Data integration in preliminary Airplane Design

diploma thesis
cand. aer. Daniel Böhnke

Distributed Systems and Component Software Department
Institution for Simulation and Software Technology
at the German Aerospace Center (DLR)

Similarity Mechanics Group
Institute for Statics and Dynamics of Aerospace Structures
at Stuttgart University

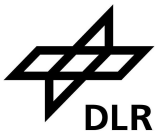
Cologne, July 2009

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe bzw. unerlaubte Hilfsmittel angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Köln, 31.Juli 2009

Task



At the German Aerospace Center (DLR) development is carried out concerning interfaces, program libraries and tools for the integration of numerical applications in the areas of preliminary airplane design and air transport missions (e.g. for the domains of aeronautics, structure, propulsion, combustion chamber, atmosphere, cosmic radiation, and mission analysis).

Designed for this task a new DLR data format (CPACS) was specified for the combined storage of data from all involved tools in XML. Other formats known in preliminary airplane design are ISO 10303, wide spread in industry, and the UML, originating from software development. The goal of this work is to analyze the named models. Additionally, the possibilities to store CPACS information in ISO 10303 are to be elaborated.

For CPACS a software interface for tools exists that allows the export and import of data. As an additional (optional) part of the diploma thesis the interface is to be extended to interface relating ISO 10303 data. This can be accomplished be developing import/export filters or by the direct integration of ISO 10303 data interfaces. The development of the interface needs to carried out either in Java or Python.

Abstract

A multidisciplinary approach to preliminary airplane design is seen as one of the major improvements for future design tasks. Several approaches can be found in the literature. The key to a multidisciplinary approach is a central model that contains all data and is accessible for all domains. The goal of this work is to analyze three different models in respect to their benefits for data integration in preliminary airplane design. The models are CPACS, developed at the DLR, STEP, published by the ISO, and the UML, released by the OMG. For this purpose a lexical overview of the important terms is given. Additionally several methods for the classification of information are introduced and requirements for information models are set up. The main part of the work is contributed to the analysis of the different models. A conclusion is drawn that suggests a solution for a future information model using a combination of the UML / SysML and XML. Subsequently a prototype for a converter tool, that processes CPACS data to STEP is developed. The converted geometry data of the ATTAS VFW 614 is used for validation and shows the quality of the tool.

Contents

Nomenclature

API	Application Programming Interface
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CPACS	Common Parametric Aircraft Configuration Scheme
DLR	German Aerospace Center
EDM	Engineering Data Management
EXPRESS	Modeling Language
IGES	Initial Graphics Exchange Specification
ISO	International Organization for Standardization
KBE	Knowledge Based Engineering
OMG	Object Management Group
PDE	Product Data Exchange
PDM	Product Data Management
PLM	Product Lifecycle Management
RMS	Root Mean Square
SET	Standard D'Echange et de Transfert
STEP	Standard for the Exchange of Product Data
SysML	Systems Modeling Language
TIGL	TIVA Geometric Library
TIXI	TIVA XML Interface
UML	Unified Modeling Language
VDA	Verband Deutscher Automobilindustrie
XMI	XML Metadata Interchange
XML	Extensible Metadata Language
XSD	XML Schema Definition

List of Figures

1. Introduction

The introduction is split into three parts. At the beginning section 1.1 gives an outlook about the motivation for the work carried out. A short lexical overview is listed in section 1.2. This section outlines some of the major terms for information models. Finally, section 1.3 introduces the concept and structure of this work.

1.1. Research Motivation

In preliminary airplane design various engineering domains have to be combined. Aerodynamics and structure, flight control and systems as well as cabin design are just some of the domains to be named. Different calculation methods like numerical simulations based on geometrical data, or analytics based on symbolic equations come along with these disciplines. Supplementary, shorter product design cycles have to be established even though product complexity increases [7, 17].

The data from all domains has to be available instantly to allow shorter iteration cycles and multidisciplinary design. Product Lifecycle Management and Product Data Management systems have been developed to handle product data. These Systems however do not handle data in a way that it is accessible as knowledge. Different engineering domains still can not interact, as there is no common language available [55]. A holistic model accessible for all domains is the key to realize these tasks. Several information models in preliminary airplane design are known.

At the German Aerospace Center (DLR) efforts are put in combining data for preliminary airplane design in a single information model. The *Common Parametric Aircraft Configuration Scheme* (CPACS) enables project partners to adjoin and share data from one single source [32, 37].

