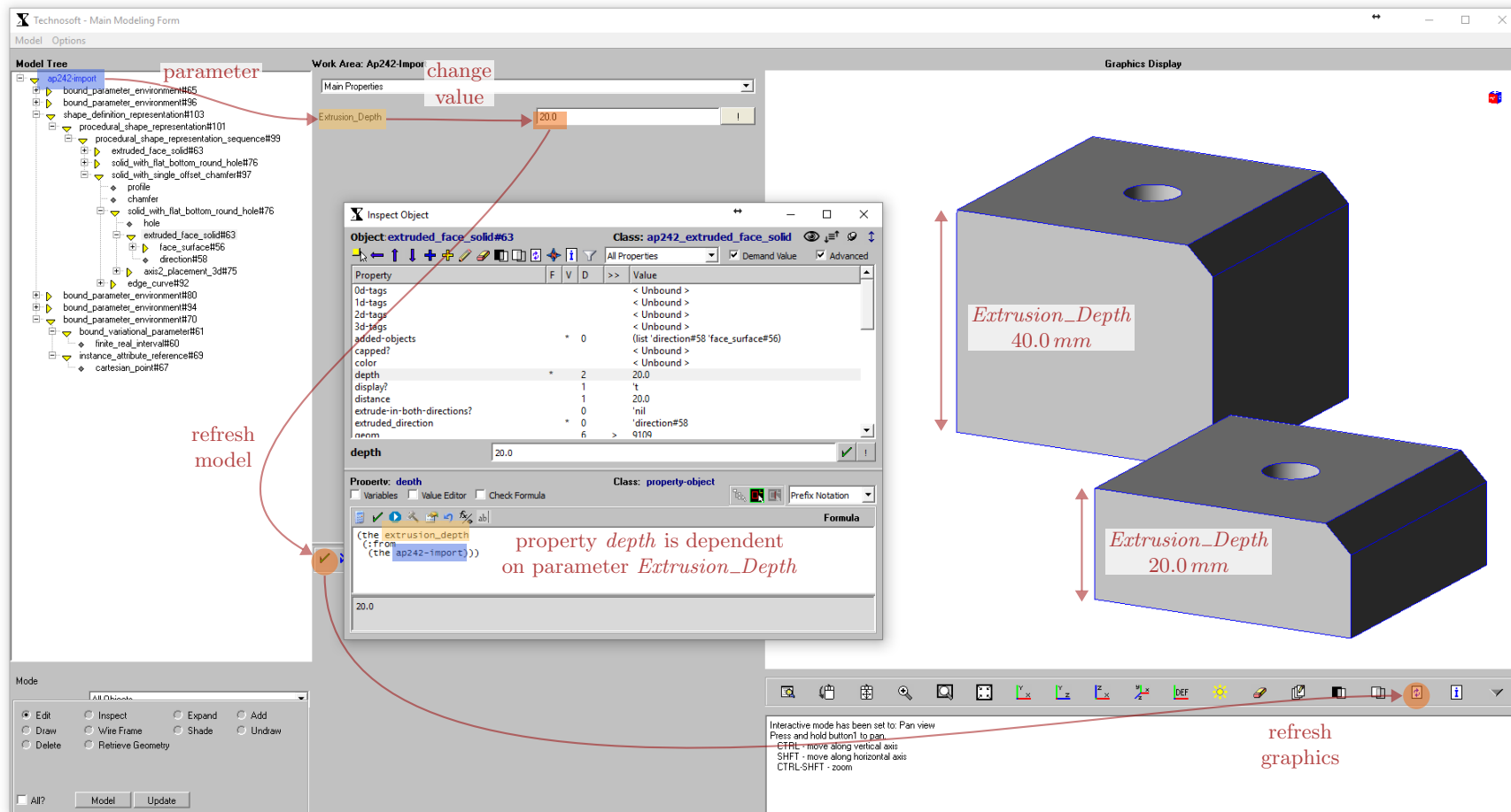


Figure 4.2.: Example geometry imported into AML. The parameter `Extrusion_Depth` is stored as attribute in the root-object `ap242-import` and can be changed in the work area. The second graphical representation shows an overlay of the same model with a different value of the parameter `Extrusion_Depth`.

5. Discussion

One key question of this thesis is, whether the STEP format can enable the knowledge transfer between CAD and KBE environments. The proof-of-concept translation of an example geometry illustrated that it is in fact possible to transfer product knowledge in the form of STEP files into the KBE framework AML. The targeted scope of knowledge – the intentions of the designer – however represents all design decisions along with the underlying processes leading to them, which raises the more specific question, whether these design intentions can be structurally represented with the STEP standard, which is considerably more difficult to investigate. Thus, the investigations that are described in section 2.4 with the aim to determine the capabilities of STEP to transfer such knowledge are focused on the elements that are commonly referred to as the *design intent* in CAD models. In particular, elements, such as construction history, parameterized and constrained dimensions and features were examined. The following section offers a more detailed discussion on the current state of the implementation of these elements in STEP. Moreover, the following section includes an attempt to answer the questions posed above. The chapter concludes with the insights gained during the development of the STEP translator and the experiences with its implementation into AML

5.1. Knowledge transfer with the STEP standard

In section 1.4, the states of research and standardization in the field of product data exchange are reviewed. One important conclusion of this chapter is that the STEP standard is commonly viewed as the most capable neutral file format, when it comes to the transfer and storage of product data. The standard already covers large parts of the data structures that are generated during a product's life cycle. Moreover, the projects discussed in section 1.4 as well as the investigations of this thesis show that the STEP standard provides the necessary structures to represent the elements of *design intent* mentioned above. These structures cover a considerable amount of today's typical data structures in this area, such as the features in CAD software, but still have to prove their applicability in larger scale real world applications. The development of the standard is pursued with the effort of several international associations and project groups who are focused on the extension of the standard's capabilities. Due to its modular architecture, the standard can be gradually extended and can therefore react to changing requirements.

However, there are different reasons, why the standard cannot provide a universal exchange format that covers all the possible data structures used in any platform and why the vendors of CAE software do not necessarily implement all parts of the standard.

First, the development of the standard is aimed at mapping the structures of existing systems, such as CAD tools. These tools from different vendors gradually evolved over the years and although their data structures resemble each other, they are not completely congruent. One example for such differences are the features in CAD systems. The most common features, such as holes, chamfers or blendings can be found in most of the CAD systems, but possibly with a different set of sub-features, such as the tapered hole feature in Siemens NX 10, which is not available in PTC Creo Parametric 3.0. Thus, the STEP standard has to define a set of structures that covers ideally all implementations, but at the same time must adhere to one of its main design goals – minimal redundancy. Therefore, the standard has to find a balance between the unification of features and the support of existing structures. This in turn means that the vendors possibly have to adapt their structures to be able to translate them with the standard.

Second, vendors that are established on the market benefit from the lack of a neutral file format that can transfer data models without loss between systems from different vendors. Once a company uses one particular system, the barrier is high for a change to another system, as this would cause immense costs and the loss of data (Stiteler 2004).

Third, it is expensive for the vendors to extend the capabilities of their STEP interfaces. This means that the implementation of additional Application Protocols or Conformance Classes is demand-driven and the vendors must see a financial benefit in putting effort into the development of their translators.

And fourth, the validation of the conformance of the implemented interfaces is a general challenge with exchange formats. As discussed in section 2.4.1, the STEP standard provides conformance tests to validate implementations of the standard. However, a successful conformance test does not guarantee interoperability with other systems (ISO 10303–31:1994). One reason for this is that the systems handle the received data differently on binary level, such as different precision in the representation of real numbers or different geometric modeling kernels. Therefore, implementations of the STEP standard, such as EXPRESS parsers or STEP translators, have to be classified into different conformance levels and have to provide information about the internal data handling (see appendices E in ISO 10303–11:2004 and ISO 10303–21:2002). Despite the fact that complete conformance between different implementations cannot be proved without a complete insight into the corresponding systems, the

conformance tests increase the probability that these implementations are able to interoperate (ISO 10303–31:1994).

The investigations in section 3.2 confirm these points and show that AP242 is not yet supported in the latest CAD systems. Some vendors claim to work on an implementation of the AP242 structures, but they offer no specific information about the parts that enable the transfer of *design intent*, namely part 55, part 108, part 111 and part 112, as specified in section 2.4. This lack of implementation of certain STEP parts leads to alternative approaches in the use of the STEP format as intermediate exchange format. As described in section 1.4, most of the projects that use capabilities of the STEP standard that are not yet implemented in the corresponding software access the native data model of that software with the help of an API or with journal files. With direct access to each step of the construction process, they can create their own STEP files with a translator tool. Thus, they are independent from the implementation level of the system’s translators. However, this approach requires that the vendors provide the necessary interfaces.

It is difficult to predict whether and when AP242 will be supported by commercial CAE products, since many parties with different interests are involved. As discussed in this section, the software vendors will only implement the standard when large commercial users require them to validate their STEP translators to the new standard. Since the STEP format is nowadays the most widely used 3D exchange format between different CAE platforms (based on the development of the past years, as seen in Prawel (2010)), most of the platforms support parts of it – in most cases AP203 and AP214. As AP242 merges and extends these two most widely used STEP APs, it will play a decisive role in the future of the format. STEP AP242 only has one Conformance Class, the `managed_model_based_3d_engineering_cc1`. Therefore, an implementation of AP242 has to support all structures defined in the AP in order to comply with the standard. Thus, despite the fact that the modules discussed in this thesis are currently not the main motivation for the CAD vendors for the implementation of AP242, their implementation together with the other modules of the standard seems possible.

In conclusion, the STEP standard in general and STEP AP242 in particular enable the representation – and therefore at least theoretically the transfer – of knowledge elements, such as construction history, parameterized and constrained dimensions and features. At the moment however, most of these capabilities of the STEP format are not used by commercial products. Thus, the transfer of such data structures is not limited by the standard itself, but rather by the lack of its implementation in the corresponding systems.

The implementation of an import tool that supports these structures in the KBE framework AML is feasible, as shown in this work with the transfer of an example geometry. In any case,

a complete implementation is a great development effort. The reasons for that are specified in the next section.

5.2. Implementation of a STEP translator

The STEP import functionality into AML is implemented in two levels. First, the structures that are defined in the standard's schema are converted into an AML class structure. With this library of classes, AML can represent the objects defined in the STEP exchange file. Second, the exchange file is parsed for the objects contained in the file, and a set of AML classes and functions populate the objects into the AML data model. This section contains a discussion of the implementation of both levels and an outlook on what remains to be implemented.

The first level, the mapping of EXPRESS structures into AML, is accomplished with a Python parser, as addressed in section 3.3.3. An EXPRESS schema contains declarations of the types CONSTANT, TYPE, ENTITY, RULE and FUNCTION. The central data structure is defined by the entities, which represent the object structures in the sending or receiving systems. The other structures are used by the entity declarations to specify these entities, as described in section 2.4.2. The parser implemented in the scope of this work can read EXPRESS files that are defined according to the syntax rules in ISO 10303–11:2004. However, the code generating part of the parser only processes the entity structures and not the other declarations. That means that the entities are mapped into AML classes, but are limited when it comes to data types, derived properties, restricting rules and other attributes defined in the EXPRESS schema. Nevertheless, the mapping is sufficient to represent the data structure defined in the exchange file in AML, since all entities (in the case of AP242: 1727 entities) can be instantiated with the appropriate properties.

In order to enable the import of the STEP files into AML, several classes and functions are defined that provide the necessary functionalities. The most fundamental functionality is the population of the data structures defined in the exchange format, which is described in section 3.4.

Another important functionality, which is system-specific and therefore not defined in the STEP exchange file or the EXPRESS schema, is the graphical representation of the geometry model. This is accomplished with a mapping of the translated AP242 classes to similar native AML classes that provide a pre-defined graphical representation. As the structures in AML and STEP are considerably different, this approach is limited. Thus, for a complete implementation, the low level `:vgl` functions have to be used to define a graphical representation

for each of the AP242 classes that represent geometry. Similarly, these functions are used by the pre-defined AML classes, such as the `box-object`, to generate the graphical output.

Furthermore, the features have to be recreated in AML. Similar to the graphical representation, this is implemented with the help of pre-defined AML classes, such as the `difference-object` for the hole feature. The chamfer feature in the example geometry shows that this approach is also limited, as the pre-defined AML classes cannot cover all the structures that can be represented with STEP. Hence, for a complete implementation, the definition of custom mapping classes is unavoidable.

The implementation of parameterized properties is covered in section 3.3.4 and is accomplished by processing the `Bound_Parameter_Environments` and by subsequently changing the formulas of the properties accordingly. Constraints can be implemented in a similar way.

The remaining element of *design intent* – the construction history – is intrinsically present in the object structure defined in the `Procedural_Shape_Representation_Sequence` and therefore does not need a specific mapping to AML other than via its structure.

Despite the implementation of the described functionalities, the object tree structure in the AML environment, as presented in chapter 4, is different to the one in a CAD environment. The current implementation of the STEP importer functions map the complete model structure in the STEP file as object – sub-object relations into the AML object tree. As a result, not only the elements that are represented in CAD model trees, such as features and the construction history, are present in the tree structure, but also other elements, which are usually encapsulated in construction elements in CAD systems, such as cartesian points in sketches. This can however be adapted by moving the sub-objects that should not appear in the object tree into the property declaration of the corresponding class.

The successful implementation of a subset of the structures provided by AP242, illustrates – even though this subset is only a small part of the complete implementation – that a (mostly) complete implementation of the STEP AP242 structures is in fact possible and only a matter of effort. Once the corresponding mapping classes for the graphical representation, the CAD features and the other STEP structures are implemented, the import of physical STEP files is fully automated.

In order to fully exploit the synergy between CAD and KBE environments, as mentioned in section 1.2, the other direction of the knowledge transfer has to be implemented as well. In the case of AML that means that the AML classes have to be mapped to the corresponding EXPRESS entities. As discussed before, this mapping is not implementable one-to-one as the structures differ too much. Especially considering that it is possible to define any custom

class structure in AML, a complete representation of these structures in STEP is not feasible. Still, the classes that build on known elements, such as the classes that were mapped from AP242, could be exported. This could be interesting for several scenarios, such as the initial definition of a complex system in AML and the final modeling of the geometrical details in a CAD system. Furthermore, the STEP standard could provide the structures to represent codified knowledge for the KBE knowledge acquisition process discussed in section 1.4.

6. Conclusion

Based on a detailed review of the STEP standard and the related literature, this thesis evaluates the standard's potential in regard to the transfer of elements of *design intent* between CAD and KBE environments. In order to substantiate the theoretical findings and to provide a proof-of-concept, an example geometry is defined and transferred from a STEP file into the KBE framework AML.

The specifications of the STEP standard, in particular of STEP AP242, demonstrate that the format enables the transfer of data structures that contain information about the intention behind certain design decisions. In the context of CAD systems, the construction history, parameterized and constrained dimensions and features mainly represent such information. A transfer of these elements between two systems can enable the modification of the model according to the intentions of the designer also in the receiving system. In the case of KBE and CAD environments, this can enable a complementary development process, in which the advantages of both environments can be exploited. For a successful transfer between those systems however, a suitable transfer file is not enough. In addition, the import and export interfaces of the standard have to be provided by the systems themselves.

The level of implementation of the STEP Application Protocols in CAD systems is very limited for several reasons. The current implementations are focused on the exchange of purely geometrical data, but according to some of the biggest CAD vendors and the associations that coordinate the development efforts of the standard, AP242 will be implemented in the next years. Similarly, KBE frameworks, such as AML, only provide interfaces for the import and export of STEP files that contain purely geometrical data. Nevertheless, since AML is highly customizable, the implementation of an interface that supports other data structures is only a matter of effort.

For an interface that supports a STEP schema, such as AP242, both, the class structure in the form of an EXPRESS schema and the data structure in the form of STEP files, have to be translated into the AML environment. Via the implementation of this interface and the translation of an example file, this thesis elucidates the feasibility and at the same time the challenges of such an approach. The challenges are mainly related to the graphical representation and the mapping of features in AML. Moreover, general challenges that are issued, when non-congruent structures have to be mapped to each other, complicate the implementation.

In summary, STEP AP242 provides the structures to represent knowledge elements that contain *design intent* and therefore also enables the transfer of such knowledge elements between CAD and KBE environments. Still, the data transfer between the two systems is limited by the implementation of their STEP interfaces. Additionally, the complete transfer of data models is only possible, if the data structures in both systems can be mapped to the neutral exchange structures without loss of information. This in turn is only possible, if the vendors commit themselves to only using structures that can be mapped to the neutral exchange file.

The final chapter contains suggestions for potential next steps to further investigate the possibilities of knowledge transfer between KBE and CAD environments including possibilities that exceeded the scope of this work and ones that build on the findings of this work.

7. Future work

The implementation of the translator in this thesis is reasonably simplified, such that the typically very extensive development efforts of a STEP binding are reduced. The main shortcut in each implementation step is the implementation of only those elements that are necessary to translate the example geometry and the disregard of the rest. This section discusses the different steps and their shortcomings to provide starting points for further developments.

First, generating the example STEP file with the EDM from *Jotne* by using query schemas was an effective way to produce a single geometry. However, this is not a very universal approach. In order to exploit the full capacity of EDM, the database should be accessed with one of the bindings. Additionally, the possibilities of a direct binding between EDM and AML should be explored.

Next, the PLY parser that reads the EXPRESS schemas is complete regarding the grammatical interpretation of the EXPRESS structures. The part that generates the AML code however only uses the entity declarations and ignores the rest. Therefore, TYPES, CONSTANTS, RULES and FUNCTIONS are not mapped to AML classes. Hence, a strategy has to be developed on how to implement these declarations into AML. In order to use the TYPES for example, type checking has to be implemented. Together with the RULES, the DERIVED ATTRIBUTES can be implemented.

The STEP file parser is able to read the STEP files and to store the entities in an AML file. While the resulting transfer file can be processed with an AML function, it only supports the population of simple entity types. Therefore, in order to be able to import other files than the one created with the EDM, the implementation of complex entity types is an essential next step. Other functionalities that still have to be implemented are CONSTRAINTS, FUNCTIONS, RULES, DERIVED ATTRIBUTES and more. Instead of using the native AML geometrical object classes to implement the graphical representation, the `:vg1` functions should be used to define a representation of each of the mapped AML classes that contains geometry.

Finally, the converters could be written in C in order to implement their compiled code into AML. This would enable the direct import from within AML, without the need to run external programs.

Bibliography

- AFNet (2015). *AFNeT STEP AP242 Benchmark: Test report for the STEP AP242 Benchmark #1: Short Report* (cited on p. 38).
- Barber, Sharon L. et al. (2010). “Experience in Development of Translators for AP203 Edition 2 Construction History”. In: *Computer-Aided Design and Applications* 7.4, pp. 565–578. ISSN: 1686-4360 (cited on pp. 2, 6, 40).
- Chapman, C. and M. Pinfold (1999). “Design engineering—a need to rethink the solution using knowledge based engineering”. In: *Knowledge-Based Systems* 12.5-6, pp. 257–267. ISSN: 09507051 (cited on p. 2).
- Coronado and Jose (2014). *STEP standard support in Creo: AP 242 implementation in Creo*. URL: http://www.asd-ssg.org/c/document_library/get_file?uuid=f11b5a3a-5594-4f79-90be-384f452ac94f&groupId=11317 (visited on 09/15/2015) (cited on p. 38).
- Feeney, Allison Barnard and Thomas Hedberg (2014). *STEP: Standard for the Exchange of Product model data*. MBE Summit 2014. URL: http://www.nist.gov/el/msid/upload/16_aBarnardFeeney.pdf (visited on 09/15/2015) (cited on p. 33).
- Feeney, Allison Barnard, Simon P. Frechette, and Vijay Srinivasan (2015). “A Portrait of an ISO STEP Tolerancing Standard as an Enabler of Smart Manufacturing Systems”. In: *Journal of Computing and Information Science in Engineering* 15.2, p. 21005. ISSN: 1530-9827 (cited on pp. 31, 32).
- Feldhusen, Jörg et al., eds. (2013). *Pahl/Beitz Konstruktionslehre*. Berlin: Springer Vieweg. ISBN: 978-3-642-29568-3 (cited on p. 1).
- Fischer, Bryan R. (2015). “A Step Up: A Neutral File Format Brings More Information into Play”. In: *Mechanical Engineering-CIME* March (cited on p. 32).
- Fish, Jonathan and Stephen Scrivener (1990). “Amplifying the Mind’s Eye: Sketching and Visual Cognition”. In: *Leonardo* 23.1, p. 117. ISSN: 0024094X (cited on p. 2).
- ISO 10303. 21:2002: Clear text encoding of the exchange structure. In: *ISO 10303 - Product data representation and exchange*. 21:2002 (cited on pp. 25, 27, 59).
- ISO 10303. 203:2005: Configuration controlled 3D designs: Ed.2. In: *ISO 10303 - Product data representation and exchange*. 203:2005 (cited on p. 5).
- ISO 10303. 31:1994: Conformance testing methodology and framework: General concepts. In: *ISO 10303 - Product data representation and exchange*. 31:1994 (cited on pp. 59, 60).
- ISO 10303. 111:2007: Elements for the procedural modelling of solid shapes. In: *ISO 10303 - Product data representation and exchange*. 111:2007 (cited on pp. 31, 51).

- ISO 10303. 242:2014: Managed model based 3D engineering. In: *ISO 10303 - Product data representation and exchange*. 242:2014 (cited on p. 32).
- ISO 10303. 112:2006: Modelling commands for the exchange of procedurally represented 2D CAD models. In: *ISO 10303 - Product data representation and exchange*. 112:2006 (cited on p. 6).
- ISO 10303. 1:1994: Overview and fundamental principles. In: *ISO 10303 - Product data representation and exchange*. 1:1994 (cited on pp. 16, 18).
- ISO 10303. 108:2005: Parameterization and constraints for explicit geometric product models. In: *ISO 10303 - Product data representation and exchange*. 108:2005 (cited on p. 30).
- ISO 10303. 108:2005: Parameterization and constraints for explicit geometric product models: Cor.1. In: *ISO 10303 - Product data representation and exchange*. 108:2005 (cited on p. 30).
- ISO 10303. 55:2005: Procedural and hybrid representation. In: *ISO 10303 - Product data representation and exchange*. 55:2005 (cited on pp. 28, 29).
- ISO 10303. 22:1998: Standard data access interface. In: *ISO 10303 - Product data representation and exchange*. 22:1998 (cited on p. 28).
- ISO 10303. 11:2004: The EXPRESS language reference manual. In: *ISO 10303 - Product data representation and exchange*. 11:2004 (cited on pp. 10, 21, 22, 24, 46–48, 59, 61, 112, 122, 164).
- ISO 10303 (2009). *Whitepaper AP242 ED1* (cited on p. 32).
- ISO TC 184/SC4 N1863 (2005). *Guidelines for the content of application protocols that use application modules* (cited on p. 20).
- Kim, Byung Chul et al. (2011). “A method to exchange procedurally represented 2D CAD model data using ISO 10303 STEP”. In: *Computer-Aided Design* 43.12, pp. 1717–1728. ISSN: 00104485 (cited on p. 6).
- Kim, Junhwan et al. (2008). “Standardized data exchange of CAD models with design intent”. In: *Computer-Aided Design* 40.7, pp. 760–777. ISSN: 00104485 (cited on pp. 5, 6, 12, 13, 31).
- La Rocca, Gianfranco (2012). “Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design”. In: *Advanced Engineering Informatics* 26.2, pp. 159–179. ISSN: 14740346 (cited on pp. 2, 9, 33).
- Loffredo, David (1999). *Fundamentals of STEP Implementation* (cited on pp. 18, 19, 28).
- Lützenberger, Johannes et al. (2012). *Linked Knowledge in Manufacturing, Engineering and Design for Next-Generation Production* (cited on p. 6).
- Owen, Jon (1993). *STEP - An introduction*. Product data engineering. Winchester: Information Geometers. ISBN: 1874728046 (cited on pp. 11, 16).

- Pratt, Michael J. (1997). “Extension of STEP for the Representation of Parametric and Variational Models”. In: *CAD Systems Development*, pp. 237–250 (cited on p. 14).
- Pratt, Michael J. (2001). “Introduction to ISO 10303—the STEP Standard for Product Data Exchange”. In: *Journal of Computing and Information Science in Engineering* 1.1, p. 102. ISSN: 1530-9827 (cited on p. 16).
- Pratt, Michael J. and Junhwan Kim (2006). “Experience in the Exchange of Procedural Shape Models using ISO 10303 (STEP)”. In: (cited on pp. 1, 6, 13–15, 29).
- Prawel, David (2010). *Collaboration & Interoperability Market Report 2010* (cited on pp. 4, 17, 60).
- Reynolds, Richard (2002). *Strategy for Product Data throughout the Life Cycle: Memorandum for the Air Force, Navy, and Army Acquisition* (cited on p. 4).
- SCRA (2006). “STEP APPLICATION HANDBOOK: ISO 10303: VERSION 3”. In: (cited on pp. 19, 24, 31).
- Seo, Tae-Sul et al. (2005). “Sharing CAD models based on feature ontology of commands history”. In: *International Journal of CAD/CAM* Vol 5, No 1 (cited on p. 5).
- STEP Tools Software (2010). *ST-Developer Tools Reference Manual*. Ed. by STEPtools (cited on pp. 19, 20).
- Stiteler, Mike (2004). *CHAPS Program Final Report: Improving affordability through intelligent CAD data exchange* (cited on pp. 1, 3–5, 59).
- TechnoSoft (2010). *AML 5.0B5 Reference Manual* (cited on pp. 34, 50, 51).
- Vettermann, Steven (2015). *ProSTEP iViP Webinar: STEP AP242 - A report from practice* (cited on p. 32).

A. File structure

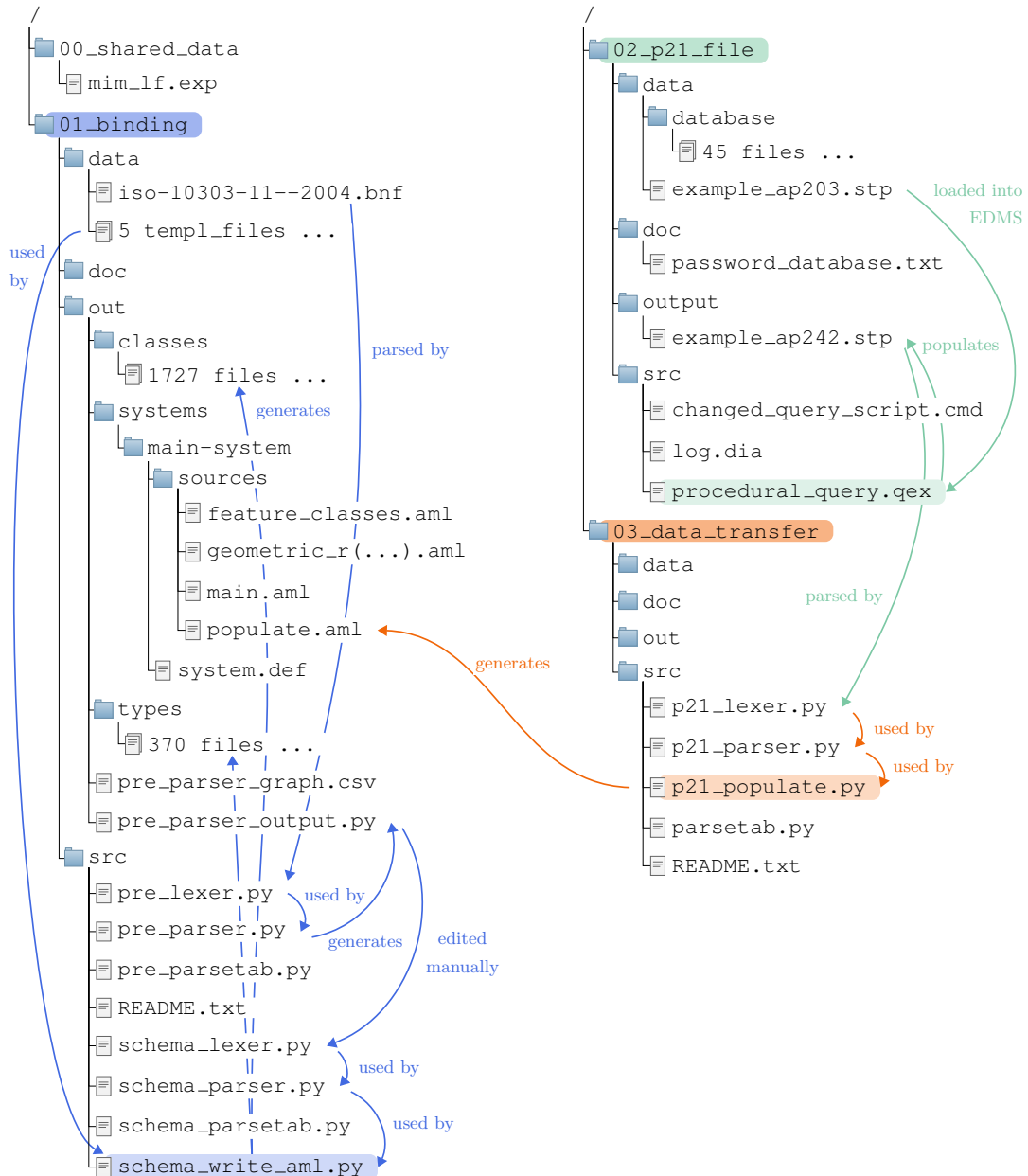
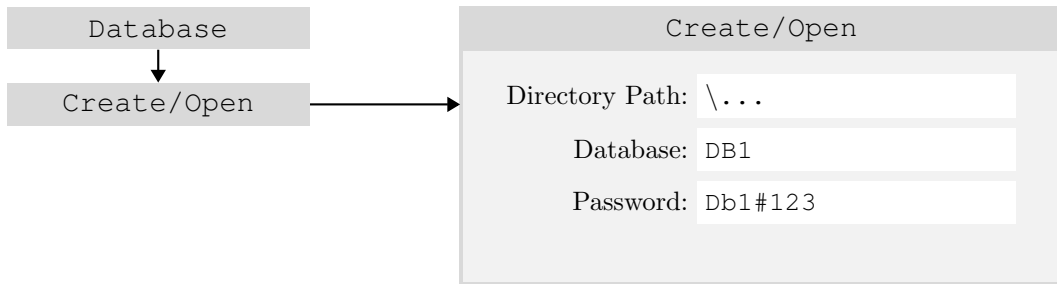


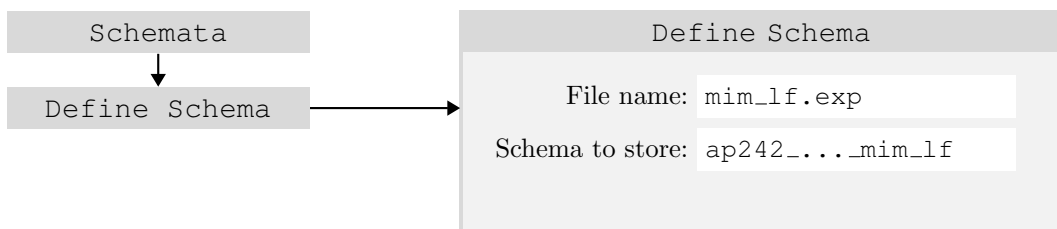
Figure A.1.: File dependencies of **binding**, **p21_file** and **data_transfer**. `schema_write_aml.py` generates AML classes out of a STEP schema, `procedural_query.qex` populates the EDM model with AP242 entities, `p21_populate.py` parses p21 STEP files and generates AML output.

B. EDM commands

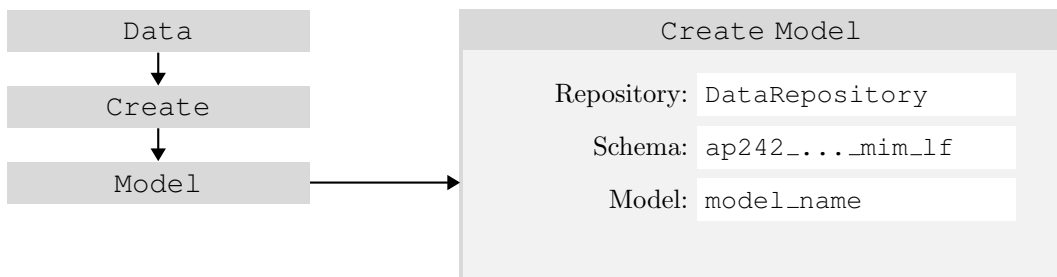
The following EDM commands represent the commands in the EDM GUI, which are described in section 3.2.2.

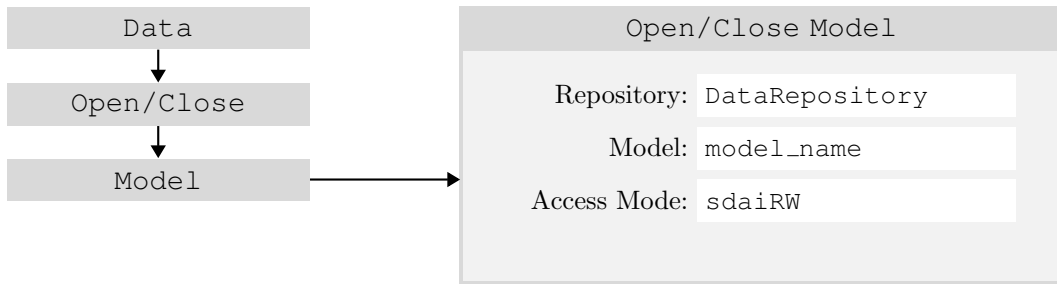


The following command loads an EXPRESS schema into EDM.

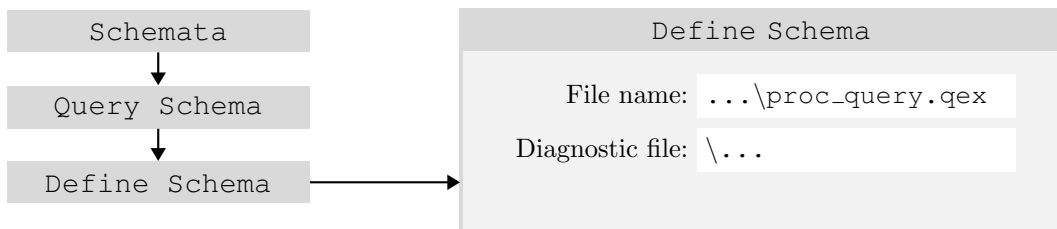


A data model has to be created in order to populate it.

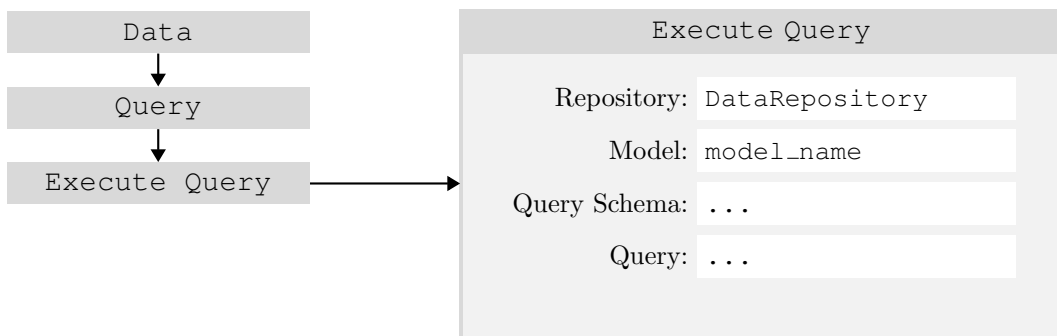




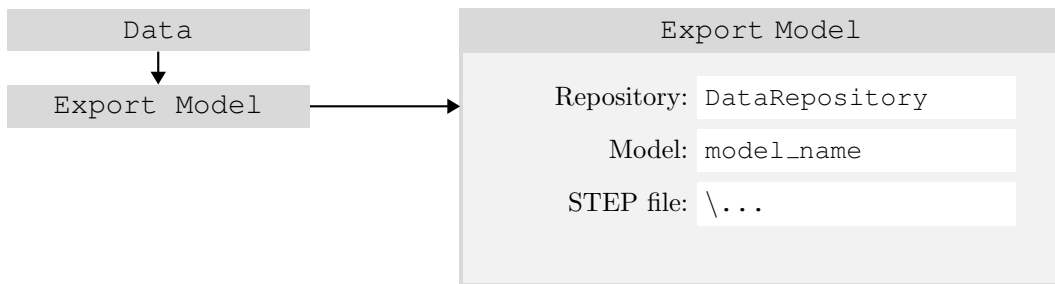
A query schema can be loaded into EDM with this command. In order to define a query schema, the data models have to be closed.



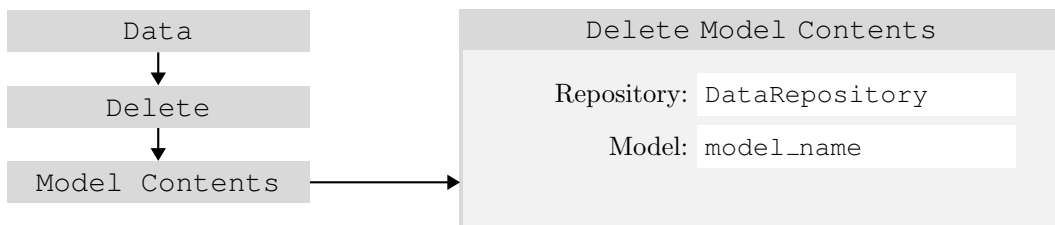
Subsequently, the query schema can be executed on the data model.



The data model is exported as STEP file with the following command.



The following command deletes the content of a data model.



C. Code listings

C.1. EDM – procedural query

```
2 QUERY_SCHEMA AP242_entity_output FOR
   ↪ AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF;
3
4 GLOBAL
5   DECLARE inst INSTANCE OF
   ↪ AP242_MANAGED_MODEL_BASED_3D_ENGINEERING_MIM_LF;
6 END_GLOBAL;
18
19 FUNCTION fct_normalize (vec : LIST[3:3] OF REAL ) : LIST[3:3] OF REAL;
20 LOCAL
21   vec_norm : LIST[3:3] OF REAL;
22   vec_abs : positive_length_measure;
23   vec_abs_quad : positive_length_measure := 0;
24 END_LOCAL;
25   REPEAT i:= LOINDEX(vec) TO HIINDEX(vec);
26     vec_abs_quad += vec[i] * vec[i];
27   END_REPEAT;
28   vec_abs := SQRT(vec_abs_quad);
29   REPEAT i:= LOINDEX(vec) TO HIINDEX(vec);
30     vec_norm[i] := vec[i] / vec_abs;
31   END_REPEAT;
32   return(vec_norm);
33 END_FUNCTION;
34
35 FUNCTION fct_edge_curve (pnt_start, pnt_end : CARTESIAN_POINT; suffix :
   ↪ STRING) : EDGE_CURVE;
36
37 LOCAL
38   --cp_start, cp_end : CARTESIAN_POINT;
39   d_1 : DIRECTION;
40   v_1 : VECTOR;
41   l_1 : LINE;
42   vp_start, vp_end : VERTEX_POINT;
43   ec_1 : EDGE_CURVE;
44
45 END_LOCAL;
46
```



```
47 NEW PERSISTENT d_1;
48 d_1.name := 'Edge Curve Direction ' + suffix;
49 REPEAT i:= LOINDEX(pnt_end.coordinates) TO HIINDEX(pnt_end.
    ↪ coordinates);
50     d_1.direction_ratios[i] := pnt_end.coordinates[i] - pnt_start.
    ↪ coordinates[i];
51 END_REPEAT;
52 d_1.direction_ratios := fct_normalize(d_1.direction_ratios);
53
54 NEW PERSISTENT v_1;
55 v_1.name := 'Edge Curve Vector ' + suffix;
56 v_1.orientation := d_1;
57 v_1.magnitude := 1;
58
59 NEW PERSISTENT l_1;
60 l_1.name := 'Line ' + suffix;
61 l_1.pnt := pnt_start;
62 l_1.dir := v_1;
63
64 NEW PERSISTENT vp_start;
65 vp_start.name := 'Vertex Point Start ' + suffix;
66 vp_start.vertex_geometry := pnt_start;
67
68 NEW PERSISTENT vp_end;
69 vp_end.name := 'Vertex Point End ' + suffix;
70 vp_end.vertex_geometry := pnt_end;
71
72 NEW PERSISTENT ec_1;
73 ec_1.name := 'Edge Curve ' + suffix;
74 ec_1.edge_start := vp_start;
75 ec_1.edge_end := vp_end;
76 ec_1.edge_geometry := l_1;
77 ec_1.same_sense := TRUE;
78
79 RETURN(ec_1);
80
81 END_FUNCTION; -- fct_edge_curve
85 QUERY_FUNCTION procedural_repr : STRING;
86
87 LOCAL
88
```

```

89  ----- edges -----
90  ec_1, ec_2, ec_3, ec_4 : EDGE_CURVE;
91  cp_1, cp_2, cp_3, cp_4 : CARTESIAN_POINT;
92
93  ----- rectangle -----
94  oe_1, oe_2, oe_3, oe_4 : ORIENTED_EDGE;
95  el_1 : EDGE_LOOP;
96  fb_1 : FACE_BOUND;
97  d_1, d_2 : DIRECTION;
98  cp_plane : CARTESIAN_POINT;
99  ap3_1 : AXIS2_PLACEMENT_3D;
100 p_1 : PLANE;
101 fs_1 : FACE_SURFACE;
102
103 ----- extrude -----
104 d_extr : DIRECTION;
105 bvp_extr_depth : BOUND_VARIATIONAL_PARAMETER;
106 bvp_current : positive_length_measure := 40;
107 fri_extr : FINITE_REAL_INTERVAL;
108 iar_extr : INSTANCE_ATTRIBUTE_REFERENCE; -- bvp_extr_depth
109 bpe_extr : BOUND_PARAMETER_ENVIRONMENT;
110 efs_1 : EXTRUDED_FACE_SOLID;
111
112 ----- hole -----
113 iar_hole : INSTANCE_ATTRIBUTE_REFERENCE; -- bvp_extr_depth
114 bpe_hole : BOUND_PARAMETER_ENVIRONMENT;
115 iar_cp : INSTANCE_ATTRIBUTE_REFERENCE; -- bvp_extr_depth
116 bpe_cp : BOUND_PARAMETER_ENVIRONMENT;
117 d_hole, d_hole_ref : DIRECTION;
118 cp_hole : CARTESIAN_POINT;
119 ap3_hole : AXIS2_PLACEMENT_3D;
120 swfbrh : SOLID_WITH_FLAT_BOTTOM_ROUND_HOLE;
121
122 ----- chamfer -----
123 iar_chamfer_cp1, iar_chamfer_cp2 : INSTANCE_ATTRIBUTE_REFERENCE; --
    ↪ bvp_extr_depth
124 bpe_chamfer_cp1, bpe_chamfer_cp2 : BOUND_PARAMETER_ENVIRONMENT;
125 cp_edge_1, cp_edge_2 : CARTESIAN_POINT;
126 ec_chamfer : EDGE_CURVE;
127 swsoc : SOLID_WITH_SINGLE_OFFSET_CHAMFER;
128

```

```
129  ----- representation -----
130  psrs : PROCEDURAL_SHAPE_REPRESENTATION_SEQUENCE;
131  psr  : PROCEDURAL_SHAPE_REPRESENTATION;
132  sdr  : SHAPE_DEFINITION_REPRESENTATION;
133
134  END_LOCAL;
135  ----- edges -----
136  NEW PERSISTENT cp_1;
137  cp_1.name := 'Cartesian Point 1';
138  cp_1.coordinates := [0,0,0];
139
140  NEW PERSISTENT cp_2;
141  cp_2.name := 'Cartesian Point 2';
142  cp_2.coordinates := [50,0,0];
143
144  NEW PERSISTENT cp_3;
145  cp_3.name := 'Cartesian Point 3';
146  cp_3.coordinates := [50,50,0];
147
148  NEW PERSISTENT cp_4;
149  cp_4.name := 'Cartesian Point 4';
150  cp_4.coordinates := [0,50,0];
151
152  ec_1 := fct_edge_curve(cp_1, cp_2, '(Edge1)');
153  ec_2 := fct_edge_curve(cp_2, cp_3, '(Edge2)');
154  ec_3 := fct_edge_curve(cp_3, cp_4, '(Edge3)');
155  ec_4 := fct_edge_curve(cp_4, cp_1, '(Edge4)');
156
157  ----- rectangle -----
158  NEW PERSISTENT oe_1;
159  NEW PERSISTENT oe_2;
160  NEW PERSISTENT oe_3;
161  NEW PERSISTENT oe_4;
162  oe_1.name := 'Oriented Edge 1';
163  oe_2.name := 'Oriented Edge 2';
164  oe_3.name := 'Oriented Edge 3';
165  oe_4.name := 'Oriented Edge 4';
166  oe_1.edge_element := ec_1;
167  oe_2.edge_element := ec_2;
168  oe_3.edge_element := ec_3;
169  oe_4.edge_element := ec_4;
```

```
170 oe_1.orientation := FALSE;
171 oe_2.orientation := FALSE;
172 oe_3.orientation := FALSE;
173 oe_4.orientation := FALSE;
174
175 NEW PERSISTENT el_1;
176 el_1.name := 'Edge Loop 1';
177 el_1.edge_list := [oe_1, oe_2, oe_3, oe_4];
178
179 NEW PERSISTENT fb_1;
180 fb_1.name := 'Face Bound 1';
181 fb_1.bound := el_1;
182 fb_1.orientation := FALSE;
183
184 NEW PERSISTENT d_1;
185 NEW PERSISTENT d_2;
186 d_1.name := 'Plane Perpendicular Direction';
187 d_2.name := 'Plane Reference Direction';
188 d_1.direction_ratios := [0, 0, 1]; -- perpendicular axis to plane
189 d_2.direction_ratios := [1, 0, 0]; -- optional second axis
190
191 NEW PERSISTENT cp_plane;
192 cp_plane.name := 'Plane Reference Point';
193 cp_plane.coordinates := [0, 0, 0];
194
195 NEW PERSISTENT ap3_1;
196 ap3_1.name := 'Plane Placement';
197 ap3_1.location := cp_plane;
198 ap3_1.axis := d_1;
199 ap3_1.ref_direction := d_2;
200
201 NEW PERSISTENT p_1;
202 p_1.name := 'Base Plane';
203 p_1.position := ap3_1;
204
205 NEW PERSISTENT fs_1;
206 fs_1.name := 'Base Face Surface';
207 fs_1.bounds := [fb_1];
208 fs_1.face_geometry := p_1;
209 fs_1.same_sense := FALSE;
210
```

```
211 ----- extrude -----
212 NEW PERSISTENT d_extr;
213 d_extr.name := 'Extrusion Direction';
214 d_extr.direction_ratios := [0, 0, 1];
215
216 NEW PERSISTENT fri_extr; -- FINITE_REAL_INTERVAL
217 fri_extr.min := 0;
218 fri_extr.min_closure := CLOSED;
219 fri_extr.max := 50;
220 fri_extr.max_closure := CLOSED;
221
222 NEW PERSISTENT bvp_extr_depth; -- BOUND_VARIATIONAL_PARAMETER
223 bvp_extr_depth.name := 'extrusion_depth';
224 bvp_extr_depth.values_space := fri_extr;
225 bvp_extr_depth.parameter_description := 'Defines the Height of the
    ↪ Cube';
226 -- bvp_extr_depth.name := 'Extrusion Depth'; maths_variable.name
    ↪ changed manually in EDMS
227 -- bvp_extr_depth.parameter_current_value := 23;
228
229 NEW PERSISTENT efs_1; -- EXTRUDED_FACE_SOLID
230 efs_1.name := 'Extruded Cube';
231 efs_1.swept_face := fs_1;
232 efs_1.extruded_direction := d_extr;
233 efs_1.depth := bvp_current; -- parameter_current_value;
234
235 NEW PERSISTENT iar_extr; -- INSTANCE_ATTRIBUTE_REFERENCE
236 iar_extr.attribute_name := 'GEOMETRIC_MODEL_SCHEMA.
    ↪ EXTRUDED_FACE_SOLID.DEPTH';
237 iar_extr.owning_instance := efs_1;
238
239 NEW PERSISTENT bpe_extr; -- BOUND_PARAMETER_ENVIRONMENT
240 bpe_extr.syntactic_representation := bvp_extr_depth;
241 bpe_extr.semantics := iar_extr;
242
243 ----- hole -----
244 NEW PERSISTENT cp_hole; -- CARTESIAN_POINT
245 cp_hole.name := 'Hole Center Point';
246 cp_hole.coordinates := [25, 25, bvp_current];
247
248 NEW PERSISTENT iar_hole; -- INSTANCE_ATTRIBUTE_REFERENCE
```

```
249   iar_hole.attribute_name := 'GEOMETRIC_MODEL_SCHEMA.CARTESIAN_POINT.  
      ↪ COORDINATES[2]';  
250   iar_hole.owning_instance := cp_hole;  
251  
252   NEW PERSISTENT bpe_hole; -- BOUND_PARAMETER_ENVIRONMENT  
253   bpe_hole.syntactic_representation := bvp_extr_depth;  
254   bpe_hole.semantics := iar_hole;  
255  
256   NEW PERSISTENT d_hole; -- DIRECTION  
257   d_hole.name := 'Hole Direction';  
258   d_hole.direction_ratios := [0, 0, -1];  
259  
260   NEW PERSISTENT d_hole_ref; -- DIRECTION  
261   d_hole_ref.name := 'Hole Reference Direction';  
262   d_hole_ref.direction_ratios := [1, 0, 0];  
263  
264   NEW PERSISTENT ap3_hole; -- AXIS2_PLACEMENT_3D  
265   ap3_hole.name := 'Hole Placement';  
266   ap3_hole.location := cp_hole;  
267   ap3_hole.axis := d_hole;  
268   ap3_hole.ref_direction := d_hole_ref;  
269  
270   NEW PERSISTENT swfbrh; -- SOLID_WITH_FLAT_BOTTOM_ROUND_HOLE  
271   swfbrh.name := 'Through Hole';  
272   swfbrh.rationale := '';  
273   swfbrh.base_solid := efs_1;  
274   swfbrh.placing := ap3_hole;  
275   -- derived: swfbrh.depth  
276   swfbrh.segments := 1; -- for stepped holes  
277   swfbrh.segment_radii := [10];  
278   swfbrh.segment_depths := [bvp_current];  
279   swfbrh.fillet_radius := 0;  
280  
281   NEW PERSISTENT iar_cp; -- INSTANCE_ATTRIBUTE_REFERENCE  
282   iar_cp.attribute_name := 'GEOMETRIC_MODEL_SCHEMA.  
      ↪ SOLID_WITH_FLAT_BOTTOM_ROUND_HOLE.SEGMENT_DEPTHS[0]';  
283   iar_cp.owning_instance := swfbrh;  
284  
285   NEW PERSISTENT bpe_cp; -- BOUND_PARAMETER_ENVIRONMENT  
286   bpe_cp.syntactic_representation := bvp_extr_depth;  
287   bpe_cp.semantics := iar_cp;
```

```
288
289 ----- chamfer -----
290 NEW PERSISTENT cp_edge_1;
291 cp_edge_1.name := 'Start Point Chamfer';
292 cp_edge_1.coordinates := [50,0,bvp_current];
293
294 NEW PERSISTENT cp_edge_2;
295 cp_edge_2.name := 'End Point Chamfer';
296 cp_edge_2.coordinates := [50,50,bvp_current];
297
298 ec_chamfer := fct_edge_curve(cp_edge_1, cp_edge_2, '(Chamfer)');
299
300 NEW PERSISTENT iar_chamfer_cp1; -- INSTANCE_ATTRIBUTE_REFERENCE
301 iar_chamfer_cp1.attribute_name := 'GEOMETRIC_MODEL_SCHEMA.
    ↪ CARTESIAN_POINT.COORDINATES[2]';
302 iar_chamfer_cp1.owning_instance := cp_edge_1;
303
304 NEW PERSISTENT bpe_chamfer_cp1; -- BOUND_PARAMETER_ENVIRONMENT
305 bpe_chamfer_cp1.syntactic_representation := bvp_extr_depth;
306 bpe_chamfer_cp1.semantics := iar_chamfer_cp1;
307
308 NEW PERSISTENT iar_chamfer_cp2; -- INSTANCE_ATTRIBUTE_REFERENCE
309 iar_chamfer_cp2.attribute_name := 'GEOMETRIC_MODEL_SCHEMA.
    ↪ CARTESIAN_POINT.COORDINATES[2]';
310 iar_chamfer_cp2.owning_instance := cp_edge_2;
311
312 NEW PERSISTENT bpe_chamfer_cp2; -- BOUND_PARAMETER_ENVIRONMENT
313 bpe_chamfer_cp2.syntactic_representation := bvp_extr_depth;
314 bpe_chamfer_cp2.semantics := iar_chamfer_cp2;
315
316 NEW PERSISTENT swsoc;
317 swsoc.name := 'Chamfer';
318 -- swsoc.rationale := '';
319 swsoc.base_solid := swfbrh;
320 swsoc.blended_edges := [ec_chamfer];
321 swsoc.offset_distance := 5;
322
323 ----- representation -----
324
325 NEW PERSISTENT psrs;
326 -- psrs.name := 'Sequence';
```

```
327 psrs.elements := [efs_1, swfbrh, swsoc];
328 -- psrs.suppressed_items := ;
329 -- psrs.rationale := 'rationale_text';
330
331 NEW PERSISTENT psr;
332 -- psr.name := '';
333 psr.items := [psrs];
334 -- psr.representation_items := ; -- @TODO
335
336 NEW PERSISTENT sdr;
337 -- sdr.definition := ; -- @TODO
338 sdr.used_representation := psr;
339
340 RETURN('SHAPE_DEFINITION_REPRESENTATION complete and populated');
341
342 END_QUERY_FUNCTION;
373 END_QUERY_SCHEMA;
```

Listing C.1: EDM query schema that populates the data model. (File: procedural_query.qex)



C.2. AML

C.2.1. Main

```
1 ;; listing_start root_class "Definition of the root-class for the import."  
2 (in-package :aml)  
3  
4 (define-class ap242-import  
5   :inherit-from (  
6     object  
7   )  
8   :properties (  
9   )  
10  :subobjects (  
11  )  
12 )  
13  
14 (define-method property-names-to-inspect ap242-import ()  
15   (remove 'added-objects (properties (the))))  
16 )  
17  
18 ;; listing_end root_class  
19  
20 ;; listing_start load_class "Function to load classes on demand."  
21 (defun load_class (class_name)  
22   (if (subclassp (make-symbol class_name) 'object)  
23     "Class already loaded"  
24     (load (logical-path :class-path (concatenate class_name ".aml"))))
```



```
25     )
26   )
27 ;; listing_end load_class
28
29 ;;; (defvar entity_hash (make-hash-table))
30
31 ;; listing_start populate "Function to walk iteratively through the hierarchy and populate the entities."
32 (defun populate (ref type_name params parent)
33   (load_class (concatenate "ap242_" type_name))
34   (let (
35     (name (make-symbol (concatenate type_name (string ref))))
36     (class (make-symbol (concatenate "ap242_" type_name)))
37     (instance (add-object parent name class))
38     (index_list (the index (:from instance)))
39   )
40   (progn
41     (loop
42       for attribute in index_list
43       for count from 1 do
44         (progn
45           (let (
46             (attr attribute)
47           )
48             (progn
49               (change-value (the-list (list attr) :from instance) (nth (- count 1) (eval params)))));NEW
50               ;; Check if element is list
51               (if (and (listp (nth (- count 1) (eval params)))
52                       (nth (- count 1) (eval params))))); excludes nil
```

```

53 ;;If
54 (progn
55   (let (
56     (elem_list (nth (- count 1) (eval params)))
57     )
58   (progn
59     ;;(print "nested list...")
60     (loop for elem in elem_list do
61
62       ;; Check if element is reference (#...)
63       ;;***** Could be solved without repetiton with recursion
64       ↪ ...
65       (progn;;(print elem)
66         (if (symbolp elem)
67
68           ;;If
69           (progn
70             (if (eq (subseq (symbol-name elem) 0 1) "#")
71               (progn
72                 (populate elem;; ref
73                   (nth 0 (gethash elem populate_hash));; type-name
74                   (nth 1 (gethash elem populate_hash));; (child) params
75                   ;;(the-list (list name) :from parent);; parent
76                   instance;; parent
77                 )
78               )
79             )