Along with CPACS there are other information models such as the *STandard for the Exchange of Product model data* (STEP) published by the International Organization for Standardization (ISO). STEP is widely spread in today's industry, especially for the exchange of geometric data. Usage of STEP is made in aerospace, automobile and ship engineering [18].

The *Unified Modeling Language (UML)*, having its origins in software engineering, being distributed by the Object Management Group (OMG), and its successor the *Systems Engineering Language (SysML)* have also been used to model airplane data [9, 38, 59].

The goal of this work is to analyze and compare these information models. We want to find out, in which ways these different information models can help to increase speed and quality of future aircraft design projects.

Along with the analysis a converter is written allowing the output of CPACS data to STEP. The converter allows the transfer of one information model to another and enables even more people to work on the same project. This can help to increase the acceptance of CPACS inside and outside the DLR.

1.2. Introduction to Information Models

"When I use a word," Humpty Dumpty said in a rather scornful tone, "it means just what I choose it to mean-neither more nor less"
Alice in Wonderland
Lewis Carroll

In this section a short lexical overview for information models is given. The major terms in information modeling are defined. Subsequently, characteristics and structure of information models are described.

At first, the term information should be defined. Several formulations can be found in the literature describing the term. The Oxford English Dictionary [1] defines information as "knowledge communicated concerning some particular fact, subject, or event" whereas knowledge is specified as "the fact, state, or condition of understanding". The german encyclopedia Brockhaus [11] defines information as "knowledge aligned in format". Additionally, a philosophical interpretation of the term information is given in [49] stating that information is "knowledge that can be communicated concerning its content and matter". The major EXPRESS book Information Modeling the EXPRESS Way [52] characterizes information as "data placed in context". The definition of data can again be found in the Oxford dictionary as "facts, esp. numerical facts, collected together for reference or information".

As it can be seen from the various quotes, there is no distinct definition of information. In this work we will stick to the common economic hierarchy of data, information and knowledge, where knowledge is on the top of this hierarchy. As mentioned in [55] many of the PDM and PLM systems neither offer a holistic representation nor do they handle knowledge. Integrating data into a preliminary airplane design process will therefore target the modeling of information, as we put data in context. Of course, the engineers view of a future product will include raw numerical data. The goal of modeling however should be to allow an understanding and reconstruction of these data.

An information model is defined in [42] as "a collection of symbol structure types [..] and a collection of general integrity rules". Symbol structure types describe entities that can be produced inside the information model. General integrity rules check for the consistency of the produced entities inside the model. A typical information model is described by the relational database model from [15]. The model is established from the three different symbol structures: table, tuple and domain. The integrity rule for the relational database is defined as: No two tuples within a table can have the same key.
Another more extensive definition for an information model is given in [36], "an information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse". One of the first attempts to create modern information models was established by the creation of the Entity Relationship Model described in [13].

The creation of an information model is described in the literature as well. During the development process of an information model, four different worlds are distinguished by [29]:

- **subject world**
  subject matter of the information system
- **system world**
  information system itself
- **usage world**
  organizational environment
- **development world**
  describes the creation of the information system

This definition goes along well with the three schemes architecture in [58] that arrogates an external (usage world) and an internal (system world) schema as well as a conceptual schema (subject world). The development world is not regarded in this paper.

In modern industry the usage and subject world are more complex as a product oriented view and a resource oriented view exist. The parallel existence of ERP and PDM-Systems shows this conflict [22].

For the creation of every information model an information modeling language is needed. The definition of an information modeling language is given in [36] as "an information modeling language is a formal syntax that allows users to capture data semantics and constraints". Several information modeling languages have been developed over the past decades. IDEF1X, EXPRESS and UML are just some of the languages to be named.

The term information model is usually used in the context of information modeling languages. In this work the models developed in STEP, CPACS and the UML are generally called information models. Accurately, the term would only refer to the specific metamodels.

1.3. Research Outline

This work analyses different information models in preliminary design and describes the development of a CPACS to STEP data conversion tool. The thesis is split into the following sections.

In chapter 1 the motivation for the work carried out is explained. More further a short introduction into information modeling can be found. The introduction elaborated some of the major terms quoting from common literature.

Chapter 2 gives a short review about information in preliminary aircraft design. The goal of this section is to name the different approaches to classify information. One classification scheme is chosen and the different objects and their relevance for preliminary design are outlined.