```





```
80         )
81     )
82     ;;*****
83     )
84     )
85     )
86     )
87 ;;Else
88 (progn
89     ;; Check if element is reference (#...)
90     ;;*****
91     (if (and (symbolp (nth (- count 1) (eval params)))
92             (nth (- count 1) (eval params));; excludes nil
93             (not (eq (nth (- count 1) (eval params)) t));; excludes t
94     ;;If
95     (progn
96         (if (eq (subseq (symbol-name (nth (- count 1) (eval params))) 0 1) "#")
97             ;;If (no Else)
98             (progn
99                 ;;(print (nth 1 (gethash (nth (- count 1) (eval params)) populate_hash));;
100                    ↪ -> child params
101                 (populate (nth (- count 1) (eval params));; ref
102                    (nth 0 (gethash (nth (- count 1) (eval params)) populate_hash));;
103                    ↪ type-name
104                    (nth 1 (gethash (nth (- count 1) (eval params)) populate_hash));; (
105                    ↪ child) params
106                ;;(the-list (list name) :from parent);; parent
107                instance;; parent
```



```
105         )
106     (change-value (the-list (list attr) :from instance)
107         (make-symbol (concatenate
108             (string (nth 0 (gethash (nth (- count 1) (eval
109                 ↪ params)) populate_hash)))
110             (string (nth (- count 1) (eval params)))
111         ))
112     )
113 )
114 )
115 )
116 ;;*****
117
118 )
119 )
120 )
121 )
122 )
123 );; loop
124 )
125 )
126 )
127 ;; listing_end populate
128
129 ;; listing_start populate_root "Initial call of the populate function with the root entities."
130 (defun populate_root (root_entity)
131     (populate root_entity
```



```
132         (first (gethash root_entity populate_hash))
133         (nth 1 (gethash root_entity populate_hash))
134         (the))
135     )
136 ;; listing_end populate_root
137
138 ;; listing_start split-at-point "{Function to split a reference path into schema, entity, property and
    ↪ index.}"
139 ;; example input: GEOMETRIC_MODEL_SCHEMA.CARTESIAN_POINT.COORDINATES[2]
140 (defun split-reference (string)
141   (setf sub_list '())
142   (setf sub_string string)
143   (setf rest_string string)
144   (loop for point_sep from 1 to (count #\. string)
145     do (let (
146           (substr (subseq rest_string 0 (position #\. rest_string)))
147           (reststr (subseq rest_string (+ (position #\. rest_string) 1) (length rest_string)))
148         )
149       (progn
150         (setf sub_string substr)
151         (setf rest_string reststr)
152         (setf sub_list (append sub_list (list sub_string)))
153       )
154     )
155   )
156 ;; Check for index -- e.g. coordinates[1]
157 (if (= 1 (count #\[ rest_string))
158     (progn
```



```
159     (print rest_string)
160     (setf sub_string (subseq rest_string 0 (position #\[ rest_string)))
161     (setf rest_string (subseq rest_string (+ (position #\[ rest_string) 1) (- (length rest_string)
    ↪ 1)))
162     (setf sub_list (append sub_list (list sub_string)))
163     )
164     )
165     (setf sub_list (append sub_list (list rest_string)))
166     (print sub_list)
167     )
168 ;; listing_end split-at-point
169
170 ;; listing_start substitute-nth "Function to substitute nth element of a list."
171 (defun substitute-nth (val n list)
172   (loop for i from 0 for j in list collect (if (= i n) val j)))
173 ;; listing_end substitute-nth
174
175 (defun replace_substr (new old string)
176   (let (
177     (pos (search old string))
178     (len (length old))
179     (prefix (subseq string 0 pos))
180     (suffix (subseq string (+ pos len) (length string)))
181     )
182     (progn
183       ;; (print prefix)
184       ;; (print suffix)
185       (concatenate prefix new suffix)
```



```
186     )
187   )
188 )
189
190 (defun add_list_string (string)
191   (let (
192     (prefix "(list ")
193     (rest (subseq string 1 (length string)))
194   )
195   (progn
196     ;;(print prefix)
197     ;;(print rest)
198     (concatenate prefix rest)
199   )
200 )
201 )
202
203 (defvar new_formula 'nil)
204
205 (defun add-params-to-root ()
206
207   (loop for param in (children (the) :class 'ap242_bound_parameter_environment) do
208     (let (
209       (owning_inst (the-list '(owning_inst) :from param))
210       (attr_ref (children param :class 'ap242_instance_attribute_reference))
211       (param_target (make-symbol (first (select-object
212                                     :test '(object-instance (the) 'parameter_target)
213                                     :from param ;(the-list (list param) :from (the))
```




```
214         :eval '(the parameter_target)))
215     )
216 (param_target_index (first (select-object
217     :test '(object-instance (the) 'parameter_target_index)
218     :from param ;(the-list (list param) :from (the))
219     :eval '(the parameter_target_index)))
220 )
221 (param_name (make-symbol (first (select-object
222     :test '(object-instance (the) 'parameter_description)
223     :from param ;(the-list (list param) :from (the))
224     :eval '(the name))))
225 )
226 (param_value (if param_target_index
227     (nth (read-from-string param_target_index)
228         (the-list (list owning_inst param_target) :from (first attr_ref)))
229     (the-list (list owning_inst param_target) :from (first attr_ref)))
230 )
231 )
232 )
233
234 (progn
235     ;(print (the cartesian_point#75 (:from attr_ref)))
236     (print owning_inst)
237     (print (first attr_ref))
238     (print (type-of (first attr_ref)))
239     (print param_name)
240     (print param_target)
241     (print param_target_index)
```



```
242 (print param_value)
243
244 (add-property (the ap242-import) param_name param_value)
245 (if param_target_index
246     ;;(list 25.0 25.0
247     ;;(the extrusion_depth (:from (the ap242-import))))
248     (setf new_formula
249         (read-from-string
250         (add_list_string
251         (replace_substr (string param_name)
252             "xxx"
253             (format nil "~S"
254                 (substitute-nth '(the xxx (:from (the ap242-import)))
255                     (read-from-string param_target_index)
256                     (the-list (list owning_inst param_target) :from (
257                         ↪ first attr_ref))
258                     )))))
259     )
260     (setf new_formula (read-from-string
261         (replace_substr (string param_name)
262             "xxx"
263             (format nil "~S"
264                 '(the xxx (:from (the ap242-import)))
265                 )))
266     )
267 )
268
```



```
269 ;(if param_target_index
270     (loop for node in (select-object :from (the)) do
271         (progn
272             (if (equal (object-name node) owning_inst)
273                 (progn
274                     ;;(print "HEUREKA!")
275                     ;;(print (equal (object-name node) owning_inst))
276                     (change-formula (the-list (list param_target) :from node) new_formula)
277                 )
278             )
279         )
280     )
281 )
282 )
283 )
284 )
285 )
286
287
288
289 ;;(remove-object-from-display (current-object))
290 (delete-all-models t)
291 (create-model 'ap242-import)
292 (select-model 'ap242-import)
293 ;;(print (first root_entities))
294
295
296 (loop for root in root_entities do
```



```
297 (progn
298   (populate_root root)
299   )
300 )
301
302 (add-params-to-root)
303
304 ;;(populate_root (first root_entities))
305
306
307 (draw (the))
308 (shade (the) 'facet)
309 (add-light :name 'light1 :color 'white :x 0.2 :y 0.6 :z 0.4)
310 (select-model 'name-generator)
311 ;;(render 'boundaryshaded)
312 ;;(remove-object-from-display (the box-object))
313 (regen)
```

Listing C.2: AML function that populates the data model and other helper functions. (File: main.aml)

C.2.2. Geometric representation classes

```
1 (in-package :aml)
2
3 (define-class aml_point_object
4   :inherit-from (
5     point-object
6   )
7   :properties (
```



```
8      ;; coordinates already defined in cartesian-point
9      coordinates (nth 0 (remove-duplicates
10                    (select-object
11                      :test '(object-instance (the) 'coordinates)
12                      :from (the superior)
13                      :eval '(the coordinates))))
14      )
15      :subobjects (
16      )
17      )
18
19      (define-class aml_line_object
20      :inherit-from (
21                    line-object
22                    )
23      :properties (
24                    point1 (nth 0 (remove-duplicates
25                                  (select-object
26                                    :test '(object-instance (the) 'coordinates)
27                                    :from (the superior)
28                                    :eval '(the coordinates))))
29                    point2 (nth 1 (remove-duplicates
30                                  (select-object
31                                    :test '(object-instance (the) 'coordinates)
32                                    :from (the superior)
33                                    :eval '(the coordinates))))
34                    )
35      :subobjects (
```



```
36         )
37     )
38
39 (define-class aml_sewn_object
40     :inherit-from (
41         sewn-object
42     )
43     :properties (
44         object-list (remove-duplicates (select-object
45                                     :class 'bounded-object
46                                     :from (the superior)))
47 ;;         object-list (remove-duplicates (select-object
48 ;;                                     :class 'line-object
49 ;;                                     :from (the superior)))
50     )
51     :subobjects (
52     )
53 )
54
55 (define-class aml_bounded_object
56     :inherit-from (
57         bounded-object
58     )
59     :properties (
60         object-list (remove-duplicates (select-object
61                                     :class 'line-object
62                                     :from (the superior)))
63         dimension (default 2)
```



```
64         )
65     :subobjects (
66         )
67 )
68
69
70 (define-class aml_extrusion_object
71     :inherit-from (
72         extrusion-object
73         tagging-object
74     )
75     :properties (
76         tag-dimensions '(2)
77         sub-geoms-indices-list '((0 1 2 3))
78         swept-object (first (remove-duplicates (select-object
79                                     :class 'bounded-object
80                                     :from (the superior))))
81         vector (the direction_ratios (:from (nth 0 (select-object
82                                     :class 'ap242_direction
83                                     :from (the superior)
84                                     :test '(equal (object-name (the)
85                                             ↳ superior extruded_direction))))))
85         distance ^depth
86         capped? t
87     )
88     :subobjects (
89     )
90 )
```



```
91
92 (define-class aml_coordinate_system_class
93   :inherit-from (
94     coordinate-system-class
95   )
96   :properties (
97     origin (nth 0 (remove-duplicates
98       (select-object
99         :test '(object-instance (the) 'coordinates)
100        :from (the superior)
101        :eval '(the coordinates))))
102     vector-i (nth 0 (remove-duplicates
103       (select-object
104         :test '(object-instance (the) 'direction_ratios)
105         :from (the superior)
106         :eval '(the direction_ratios))))
107     vector-j (nth 1 (remove-duplicates
108       (select-object
109         :test '(object-instance (the) 'direction_ratios)
110         :from (the superior)
111         :eval '(the direction_ratios))))
112   )
113   :subobjects (
114   )
115 )
116 )
```

Listing C.3: AML geometric mapping classes. (File: geometric_representation_classes.aml)



C.2.3. Feature mapping classes

```
1 (in-package :aml)
2
3 (define-class xaml_parameter
4   :inherit-from (
5     object
6   )
7   :properties (
8     owning_inst (nth 0 (remove-duplicates
9       (select-object
10        :test '(object-instance (the) 'owning_instance)
11        :from (the superior)
12        :eval '(the owning_instance))))
13     parameter_data (find_parameter_location (the superior))
14     parameter_name (first ^parameter_data)
15     parameter_target (nth 1 ^parameter_data)
16     parameter_target_index (nth 2 ^parameter_data)
17   )
18   :subobjects (
19   )
20 )
21 )
22
23
24
25 (define-method find_parameter_location xaml_parameter ()
26   (let
```



```
27 (
28   (pos 'nil)
29   (name 'nil)
30   (attr 'nil)
31   (index 'nil)
32   (hierarchy_list
33     (split-reference
34       (nth 0
35         (remove-duplicates (select-object
36                             :test '(object-instance (the) 'attribute_name)
37                             :from (the)
38                             :eval '(the attribute_name)))
39       )
40     )
41   )
42 )
43 (progn
44   (setq pos
45     (position
46       (string-upcase
47         (subseq (string (the owning_inst)) 0 (position #\# (string (the owning_inst)))))
48       hierarchy_list
49     )
50   )
51
52   (setq name (subseq (string (the owning_inst)) 0 (position #\# (string (the owning_inst)))))
53   (setq attr (nth (+ pos 1) hierarchy_list))
54   (setq index (nth (+ pos 2) hierarchy_list))
```



```
55
56     )
57
58     (list name attr index)
59     )
60
61 )
62
63
64 (define-method assign_variable xaml_parameter ()
65   (let
66     (
67       (instance (the superior owning_inst))
68       ;;(first (children (the superior) :test '(equal (object-name (the)) (the superior owning_inst))))
69     )
70
71     (print instance)
72     (print ^parameter_name)
73     ;;(add-property (instance) ^parameter_name '( (the table-width) (the tabledepth))
74
75   )
76 )
77
78 (define-class xaml_flat_bottom_round_hole_feature
79   :inherit-from (
80     difference-object
81   )
82   :properties (
```



```
83     object-list (list
84         (first (children (the superior)
85             :test '(equal (object-name (the)) (the superior base_solid))
86                 ↪ ))
87         (the hole))
88 :subobjects (;; @TODO: multiple holes for stepped holes (multiple segments)
89     (hole :class 'cylinder-object
90         height (first ^^segment_depths)
91         diameter (first ^^segment_radii)
92         solid? t
93         display? nil
94         reference-coordinate-system (nth 0 (children (the superior superior)
95             :class 'coordinate-system-class))
96         orientation (list
97             (translate (list 0.0 0.0 (half ^height)))
98             (rotate 90 '(0 1 0)))
99         )
100     )
101 )
102
103 (define-class xaml_single_offset_feature
104     :inherit-from (
105         difference-object
106         )
107     :properties (
108
109         object-list (list
```



```
110         (first (children (the superior)
111                 :test '(equal (object-name (the)) (the superior base_solid))
112                 ↪ ))
113         ;; (the box1)
114         (the chamfer))
115
116     offset ^offset_distance
117
118     edge (the-list (list
119                 (concatenate (first (gethash (first (the superior blended_edges))
120                 ↪ populate_hash))
121                 (string (first (the superior blended_edges)))))
122         :from (the superior))
123
124     d1 '(-5.0 0.0 0.0);; @TODO: automate
125     d2 '(0.0 0.0 -5.0);; @TODO: automate
126
127     (l_1 :class 'line-object
128         point1 (vertex-of-object (first (children (the superior superior) :class 'line-object)
129         ↪ ) 1)
130         point2 (add-vectors ^point1 ^^d1)
131         )
132     (l_2 :class 'line-object
133         point1 (vertex-of-object (first (children (the superior superior) :class 'line-object)
134         ↪ ) 1)
135         point2 (add-vectors ^point1 ^^d2)
136         )
137     (l_dia :class 'line-object
```



```
134         point1 (the superior superior l_1 point2)
135         point2 (the superior superior l_2 point2)
136     )
137 )
138 :subobjects (
139     (profile :class 'bounded-object
140         object-list (list
141             ^^l_1
142             ^^l_2
143             ^^l_dia
144         )
145         dimension 2
146     )
147
148     (chamfer :class 'tangential-sweep-object
149 ;;;         path-object (nth 0 (children (the superior superior) :class 'line-object))
150 swept-object ^^profile
151     (path-object :class 'line-object;; @TODO: Sweep along line-object only works when
152         ↪ point1 = (0 0 0) !?!
153         point1 '(0 0 0)
154         point2 (subtract-vectors (the point1 (:from (nth 0 (children (the
155             ↪ superior superior superior) :class 'line-object))))
156             (the point2 (:from (nth 0 (children (the
157                 ↪ superior superior superior) :class 'line-
158                 ↪ object))))
159             );; @TODO: Direction?
160     )
161 )
```



```
158 )  
159 )
```

Listing C.4: AML feature mapping classes. (File: feature_classes.aml)

C.2.4. Populate transfer file

```
1 (in-package :aml)  
2  
3 (setf root_entities (list '#65 '#96 '#103 '#80 '#94 '#70))  
4  
5 (defvar populate_hash (make-hash-table :test 'equal))  
6 (setf (gethash '#1 populate_hash) '("cartesian_point" (list "Cartesian Point 1" (list 0.0 0.0 0.0) )))  
7 (setf (gethash '#3 populate_hash) '("cartesian_point" (list "Cartesian Point 2" (list 50.0 0.0 0.0) )))  
8 (setf (gethash '#5 populate_hash) '("cartesian_point" (list "Cartesian Point 3" (list 50.0 50.0 0.0) )))  
9 (setf (gethash '#7 populate_hash) '("cartesian_point" (list "Cartesian Point 4" (list 0.0 50.0 0.0) )))  
10 (setf (gethash '#9 populate_hash) '("direction" (list "Edge Curve Direction (Edge1)" (list 1.0 0.0 0.0)  
    ↪ )))  
11 (setf (gethash '#12 populate_hash) '("vector" (list "Edge Curve Vector (Edge1)" '#9 1.0 )))  
12 (setf (gethash '#13 populate_hash) '("line" (list "Line (Edge1)" '#1 '#12 )))  
13 (setf (gethash '#14 populate_hash) '("vertex_point" (list "Vertex Point Start (Edge1)" '#1 )))  
14 (setf (gethash '#15 populate_hash) '("vertex_point" (list "Vertex Point End (Edge1)" '#3 )))  
15 (setf (gethash '#16 populate_hash) '("edge_curve" (list "Edge Curve (Edge1)" '#14 '#15 '#13 't )))  
16 (setf (gethash '#17 populate_hash) '("direction" (list "Edge Curve Direction (Edge2)" (list 0.0 1.0 0.0)  
    ↪ )))  
17 (setf (gethash '#20 populate_hash) '("vector" (list "Edge Curve Vector (Edge2)" '#17 1.0 )))  
18 (setf (gethash '#21 populate_hash) '("line" (list "Line (Edge2)" '#3 '#20 )))  
19 (setf (gethash '#22 populate_hash) '("vertex_point" (list "Vertex Point Start (Edge2)" '#3 )))  
20 (setf (gethash '#23 populate_hash) '("vertex_point" (list "Vertex Point End (Edge2)" '#5 )))
```



```
21 (setf (gethash '#24 populate_hash) '("edge_curve" (list "Edge Curve (Edge2)" '#22 '#23 '#21 't )))
22 (setf (gethash '#25 populate_hash) '("direction" (list "Edge Curve Direction (Edge3)" (list -1.0 0.0 0.0
    ↪ ) )))
23 (setf (gethash '#28 populate_hash) '("vector" (list "Edge Curve Vector (Edge3)" '#25 1.0 )))
24 (setf (gethash '#29 populate_hash) '("line" (list "Line (Edge3)" '#5 '#28 )))
25 (setf (gethash '#30 populate_hash) '("vertex_point" (list "Vertex Point Start (Edge3)" '#5 )))
26 (setf (gethash '#31 populate_hash) '("vertex_point" (list "Vertex Point End (Edge3)" '#7 )))
27 (setf (gethash '#32 populate_hash) '("edge_curve" (list "Edge Curve (Edge3)" '#30 '#31 '#29 't )))
28 (setf (gethash '#33 populate_hash) '("direction" (list "Edge Curve Direction (Edge4)" (list 0.0 -1.0 0.0
    ↪ ) )))
29 (setf (gethash '#36 populate_hash) '("vector" (list "Edge Curve Vector (Edge4)" '#33 1.0 )))
30 (setf (gethash '#37 populate_hash) '("line" (list "Line (Edge4)" '#7 '#36 )))
31 (setf (gethash '#38 populate_hash) '("vertex_point" (list "Vertex Point Start (Edge4)" '#7 )))
32 (setf (gethash '#39 populate_hash) '("vertex_point" (list "Vertex Point End (Edge4)" '#1 )))
33 (setf (gethash '#40 populate_hash) '("edge_curve" (list "Edge Curve (Edge4)" '#38 '#39 '#37 't )))
34 (setf (gethash '#41 populate_hash) '("oriented_edge" (list "Oriented Edge 1" "*" "*" '#16 nil )))
35 (setf (gethash '#42 populate_hash) '("oriented_edge" (list "Oriented Edge 2" "*" "*" '#24 nil )))
36 (setf (gethash '#43 populate_hash) '("oriented_edge" (list "Oriented Edge 3" "*" "*" '#32 nil )))
37 (setf (gethash '#44 populate_hash) '("oriented_edge" (list "Oriented Edge 4" "*" "*" '#40 nil )))
38 (setf (gethash '#45 populate_hash) '("edge_loop" (list "Edge Loop 1" (list '#41 '#42 '#43 '#44 ) )))
39 (setf (gethash '#47 populate_hash) '("face_bound" (list "Face Bound 1" '#45 nil )))
40 (setf (gethash '#48 populate_hash) '("direction" (list "Plane Perpendicular Direction" (list 0.0 0.0 1.0
    ↪ ) )))
41 (setf (gethash '#49 populate_hash) '("direction" (list "Plane Reference Direction" (list 1.0 0.0 0.0 ) ) )
    ↪ )
42 (setf (gethash '#52 populate_hash) '("cartesian_point" (list "Plane Reference Point" (list 0.0 0.0 0.0 )
    ↪ ) )
    ↪ )
43 (setf (gethash '#54 populate_hash) '("axis2_placement_3d" (list "Plane Placement" '#52 '#48 '#49 )))
```




```
44 (setf (gethash '#55 populate_hash) '("plane" (list "Base Plane" '#54 )))
45 (setf (gethash '#56 populate_hash) '("face_surface" (list "Base Face Surface" (list '#47 ) '#55 nil )))
46 (setf (gethash '#58 populate_hash) '("direction" (list "Extrusion Direction" (list 0.0 0.0 1.0 ) )))
47 (setf (gethash '#60 populate_hash) '("finite_real_interval" (list 0.0 ".CLOSED." 50.0 ".CLOSED." )))
48 (setf (gethash '#61 populate_hash) '("bound_variational_parameter" (list "extrusion_depth" '#60 "
    ↳ extrusion_depth" "Defines the Height of the Cube" "*" )))
49 (setf (gethash '#63 populate_hash) '("extruded_face_solid" (list "Extruded Cube" '#56 '#58 40.0 )))
50 (setf (gethash '#64 populate_hash) '("instance_attribute_reference" (list "GEOMETRIC_MODEL_SCHEMA.
    ↳ EXTRUDED_FACE_SOLID.DEPTH" '#63 )))
51 (setf (gethash '#65 populate_hash) '("bound_parameter_environment" (list '#61 '#64 )))
52 (setf (gethash '#67 populate_hash) '("cartesian_point" (list "Hole Center Point" (list 25.0 25.0 40.0 ) )
    ↳ ))
53 (setf (gethash '#69 populate_hash) '("instance_attribute_reference" (list "GEOMETRIC_MODEL_SCHEMA.
    ↳ CARTESIAN_POINT.COORDINATES[2]" '#67 )))
54 (setf (gethash '#70 populate_hash) '("bound_parameter_environment" (list '#61 '#69 )))
55 (setf (gethash '#71 populate_hash) '("direction" (list "Hole Direction" (list 0.0 0.0 -1.0 ) )))
56 (setf (gethash '#73 populate_hash) '("direction" (list "Hole Reference Direction" (list 1.0 0.0 0.0 ) )))
57 (setf (gethash '#75 populate_hash) '("axis2_placement_3d" (list "Hole Placement" '#67 '#71 '#73 )))
58 (setf (gethash '#76 populate_hash) '("solid_with_flat_bottom_round_hole" (list "Through Hole" "" '#63
    ↳ '#75 "*" 1 (list 10.0 ) (list 40.0 ) 0.0 )))
59 (setf (gethash '#79 populate_hash) '("instance_attribute_reference" (list "GEOMETRIC_MODEL_SCHEMA.
    ↳ SOLID_WITH_FLAT_BOTTOM_ROUND_HOLE.SEGMENT_DEPTHS[0]" '#76 )))
60 (setf (gethash '#80 populate_hash) '("bound_parameter_environment" (list '#61 '#79 )))
61 (setf (gethash '#81 populate_hash) '("cartesian_point" (list "Start Point Chamfer" (list 50.0 0.0 40.0 )
    ↳ )))
62 (setf (gethash '#83 populate_hash) '("cartesian_point" (list "End Point Chamfer" (list 50.0 50.0 40.0 ) )
    ↳ ))
```



```
63 (setf (gethash '#85 populate_hash) '("direction" (list "Edge Curve Direction (Chamfer)" (list 0.0 1.0 0.0
    ↪ ) )))
64 (setf (gethash '#88 populate_hash) '("vector" (list "Edge Curve Vector (Chamfer)" '#85 1.0 )))
65 (setf (gethash '#89 populate_hash) '("line" (list "Line (Chamfer)" '#81 '#88 )))
66 (setf (gethash '#90 populate_hash) '("vertex_point" (list "Vertex Point Start (Chamfer)" '#81 )))
67 (setf (gethash '#91 populate_hash) '("vertex_point" (list "Vertex Point End (Chamfer)" '#83 )))
68 (setf (gethash '#92 populate_hash) '("edge_curve" (list "Edge Curve (Chamfer)" '#90 '#91 '#89 't )))
69 (setf (gethash '#93 populate_hash) '("instance_attribute_reference" (list "GEOMETRIC_MODEL_SCHEMA.
    ↪ CARTESIAN_POINT.COORDINATES[2]" '#81 )))
70 (setf (gethash '#94 populate_hash) '("bound_parameter_environment" (list '#61 '#93 )))
71 (setf (gethash '#95 populate_hash) '("instance_attribute_reference" (list "GEOMETRIC_MODEL_SCHEMA.
    ↪ CARTESIAN_POINT.COORDINATES[2]" '#83 )))
72 (setf (gethash '#96 populate_hash) '("bound_parameter_environment" (list '#61 '#95 )))
73 (setf (gethash '#97 populate_hash) '("solid_with_single_offset_chamfer" (list "Chamfer" nil '#76 (list
    ↪ '#92 ) 5.0 )))
74 (setf (gethash '#99 populate_hash) '("procedural_shape_representation_sequence" (list nil (list '#63 '#76
    ↪ '#97 ) nil nil )))
75 (setf (gethash '#101 populate_hash) '("procedural_shape_representation" (list nil (list '#99 ) nil )))
76 (setf (gethash '#103 populate_hash) '("shape_definition_representation" (list nil '#101 )))
```

Listing C.5: Transfer file with an hash table containing the instances from the STEP file. (File: populate.aml)

C.3. Python

As the source code of the three developed parsers is too long to be listed here, only extracts are shown.