The following chapter 3 describes the requirements for Information Models. In this section not the information itself is outlined but severals important aspects are shown, that should be handled by information models. Additionally, some of the abstraction methods commonly applied in information models are elaborated.

Chapter 4 then describes the different information models CPACS, STEP and UML as well as their underlying data languages such as XML, XSLT, EXPRESS and XMI. The information models are checked for the requirements formulated in the chapters 2 and 3. Finally a comparison and analysis of the information models gives hints on how to use the different information models.

In chapter 5 an outline is given on the developed converter tool. Along with a description of the converter tool, the different development tools are illustrated. Additionally the test mechanisms and results for the validation of the converter are shown.

Finally, chapter 6 summarizes the results of this work and gives a short outlook on the further development of information models in preliminary airplane design. Chapter 6 is followed by the bibliography and the appendix.

2. Information in Airplane Design

Before the different information models can be analyzed and rated, benchmarks need to be set up. This chapter introduces all information that might be necessary in preliminary airplane design. The following chapter 3 then lines out how this information should be modeled.

The first section of this chapter introduces some classification methods for information that can be found in the literature. One approach is chosen and the following section outlines all objects that can be found from the previous classification. The objects are linked from their domain independent definition to their possible meaning in preliminary airplane design.

2.1. Information Classification

When breaking down information into several fragments, many approaches can be taken. Most engineering design books like [46] from Pahl et al. differ between functions and flows. A flow can be of energy, material or signal and has a direction. This portioning is however too abstract for our case.

In his lectures [31] Katzmann distinguishes product information into technical information, commercial information and quality information. The modeling activities at the DLR are limited to the introduction of new technologies into airplane design. Quality information is of more interest when a product goes into production, and is therefore of minor relevance for our consideration. Commercial information is only regarded as long as it is limited to economic factors. Product marketing is also of minor importance.

Analysis and synthesis are two opposite design steps. Executing an analysis determines the properties of a product from a given set of characteristics. Synthesis tries to find the needed characteristics to fulfill a set of requirements (properties). Weber et al. introduce this classification into PDM systems in their article [62].

In her proposal [38] for a tool for preliminary airplane design Lu defines two different kinds of tools. Those that are used for design analysis and those that are used for mission analysis. It might be possible to derive an information structure from this portioning. Some parameters in a model like for example a wing span are related to

both segments. Hence the portioning can be sensible for tool structure but is not for the related information.

Rudolph proposes a rather abstract definition of information as well. In his work [50] he defines different types of knowledge and a counter pair to each type. This can for example be continual or discrete information. Figure 2.1 gives an overview.

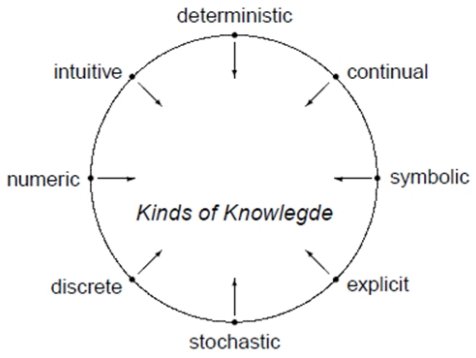


Figure 2.1.: Kinds of Knowledge, from [50]

For the following section these types are used to line out what information might need to be modeled in preliminary airplane design. Additionally to the named types from Rudolph, also implicit and explicit geometric information is described.

2.2. Information Objects

For the classification of information objects the scheme introduced by Rudolph is chosen. As it is already outlined in the introduction no static definition of information is available. Therefor any other classification scheme can lead to equal results.

2.2.1. Geometric Information

Modeling an airplane is based to a great amount to the modeling of geometry. Geometric information is important for the analysis e.g. of structure and aerodynamics. Geometric information is not only important for analysis of mission parameters, but also airlines are curious about different cabin layouts. As most products are designed for multiple purposes a freight configuration must be considerated as well. Many information stays in connection to geometric information. This information can be important from an engineering point of view (e.g. materials) as well as from an economic point of view (e.g. production information, suppliers). This section gives an outlook on two ways to hold geometric information, either implicit or explicit. It has been

stated many times that a product data management approach should not be based on geometric information as the superior element. This is elaborated as well in the following section.

Explicit Geometry

Geometric information can be stored using explicit file formats. This way a software that interprets this information finds for example a set of points, lines and solids in an already defined format. The explicit information can either be stored in a native file format from a CAD-Kernel or in a neutral format like STEP, IGES or DXF.