C.3.1. WSN converter

pre_lexer.py

```

11 import logging
12 import ply.lex as lex
13 from ply.lex import LexError
14 import ply.yacc as yacc
15 from os import path # for relative path calls
16 import datetime
17 import re
18 import csv
19
20 special_literal_dict = {
21     '<*' : 'LT_AST' ,
22     '--' : 'MIN_MIN' ,
23     '**' : 'AST_AST' ,
24     '<=' : 'LT_EQ' ,
25     '>=' : 'GT_EQ' ,
26     '<>' : 'LT_GT' ,
27     ':<>:' : 'COL_LT_GT_COL' ,
28     ':=:' : 'COL_EQ_COL' ,
29     ''' ''' : 'APO_APO' ,
30     ':=' : 'COL_EQ' ,
31     '(*' : 'LP_AST' ,
32     '*)' : 'AST_RP' ,
33     '||' : 'VB_VB' ,
34     r'\a' : 'UNICODE_SET' , # ONE OF !"#$%&'()*+,-./0123456789:;<=>?
35         ↪ @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
36         ↪ {|}~ - ignored, as only in TAIL_REMARKS which don't occur in
37         ↪ AP242
38     r'\s' : 'SPACE' , # \s SPACE -
39     r'\q' : 'APO' , # '''
40     r'\n' : 'NEWLINE' , # \n
41     r'\x8' : 'X8' , # \b BACKSPACE

```

```
39     r'\x9' : 'X9', # \t TAB
40     r'\xA' : 'XA', # \n NEWLINE
41     r'\xB' : 'XB', # \v VERTICAL_TAB
42     r'\xC' : 'XC', # \f FEED
43     r'\xD' : 'XD', # \r RETURN
44     r'''''' : 'APO',
45     r'''''''' : 'APO_APO',
46     }
47
48
49 input_file = '../data/iso-10303-11--2004.bnf' #_shortened
50
51 logger = logging.getLogger(__name__)
68 #####
69 ### TOKENS
70 #####
71 tokens = (
72     'ISO',
73     'INDEX',
74     'TOKEN',
75     'STRING',
76     'NAME',
77     'EXPR',
78     'ESCAPE',
79     'SPECIAL_LITERAL',
80 )
81
82 #####
83 ### Lexer
84 #####
85
86 class Lexer(object):
87     tokens = tokens
88
89     def __init__(self, debug=0, optimize=0, **kwargs):
90         self.lexer = lex.lex(module=self, debug=debug, optimize=
91             ↪ optimize, debuglog=logger, errorlog=logger, **kwargs)
92         self.input_length = 0
93         self.reset()
94
95     def input(self, s):
```

```
95         self.lexer.input(s)
96         self.input_length += len(s)
97
98     def reset(self):
99         self.lexer.lineno = 1
100
101     def token(self):
102         try:
103             return next(self.lexer)
104         except StopIteration:
105             return None
106
107     def t_ISO(self, t):
108         r'iso-10303-11:2004'
109         return t
110
111     def t_ESCAPE(self, t):
112         r'\\(x.|s|q|a)'
113         return t
114
115     def t_SPECIAL_LITERAL(self, t):
116         r"' [<>\\*-=: ('') \\(\\)] {1,3}' "
117         return t
118
119 #####
120 # Simple Model
121 #####
122
123 class File:
124     def __init__(self, iso, *tokens): #*tokens: list of tokens, **
125         ↪ dictionary
126         self.iso = iso
127         self.tokens = list(*tokens)
128
129 class ISO:
130     def __init__(self, iso):
131         self.iso_string = iso
132
133 class Token:
134     def __init__(self, token, regex):
135         self.token = token
136         self.regex = regex
```

```
186
187 class Production:
188     def __init__(self, prod, rule):
189         self.production_name = prod
190         self.production_rule = rule
191
192 token_dict = {}
193 prod_dict = {}
194 repetition_dict = {}
195 new_terminal_group_dict = {}
```

Listing C.6: Lexer for the WSN rules defined in ISO 10303-11:2004. (File: pre_lexer.py)

pre_parser.py

```
11 from pre_lexer import *
12 from os import path # for relative path calls
13 import datetime
14 import re
15 import csv
16
17 input_file = '../data/iso-10303-11--2004.bnf' #_shortened
18
19 #####
20 ### LEXER Test
21 #####
22
23 lexer = Lexer()
24 #lexer.build()
25 #lexer.token_count = 0          # Set the initial count
26
27 # Give the lexer some input
28 f = open(input_file, 'r')
29 data = ""
30 while 1:
31     line = f.readline()
32     if not line:break
33     data += line
34 f.close()
35 data = "<*"
36 print data
```

```
37 lexer.input(data)
38
39 # Tokenize
40 while True:
41     tok = lexer.token()
42     if not tok:
43         break      # No more input
44     print(tok)
45
46 print 'LEXER Test done.'
47
48 #####
49 ### Parser
50 #####
51 class Parser(object):
52     tokens = tokens
53
54
55     def __init__(self, lexer=None, debug=0):
56         self.lexer = lexer if lexer else Lexer()
57         self.add_group_count = 0
58         self.add_stack_count = 0
59
60         try: self.tokens = lexer.tokens
61         except AttributeError: pass
62
63         self.parser = yacc.yacc(module=self, debug=debug, debuglog=
64             ↪ logger, errorlog=logger, tabmodule='pre_parsetab')
65         self.reset()
66
67     def parse(self, data, **kwargs):
68         self.lexer.reset()
69         self.lexer.input(data)
70
71         if 'debug' in kwargs:
72             result = self.parser.parse(lexer=self.lexer, debug=logger,
73                 ** dict((k, v) for k, v in
74                     ↪ kwargs.iteritems() if k
75                     ↪ != 'debug'))
76
77         else:
78             result = self.parser.parse(lexer=self.lexer, **kwargs)
```

```
75         return result
76
77     def reset(self):
78         self.refs = {}
79         self.is_in_exchange_structure = False
80
81     def p_file(self, p):
82         '''p_file : p_iso p_token_list p_production_list'''
83         p[0] = File(p[1], p[2])
84
85     def p_iso(self, p): # ISO Part
86         '''p_iso : ';' ';' ISO'''
87         p[0] = ISO(p[3])
88
89     def p_token(self, p):
90         '''p_token : INDEX TOKEN '=' p_AND_string '.' '''
91         token_dict[p[2]] = p[4]
92         p[0] = Token(p[2], p[4])
93
94     def p_token_list(self, p):
95         '''p_token_list : p_token
96             | p_token_list p_token'''
97         try: p[0] = p[1] + [p[2]]
98         except IndexError: p[0] = [p[1]]
99
100    def p_empty(self, p):
101        '''empty :'''
102        pass
103
104    def p_production(self, p):
105        '''p_production : INDEX p_AND_string '=' p_AND_string '.'
106            | INDEX p_AND_string '=' p_OR_list '.' '''
107        prod_dict[p[2]] = p[4]
108        p[0] = Production(p[2], p[4])
109
110    def p_production_list (self, p):
111        '''p_production_list : p_production
112            | p_production_list p_production'''
113    ##         for i in p:
114    ##             print i
115        try: p[0] = p[1] + [p[2]]
```



```

116         except IndexError: p[0] = [p[1]]
117
118     def p_name_to_string(self, p): # makes it much easier, distinguish
119         ↪ later
120         '''p_string : STRING
121             | NAME
122             | EXPR
123             | TOKEN
124             | ESCAPE
125             | SPECIAL_LITERAL'''
126
127     #print p[1]
128     p[0] = p[1]
129     for spec in special_literal_dict:
130         if spec in p[1]:# and p[1] != "" and p[1] != "'''" :
131             #arg = re.sub[r'','',p[1]]
132             #print arg
133             try:
134                 p[0] = special_literal_dict[p[1]]
135                 print p[1] + ' --> ' +special_literal_dict[p[1]]
136             except KeyError:
137                 p[0] = special_literal_dict[p[1][1:-1]]
138                 print p[1] + ' --> ' +special_literal_dict[p
139                     ↪ [1][1:-1]]
140
141     ### AND/OR Syntax:
142     def p_AND_string(self, p):
143         '''p_AND_string : p_string
144             | p_AND_string p_string ''' # ?
145         try: p[0] = p[1] + ' ' + p[2]
146         except IndexError: p[0] = p[1]
147
148     # AND & AND
149     def p_AND_and_AND(self, p):
150         '''p_AND_string : p_AND_string p_AND_string ''' # ?
151         p[0] = p[1] + ' ' + p[2]
152
153     # OR & AND
154     def p_OR_and_AND(self, p):
155         '''p_OR_list : p_OR_list p_AND_string'''
156         p_temp = []

```

```
155         for elem in p[1]:
156             p_temp = p_temp + [str(elem) + ' ' + p[2]]
157         p[0] = p_temp
158
159     # AND & OR
160     def p_AND_and_OR(self, p):
161         '''p_OR_list : p_AND_string p_OR_list'''
162         p_temp = []
163         for elem in p[2]:
164             p_temp = p_temp + [p[1] + ' ' + str(elem)]
165         p[0] = p_temp
166
167     # OR & OR
168     def p_OR_and_OR(self, p):
169         '''p_OR_list : p_OR_list p_OR_list'''
170         p_temp = []
171         for elem1 in p[1]:
172             for elem2 in p[2]:
173                 p_temp = p_temp + [str(elem1) + ' ' + str(elem2)]
174         p[0] = p_temp
175
176     # AND | AND
177     def p_AND_or_AND(self, p):
178         '''p_OR_list : p_AND_string '|' p_AND_string'''
179         p[0] = [p[1]] + [p[3]]
180
181     # OR | AND
182     def p_OR_or_AND(self, p):
183         '''p_OR_list : p_OR_list '|' p_AND_string'''
184         p[0] = p[1] + [p[3]]
185
186     # AND | OR
187     def p_AND_or_OR(self, p):
188         '''p_OR_list : p_AND_string '|' p_OR_list'''
189         p[0] = [p[1]] + p[3]
190
191     # OR | OR
192     def p_OR_or_OR(self, p):
193         '''p_OR_list : p_OR_list '|' p_OR_list'''
194         p[0] = p[1] + p[3]
195
```

```

196     ### EBNF -> BNF
197     def p_AND_repetition(self, p): # Extended Backus Naur Form {...}
198         ↪ Repetition zero or more times
199         '''p_OR_list : '{' p_AND_string '}' '''
200         repetition_dict['stack_' + str(self.add_stack_count)] = [str(p
201             ↪ [2]) + ' ' + str([]), 'stack_' + str(self.
202             ↪ add_stack_count) + ' ' + str(p[2])]
203         p[0] = [[]] + ['stack_' + str(self.add_stack_count)]
204         self.add_stack_count += 1
205
206     def p_OR_repetition(self, p): # Extended Backus Naur Form {...}
207         ↪ Repetition zero or more times
208         '''p_OR_list : '{' p_OR_list '}' '''    ### p_OR_list | (
209             ↪ p_OR_list + [[]])
210         p_temp = []
211         for elem in p[2]:
212             p_temp = p_temp + [str(elem) + ' ' + 'stack_' + str(self.
213                 ↪ add_stack_count)]
214             p_temp = p_temp + [str(elem)]
215         repetition_dict['stack_' + str(self.add_stack_count)] = p_temp
216         p[0] = [[]] + ['stack_' + str(self.add_stack_count)]
217         self.add_stack_count += 1
218
219     #####
220     ### Data
221     #####
222
223     logging.basicConfig()
224     logger.setLevel(logging.DEBUG)
225
226     parser = Parser()
227     parser.reset()
228
229     # Read file and feed to parser
230     p = input_file
231     with open(p, 'rU') as f:
232         s = f.read()
233         try:
234             result = parser.parse(s)#, debug=1)
235             print result
236
237         ##         for obj in gc.get_objects():
238         ##             if isinstance(obj, P21File):

```

```

261 ##             print obj
262     except SystemExit:
263         pass
264
265 def write_to_file(name, cont):
266     f = open(path.relpath("../out/" + name), 'w')
267     f.write(cont)
268     f.close()
269     print 'File written...'
270
271 #look for predefined repetitions (digits...) and write repetitions into
    ↪ prod_dict
272 for rep in repetition_dict:
273     dupl = False
274     for prod in prod_dict:
280         if sorted(re.findall('[^ \[\]\'\\""]+', str(prod_dict[prod]))) ==
            ↪ sorted(re.findall('[^ \[\]\'\\""]+', str(repetition_dict[
            ↪ rep]])):
281             dupl = True #'[\w]+'
282             print str(rep + ' (' + repetition_dict[rep][1].split()[1])
                ↪ + ') is a duplicate of ' + prod + ' - skipped.'
283             # format prod
284             ##             print repetition_dict[rep][1].split()[1]
285             ##             print str(prod)
286             ##             print rep
287             prod_dict[prod] = [repetition_dict[rep][1].split()[1] + ' '
                ↪ + str([]), str(prod) + ' ' + repetition_dict[rep
                ↪ ][1].split()[1]]
288         if dupl == False:
289             #prod_dict[str(repetition_dict[rep])] = [rep + ' ' + str([]),
                ↪ str(repetition_dict[rep]) + ' ' + rep]
290             prod_dict[rep] = repetition_dict[rep]
291
292 for group in new_terminal_group_dict:
293     prod_dict[group] = new_terminal_group_dict[group]
294
295 #####
296 ### Write to file
297 #####
298
299 # External text files
300
301
302
303 with open("../data/templ_main_header.txt", "r") as f:

```

```

304     templ_main_header = f.read()
305 with open("../data/templ_lexer_header.txt","r") as f:
306     templ_lexer_header = f.read()
307 with open("../data/templ_lexer_test.txt","r") as f:
308     templ_lexer_test = f.read()
309 with open("../data/templ_additional_token_rules.txt","r") as f:
310     templ_additional_token_rules = f.read()
311 with open("../data/templ_productions_header.txt","r") as f:
312     templ_productions_header = f.read()
313
314 # Generated text
315 token_list = '\n'
316 token_rules = ''
317 prod_rules = '\n\n'
318
319 # Token list & rules
320 for token in token_dict:
321     token_list += '\t' + str(token) + '\n'
322     token_rules += '\t\t' + token + ' = ' + str(token_dict[token]).
        ↪ upper() + '\n'
323
324 # Delete empty [] in strings (necessary, because of {...}-repetitions
325 # Replace [] with '' in lists
326 # Add prefix p_ in references to known productions
327 def clean_and_prefix(prod_rule):
328
329     def add_prefix_AND_string(string):
330         temp_string = ''
331         string = re.sub('\[\]\s', '', string)
332         string = re.sub('\s\[\]', '', string)
333         string = re.sub('\[\]', '', string)
334         for split in string.split():
335             #print 'AND - split: ' + split
336             if 'p_' + split in prod_dict:
337                 temp_string += 'p_' + split + ' '
338                 #print 'AND - temp: ' + temp_string
339             else:
340                 temp_string += split + ' '
341                 #print 'AND - temp: ' + temp_string
342         return temp_string[:-1] # cut last ' '
343

```

```
344     def add_prefix(list_or_string):
345         if type(list_or_string).__name__ == 'str':
346             list_or_string = add_prefix_AND_string(list_or_string)
347             return list_or_string
348         else:
349             or_list = list_or_string
350
351             elem_index = 0
352             for elem in or_list:
353                 #print 'OR - elem: ' + str(elem)
354                 #print type(elem).__name__
355                 if type(elem).__name__ == 'str':
356                     #print 'OR - go to AND: ' + elem
357                     #print 'Index: ' + str(elem_index)
358                     or_list[elem_index] = add_prefix_AND_string(elem)
359                     elem_index +=1
360                 elif type(elem).__name__ == 'list': # never goes in
361                     ↪ here :)
362                     print elem
363                     if elem == []:
364                         or_list[elem_index] = '' # replace [] with ''
365                         print elem + ' transformend into: ' + or_list[
366                             ↪ elem_index]
367                     else:
368                         or_list[elem_index] = add_prefix_AND_string(
369                             ↪ elem)
370                 else:
371                     print 'add_prefix for type: ' + type(elem).__name__
372                     ↪ + ' not defined!'
373                 # print 'or_list: ' + str(or_list)
374             return or_list
375
376     return add_prefix(prod_rule)
377
378 prod_edges = []
379
380 prod_list = []
381 for prod in prod_dict:
382     prod_list += [prod]
383
384 for prod_elem in prod_list:
```

```

381     p_prod = 'p_' + prod_elem
382     prod_dict[p_prod] = prod_dict.pop(prod_elem)
383
384 # Generate production rules
385 for prod in prod_dict:
386     prod_dict[prod] = clean_and_prefix(prod_dict[prod])
387     if type(prod_dict[prod]).__name__ == 'str':
388         # write edges to csv file for visualisation in yEd
389         ##           print '_____ '
390         ##           print prod
391         ##           print prod_dict[prod]
392         for ref in prod_dict[prod].split():
393             prod_edges += [[prod, prod_dict[prod], ref, 'production']]
394         prod_rules += '\tdef ' + prod + '(self, p):\n\t\t' + '\"\"\" '
395             ↪ + prod + ' : ' + prod_dict[prod] + ' \"\"\" \n\t\t p[0] =
396             ↪ p[1] #Standard - CHANGE!? \n\n'
397     elif type(prod_dict[prod]).__name__ == 'list':
398         # print str(prod_dict[prod])
399         # write edges to csv file for visualisation in yEd
400         for list_elem in prod_dict[prod]:
401             for ref in list_elem.split():
402                 prod_edges += [[prod, prod_dict[prod], ref, 'production
403                 ↪ ']]
404         prod_rules += '\tdef ' + prod + '(self, p):\n\t\t' + '\"\"\" '
405             ↪ + prod + ' : ' + prod_dict[prod][0]
406         first = True
407         for item in prod_dict[prod]:
408             if not first:
409                 prod_rules += '\n\t\t\t| ' + item
410             first = False
411         if 'stack' in prod:
412             prod_rules += ' \"\"\" \n\t\t' + \
413             '''try: p[0] = p[1] + ' ' + p[2]
414             except IndexError: p[0] = p[1] #Standard - CHANGE!?
415             ''' + '\n'
416         else:
417             prod_rules += ' \"\"\" \n\t\t p[0] = p[1] #Standard - CHANGE
418             ↪ !? \n\n'
419     else:
420         print 'Procedure for type ' + type(prod_dict[prod]).__name__ +
421             ↪ ' is not defined!'

```

```
416
417
418 with open('../out/pre_parser_graph.csv', 'wb') as myfile:
419     wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
420     for elem in prod_edges:
421         wr.writerow(elem)
422
423 # Add all together
424 file_text = templ_main_header + \
425             token_list + \
426             templ_lexer_header + \
427             token_rules + \
428             templ_additional_token_rules + \
429             templ_lexer_test + \
430             templ_productions_header + \
431             prod_rules
432
433 # Write to file
434 write_to_file('pre_parser_output.py', file_text)
```

Listing C.7: Parser for the WSN rules defined in ISO 10303–11:2004. Generates the schema_lexer.py file. (File: pre_parser.py)

C.3.2. Schema Parser

schema_lexer.py

```
24 import logging
25 import ply.lex as lex
26 from ply.lex import LexError
27 import ply.yacc as yacc
28
29 logger = logging.getLogger(__name__)
30
31 # ensure Python 2.6 compatibility
32 if not hasattr(logging, 'NullHandler'):
33     class NullHandler(logging.Handler):
34         def handle(self, record):
35             pass
36         def emit(self, record):
37             pass
38         def createLock(self):
39             self.lock = None
40
41     setattr(logging, 'NullHandler', NullHandler)
42
43 logger.addHandler(logging.NullHandler())
44
45
46 ## listing_start tokens "Tokens defined in ISO 10303-11."
47 #####
48 ### Tokens
49 #####
50
51 #Tokend defined in 10303-11
52 tokens = (
53     'FORMAT' ,
54     'GENERIC' ,
55     'ENTITY' ,
56     'EXP' ,
57     'TRUE' ,
58     'BY' ,
59     'END_REPEAT' ,
60     'LOG2' ,
```

```
194     'UNICODE_SET',
195     'SPACE',
196     'X8', # \x8 # only used in simple_string_literal
197     'X9', # \x9
198     #'XA', # \xA
199     'XB', # \xB
200     'XC', # \xC
201     'XD', # \xD
202     #special characters
203     'BS', # '\
204 )
205 ## listing_end tokens
206
207 ## listing_start token_rules "Token Rules defined in ISO 10303-11."
208 #####
209 ### Lexer (Token Rules)
210 #####
211
212 class Lexer(object):
213     tokens = tokens
214     states = (
215         ('preamble', 'exclusive'),
216     )
217     def __init__(self, debug=0, optimize=0, header_limit=4096, **kwargs
218         ↪ ):
219         self.lexer = lex.lex(module=self, debug=debug, optimize=
220             ↪ optimize, debuglog=logger, errorlog=logger, **kwargs)
221         self.input_length = 0
222         self.header_limit = header_limit
223         self.reset()
224
225     def input(self, s):
226         self.lexer.input(s)
227         self.input_length += len(s)
228
229     def reset(self):
230         self.lexer.lineno = 1
231         self.lexer.begin('preamble')
232         print 'preamble state'
233
234     def token(self):
```

```
243         try:
244             return next(self.lexer)
245         except StopIteration:
246             return None
247
248     def t_preamble_SCHEMA(self, t):
249         r'SCHEMA'
250         t.lexer.begin('INITIAL')
251         print 'initial state'
252         return t
253
254     def t_preamble_error(self, t):
255         t.lexer.skip(1)
256
257     t_FORMAT = 'FORMAT'
258     t_GENERIC = 'GENERIC'
259     t_ENTITY = 'ENTITY'
260     t_EXP = 'EXP'
261     t_TRUE = 'TRUE'
262     t_BY = 'BY'
408     #special characters
409     t_BS = r'\'\'
410
411     # ' literal?
412     literals = 'abcdefghijklmnopqrstuvwxyz' + \
413               'ABCDEFGHIJKLMNOPQRSTUVWXYZ' + \
414               '0123456789' + \
415               '()' + \
416               '"' + \
417               "!#$%&+,-./:;<=>?@[\\]^_`{|}~" + \
418               '*'
428     def t_preamble_COMMENT(self, t): # _ALL_ doesn't work!?
429         r'\(\\*[^\*]*\\*)'
430         return t
431
432     def t_COMMENT(self, t):
433         r'\(\\*[^\*]*\\*)'
434         return t
435
436     def t_ALL_newline(self, t):
437         r'\n+'
```

```

438         t.lexer.lineno += len(t.value)
439
440     # Ignored characters (whitespace)
441     ##     t_ignore = ' \t'
442     def t_SPACETAB(self,t):
443         r' [ \t]+'
444         #print "Space(s) and/or tab(s)"
445     ##     def t_SPACETAB(self,t):
446     ##         r' [ ]{2,}|\[\t]+'
447     ##         print "Spaces and/or tab(s)"
448
449     # Error handling rule
450     def t_error(self, t):
451         print("Illegal character '%s'" % t.value[0])
452         t.lexer.skip(1)
453
454     #literals = ' (=;,*$.|{}[]%\'
455
456     def test(self,data):
457         self.lexer.input(data)
458         while True:
459             tok = self.lexer.token()
460             if not tok:
461                 break
462             print(tok)

```

Listing C.8: EXPRESS schema lexer. (File: schema_lexer.py)

schema_parser.py

```

1 from schema_lexer import *
2 from os import path # for relative path calls
3 import re
4 import pdb
5
6 input_file = '../..//00_shared_data/mim_lf.exp'
7
8
9 #####
10 ### LEXER Test
11 #####
12 def lexer_test():

```

```
13     lexer = Lexer()
14     #lexer.build()
15     #lexer.token_count = 0           # Set the initial count
16
17     # Give the lexer some input
18     f = open(input_file, 'r')
19     data = ""
20     while 1:
21         line = f.readline()
22         if not line:break
23         data += line
24     f.close()
25     #data= 'UNICODE_empty'
26     lexer.input(data)
27
28     # Tokenize
29     while True:
30         tok = lexer.token()
31         if not tok:
32             break           # No more input
33         print(tok)
34
35     print 'LEXER Test done.'
36
37
38     #####
39     # Simple Model
40     #####
41
42     #p_type_decl : TYPE p_type_id '=' p_underlying_type ';' p_where_clause
43     ↪ END_TYPE ';'
44     class Type_decl:
45         def __init__(self, type_id, underlying_type, where_clause):
46             self.type_id = type_id
47             self.underlying_type = underlying_type
48             self.where_clause = where_clause
49
50     #p_explicit_attr : p_attribute_decl ':' OPTIONAL p_parameter_type ';'
51     class Explicit_attr:
52         def __init__(self, attribute_decl, OPTIONAL, parameter_type):
53             self.attribute_decl = attribute_decl
```

```
53         self.OPTIONAL = OPTIONAL
54         self.parameter_type = parameter_type
55
56 #p_subsuper : p_supertype_constraint p_subtype_declaration
57 class Subsuper:
58     def __init__(self, supertype_constraint, subtype_declaration):
59         self.supertype_constraint = supertype_constraint
60         self.subtype_declaration = subtype_declaration
61
62
63 # ENTITY p_entity_id p_subsuper ';'
64 class Entity_head:
65     def __init__(self, entity_id, subsuper):
66         self.entity_id = entity_id
67         self.subsuper = subsuper
68
69 # p_stack_25 p_derive_clause p_inverse_clause p_unique_clause
70     ↪ p_where_clause
71 class Entity_body:
72     def __init__(self, explicit_attr, derive_clause, inverse_clause,
73     ↪ unique_clause, where_clause):
74         self.explicit_attr = explicit_attr
75         self.derive_clause = derive_clause
76         self.inverse_clause = inverse_clause
77         self.unique_clause = unique_clause
78         self.where_clause = where_clause
79
80 # p_entity_head p_entity_body END_ENTITY ';'
81 class Entity_decl:
82     def __init__(self, entity_head, entity_body):
83         self.entity_head = entity_head
84         self.entity_body = entity_body
85
86 ## listing_start class_model "Extract of the class model representing
87     ↪ the EXPRESS Structure."
88 # p_schema_body : p_stack_45 p_constant_decl p_stack_46 # p_stack_45:
89     ↪ p_interface_specification # p_stack_46: declarations
90 class Schema_body:
91     def __init__(self, interface_specification , declarations):
92         self.interface_specification = interface_specification
93         self.declarations = declarations
```

```
90
91 #p_schema_decl : SCHEMA p_schema_id p_schema_version_id ';'
          ↪ p_schema_body END_SCHEMA
92 class Schema_decl:
93     def __init__(self, schema_id, schema_version_id, schema_body):
94         self.schema_id = schema_id
95         self.schema_version_id = schema_version_id
96         self.schema_body = schema_body
97
98 #p_syntax : p_schema_decl
99 class Syntax:
100     def __init__(self, schema_decl):
101         self.schema_decl = schema_decl
102 # ...
103 ## listing_end class_model
104
105 #####
106 ### Productions
107 #####
108
109 class Parser(object):
110     tokens = tokens
111
112     def __init__(self, lexer=None, debug=0):
113         self.lexer = lexer if lexer else Lexer()
114
115         try: self.tokens = lexer.tokens
116         except AttributeError: pass
117
118         self.parser = yacc.yacc(module=self, debug=debug, debuglog=
          ↪ logger, errorlog=logger, tabmodule='schema_parsetab')
119         self.reset()
120
121     def parse(self, data, **kwargs):
122         self.lexer.reset()
123         self.lexer.input(data)
124
125         if 'debug' in kwargs:
126             result = self.parser.parse(lexer=self.lexer, debug=logger,
          ↪ ** dict((k, v) for k, v in kwargs.iteritems() if k
          ↪ != 'debug'))
```

```
127         else:
128             result = self.parser.parse(lexer=self.lexer, **kwargs)
129         return result
130
131     def reset(self):
132         self.refs = {}
133         self.is_in_exchange_structure = False
134
135     def p_result(self, p):
136         """ p_result : p_syntax """
137         p[0] = p[1]
138     def p_syntax(self, p):
139         """ p_syntax : p_schema_decl
140             | p_syntax p_schema_decl """
141         if len(p) == 3:
142             p[0] = p[1] + p[2]
143         elif len(p) == 2:
144             p[0] = Syntax(p[1])
145         else:
146             print 'Indexerror in p_syntax with len(p):'
147             print len(p)
148     def p_entity_body_1234_5(self, p): #p_stack_25: p_explicit_attr
149         """ p_entity_body : p_stack_25 p_derive_clause p_inverse_clause
150             ↪ p_unique_clause p_where_clause
151             | p_stack_25 p_derive_clause p_inverse_clause
152             ↪ p_unique_clause """
153         if len(p) == 6:
154             p[0] = Entity_body(p[1], p[2], p[3], p[4], p[5])
155         elif len(p) == 5:
156             p[0] = Entity_body(p[1], p[2], p[3], p[4], [])
157         else:
158             print 'Indexerror in p_entity_body with len(p):'
159             print len(p)
160
161     def p_entity_body_123_4(self, p): #p_stack_25: p_explicit_attr
162         """ p_entity_body : p_stack_25 p_derive_clause p_inverse_clause
163             ↪ p_where_clause
164             | p_stack_25 p_derive_clause p_inverse_clause
165             ↪ """
166         if len(p) == 5:
167             p[0] = Entity_body(p[1], p[2], p[3], p[4], [])
```



```
599     elif len(p) == 4:
600         p[0] = Entity_body(p[1], p[2], p[3], [], [])
601     else:
602         print 'Indexerror in p_entity_body with len(p):'
603         print len(p)
604
605 def p_entity_body_124_5(self, p): #p_stack_25: p_explicit_attr
606     """ p_entity_body : p_stack_25 p_derive_clause p_unique_clause
607         ↪ p_where_clause
608             | p_stack_25 p_derive_clause p_unique_clause
609             ↪ """
610
611     if len(p) == 5:
612         p[0] = Entity_body(p[1], p[2], [], p[3], p[4])
613     elif len(p) == 4:
614         p[0] = Entity_body(p[1], p[2], [], p[3], [])
615     else:
616         print 'Indexerror in p_entity_body with len(p):'
617         print len(p)
618
619 def p_entity_body_12_5(self, p): #p_stack_25: p_explicit_attr
620     """ p_entity_body : p_stack_25 p_derive_clause p_where_clause
621             | p_stack_25 p_derive_clause """
622
623     if len(p) == 4:
624         p[0] = Entity_body(p[1], p[2], [], [], p[3])
625     elif len(p) == 3:
626         p[0] = Entity_body(p[1], p[2], [], [], [])
627     else:
628         print 'Indexerror in p_entity_body with len(p):'
629         print len(p)
630
631 def p_entity_body_134_5(self, p): #p_stack_25: p_explicit_attr
632     """ p_entity_body : p_stack_25 p_inverse_clause p_unique_clause
633         ↪ p_where_clause
634             | p_stack_25 p_inverse_clause p_unique_clause
635             ↪ """
636
637     if len(p) == 5:
638         p[0] = Entity_body(p[1], [], p[2], p[3], p[4])
639     elif len(p) == 4:
640         p[0] = Entity_body(p[1], [], p[2], p[3], [])
641     else:
642         print 'Indexerror in p_entity_body with len(p):'
```

```
636         print len(p)
637
638     def p_entity_body_13_5(self, p): #p_stack_25: p_explicit_attr
639         """ p_entity_body : p_stack_25 p_inverse_clause p_where_clause
640             | p_stack_25 p_inverse_clause """
641         if len(p) == 4:
642             p[0] = Entity_body(p[1], [], p[2], [], p[3])
643         elif len(p) == 3:
644             p[0] = Entity_body(p[1], [], p[2], [], [])
645         else:
646             print 'Indexerror in p_entity_body with len(p):'
647             print len(p)
648
649     def p_entity_body_14_5(self, p): #p_stack_25: p_explicit_attr
650         """ p_entity_body : p_stack_25 p_unique_clause p_where_clause
651             | p_stack_25 p_unique_clause """
652         if len(p) == 4:
653             p[0] = Entity_body(p[1], [], [], p[2], p[3])
654         elif len(p) == 3:
655             p[0] = Entity_body(p[1], [], [], p[2], [])
656         else:
657             print 'Indexerror in p_entity_body with len(p):'
658             print len(p)
659
660     def p_entity_body_1_5(self, p): #p_stack_25: p_explicit_attr
661         """ p_entity_body : p_stack_25 p_where_clause
662             | p_stack_25 """
663         if len(p) == 3:
664             p[0] = Entity_body(p[1], [], [], [], p[2])
665         elif len(p) == 2:
666             p[0] = Entity_body(p[1], [], [], [], [])
667         else:
668             print 'Indexerror in p_entity_body with len(p):'
669             print len(p)
670
671     def p_entity_body_234_5(self, p):
672         """ p_entity_body : p_derive_clause p_inverse_clause
673             ↪ p_unique_clause p_where_clause
674             | p_derive_clause p_inverse_clause
675             ↪ p_unique_clause """
676         if len(p) == 5:
```

```
675         p[0] = Entity_body([], p[1], p[2], p[3], p[4])
676     elif len(p) == 4:
677         p[0] = Entity_body([], p[1], p[2], p[3], [])
678     else:
679         print 'Indexerror in p_entity_body with len(p):'
680         print len(p)
681
682 def p_entity_body_23_4(self, p):
683     """ p_entity_body : p_derive_clause p_inverse_clause
684         ↪ p_where_clause
685         | p_derive_clause p_inverse_clause """
686     if len(p) == 4:
687         p[0] = Entity_body([], p[1], p[2], p[3], [])
688     elif len(p) == 3:
689         p[0] = Entity_body([], p[1], p[2], [], [])
690     else:
691         print 'Indexerror in p_entity_body with len(p):'
692         print len(p)
693
694 def p_entity_body_24_5(self, p):
695     """ p_entity_body : p_derive_clause p_unique_clause
696         ↪ p_where_clause
697         | p_derive_clause p_unique_clause """
698     if len(p) == 4:
699         p[0] = Entity_body([], p[1], [], p[2], p[3])
700     elif len(p) == 3:
701         p[0] = Entity_body([], p[1], [], p[2], [])
702     else:
703         print 'Indexerror in p_entity_body with len(p):'
704         print len(p)
705
706 def p_entity_body_2_5(self, p):
707     """ p_entity_body : p_derive_clause p_where_clause
708         | p_derive_clause """
709     if len(p) == 3:
710         p[0] = Entity_body([], p[1], [], [], p[2])
711     elif len(p) == 2:
712         p[0] = Entity_body([], p[1], [], [], [])
713     else:
714         print 'Indexerror in p_entity_body with len(p):'
715         print len(p)
```

```
714
715 def p_entity_body_34_5(self, p):
716     """ p_entity_body : p_inverse_clause p_unique_clause
717         ↪ p_where_clause
718         | p_inverse_clause p_unique_clause """
719     if len(p) == 4:
720         p[0] = Entity_body([], [], p[1], p[2], p[3])
721     elif len(p) == 3:
722         p[0] = Entity_body([], [], p[1], p[2], [])
723     else:
724         print 'Indexerror in p_entity_body with len(p):'
725         print len(p)
726
727 def p_entity_body_3_5(self, p):
728     """ p_entity_body : p_inverse_clause p_where_clause
729         | p_inverse_clause """
730     if len(p) == 3:
731         p[0] = Entity_body([], [], p[1], [], p[2])
732     elif len(p) == 2:
733         p[0] = Entity_body([], [], p[1], [], [])
734     else:
735         print 'Indexerror in p_entity_body with len(p):'
736         print len(p)
737
738 def p_entity_body_4_5(self, p):
739     """ p_entity_body : p_unique_clause p_where_clause
740         | p_unique_clause """
741     if len(p) == 3:
742         p[0] = Entity_body([], [], [], p[1], p[2])
743     elif len(p) == 2:
744         p[0] = Entity_body([], [], [], p[1], [])
745     else:
746         print 'Indexerror in p_entity_body with len(p):'
747         print len(p)
748
749 def p_entity_body__5(self, p):
750     """ p_entity_body : p_where_clause
751         | """
752     if len(p) == 2:
753         p[0] = Entity_body([], [], [], [], p[1])
754     elif len(p) == 1:
```

```
754         p[0] = Entity_body([], [], [], [], [])
755     else:
756         print 'Indexerror in p_entity_body with len(p):'
757         print len(p)
1007 def p_schema_body_int_spec(self, p): # p_stack_45:
    ↪ p_interface_specification # p_stack_46: declarations
1008     """ p_schema_body : p_stack_45 p_constant_decl p_stack_46
1009         | p_stack_45 p_stack_46
1010         | p_stack_45 p_constant_decl
1011         | p_stack_45 """
1012     if len(p) == 4:
1013         p[0] = Schema_body(p[1], p[2] + p[3])
1014     elif len(p) == 3:
1015         p[0] = Schema_body(p[1], p[2])
1016     elif len(p) == 2:
1017         p[0] = Schema_body(p[1], [])
1018     else:
1019         print 'Indexerror in p_schema_body_int_spec with len(p):'
1020         print len(p)
1021
1022 def p_schema_body(self, p):
1023     """ p_schema_body : p_constant_decl p_stack_46
1024         | p_stack_46
1025         | p_constant_decl
1026         | """
1027     if len(p) == 3:
1028         p[0] = Schema_body([], [p[1]] + p[2])
1029     elif len(p) == 2:
1030         p[0] = Schema_body([], p[1])
1031     elif len(p) == 1:
1032         p[0] = Schema_body([], [])
1033     else:
1034         print 'Indexerror in p_schema_body with len(p):'
1035         print len(p)
2383 def p_subsuper_supertype(self, p):
2384     """ p_subsuper : p_supertype_constraint p_subtype_declaration
2385         | p_supertype_constraint """
2386     if len(p) == 3:
2387         p[0] = Subsuper(p[1], p[2])
2388     elif len(p) == 2:
2389         p[0] = Subsuper(p[1], [])
```

```
2390         else:
2391             print 'Indexerror in p_subsuper_supertype with len(p):'
2392             print len(p)
2393
2394     def p_subsuper(self, p):
2395         """ p_subsuper : p_subtype_declaration
2396             | """
2397         if len(p) == 2:
2398             p[0] = Subsuper([], p[1])
2399         elif len(p) == 1:
2400             p[0] = Subsuper([], [])
2401         else:
2402             print 'Indexerror in p_subsuper with len(p):'
2403             print len(p)
2404
2405     #####
2406     ### Run Parser
2407     #####
2408
2409     paragraph_dict = {}
2410
2411     def parser_test(debug = False):
2412
2413         paragraph = ''
2414         first = True
2415
2416         logging.basicConfig()
2417         logger.setLevel(logging.DEBUG)
2418
2419         parser = Parser()
2420         parser.reset()
2421
2422         # Read file and feed to parser
2423         p = input_file
2424         with open(p, 'rU') as f:
2425             s = ''
2426             line_nr = 0
2427             while True:
2428                 line_nr += 1
2429                 line = f.readline()
2430                 if not line:break
```

```
2860
2861     s += line
2862
2863     # Generate paragraph_dict with original code snippets for
2864     ↪ AML code generation
2865     if (
2866         line.lstrip().startswith('ENTITY ') or
2867         line.lstrip().startswith('TYPE ') or
2868         line.lstrip().startswith('CONSTANT') or
2869         line.lstrip().startswith('FUNCTION ') or
2870         line.lstrip().startswith('RULE ')
2871     ):
2872         first_line = line
2873         #print first_line
2874         first = False
2875         paragraph = ''
2876
2877     paragraph += ';; ' + line
2878
2879     if (
2880         line.lstrip().startswith('END_ENTITY') or
2881         line.lstrip().startswith('END_TYPE') or
2882         line.lstrip().startswith('END_CONSTANT') or
2883         line.lstrip().startswith('END_FUNCTION') or
2884         line.lstrip().startswith('END_RULE')
2885     ):
2886         #print line
2887         try:
2888             paragraph_dict[first_line.split()[0] + ' ' +
2889                 ↪ first_line.split()[1].replace(';', '')] =
2890                 ↪ paragraph
2891         except IndexError: pass
2892         first = True
2893         paragraph = ''
2894
2895     try:
2896         if debug == 'debug':
2897             result = parser.parse(s, debug=1) #debug=1
2898         else:
2899             result = parser.parse(s)
2900     print result
```

```

2898         return result
2899     ##         for obj in gc.get_objects():
2900     ##             if isinstance(obj, P21File):
2901     ##                 print obj
2902     except SystemExit:
2903         pass
2904
2905
2906
2907 result = parser_test() #'debug')