Native data benefits from the fact that the operations to create the geometry are usually still retraceable. The native files are however restricted to solely one CAD system. This makes it vulnerable to updates and extensive costs. Additionally, new engineers working on the model might be used to work with different software. The key to a multidisciplinary information model should be, that people can still work with their familiar tools. Also the exchange with other partners is only possible when changing to a neutral file format.

A short list of the most used neutral formats can be found within Choi's et al.'s works in [14]. As it can be deduced from the name, neutral file formats can be run on almost every CAD system. This is important for multi-company solutions, for example in aerospace industry with usually one OEM and many suppliers. In this case however solutions are sometimes established by using the same commercial CAD tool in the whole product chain. This, of course, implies major cost effects for suppliers working for several OEMs. One of the major shortcomings of a neutral file format is that all information about the creation of the information (parameter and operation) gets lost. Changes to the geometry are therefore costly.

Implicit Geometry

In the same way as an engineer develops geometry by running different operations with parameters, geometric information can be stored as a series of these operations. Geometric information stored this way is called implicit or history based. The benefit of this approach is that as all operations describing the model are accessible, changes in the geometry can be established easily. Approaches to generate implicit geometry through scripts are described by Reichwein and Hertkorn in [48].

The approach itself is again bound to one CAD system as most CAD kernels know different operations and parameters. Through mapping procedures it is possible to bring together similar operations from different CAD systems. For this purpose it is important to model an auxiliary geometric representation of the model, as references

to existing parts must be recreated. This can be found in the work of Choi et al. in [14] as well as by Mun et al. in [41].

Handling geometric information implicitly might be the best way for future information models. The implicit information is highly modular through changes and holds all parameters that can be accessed in other domains as well.

CAD Dilemma

Besides specialized analysis and simulation software, CAD systems belong to the most widely spread and expensive tools in the engineering domain. One of the major problems with CAD is that all these extensive tools are not inter operable. Usually a supplier must provide the CAD system its customers use. When the supplier tries to stay independent and works for several OEMs he needs to provide additional CAD tools. This is not only cost intensive but also inefficient because the same product needs to be modeled twice as it can not be transfered from one system to another. Meier calls this effect the "idle power"of CAD exchange. In his article [39] he also lists some of the different CAD systems used by the major OEMs from the german automobile industry. It can be seen that while Volkswagen and Daimler Benz mostly use Catia V4/V5 tools, Opel sticks to Unigraphics and Ford uses Ideas. Along with the CAD tools usually extensive PDM and PLM tools are used.

These tools are a step forward from a pure geometry based representation of product data. The problem is that modeling of a product should start as early as the the first concepts come up. The first elements of a product are however not geometry related as much more requirements and properties need to be considered. Stark names these issues in [55]. Katzenbach underlines these theses as he states in his lectures [31] that geometric information is not at the top of product topology.

2.2.2. Numeric Information

Numeric information describes all parameters of an information model that can be interpreted in a mathematical way. These parameters can therefore hold integer values as well as real. Numeric data can be classified using a unit system. A partially dimensionless treatment of parameters (aspect ratio and wingspan instead of wingspan and area) is possible as well. Numeric data are not necessarily to be retrieved from mathematical coherences.

2.2.3. Symbolic Information

Equations are a major contributor for describing product data as well as for analyzing them. Symbolic data can be used for some different reasons while modeling a product.

First of all, many aspects describing the product itself can be modeled via equations. The most simple example in this case is a fuselage section with a circular cross section. The location for fixing points on this section can be described using simple equations as well as the number of stringers.

It is possible to use equations to validate the model up to a certain point. All values that are brought into relationships via symbolic information can be recalculated and changes can be monitored. Additionally symbolic information can be used to access different analysis tools. To pick up the previous example a tool might work using the aspect ratio and wingspan but the model only holds wingspan and area. In this case equations can be helpful to ease the mapping process.

2.2.4. Continual Information

Regardless of the fact that continual data are hard to manage on a digital system such as a computer it comes up in preliminary airplane design on several occasions. Material data can be described using diagrams for tensile strength or elongation. This data usually comes from data sheets or standards and is neither retrievable nor can it be expressed via equations.

Continual data can also appear multi dimensionally such as in climate data which are also handled at the DLR. Performance maps are an additional example where continual data may arise in preliminary airplane design, for example with engine data.

2.2.5. Discrete Information

Like already mentioned, in the previous section, on a digital system like a computer data is always stored discretely. In this case the term discrete information is used in a different association.