```

Listing C.9: EXPRESS schema parser. (File: schema_parser.py)

schema_write_aml.py

```

1 from schema_parser import *
2 import datetime
3
4 timestamp = datetime.datetime.now().strftime('%Y/%m/%d %H:%M:%S')
5 aml_header = ';;;;;;;;;;;;;;\n' + \
6             ';;\n' + \
7             ';; ' + timestamp + ' jschaetzle\n' + \
8             ';;\n' + \
9             ';; This AML class was created by parsing the following
10             ↪ EXPRESS Schema: \n' + \
11             ';; ISO/TS 10303-442 AP242 managed model based 3d
12             ↪ engineering - EXPRESS MIM Long form Schema v
13             ↪ 1.36\n' + \
14             ';;\n' + \
15             ';; The original CONSTANT/TYPE/ENTITY/RULE/FUNCTION
16             ↪ looks like this:\n' + \
17             ';;\n' + \
18             ';;;;;;;;;;;;;;\n' + \
19             ';;\n'
20
21 ## listing_start aml_mapping "Mapping between AP242 classes and AML
22     ↪ geometric objects."
23 aml_entity_suffix = ''
24 aml_type_suffix = ''
25 aml_entity_prefix = 'ap242_'
26 aml_type_prefix = 'ap242_'
27 system_name = 'main-system' # defined in logical.pth in AML path

```



```
23 class_path = 'class-path' # defined in logical.pth in AML path
24
25 aml_class_mapping = {
26     'cartesian_point'      : 'aml_point_object',
27     #'vector'              : 'vector-class',
28     #'line'                : 'line-object',
29     'edge_curve'          : 'aml_line_object',
30     'edge_loop'           : 'aml_sewn_object',
31     'face_bound'          : 'aml_bounded_object',
32     'extruded_face_solid' : 'aml_extrusion_object',
33     'solid_with_flat_bottom_round_hole' : '
34         ↪ xaml_flat_bottom_round_hole_feature',
35     'axis2_placement_3d'   : 'aml_coordinate_system_class',
36     'solid_with_single_offset_chamfer' : 'xaml_single_offset_feature',
37     'bound_parameter_environment' : 'xaml_parameter'
38 }
39
40 ## listing_end aml_mapping
41
42 #####
43 ### Create Dictionary "Database" for entities (inheritance...)
44 #####
45
46 entity_supertype_dict = {}
47 entity_subtype_dict = {}
48 entity_attributes_dict = {}
49 entity_dependencies_dict = {}
50 entity_inherited_attributes_dict = {}
51
52 for decl in result.schema_decl.schema_body.declarations:
53     if decl.__class__.__name__ == 'Entity_decl':
54         ### Supertype of
55         entity_supertype_dict[decl.entity_head.entity_id] = decl.
56             ↪ entity_head.subsuper.supertype_constraint # ?
57         ### Subtype of
58         entity_subtype_dict[decl.entity_head.entity_id] = decl.
59             ↪ entity_head.subsuper.subtype_declaration
60         if decl.entity_head.subsuper.subtype_declaration == []:
61             entity_subtype_dict[decl.entity_head.entity_id] = ['object'
62                 ↪ ]
63         ### Attributes
```

```

60     #attribute = ''
61     attribute_dict = {'__index__': []}
62     for attr in decl.entity_body.explicit_attr:
63         attribute_dict['__index__'] += [attr.attribute_decl]
64         attribute_dict[attr.attribute_decl] = attr.parameter_type #
        ↪ List or Dictionary?
65         #attribute_stack += [attribute]
66     entity_attributes_dict[decl.entity_head.entity_id] =
        ↪ attribute_dict # List of Lists/Dics
67
68 ### Dependencies
69 def get_dependencies(ap242_class):
70     #print 'ap242_class: ' + ap242_class
71
72     def get_ancestors(child_class, ancestors = []):
73         #print 'child_class: ' + child_class
74         #print 'ancestors: ' + str(ancestors)
75         parents = entity_subtype_dict[child_class]
76         #print 'parents: ' + str(parents)
77         for parent in parents:
78             #print 'parent: ' + parent
79
80             if parent not in ancestors:
81                 ancestors += [parent]
82             if parent != 'object':
83                 ancestors = get_ancestors(parent, ancestors)
84     return ancestors
85
86     ancestors = get_ancestors(ap242_class)
87
88     dependencies = ancestors #TODO
89     return dependencies
90
91 def get_inherited_attributes(child_entity, attributes = {}, parent_list
    ↪ = []):
92     parents = entity_subtype_dict[child_entity]
93     for parent in reversed(parents):
94         if parent is not 'object' and parent not in parent_list:
95             attributes['__index__'] += [parent]
96             attributes[parent] = {}

```

```

97         attributes[parent]['__index__'] = entity_attributes_dict[
           ↪ parent]['__index__']
98         for i in entity_attributes_dict[parent]['__index__']:
99             attributes[parent][i] = entity_attributes_dict[parent][
           ↪ i]
100        attributes = get_inherited_attributes(parent, attributes,
           ↪ parent_list)
101        parent_list += [parent]
102    return attributes
103
104 for entity in entity_subtype_dict:
105     entity_dependencies_dict[entity] = get_dependencies(entity)
106     entity_inherited_attributes_dict[entity] = get_inherited_attributes
           ↪ (entity, {'__index__' : []}, [])
107
108
109 #####
110 ### Write to AML
111 #####
112
113 type_dict = {}
114
115 entity_counter = 0
116
117 def write_aml_file(class_name, cont, out_folder):
118     "Writes class to ./classes/class_name.aml"
119     f = open(path.relpath("../out/" + out_folder + class_name + '.aml')
           ↪ , 'w')
120     f.write(cont)
121     f.close()
122     #print 'Class: "' + class_name + '"' written to AML class.'
123     return 'done'
124
125 ### Entity content
126 # Go through supertype nested list:
127 supertype = ''
128 def rec_list(supertype_list, pos = 'start', sup = ''):
129     supertype = sup
130     if type(supertype_list).__name__ == 'list':
131         supertype += '(list\n'
132         for i, elem in enumerate(supertype_list):

```

```
133         #print i
134         #print len(supertype_list) - 1
135         if i == len(supertype_list) - 1: # last element
136             #print 'Goto rec_list with last: ' + str(elem)
137             supertype = rec_list(elem, 'last', sup = supertype)
138         else:
139             #print 'Goto rec_list with: ' + str(elem)
140             supertype = rec_list(elem, sup = supertype)
141         if pos == 'last': supertype += ')\n'
142
143     elif type(supertype_list).__name__ == 'str':
144         elem = supertype_list
145         if elem == 'ONEOF':
146             supertype += ' ;; ONEOF\n'
147         elif elem == 'AND':
148             supertype += ' ;; AND\n'
149         elif elem == 'ANDOR':
150             supertype += ' ;; ANDOR\n'
151         else:
152             supertype += '\ ' + aml_entity_prefix + elem +
153                 '\ ' + aml_entity_suffix + '\n'
154             #print 'Error: unknown string: ' + elem
155
156         if pos == 'last': # last element
157             supertype += ')\n'
158         return supertype
159
160     else: print 'Error: rec_list() not defined for type: ' + type(
161         supertype_list).__name__
162     return supertype
163     #print supertype
164
165 for entity in entity_supertype_dict: # all entities
166     # Subtypes
167     subtypes = ''
168     load_classes = ''
169     for subtype in entity_subtype_dict[entity]:
170         if subtype == 'object':
171             subtypes += subtype + '\n'
```

```

172     else:
173         subtypes += aml_entity_prefix + subtype + aml_entity_suffix
174                 ↪ + '\n'
175         load_classes += '(load_class ' + \
176                 ↪ ' "' + aml_entity_prefix + subtype +
177                 ↪ aml_entity_suffix + '")\n'
178
179     if entity in aml_class_mapping:
180         subtypes += aml_class_mapping[entity] + '\n'
181
182     # Index of Instantiable Subobjects & Inheritance History
183     index = 'index (list\n'
184     index_list = []
185     inh_history = ''
186     for inh_class in entity_inherited_attributes_dict[entity]['
187                 ↪ __index__']:
188         inh_history += ';; inherited from ' + inh_class + ':\n'
189         class_index_list = []
190         for inh_attr in entity_inherited_attributes_dict[entity][
191                 ↪ inh_class]['__index__']:
192             attr_type = entity_inherited_attributes_dict[entity][
193                 ↪ inh_class][inh_attr]
194             if True: #inh_attr not in index_list:
195                 class_index_list += [inh_attr]
196                 if type(attr_type).__name__ == 'str':
197                     inh_history += ';;-> ' + inh_attr + ' \'' + attr_type +
198                             ↪ '\n'
199                 elif type(attr_type).__name__ == 'list':
200                     inh_history += ';;-> ' + inh_attr + ' \'' + attr_type
201                             ↪ [0]
202                     inh_history += ' ;; ' + attr_type[1] + ' \'' + str(
203                             ↪ attr_type[2]) + '\n'
204             else:
205                 print 'undefined type in definition of inherited
206                         ↪ attributes: ' + \
207                         ↪ type(attr_type).__name__
208         index_list = class_index_list + index_list
209     for attr in entity_attributes_dict[entity]['__index__']:
210         if True: #attr not in index_list:
211             index_list += [attr]
212     for elem in index_list:
213         index += '\n' + elem + '\n'

```

```

204     index += ')\n'
205
206     # Explicit attributes
207     attribute_string = ''
208     attribute_stack_string = ''
209     for attr in entity_attributes_dict[entity]['__index__']:
210         attr_type = entity_attributes_dict[entity][attr]
211         if type(attr_type).__name__ == 'str':
212             attribute_string = attr + ' \'' + attr_type + '\n'
213         elif type(attr_type).__name__ == 'list':
214             attribute_string = attr + ' \'' + attr_type[0] + ' ;; ' +
                ↪ attr_type[1] + ' ' + str(attr_type[2]) + '\n'
215         else:
216             print 'undefined type in definition of explicit attributes:
                ↪ ' + \
                type(attr).__name__
217             attribute_stack_string += attribute_string
218
219
220     # Derived Attributes
221     # @TODO
222
223     # Supertype
224     if entity_supertype_dict[entity] != []:
225         supertype = rec_list(entity_supertype_dict[entity])
226         #print supertype
227         supertype = 'supertype (\n' + \
228             supertype + \
229             ')\n'
230     else: supertype = ''
231
232     aml_entity = load_classes + \
233         '\n' + \
234         '(define-class ' + aml_entity_prefix + entity +
                ↪ aml_entity_suffix + '\n' + \
235         ':inherit-from (\n' + \
236         subtypes + \
237         ')\n' + \
238         ':properties (\n' + \
239         index + \
240         ';; SUPERTYPE ;;\n' + \
241         supertype + \

```

```
242         ';; INHERITED ATTRIBUTES ;;\n' + \  
243         inh_history + \  
244         ';; EXPLICIT ATTRIBUTES ;;\n' + \  
245         attribute_stack_string + \  
246         ')\n' + \  
247         ':subobjects (\n' + \  
248         '' + '\n' + \  
249         ')\n' + \  
250         ')\n'  
251 # aml_entity = ''  
252  
253     entity_counter += 1  
254     write_aml_file(aml_entity_prefix + entity + aml_entity_suffix,  
255                   ↪ aml_header + \  
256                       paragraph_dict['ENTITY ' + entity] + \  
257                       ';;\n' + \  
258                       '\n' + \  
259                       '\n' + \  
260                       '(in-package :aml)\n' + \  
261                       '\n' + \  
262                       aml_type, 'types/')  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305 print str(entity_counter) + ' entities written to AML.' #@TODO: Add  
306     ↪ counter for number of entities
```

Listing C.10: Script that writes the EXPRESS entities from the parser to AML classes.
(File: schema_write_aml.py)

C.3.3. P21 Parser

p21_lexer.py

```
4 # STEP Part 21 Parser
5 #
6 # Copyright (c) 2011, Thomas Paviot (tpaviot@gmail.com)
7 # Copyright (c) 2014, Christopher HORLER (cshorler@googlemail.com)
8 #
9 # All rights reserved.
10 #
11 # This file is part of the StepClassLibrary (SCL).
12 #
13 # Redistribution and use in source and binary forms, with or without
14 # modification, are permitted provided that the following conditions
15 #   ↪ are met:
16 #
17 #   Redistributions of source code must retain the above copyright
18 #   ↪ notice,
19 #   this list of conditions and the following disclaimer.
20 #
21 #   Redistributions in binary form must reproduce the above copyright
22 #   ↪ notice,
23 #   this list of conditions and the following disclaimer in the
24 #   ↪ documentation
25 #   and/or other materials provided with the distribution.
26 #
27 #   Neither the name of the <ORGANIZATION> nor the names of its
28 #   ↪ contributors may
29 #   be used to endorse or promote products derived from this software
30 #   ↪ without
31 #   specific prior written permission.
32 #
33 # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "
34 #   ↪ AS IS"
35 # AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
36 #   ↪ THE
37 # IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
38 #   ↪ PURPOSE
39 # ARE DISCLAIMED.
```



```
31 # IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
    ↪ ANY
32 # DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
    ↪ DAMAGES
33 # (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
    ↪ SERVICES;
34 # LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
    ↪ CAUSED AND
35 # ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
    ↪ TORT
36 # (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
    ↪ OF
37 # THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
38
39 import logging
40 import ply.lex as lex
41 import ply.yacc as yacc
42 from ply.lex import LexError
43 import os.path#
44
45
46 logger = logging.getLogger(__name__)
47
48 # ensure Python 2.6 compatibility
49 if not hasattr(logging, 'NullHandler'):
50     class NullHandler(logging.Handler):
51         def handle(self, record):
52             pass
53         def emit(self, record):
54             pass
55         def createLock(self):
56             self.lock = None
57
58     setattr(logging, 'NullHandler', NullHandler)
59
60 logger.addHandler(logging.NullHandler())
61
62 #####
63 # Common Code for Lexer / Parser
64 #####
```

```
65 base_tokens = ['INTEGER', 'REAL', 'USER_DEFINED_KEYWORD', '
    ↪ STANDARD_KEYWORD', 'STRING', 'BINARY',
66             'ENTITY_INSTANCE_NAME', 'ENUMERATION', 'PART21_END', '
    ↪ PART21_START', 'HEADER_SEC',
67             'ENDSEC', 'DATA']
68
69 #####
70 # Lexer
71 #####
72 class P21_Lexer(object):
73     tokens = list(base_tokens)
74     states = (('slurp', 'exclusive'),)
75
76     def __init__(self, debug=0, optimize=0, compatibility_mode=False,
    ↪ header_limit=4096):
77         self.base_tokens = list(base_tokens)
78         self.schema_dict = {}
79         self.active_schema = {}
80         self.input_length = 0
81         self.compatibility_mode = compatibility_mode
82         self.header_limit = header_limit
83         self.lexer = lex.lex(module=self, debug=debug, debuglog=logger,
    ↪ optimize=optimize,
84                               errorlog=logger)
85         self.reset()
86
87     def __getattr__(self, name):
88         if name == 'lineno':
89             return self.lexer.lineno
90         elif name == 'lexpos':
91             return self.lexer.lexpos
92         else:
93             raise AttributeError
94
95     def input(self, s):
96         self.lexer.input(s)
97         self.input_length += len(s)
98
99     def reset(self):
100         self.lexer.lineno = 1
101         self.lexer.begin('slurp')
```

```
102
103     def token(self):
104         try:
105             return next(self.lexer)
106         except StopIteration:
107             return None
108
109     def activate_schema(self, schema_name):
110         if schema_name in self.schema_dict:
111             self.active_schema = self.schema_dict[schema_name]
112         else:
113             raise ValueError('schema not registered')
114
115     def register_schema(self, schema_name, entities):
116         if schema_name in self.schema_dict:
117             raise ValueError('schema already registered')
118
119         for k in entities:
120             if k in self.base_tokens: raise ValueError('schema cannot
121                                     ↪ override base_tokens')
122
123         if isinstance(entities, list):
124             entities = dict((k, k) for k in entities)
125
126         self.schema_dict[schema_name] = entities
127
128     def t_slurp_PART21_START(self, t):
129         r'ISO-10303-21;'
130         t.lexer.begin('INITIAL')
131         return t
132
133     def t_slurp_error(self, t):
134         offset = t.value.find('\nISO-10303-21;', 0, self.header_limit)
135         if offset == -1 and self.header_limit < len(t.value): # not
136             ↪ found within header_limit
137             raise LexError("Scanning error. try increasing lexer
138                             ↪ header_limit parameter",
139                             "{0}...".format(t.value[0:20]))
140         elif offset == -1: # not found before EOF
141             t.lexer.lexpos = self.input_length
142         else: # found ISO-10303-21;
```

```
140         offset += 1 # also skip the \n
141         t.lexer.lineno += t.value[0:offset].count('\n')
142         t.lexer.skip(offset)
143
144     # Comment (ignored)
145     def t_COMMENT(self, t):
146         r'\/*(.|\n)*?\/'
147         t.lexer.lineno += t.value.count('\n')
148
149     def t_PART21_END(self, t):
150         r'END-ISO-10303-21;'
151         t.lexer.begin('slurp')
152         return t
153
154     def t_HEADER_SEC(self, t):
155         r'HEADER;'
156         return t
157
158     def t_ENDSEC(self, t):
159         r'ENDSEC;'
160         return t
161
162     # Keywords
163     def t_STANDARD_KEYWORD(self, t):
164         r'(?:!)[A-Za-z_][0-9A-Za-z_]*'
165         if self.compatibility_mode:
166             t.value = t.value.upper()
167         elif not t.value.isupper():
168             raise LexError('Scanning error. Mixed/lower case keyword
169                 ↪ detected, please use compatibility_mode=True', t.
170                 ↪ value)
171
172         if t.value in self.base_tokens:
173             t.type = t.value
174         elif t.value in self.active_schema:
175             t.type = self.active_schema[t.value]
176         elif t.value.startswith('!'):
177             t.type = 'USER_DEFINED_KEYWORD'
178         return t
179
180     def t_newline(self, t):
```

```
179         r'\n+'
180         t.lexer.lineno += len(t.value)
181
182     # Simple Data Types
183     def t_REAL(self, t):
184         r'[+-]*[0-9][0-9]*\.[0-9]*(?:E[+-]*[0-9][0-9]*)?'
185         t.value = float(t.value)
186         return t
187
188     def t_INTEGER(self, t):
189         r'[+-]*[0-9][0-9]*'
190         t.value = int(t.value)
191         return t
192
193     def t_STRING(self, t):
194         r"'(?:[!\"*$%&.#+, \- () ? / : ; <=> @ { } | ^ ` ~ 0-9 a-z A-Z _ \ ] | '' ) *'"
195         t.value = t.value[1:-1]
196         return t
197
198     def t_BINARY(self, t):
199         r'"[0-3][0-9A-F]*"'
200         try:
201             t.value = int(t.value[2:-1], base=16)
202         except ValueError:
203             t.value = None
204         return t
205
206     t_ENTITY_INSTANCE_NAME = r'\#[0-9]+'
207     t_ENUMERATION = r'\.[A-Z_][A-Z0-9_]*\.'
208
209     # Punctuation
210     literals = ' (= ; , * $ '
211
212     t_ANY_ignore = ' \t'
213
214
215     #####
216     # Simple Model
217     #####
218     class P21File:
219         def __init__(self, header, *sections):
```

```
220         self.header = header
221         self.sections = list(*sections)
222
223 class P21Header:
224     def __init__(self, file_description, file_name, file_schema):
225         self.file_description = file_description
226         self.file_name = file_name
227         self.file_schema = file_schema
228         self.extra_headers = []
229
230 class HeaderEntity:
231     def __init__(self, type_name, *params):
232         self.type_name = type_name
233         self.params = list(params) if params else []
234
235 class Section:
236     def __init__(self, entities):
237         self.entities = entities
238
239 class SimpleEntity:
240     def __init__(self, ref, type_name, *params):
241         self.ref = ref
242         self.type_name = type_name
243         self.params = list(params) if params else []
244
245 class ComplexEntity:
246     def __init__(self, ref, *params):
247         self.ref = ref
248         self.params = list(params) if params else []
249
250 class TypedParameter:
251     def __init__(self, type_name, *params):
252         self.type_name = type_name
253         self.params = list(params) if params else None
```

Listing C.11: STEP file lexer.¹(File: p21_lexer.py)

¹Based on <https://github.com/stepcode/stepcode/wiki/python-generator>, last accessed: 2016-03-20

p21_parser.py

```
39 from p21_lexer import *
40 import os.path#
41
42
43 P21_input_file = '../..//02_p21_file/output/example_ap242.stp'
44 #####
45 # Simple Model
46 #####
47 class P21File:
48     def __init__(self, header, *sections):
49         self.header = header
50         self.sections = list(*sections)
51
52 class P21Header:
53     def __init__(self, file_description, file_name, file_schema):
54         self.file_description = file_description
55         self.file_name = file_name
56         self.file_schema = file_schema
57         self.extra_headers = []
58
59 class HeaderEntity:
60     def __init__(self, type_name, *params):
61         self.type_name = type_name
62         self.params = list(params) if params else []
63
64 class Section:
65     def __init__(self, entities):
66         self.entities = entities
67
68 class SimpleEntity:
69     def __init__(self, ref, type_name, *params):
70         self.ref = ref
71         self.type_name = type_name
72         self.params = list(params) if params else []
73
74 class ComplexEntity:
75     def __init__(self, ref, *params):
76         self.ref = ref
77         self.params = list(params) if params else []
```

```
108
109 class TypedParameter:
110     def __init__(self, type_name, *params):
111         self.type_name = type_name
112         self.params = list(params) if params else None
113
114 #####
115 # Parser
116 #####
117 class P21_Parser(object):
118     tokens = list(base_tokens)
119     start = 'exchange_file'
120
121     def __init__(self, lexer=None, debug=0):
122         self.lexer = lexer if lexer else P21_Lexer()
123
124         try: self.tokens = lexer.tokens
125         except AttributeError: pass
126
127         self.parser = yacc.yacc(module=self, debug=debug, debuglog=
128             ↪ logger, errorlog=logger)
129         self.reset()
130
131     def parse(self, p21_data, **kwargs):
132         #TODO: will probably need to change this function if the lexer
133         ↪ is ever to support t_eof
134         self.lexer.reset()
135         self.lexer.input(p21_data)
136
137         if 'debug' in kwargs:
138             result = self.parser.parse(lexer=self.lexer, debug=logger,
139                 ↪ ** dict((k, v) for k, v in
140                 ↪ kwargs.iteritems() if k
141                 ↪ != 'debug'))
142         else:
143             result = self.parser.parse(lexer=self.lexer, **kwargs)
144         return result
145
146     def reset(self):
147         self.refs = {}
148         self.is_in_exchange_structure = False
```



```
145
146 def p_exchange_file(self, p):
147     """exchange_file : check_p21_start_token header_section
148         ↪ data_section_list check_p21_end_token"""
149     p[0] = P21File(p[2], p[3])
150
151 def p_check_start_token(self, p):
152     """check_p21_start_token : PART21_START"""
153     self.is_in_exchange_structure = True
154     p[0] = p[1]
155
156 def p_check_end_token(self, p):
157     """check_p21_end_token : PART21_END"""
158     self.is_in_exchange_structure = False
159     p[0] = p[1]
160
161 # TODO: Specialise the first 3 header entities
162 def p_header_section(self, p):
163     """header_section : HEADER_SEC header_entity header_entity
164         ↪ header_entity ENDSEC"""
165     p[0] = P21Header(p[2], p[3], p[4])
166
167 def p_header_section_with_entity_list(self, p):
168     """header_section : HEADER_SEC header_entity header_entity
169         ↪ header_entity header_entity_list ENDSEC"""
170     p[0] = P21Header(p[2], p[3], p[4])
171     p[0].extra_headers.extend(p[5])
172
173 def p_header_entity(self, p):
174     """header_entity : keyword '(' parameter_list ')' ';'"""
175     p[0] = HeaderEntity(p[1], p[3])
176
177 def p_check_entity_instance_name(self, p):
178     """check_entity_instance_name : ENTITY_INSTANCE_NAME"""
179     if p[1] in self.refs:
180         logger.error('Line: {0}, SyntaxError - Duplicate Entity
181             ↪ Instance Name: {1}'.format(p.lineno(1), p[1]))
182         raise SyntaxError
183     else:
184         self.refs[p[1]] = None
185     p[0] = p[1]
```

```
182
183 def p_simple_entity_instance(self, p):
184     """simple_entity_instance : check_entity_instance_name '='
185         ↪ simple_record ';' """
186     p[0] = SimpleEntity(p[1], *p[3])
187
188 def p_entity_instance_error(self, p):
189     """simple_entity_instance : error '=' simple_record ';'
190     complex_entity_instance : error '=' subsuper_record ';' """
191     pass
192
193 def p_complex_entity_instance(self, p):
194     """complex_entity_instance : check_entity_instance_name '='
195         ↪ subsuper_record ';' """
196     #p[0] = ComplexEntity(p[1], p[3]) # @Todo: Ignored for now,
197         ↪ throws errors in populate.aml
198
199 def p_subsuper_record(self, p):
200     """subsuper_record : '(' simple_record_list ')' """
201     p[0] = [TypedParameter(*x) for x in p[2]]
202
203 def p_data_section_list(self, p):
204     """data_section_list : data_section_list data_section
205         | data_section """
206     try: p[0] = p[1] + [p[2],]
207     except IndexError: p[0] = [p[1],]
208
209 def p_header_entity_list(self, p):
210     """header_entity_list : header_entity_list header_entity
211         | header_entity """
212     try: p[0] = p[1] + [p[2],]
213     except IndexError: p[0] = [p[1],]
214
215 def p_parameter_list(self, p):
216     """parameter_list : parameter_list ',' parameter
217         | parameter """
218     try: p[0] = p[1] + [p[3],]
219     except IndexError: p[0] = [p[1],]
220
221 def p_keyword(self, p):
222     """keyword : USER_DEFINED_KEYWORD
```

```
220         | STANDARD_KEYWORD"""
221     p[0] = p[1]
222
223     def p_parameter_simple(self, p):
224         """parameter : STRING
225             | INTEGER
226             | REAL
227             | ENTITY_INSTANCE_NAME
228             | ENUMERATION
229             | BINARY
230             | '*'
231             | '$'
232             | typed_parameter
233             | list_parameter"""
234     p[0] = p[1]
235
236     def p_list_parameter(self, p):
237         """list_parameter : '(' parameter_list ')'"""
238     p[0] = p[2]
239
240     def p_typed_parameter(self, p):
241         """typed_parameter : keyword '(' parameter ')'"""
242     p[0] = TypedParameter(p[1], p[3])
243
244     def p_parameter_empty_list(self, p):
245         """parameter : '(' ')'"""
246     p[0] = []
247
248     def p_data_start(self, p):
249         """data_start : DATA '(' parameter_list ')' ';'"""
250     pass
251
252     def p_data_start_empty(self, p):
253         """data_start : DATA '(' ')' ';'
254             | DATA ';'"""
255     pass
256
257     def p_data_section(self, p):
258         """data_section : data_start entity_instance_list ENDSEC"""
259     p[0] = Section(p[2])
260
```

```
261     def p_entity_instance_list(self, p):
262         """entity_instance_list : entity_instance_list entity_instance
263             | entity_instance"""
264         try: p[0] = p[1] + [p[2],]
265         except IndexError: p[0] = [p[1],]
266
267     def p_entity_instance_list_empty(self, p):
268         """entity_instance_list : empty"""
269         p[0] = []
270
271     def p_entity_instance(self, p):
272         """entity_instance : simple_entity_instance
273             | complex_entity_instance"""
274         p[0] = p[1]
275
276
277     def p_simple_record_empty(self, p):
278         """simple_record : keyword '(' ')'"""
279         p[0] = (p[1], [])
280
281     def p_simple_record_with_params(self, p):
282         """simple_record : keyword '(' parameter_list ')'"""
283         p[0] = (p[1], p[3])
284
285     def p_simple_record_list(self, p):
286         """simple_record_list : simple_record_list simple_record
287             | simple_record"""
288         try: p[0] = p[1] + [p[2],]
289         except IndexError: p[0] = [p[1],]
290
291     def p_empty(self, p):
292         """empty :"""
293         pass
294
295
296 def test_debug():
297     import os.path
298     import gc #
299
300     logging.basicConfig()
301     logger.setLevel(logging.DEBUG)
```

```
302
303     parser = P21_Parser()
304     parser.reset()
305
306     logger.info("***** parser debug *****")
307
308     p = input_file
309     with open(p, 'rU') as f:
310         s = f.read()
311         try:
312             print 'P21Object instanciated'
313             result = parser.parse(s)#, debug=1)
314             print result
315             print result.header
316         except SystemExit:
317             pass
318
319     logger.info("***** finished *****")
320
321 def test():
322     import os, os.path, itertools, codecs
323
324
325     logging.basicConfig()
326     logger.setLevel(logging.INFO)
327
328     parser = P21_Parser()
329     compat_list = []
330
331     def parse_check(p):
332         logger.info("processing {0}".format(p))
333         parser.reset()
334         with open(p, 'rU') as f:
335             iso_wrapper = codecs.EncodedFile(f, 'iso-8859-1')
336             s = iso_wrapper.read()
337             parser.parse(s)
338
339     logger.info("***** standard test *****")
340     for d, _, files in os.walk(os.path.expanduser('~\Documents\02_Work
        ↪ \01_NTNU\01_Masterthesis\05_STEP\01_STEPcode')): #change
        ↪ path!
```

```
341     for f in itertools.ifilter(lambda x: x.endswith('.stp'), files)
342         ↪ :
343         p = os.path.join(d, f)
344         try:
345             parse_check(p)
346         except LexError:
347             logger.exception('Lexer issue, adding {0} to
348                 ↪ compatibility test list'.format(os.path.basename
349                 ↪ (p)))
350             compat_list.append(p)
351
352     lexer = P21_Lexer(compatibility_mode=True)
353     parser = P21_Parser(lexer=lexer)
354
355     logger.info("***** compatibility test *****")
356     for p in compat_list:
357         parse_check(p)
358
359     logger.info("***** finished *****")
```

Listing C.12: STEP file parser.²(File: p21_parser.py)

p21_populate.py

```
1 from p21_parser import *
2 #from write_aml import *
3
4 logging.basicConfig()
5 logger.setLevel(logging.DEBUG)
6
7 parser = P21_Parser()
8 parser.reset()
9
10 P21_input_file = '../..//02_p21_file/out/example_ap242.stp'
11
12 p = P21_input_file
13 with open(p, 'rU') as f:
14     s = f.read()
15     try:
```

²Based on <https://github.com/stepcode/stepcode/wiki/python-generator>, last accessed: 2016-03-20

```
16     P21_result = parser.parse(s)#, debug=1)
17     print P21_result
18     print P21_result.header.file_name
19     except SystemExit:
20         pass
21
22 logger.info("***** finished *****")
23
24
25 reference_dict = {}
26 referenced_by_dict = {}
27 aml_populate = ''
28 populate_hash_table = 'populate_hash'
29
30 def convert_to_aml(input, aml_populate):
31
32     def convert_string(str, aml_populate):
33         if str.startswith('#'):
34             aml_populate += '\\' + str + ' '
35         elif str == '$':
36             aml_populate += 'nil '
37         elif str == '.T.':
38             aml_populate += '\\t '
39         elif str == '.F.':
40             aml_populate += 'nil '
41         else:
42             aml_populate += '"' + str + '" '
43         return aml_populate
44
45     if type(input).__name__ == 'str':
46         aml_populate = convert_string(input, aml_populate)
47     elif type(input).__name__ == 'list':
48         aml_populate += '(list '
49         for elem in param:
50             if type(elem).__name__ == 'str':
51                 aml_populate = convert_string(elem, aml_populate)
52             elif type(elem).__name__ == 'list':
53                 aml_populate = convert_to_aml(elem, aml_populate)
54             elif type(elem).__name__ == 'float':
55                 aml_populate += str(elem) + ' '
56             elif type(elem).__name__ == 'int':
```

```
57         aml_populate += str(elem) + ' '
58     else:
59         print 'unknown type: ' + type(elem).__name__
60         aml_populate += ' ) '
61 elif type(input).__name__ == 'float':
62     aml_populate += str(input) + ' '
63 elif type(input).__name__ == 'int':
64     aml_populate += str(input) + ' '
65 else:
66     print 'unknown type: ' + type(input).__name__
67 return aml_populate
68
69 for entity in P21_result.sections[0].entities:
70     print entity.ref
71     referenced_by_dict[entity.ref] = []
72
73
74 for i, entity in enumerate(P21_result.sections[0].entities):
75
76     aml_populate += '(setf (gethash \'' + entity.ref + ' ' +
77         ↪ populate_hash_table + \
78         ') \'' + entity.type_name.lower() + '" (list '
79
80     reference_dict[entity.ref] = []
81
82     # write references into dict
83     for param in entity.params[0]:
84         aml_populate = convert_to_aml(param, aml_populate)
85         if type(param).__name__ == 'str':
86             if param.startswith('#'):
87                 reference_dict[entity.ref] += [param]
88                 referenced_by_dict[param] += [entity.ref]
89         if type(param).__name__ == 'list':
90             for elem in param:
91                 if type(elem).__name__ == 'str':
92                     if elem.startswith('#'):
93                         reference_dict[entity.ref] += [elem]
94                         referenced_by_dict[elem] += [entity.ref]
95
96     print reference_dict[entity.ref]
```



```
97     aml_populate += '))\n'
98
99
100 def find_roots():
101     roots = []
102     for entity in referenced_by_dict:
103         if referenced_by_dict[entity] == []:
104             roots += [entity]
105     return roots
106
107 roots = find_roots()
108
109 def write_file(path, cont):
110     with open("../01_binding/out/systems/main-system/sources/" +
111             ↪ path, 'w') as f:
112         f.write(cont)
113
114 root_list = ''
115 for root in roots:
116     root_list += '\\ ' + root + ' '
117
118 root_list = root_list[:-1]
119
120 #(defvar root_entities (list "" + root_list + ""))
121 aml_header = ""(in-package :aml)
122
123 (setf root_entities (list "" + root_list + ""))
124
125 (defvar populate_hash (make-hash-table :test 'equal))
126 ""
127
128 write_file('populate.aml', aml_header + aml_populate)
```

Listing C.13: Python script that generates the transfer AML file. (File: p21_populate.py)

D. WSN rules EXPRESS language

The railroad diagrams of the EXPRESS language WSN (similar to EBNF) rules that are defined in ISO 10303–11:2004 are generated with the EBNF2PS tool by Franklin Chen.¹

```
1 $ ebnf2ps -verbose -titleFont Palatino-BoldItalic -ntFont Palatino-Bold
   ↪ -ntBoxColor RoyalBlue -tFont Courier -tColor White -tBg
   ↪ RoyalBlue -fatLineWidth 50 +simplify 10303-11.bnf '.*'
```

Listing D.1: ebnf2ps tool executed on Linux.

The Encapsulated Postscript (.eps) files are converted to PDF files (listing D.2) and a L^AT_EX file that includes all the PDFs is generated with the help of the Python script shown in listing D.3. The production rules are linked to each other to enable a fast traversing of the rules in the PDF version of the thesis.

```
1 forfiles /m *.eps /C "cmd /c echo @file @fname.pdf"
2 forfiles /m *.eps /C "cmd /c epstopdf @file --outfile=@fname.pdf"
```

Listing D.2: Batch conversion from eps to pdf file.

```
1 import os
2 import re
6 tex = ''
7 prod_dict = {}
8
9 def write_tex(tex, line):
10     tex_line = line.replace('\\', '\\textbackslash ').replace('_', '\_')
   ↪ ).replace('$', '\$').replace('%', '\%')
11     tex_line = tex_line.replace('|', '\\textbar\\enspace').replace('#',
   ↪ '\#').replace('&', '\&')
12     tex_line = tex_line.replace('{', '\\{').replace('}', '\\}')
13     tex_line = tex_line.replace('~', '\\textasciitilde').replace('^',
   ↪ '\\textasciicircum')
14     tex_line = tex_line.replace('=', '$=$').replace('<', '$<$').replace
   ↪ ('>', '$>$').replace('$$', '')
15
16     for word in tex_line.split():
17         done = [tex_line.split()[1],]
```

¹<https://github.com/FranklinChen/Ebnf2ps>, last accessed: 2016-04-02

```

18     if re.sub(r'\\',r'',word) in prod_dict and word not in done:
19         done += [word]
20         tex_line = tex_line.replace(' ' + word + ' ', ' \hyperref[
                ↪ fig:ebnf_' + re.sub(r'\\',r'',word) + ']{\textbf{'
                ↪ + word + '}} ')
21
22     # change SUBTYPE_CONSTRAINT and TOTAL_OVER manually to CAPS_XXX,
                ↪ because windows is not case sensitive and these two are
                ↪ defined as token and as production...
23     if line.split()[1] == 'SUBTYPE_CONSTRAINT':
24         pdf_name = 'CAPS_SUBTYPE_CONSTRAINT'
25     elif line.split()[1] == 'TOTAL_OVER':
26         pdf_name = 'CAPS_TOTAL_OVER'
27     else:
28         pdf_name = line.split()[1]
29
30     tex += """\begin{figure}[H]
31 \centering
32     \label{fig:ebnf_""" + line.split()[1] + """}
33     \captionsetup{justification=justified,singlelinecheck=false}
34     \caption*{""" + tex_line[:-1] + """}
35     \includegraphics[scale=1, trim=-10 0 0 15, clip]{src/pic/
                ↪ railroad_diagrams/pdf/""" + pdf_name + """.pdf}
36     \end{figure}
37
38 \vspace{-8mm}
39 \hrulefill
40 \vspace{-5mm}
41
42 """
43     return tex
44
45
46 f = open('iso_10303_11.bnf', 'r')
47 while 1:
48     line = f.readline()
49     if not line:break
50     prod_dict[line.split()[1]] = line.split()[0]
51 f.close()
52
53 f = open('iso_10303_11.bnf', 'r')

```

```
54 while 1:
55     line = f.readline()
56     if not line:break
57     tex = write_tex(tex, line)
58 f.close()
59
60 f = open('railroad_diagram.tex', 'w')
61 f.write(tex)
62 f.close()
63
64
65 ##for pdf in os.listdir("./pdf/"):
66 ##    if pdf.endswith(".pdf"):
67 ##        print(pdf)
```

Listing D.3: Python script that generates the TeX file with all the railroad diagrams. The productions are linked together with hyperrefs. (File: railroad2tex.py)

0 ABS = 'abs' .

→ abs →

1 ABSTRACT = 'abstract' .

→ abstract →

2 ACOS = 'acos' .

→ acos →

3 AGGREGATE = 'aggregate' .

→ aggregate →

4 ALIAS = 'alias' .

→ alias →

5 AND = 'and' .

→ and →

6 ANDOR = 'andor' .

→ andor →

7 ARRAY = 'array' .

→ array →

8 AS = 'as' .

→ as →

9 ASIN = 'asin' .

→ asin →

10 ATAN = 'atan' .

→ atan →

11 BAG = 'bag' .

→ bag →

12 BASED_ON = 'based_on' .

→ based_on →

13 BEGIN = 'begin' .

→ begin →

14 BINARY = 'binary' .

→ binary →

15 BLENGTH = 'blength' .

→ blength →

16 BOOLEAN = 'boolean' .

→ boolean →

17 BY = 'by' .

→ by →

18 CASE = 'case' .

→ case →

19 CONSTANT = 'constant' .

→ constant →

20 CONST_E = 'const_e' .

→ const_e →

21 COS = 'cos' .

→ cos →

22 DERIVE = 'derive' .

→ derive →

23 DIV = 'div' .

→ div →

24 ELSE = 'else' .

→ else →

25 END = 'end' .

→ end →

26 END_ALIAS = 'end_alias' .

→ end_alias →

27 END_CASE = 'end_case' .

→ end_case →

28 END_CONSTANT = 'end_constant' .

→ end_constant →

29 END_ENTITY = 'end_entity' .

→ end_entity →

30 END_FUNCTION = 'end_function' .

→ end_function →

31 END_IF = 'end_if' .

→ end_if →

32 END_LOCAL = 'end_local' .

→ end_local →

33 END_PROCEDURE = 'end_procedure' .

→ end_procedure →

34 END_REPEAT = 'end_repeat' .

→ end_repeat →

35 END_RULE = 'end_rule' .

→ end_rule →

36 END_SCHEMA = 'end_schema' .

→ end_schema →

37 END_SUBTYPE_CONSTRAINT = 'end_subtype_constraint' .

→ end_subtype_constraint →

38 END_TYPE = 'end_type' .

→ end_type →

39 ENTITY = 'entity' .

→ entity →

40 ENUMERATION = 'enumeration' .

→ enumeration →

41 ESCAPE = 'escape' .

→ escape →

42 EXISTS = 'exists' .

→ exists →

43 EXTENSIBLE = 'extensible' .

→ extensible →

44 EXP = 'exp' .

→ exp →

45 FALSE = 'false' .

→ false →

46 FIXED = 'fixed' .

→ fixed →

47 FOR = 'for' .

→ for →

48 FORMAT = 'format' .

→ format →

49 FROM = 'from' .

→ from →

50 FUNCTION = 'function' .

→ function →

51 GENERIC = 'generic' .

→ generic →

52 GENERIC_ENTITY = 'generic_entity' .

→ generic_entity →

53 HIBOUND = 'hibound' .

→ hibound →

54 HIINDEX = 'hiindex' .

→ hiindex →

55 IF = 'if' .

→ if →

56 IN = 'in' .

→ in →

57 INSERT = 'insert' .

→ insert →

58 INTEGER = 'integer' .

→ integer →

59 INVERSE = 'inverse' .

→ inverse →

60 LENGTH = 'length' .

→ length →

61 LIKE = 'like' .

→ like →

62 LIST = 'list' .

→ list →

63 LOBOUND = 'lobound' .

→ lobound →

64 LOCAL = 'local' .

→ local →

65 LOG = 'log' .

→ log →

66 LOG10 = 'log10' .

→ log10 →

67 LOG2 = 'log2' .

→ log2 →

68 LOGICAL = 'logical' .

→ logical →

69 LOINDEX = 'loindex' .

→ loindex →

70 MOD = 'mod' .

→ mod →

71 NOT = 'not' .

→ not →

72 NUMBER = 'number' .

→ number →

73 NVL = 'nvl' .

→ nvl →

74 ODD = 'odd' .

→ odd →

75 OF = 'of' .

→ of →

76 ONEOF = 'oneof' .

→ oneof →

77 OPTIONAL = 'optional' .

→ optional →

78 OR = 'or' .

→ or →

79 OTHERWISE = 'otherwise' .

→ otherwise →



80 PI = 'pi' .



81 PROCEDURE = 'procedure' .



82 QUERY = 'query' .



83 REAL = 'real' .



84 REFERENCE = 'reference' .



85 REMOVE = 'remove' .



86 RENAMED = 'renamed' .



87 REPEAT = 'repeat' .



88 RETURN = 'return' .



89 ROLESOF = 'rolesof' .



90 RULE = 'rule' .



91 SCHEMA = 'schema' .



92 SELECT = 'select' .



93 SELF = 'self' .



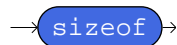
94 SET = 'set' .



95 SIN = 'sin' .



96 SIZEOF = 'sizeof' .



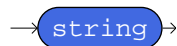
97 SKIP = 'skip' .



98 SQRT = 'sqrt' .



99 STRING = 'string' .



100 SUBTYPE = 'subtype' .

→ subtype →

101 SUBTYPE_CONSTRAINT = 'subtype_constraint' .

→ subtype_constraint →

102 SUPERTYPE = 'supertype' .

→ supertype →

103 TAN = 'tan' .

→ tan →

104 THEN = 'then' .

→ then →

105 TO = 'to' .

→ to →

106 TOTAL_OVER = 'total_over' .

→ total_over →

107 TRUE = 'true' .

→ true →

108 TYPE = 'type' .

→ type →

109 TYPEOF = 'typeof' .

→ typeof →

110 UNIQUE = 'unique' .

→ unique →

111 UNKNOWN = 'unknown' .

→ unknown →

112 UNTIL = 'until' .

→ until →

113 USE = 'use' .

→ use →

114 USEDIN = 'usedin' .

→ usedin →

115 VALUE = 'value' .

→ value →

116 VALUE_IN = 'value_in' .

→ value_in →

117 VALUE_UNIQUE = 'value_unique' .

→ value_unique →

118 VAR = 'var' .

→ var →

119 WHERE = 'where' .

→ where →

D WSN rules EXPRESS language

120 WHILE = 'while' .



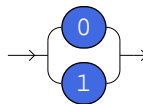
121 WITH = 'with' .



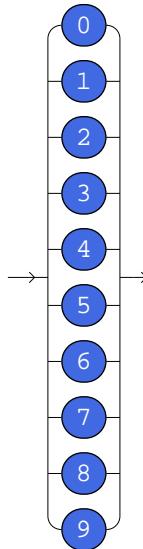
122 XOR = 'xor' .



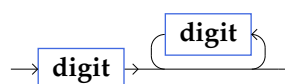
123 bit = '0' | '1' .



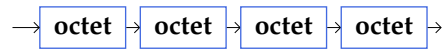
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .



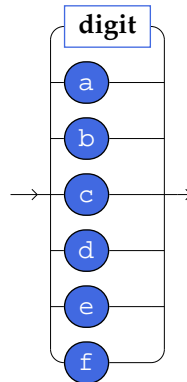
125 digits = **digit** { **digit** } .



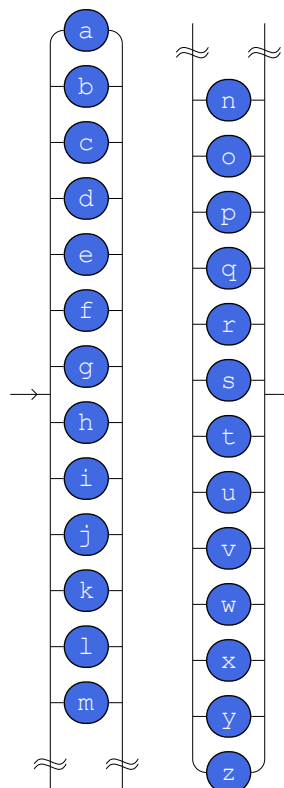
126 encoded_character = **octet octet octet octet** .



127 hex_digit = **digit** | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .



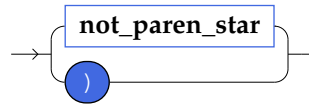
128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' .



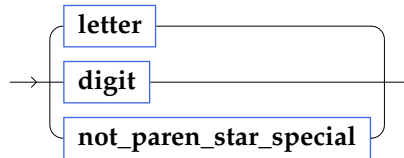
129 `lparen_then_not_lparen_star = '(' { '(' } not_lparen_star { not_lparen_star } .`



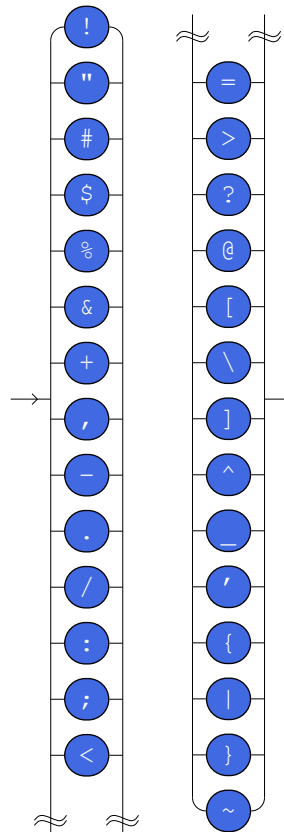
130 `not_lparen_star = not_paren_star | ')' .`



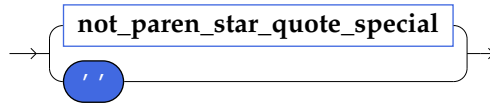
131 `not_paren_star = letter | digit | not_paren_star_special .`



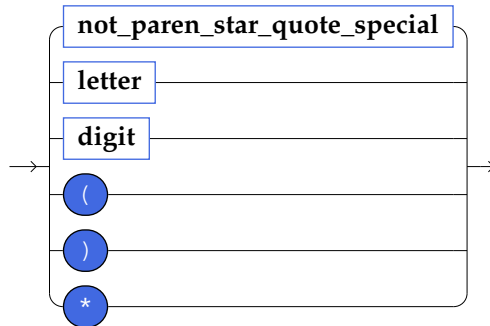
132 `not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' | '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' | '{' | '|' | '}' | '~' .`



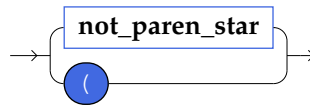
133 not_paren_star_special = not_paren_star_quote_special | """ .



134 not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .



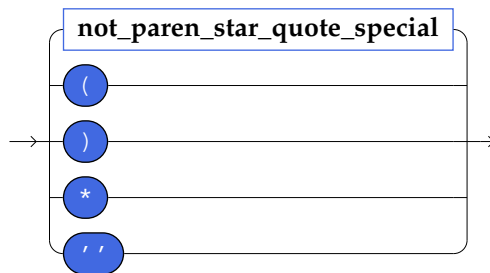
135 not_rparen_star = not_paren_star | '(' .



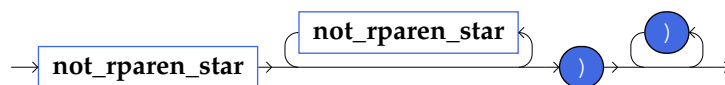
136 octet = hex_digit hex_digit .



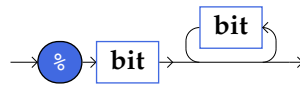
137 special = not_paren_star_quote_special | '(' | ')' | '*' | ' ' .



138 not_rparen_star_then_rparen = not_rparen_star { not_rparen_star } ')' { ')' } .



139 `binary_literal = '%' bit { bit } .`



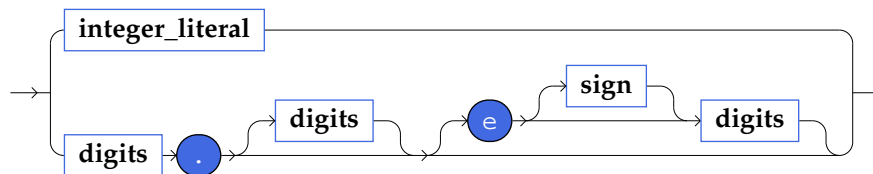
140 `encoded_string_literal = ''' encoded_character { encoded_character } ''' .`



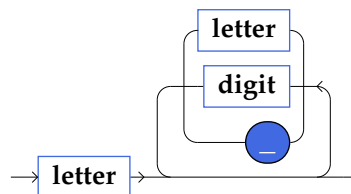
141 `integer_literal = digits .`



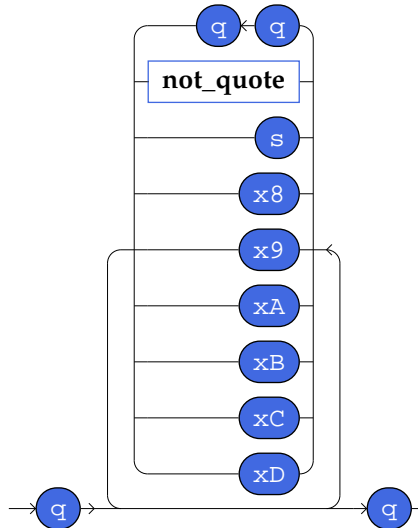
142 `real_literal = integer_literal | (digits '.' [digits] ['e' [sign] digits]) .`



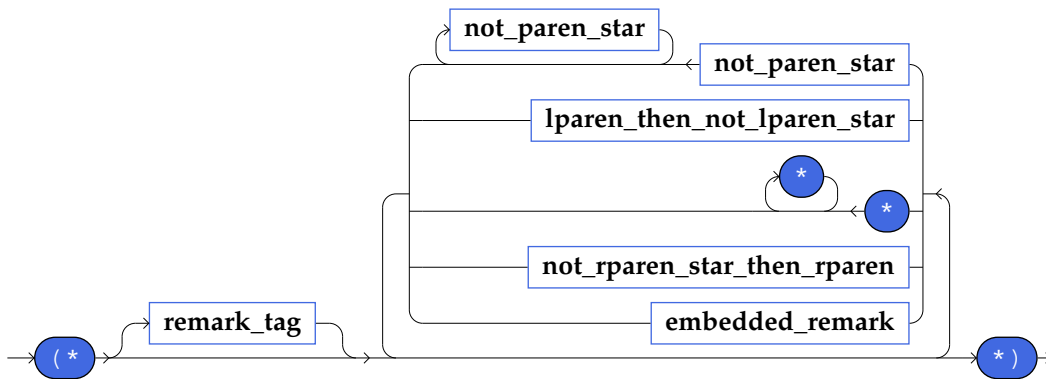
143 `simple_id = letter { letter | digit | '_' } .`



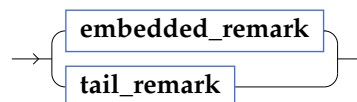
144 `simple_string_literal = \q { (\q \q) | not_quote | \s | \x8 | \x9 | \xA | \xB | \xC | \xD } \q .`



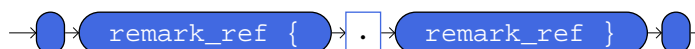
145 `embedded_remark = '(* [remark_tag] { (not_paren_star { not_paren_star }) | lparen_then_not_lparen_star | ('*' { '*' }) | not_rparen_star_then_rparen | embedded_remark } *)'` .



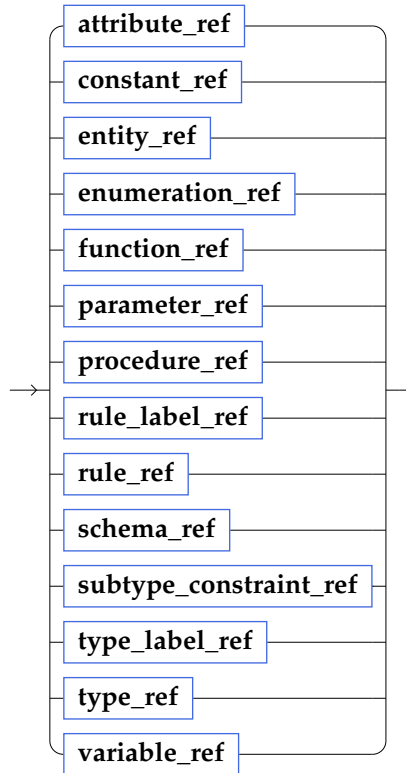
146 `remark = embedded_remark | tail_remark .`



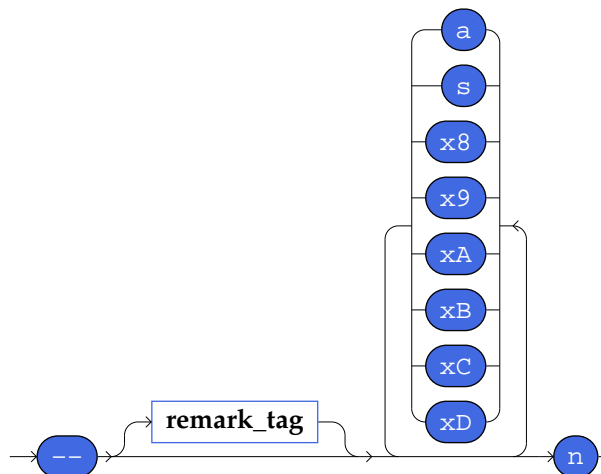
147 `remark_tag = ''' remark_ref { '.' remark_ref } ''' .`



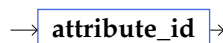
148 remark_ref = **attribute_ref** | **constant_ref** | **entity_ref** | **enumeration_ref** | **function_ref** | **parameter_ref** | **procedure_ref** | **rule_label_ref** | **rule_ref** | **schema_ref** | **subtype_constraint_ref** | **type_label_ref** | **type_ref** | **variable_ref** .



149 tail_remark = ' ' [**remark_tag**] { \a | \s | \x8 | \x9 | \xA | \xB | \xC | \xD } \n .



150 attribute_ref = **attribute_id** .



151 constant_ref = **constant_id** .

→ **constant_id** →

152 entity_ref = **entity_id** .

→ **entity_id** →

153 enumeration_ref = **enumeration_id** .

→ **enumeration_id** →

154 function_ref = **function_id** .

→ **function_id** →

155 parameter_ref = **parameter_id** .

→ **parameter_id** →

156 procedure_ref = **procedure_id** .

→ **procedure_id** →

157 rule_label_ref = **rule_label_id** .

→ **rule_label_id** →

158 rule_ref = **rule_id** .

→ **rule_id** →

159 schema_ref = **schema_id** .

→ **schema_id** →

160 subtype_constraint_ref = **subtype_constraint_id** .

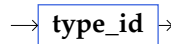
→ **subtype_constraint_id** →



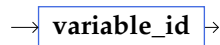
161 type_label_ref = **type_label_id** .



162 type_ref = **type_id** .



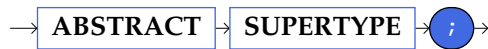
163 variable_ref = **variable_id** .



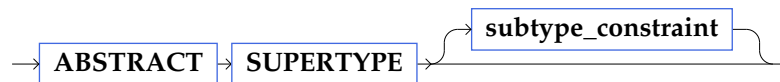
164 abstract_entity_declaration = **ABSTRACT** .



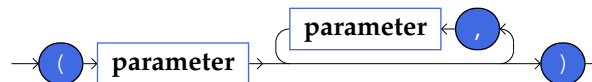
165 abstract_supertype = **ABSTRACT SUPERTYPE** ';' .



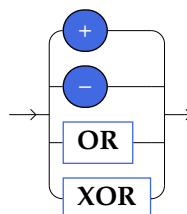
166 abstract_supertype_declaration = **ABSTRACT SUPERTYPE** [**subtype_constraint**] .



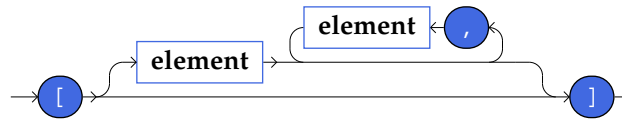
167 actual_parameter_list = '(' parameter { ',' parameter } ')'



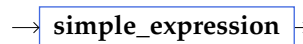
168 add_like_op = '+' | '-' | **OR** | **XOR** .



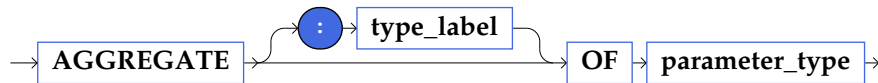
169 aggregate_initializer = '[' [element { ',' element }] ']' .



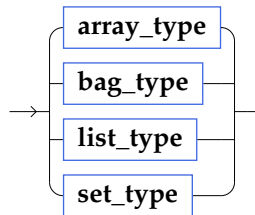
170 aggregate_source = simple_expression .



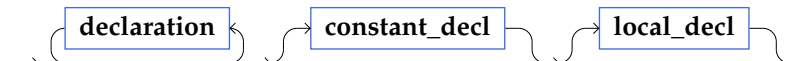
171 aggregate_type = AGGREGATE [':' type_label] OF parameter_type .



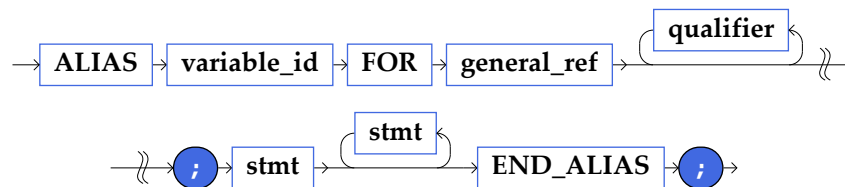
172 aggregation_types = array_type | bag_type | list_type | set_type .



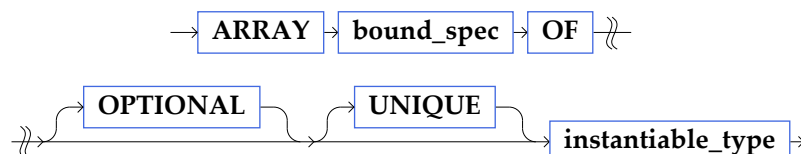
173 algorithm_head = { declaration } [constant_decl] [local_decl] .



174 alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';' stmt { stmt } END_ALIAS ';' .



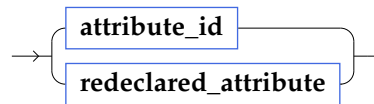
175 array_type = ARRAY bound_spec OF [OPTIONAL] [UNIQUE] instantiable_type .



176 assignment_stmt = **general_ref** { **qualifier** } ':=' **expression** ';' .



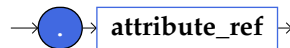
177 attribute_decl = **attribute_id** | **redeclared_attribute** .



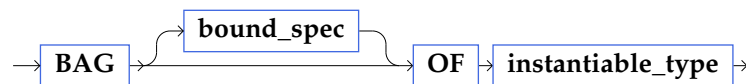
178 attribute_id = **simple_id** .



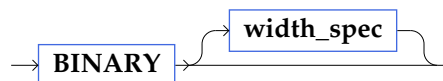
179 attribute_qualifier = '.' **attribute_ref** .



180 bag_type = **BAG** [**bound_spec**] **OF** **instantiable_type** .



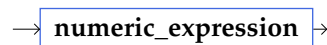
181 binary_type = **BINARY** [**width_spec**] .



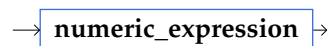
182 boolean_type = **BOOLEAN** .



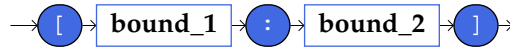
183 bound_1 = **numeric_expression** .



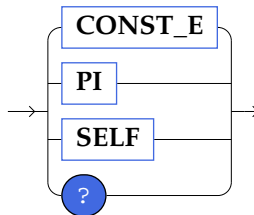
184 bound_2 = **numeric_expression** .



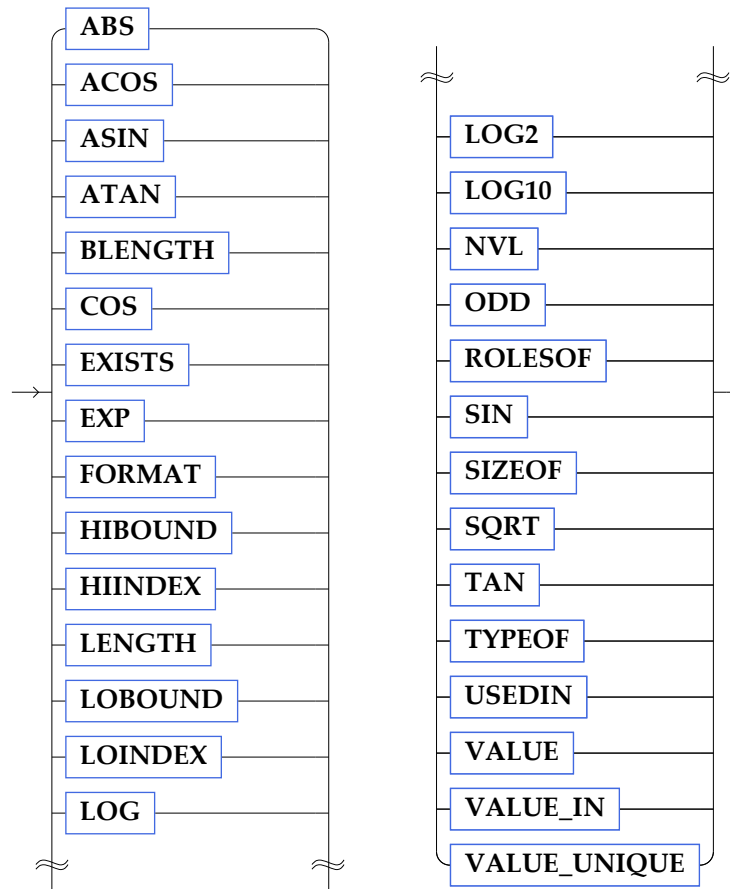
185 bound_spec = '[' bound_1 ':' bound_2 ']' .



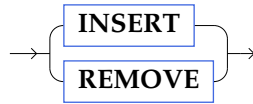
186 built_in_constant = CONST_E | PI | SELF | '?' .



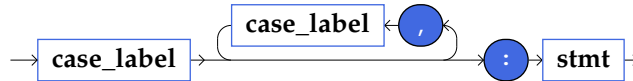
187 built_in_function = ABS | ACOS | ASIN | ATAN | BLENGTH | COS | EXISTS | EXP | FORMAT | HIBOUND | HIINDEX | LENGTH | LOBOUND | LOINDEX | LOG | LOG2 | LOG10 | NVL | ODD | ROLESOF | SIN | SIZEOF | SQRT | TAN | TYPEOF | USEDIN | VALUE | VALUE_IN | VALUE_UNIQUE .



188 built_in_procedure = **INSERT** | **REMOVE** .



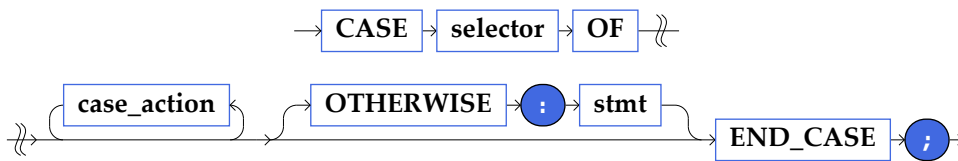
189 case_action = case_label { ',' case_label } ':' stmt .



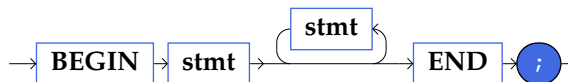
190 case_label = **expression** .



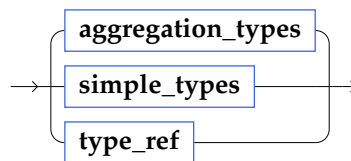
191 case_stmt = **CASE** selector **OF** { case_action } [**OTHERWISE** ':' stmt] **END_CASE** ';' .



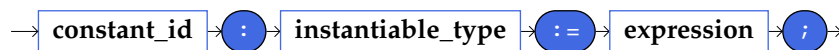
192 compound_stmt = **BEGIN** stmt { stmt } **END** ';' .



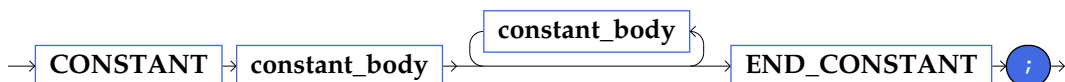
193 concrete_types = **aggregation_types** | **simple_types** | **type_ref** .



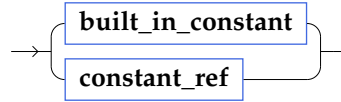
194 constant_body = **constant_id** ':' **instantiable_type** ':=' **expression** ';' .



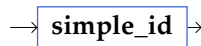
195 constant_decl = **CONSTANT** constant_body { constant_body } **END_CONSTANT** ';' .



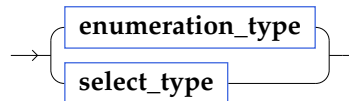
196 constant_factor = **built_in_constant** | **constant_ref** .



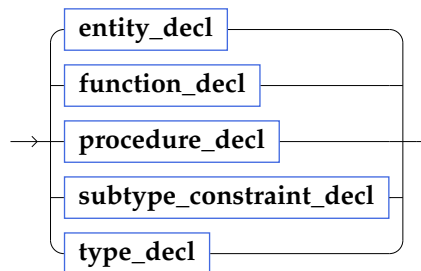
197 constant_id = **simple_id** .



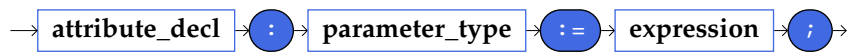
198 constructed_types = **enumeration_type** | **select_type** .



199 declaration = **entity_decl** | **function_decl** | **procedure_decl** | **subtype_constraint_decl** | **type_decl** .



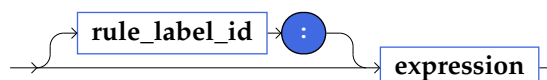
200 derived_attr = **attribute_decl** ':' **parameter_type** ':=' **expression** ';' .



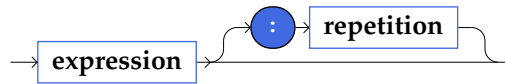
201 derive_clause = **DERIVE** **derived_attr** { **derived_attr** } .



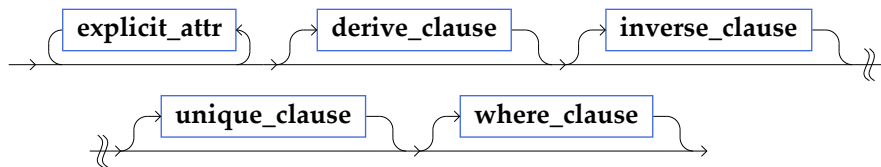
202 domain_rule = [**rule_label_id** ':'] **expression** .



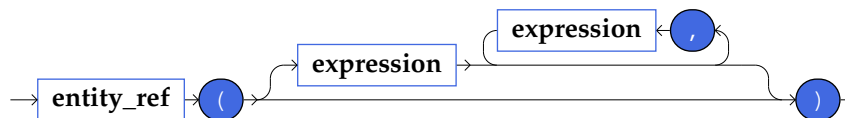
203 element = **expression** [':' repetition] .



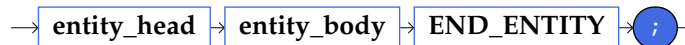
204 entity_body = { **explicit_attr** } [**derive_clause**] [**inverse_clause**] [**unique_clause**] [**where_clause**] .



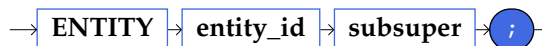
205 entity_constructor = **entity_ref** '(' [**expression** { ',' **expression** }] ')' .



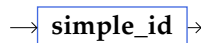
206 entity_decl = **entity_head** **entity_body** **END_ENTITY** ';' .



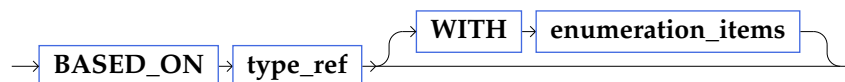
207 entity_head = **ENTITY** **entity_id** **subsuper** ';' .



208 entity_id = **simple_id** .



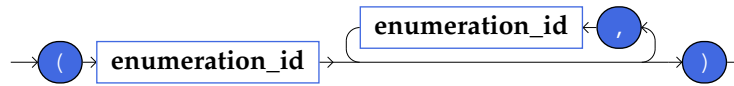
209 enumeration_extension = **BASED_ON** **type_ref** [**WITH** **enumeration_items**] .



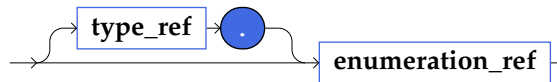
210 enumeration_id = **simple_id** .



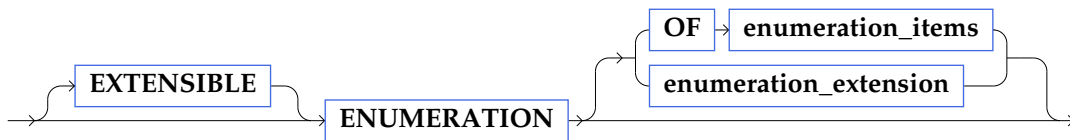
211 enumeration_items = '(' enumeration_id { ',' enumeration_id } ')'



212 enumeration_reference = [type_ref '?'] enumeration_ref



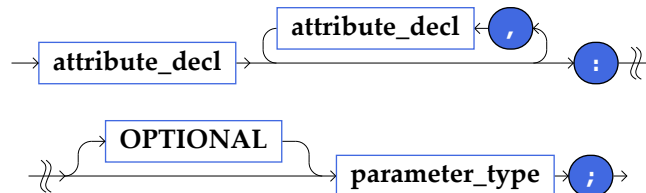
213 enumeration_type = [EXTENSIBLE] ENUMERATION [(OF enumeration_items) | enumeration_extension]



214 escape_stmt = ESCAPE ';' .



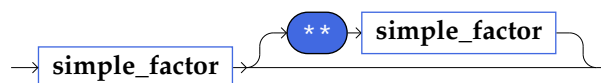
215 explicit_attr = attribute_decl { ',' attribute_decl } ':' [OPTIONAL] parameter_type ';' .



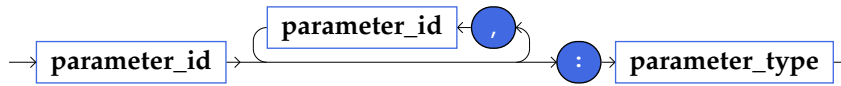
216 expression = simple_expression [rel_op_extended simple_expression] .



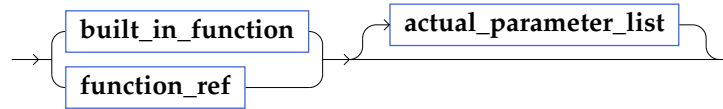
217 factor = simple_factor ['**' simple_factor] .



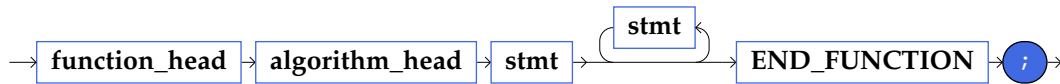
218 formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .



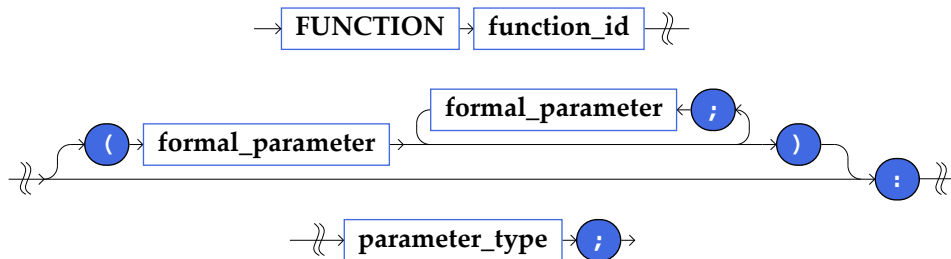
219 function_call = (built_in_function | function_ref) [actual_parameter_list] .



220 function_decl = function_head algorithm_head stmt { stmt } END_FUNCTION ';' .



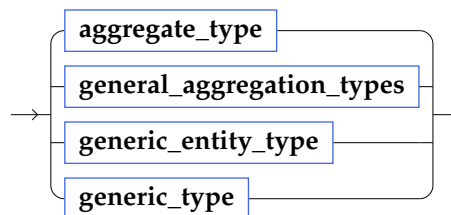
221 function_head = FUNCTION function_id ['(' formal_parameter { ';' formal_parameter } ')'] ':' parameter_type ';' .



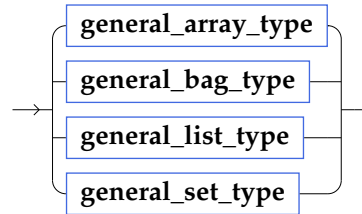
222 function_id = simple_id .



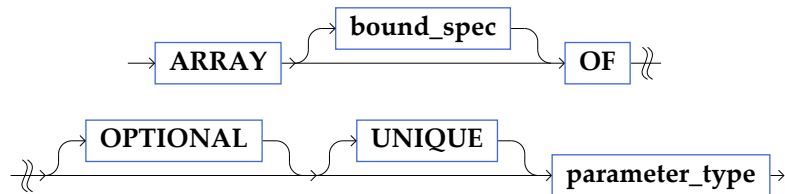
223 generalized_types = aggregate_type | general_aggregation_types | generic_entity_type | generic_type .



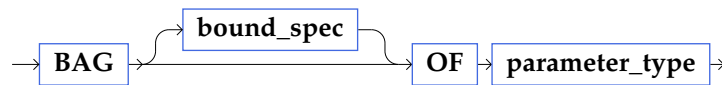
224 `general_aggregation_types = general_array_type | general_bag_type | general_list_type | general_set_type .`



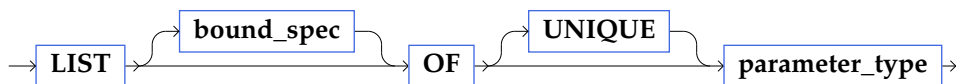
225 `general_array_type = ARRAY [bound_spec] OF [OPTIONAL] [UNIQUE] parameter_type .`



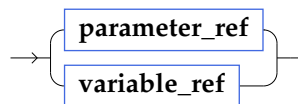
226 `general_bag_type = BAG [bound_spec] OF parameter_type .`



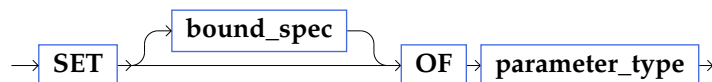
227 `general_list_type = LIST [bound_spec] OF [UNIQUE] parameter_type .`



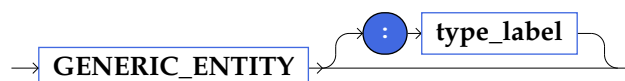
228 `general_ref = parameter_ref | variable_ref .`



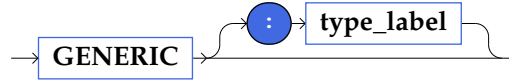
229 `general_set_type = SET [bound_spec] OF parameter_type .`



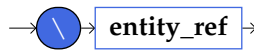
230 `generic_entity_type = GENERIC_ENTITY [':' type_label] .`



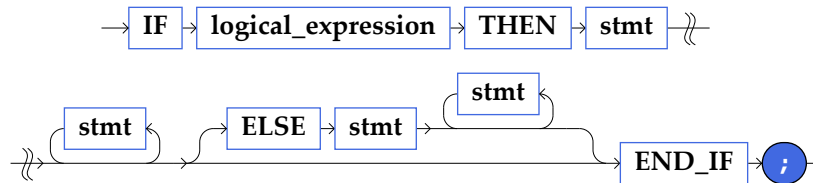
231 generic_type = **GENERIC** [':' type_label] .



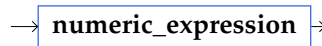
232 group_qualifier = '\' entity_ref .



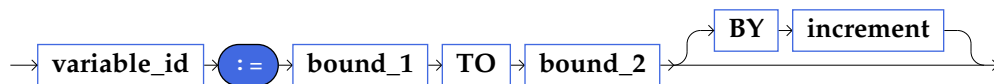
233 if_stmt = **IF** logical_expression **THEN** stmt { stmt } [**ELSE** stmt { stmt }] **END_IF** ';' .



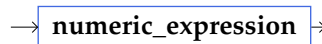
234 increment = **numeric_expression** .



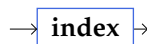
235 increment_control = variable_id ':= ' bound_1 **TO** bound_2 [**BY** increment] .



236 index = **numeric_expression** .



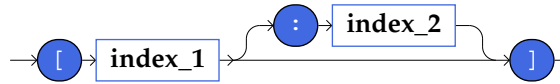
237 index_1 = **index** .



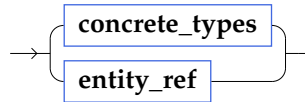
238 index_2 = **index** .



239 index_qualifier = '[' index_1 [':' index_2]]' .



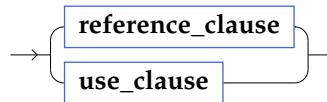
240 instantiable_type = concrete_types | entity_ref .



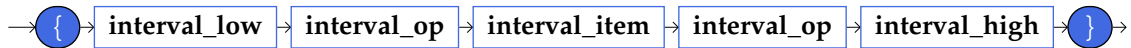
241 integer_type = INTEGER .



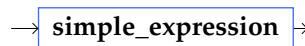
242 interface_specification = reference_clause | use_clause .



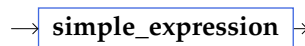
243 interval = '{' interval_low interval_op interval_item interval_op interval_high '}' .



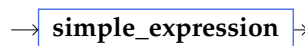
244 interval_high = simple_expression .



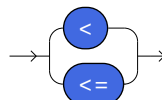
245 interval_item = simple_expression .



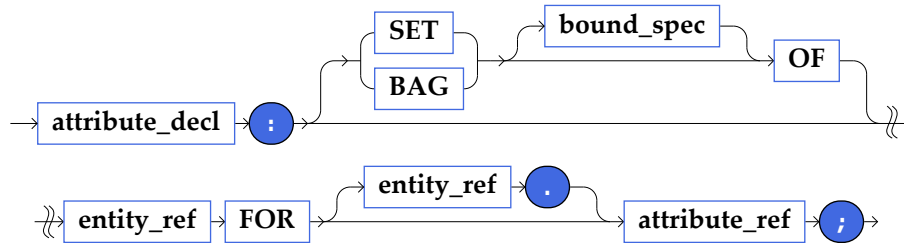
246 interval_low = simple_expression .



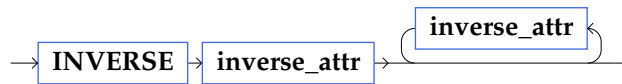
247 interval_op = '<' | '<=' .



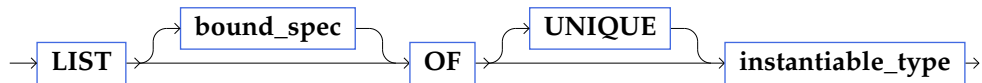
248 `inverse_attr = attribute_decl ':' [(SET | BAG) [bound_spec] OF] entity_ref FOR [entity_ref ','] attribute_ref ';' .`



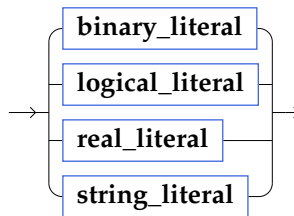
249 `inverse_clause = INVERSE inverse_attr { inverse_attr } .`



250 `list_type = LIST [bound_spec] OF [UNIQUE] instantiable_type .`



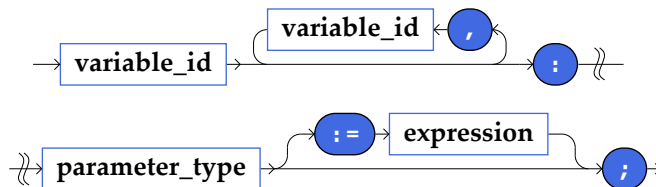
251 `literal = binary_literal | logical_literal | real_literal | string_literal .`



252 `local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' .`



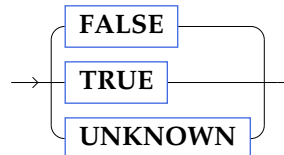
253 `local_variable = variable_id { ',' variable_id } ':' parameter_type [':' expression] ';' .`



254 logical_expression = **expression** .



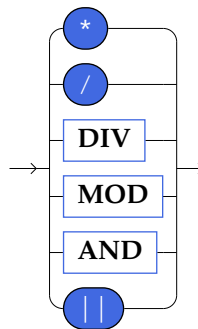
255 logical_literal = **FALSE** | **TRUE** | **UNKNOWN** .



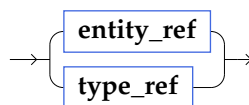
256 logical_type = **LOGICAL** .



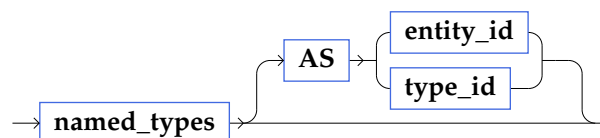
257 multiplication_like_op = '*' | '/' | **DIV** | **MOD** | **AND** | '|' | ' ' .



258 named_types = **entity_ref** | **type_ref** .



259 named_type_or_rename = **named_types** [**AS** (**entity_id** | **type_id**)] .



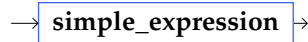
260 null_stmt = ';' .



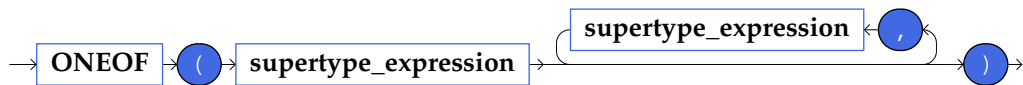
261 number_type = NUMBER .



262 numeric_expression = simple_expression .



263 one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .



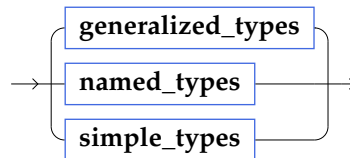
264 parameter = expression .



265 parameter_id = simple_id .



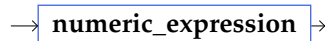
266 parameter_type = generalized_types | named_types | simple_types .



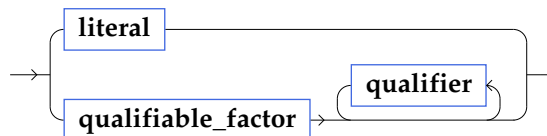
267 population = entity_ref .



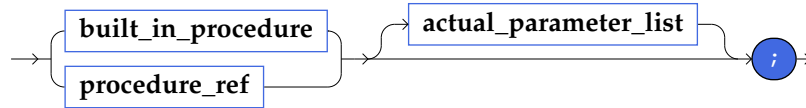
268 precision_spec = numeric_expression .



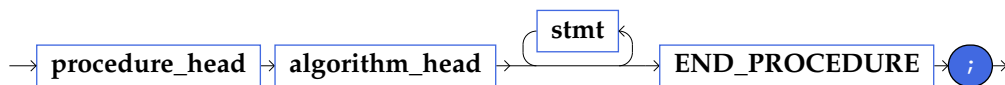
269 primary = literal | (qualifiable_factor { qualifier }) .



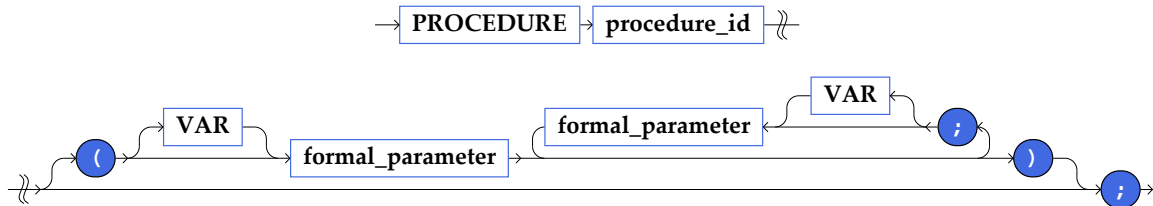
270 procedure_call_stmt = (built_in_procedure | procedure_ref) [actual_parameter_list] ';' .



271 procedure_decl = procedure_head algorithm_head { stmt } END_PROCEDURE ';' .



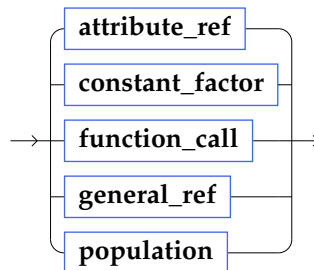
272 procedure_head = PROCEDURE procedure_id ['(' [VAR] formal_parameter { ';' [VAR] formal_parameter } ')'] ';' .



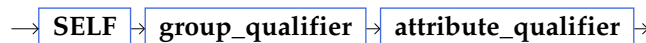
273 procedure_id = simple_id .



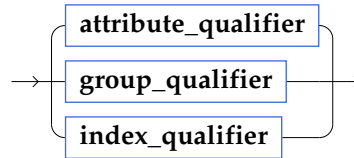
274 qualifiable_factor = attribute_ref | constant_factor | function_call | general_ref | population .



275 qualified_attribute = SELF group_qualifier attribute_qualifier .



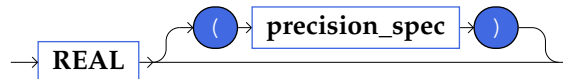
276 `qualifier = attribute_qualifier | group_qualifier | index_qualifier .`



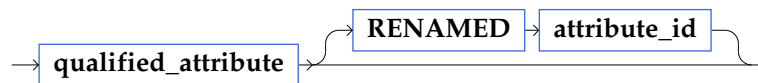
277 `query_expression = QUERY '(' variable_id '<*' aggregate_source '|' logical_expression ')'` .



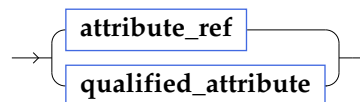
278 `real_type = REAL ['(' precision_spec ')'] .`



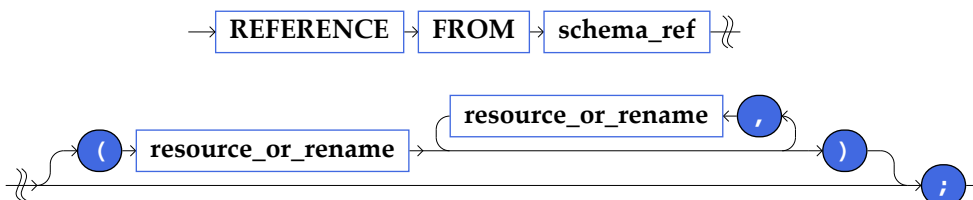
279 `redeclared_attribute = qualified_attribute [RENAMED attribute_id] .`



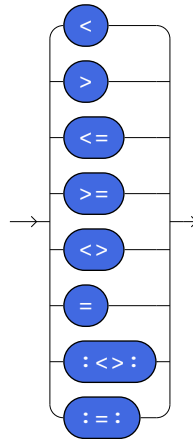
280 `referenced_attribute = attribute_ref | qualified_attribute .`



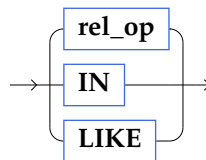
281 `reference_clause = REFERENCE FROM schema_ref ['(' resource_or_rename { ',' resource_or_rename } ')'] ';' .`



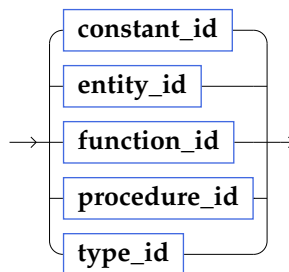
282 rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | '==:' .



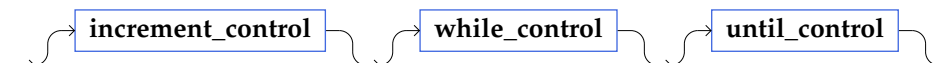
283 rel_op_extended = rel_op | IN | LIKE .



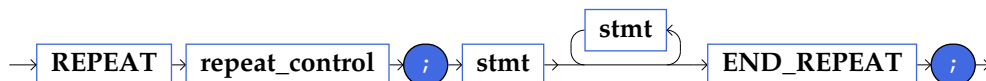
284 rename_id = constant_id | entity_id | function_id | procedure_id | type_id .



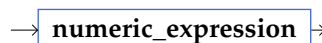
285 repeat_control = [increment_control] [while_control] [until_control] .



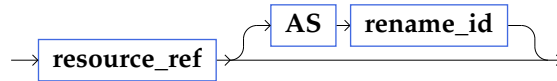
286 repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT ';' .



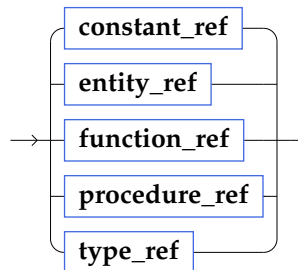
287 repetition = numeric_expression .



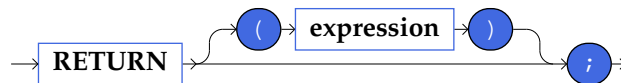
288 resource_or_rename = resource_ref [AS rename_id] .



289 resource_ref = constant_ref | entity_ref | function_ref | procedure_ref | type_ref .



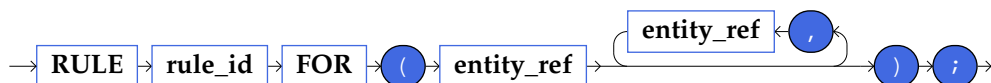
290 return_stmt = RETURN ['(' expression ')'] ';' .



291 rule_decl = rule_head algorithm_head { stmt } where_clause END_RULE ';' .



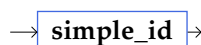
292 rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')' ';' .



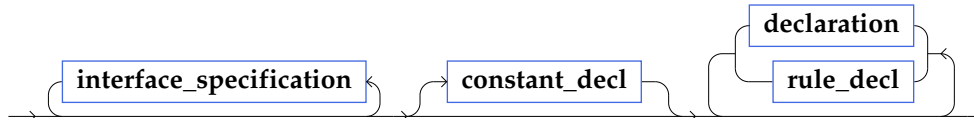
293 rule_id = simple_id .



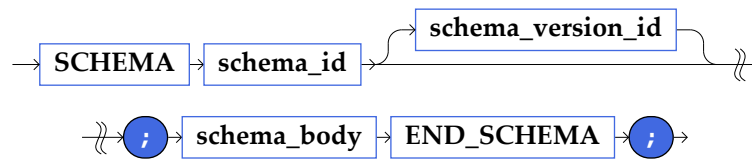
294 rule_label_id = simple_id .



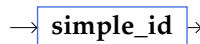
295 `schema_body = { interface_specification } [constant_decl] { declaration | rule_decl } .`



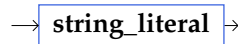
296 `schema_decl = SCHEMA schema_id [schema_version_id] ';' schema_body END_SCHEMA ';' .`



297 `schema_id = simple_id .`



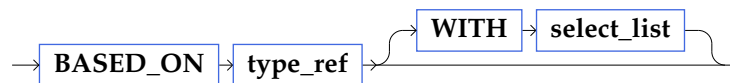
298 `schema_version_id = string_literal .`



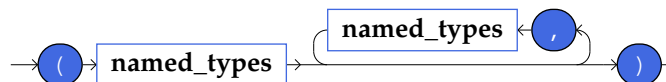
299 `selector = expression .`



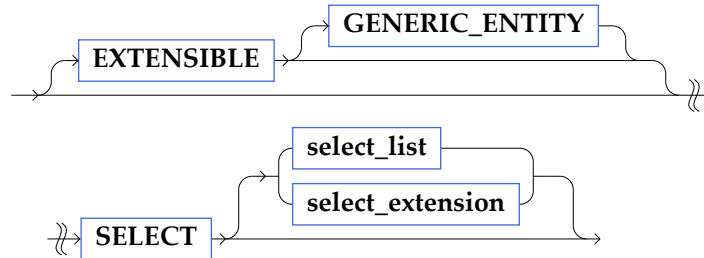
300 `select_extension = BASED_ON type_ref [WITH select_list] .`



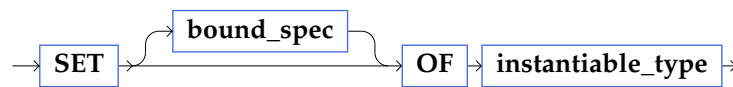
301 `select_list = '(' named_types { ',' named_types } ')'` .



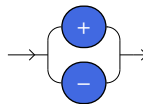
302 select_type = [EXTENSIBLE [GENERIC_ENTITY]] SELECT [select_list | select_extension] .



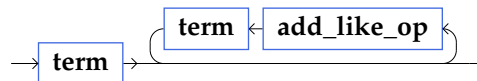
303 set_type = SET [bound_spec] OF instantiable_type .



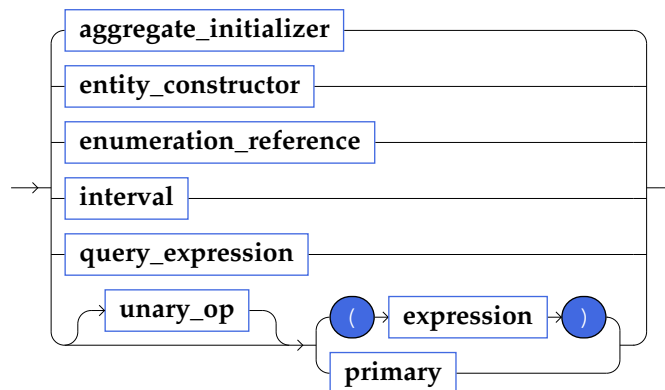
304 sign = '+' | '-' .



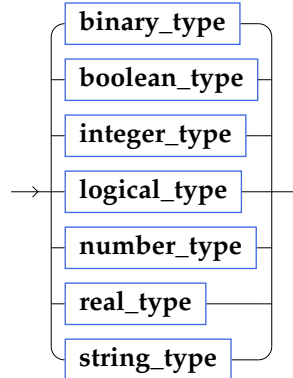
305 simple_expression = term { add_like_op term } .



306 simple_factor = aggregate_initializer | entity_constructor | enumeration_reference | interval | query_expression | ([unary_op] ('(' expression ')' | primary)) .



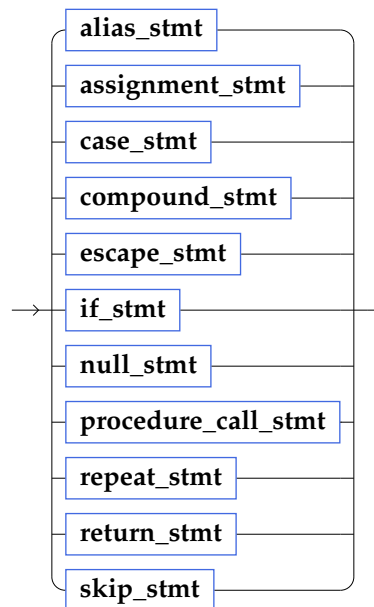
307 simple_types = **binary_type** | **boolean_type** | **integer_type** | **logical_type** | **number_type** | **real_type** | **string_type** .



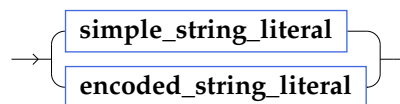
308 skip_stmt = **SKIP** ';' .



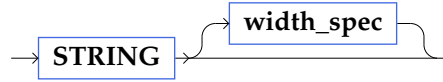
309 stmt = **alias_stmt** | **assignment_stmt** | **case_stmt** | **compound_stmt** | **escape_stmt** | **if_stmt** | **null_stmt** | **procedure_call_stmt** | **repeat_stmt** | **return_stmt** | **skip_stmt** .



310 string_literal = **simple_string_literal** | **encoded_string_literal** .



311 string_type = **STRING** [width_spec] .



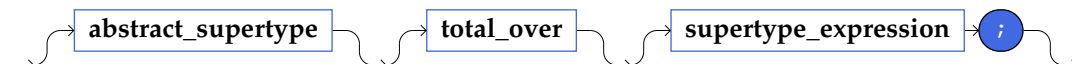
312 subsuper = [supertype_constraint] [subtype_declaration] .



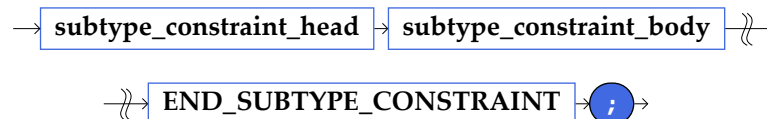
313 subtype_constraint = **OF** '(' supertype_expression ')' .



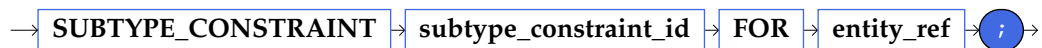
314 subtype_constraint_body = [abstract_supertype] [total_over] [supertype_expression ';'] .



315 subtype_constraint_decl = subtype_constraint_head subtype_constraint_body **END_SUBTYPE_CONSTRAINT** ';' .



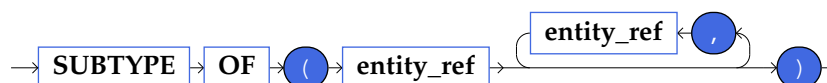
316 subtype_constraint_head = **SUBTYPE_CONSTRAINT** subtype_constraint_id **FOR** entity_ref ';' .



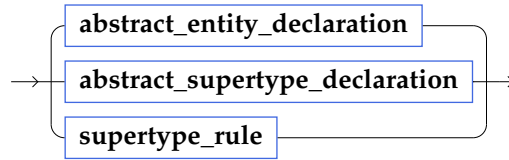
317 subtype_constraint_id = simple_id .



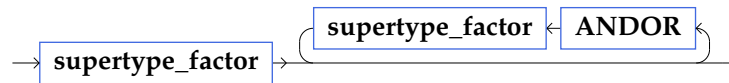
318 subtype_declaration = **SUBTYPE OF** '(' entity_ref { ',' entity_ref } ')' .



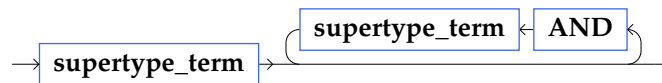
319 supertype_constraint = abstract_entity_declaration
 | abstract_supertype_declaration | supertype_rule .



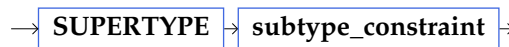
320 supertype_expression = supertype_factor { ANDOR supertype_factor } .



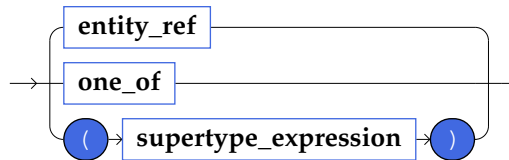
321 supertype_factor = supertype_term { AND supertype_term } .



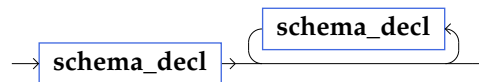
322 supertype_rule = SUPERTYPE subtype_constraint .



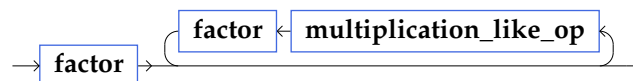
323 supertype_term = entity_ref | one_of | '(' supertype_expression ')' .



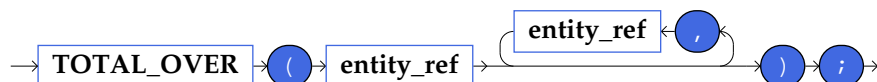
324 syntax = schema_decl { schema_decl } .



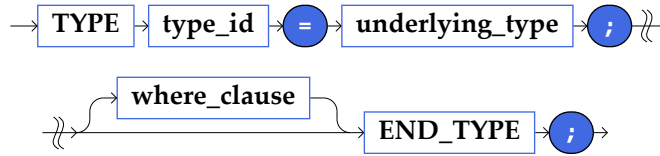
325 term = factor { multiplication_like_op factor } .



326 total_over = TOTAL_OVER '(' entity_ref { ',' entity_ref } ')' ;' .



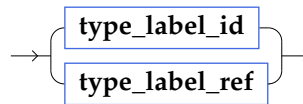
327 type_decl = **TYPE** type_id '=' underlying_type ';' [where_clause] **END_TYPE** ';' .



328 type_id = simple_id .



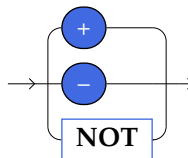
329 type_label = type_label_id | type_label_ref .



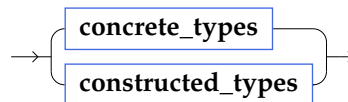
330 type_label_id = simple_id .



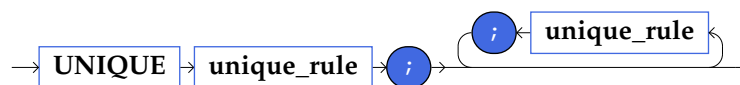
331 unary_op = '+' | '-' | NOT .



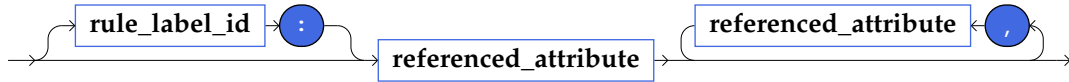
332 underlying_type = concrete_types | constructed_types .



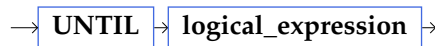
333 unique_clause = **UNIQUE** unique_rule ';' { unique_rule ';' } .



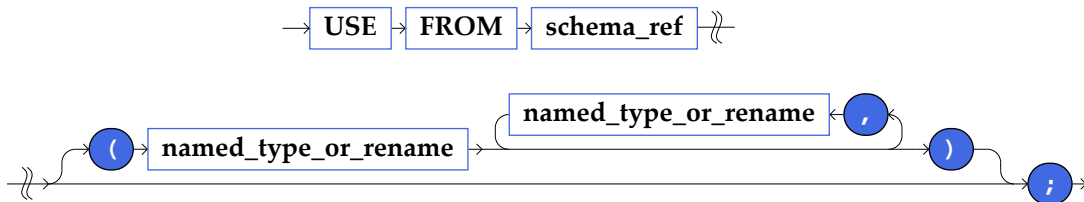
334 unique_rule = [rule_label_id ':'] referenced_attribute { ',' referenced_attribute } .



335 until_control = UNTIL logical_expression .



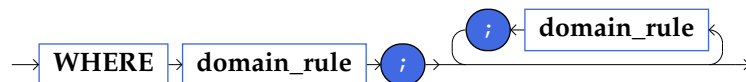
336 use_clause = USE FROM schema_ref ['(' named_type_or_rename { ',' named_type_or_rename } ')'] ';' .



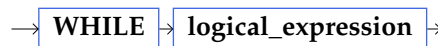
337 variable_id = simple_id .



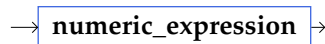
338 where_clause = WHERE domain_rule ';' { domain_rule ';' } .



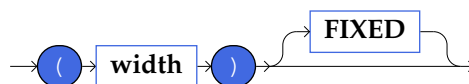
339 while_control = WHILE logical_expression .





340 width = numeric_expression .



341 width_spec = '(' width ')' [FIXED] .



NTNU	Risk assessment	Prepared by	Number	Date	
		HSE section	HMSRV2603E	02.11.2015	
HSE/KS		Approved by			
		The Rector			

Unit: (Department)

Date: 02.11.2015

Line manager: *Torgeir Welo*

Participants in the identification process (including their function):

Short description of the main activity/main process: Master project for student Jerome Schätzle. EVALUATE HOW THE STEP STANDARD AP 242 COULD ENABLE KNOWLEDGE TRANSFER BETWEEN CAD AND KBE ENVIRONMENTS.

Signatures: Responsible supervisor: *Olav Siversten*

Student: *J. Schätzle*

Activity from the identification process form	Potential undesirable incident/strain	Likelihood:	Consequence:			Risk Value (human)	Comments/status Suggested measures
		Likelihood (1-5)	Human (A-E)	Environment (A-E)	Economy/material (A-E)		
Purely theoretical work, does not contain any activities that involve risks.							

Likelihood, e.g.:

1. Minimal
2. Low
3. Medium
4. High
5. Very high

Consequence, e.g.:

- A. Safe
- B. Relatively safe
- C. Dangerous
- D. Critical
- E. Very critical

Risk value (each one to be estimated separately):

Human = Likelihood x Human Consequence

Environmental = Likelihood x Environmental consequence

Financial/material = Likelihood x Consequence for Economy/material