Discrete data is mostly important because of technical reasons since not all product data is scalable. This can be seen in a simple example. As an airplane is supposed to reach a certain range and carry a load over this distance a wing is designed for a certain amount of fuel, aerodynamic drag and structural forces. To meet the requirements the designer needs to choose an engine for the airplane. As there is no new engine for the airplane the designer can only choose from pre-existing engines. The engines differ in weight, efficiency and thrust and are relevant for the design of the wing. This information influences the design process as it implies discrete steps.

2.2.6. Stochastic Information

Especially in preliminary airplane design, stochastic data plays a major role. Many target values for a construction are derived from statistical data. Usually common products in the same area are reflected via statistics and a delta is agreed on, by which the new product should exceed previous approaches.

Stochastic data is also used in areas where a deterministic description is not possible. For example in areas where research has not advanced enough for a reproducible result. Or in areas where a calculation is too time and cost intensive (e.g. engine noise).

2.2.7. Deterministic Information

As an airplane is a product developed by engineers and usually follows the laws of physics, all pieces of information concerning an airplane are in some way deterministic. Principles can be derived from physics as well as from self determined guidelines. An airline might provide deterministic information by introducing a ratio for seat length and height that guarantees comfort for the customer. This allows the description of a factor that is usually intuitive. If something like the introduced ratio is appropriate is therefore disputable.

2.2.8. Intuitive Information

Intuition is derived from creativity and can therefor hardly be represented in a model. Intuitive decisions can lead to new concepts or the development of new technologies. An intuitive decision is made without an explicit understanding of the material of interest. Something intuitive must not be something irrational. The decision or idea can still be declared with explicit information. In the first place it might be *something no one ever thought of.*

2.2.9. Explicit Information

Explicit information can be derived from intuitive information. An explicit term is created through abstraction of a previously intuitive information. The explication for the intuitive term is established via explicit rules. In terms of airplane preliminary design this process would occur when a new winglet design that was chosen intuitively creates major benefits for the aerodynamic of the airplane and the reasons for this behavior can be found so that the effect is reproducible

3. Requirements for Information Models

After a short introduction about information classification and information objects was given in the previous chapter, this chapter describes some of the requirements for information models in preliminary airplane design. The requirements are divided into two subsets.

Primarily, some important aspects are named, that need to be fulfilled by the model, e.g. holism. These aspects can however not always be accomplished solely by the process of modeling. For instance, visualization needs a proper model to be based on, but also a stable and easy-to-use software interpreting the model. Section 3.1 gives an outlook on these aspects.

The second part of the requirements chapter elaborates some of the modeling mechanisms and abstraction methods that are commonly used in section 3.2. These mechanisms are mostly based on object oriented languages and can be identified in chapter 4 as all introduced models use these mechanisms in one or another way.

3.1. Important Aspects

In her paper [36] at the National Institute of Standards and Technology, Lee states that a quality information model is described as a model that is: "*complete, sharable, stable, extensible, well-structured, precise, and unambiguous*". In their introduction [4] to STEP Anderl and Trippner outline the central functions of a product model as the exchange, storage, archive and transformation of product data. While the list can probably be continued beyond this point, we try to give an outlook on the most important aspects of information models.

3.1.1. Holism

The first and foremost important aspect of an information model in preliminary airplane design is holism. The model does not only need to include all engineering domains but also other departments like marketing, sales and production. As the granularity of a model increases from preliminary design to production, misconceptions at an early stage can cause major harm to a project. The holistic approach can be subdivided into two sections. The first and most obvious requirement is the multidisciplinarity of the model. As the data has to be within the model, it also has to

be available for design work. Accessibility is another issue that can be linked with holism.

Multidisciplinarity

A multidisciplinary model can be defined as a model that holds a common subset of parameters and objects that have a relation to more than one domain concerned with the design of the product. This model definition for example goes beyond the point of dual disciplinary models that are used for fluid-structure interaction. These models are related to one specific tool and are from an abstract point of view handled as one domain.

Working in a multidisciplinary environment is an uncommon task for a design engineer who usually is concerned with single domain solutions. The design engineer should therefore be able to still work with his known single domain tools. This goal is set up by Hoofman et al. in [25]. Working in a common environment is one reason for this organization of tools. Additionally, there is no need to re-implement parts of a model that are not related to other domains. For example an engineer that is currently working on aeroelastic fitting might be interested in several different layer structures of a carbon fiber material. The engineer tries out different structures using a parameter study. He can trigger a calculation program for the characteristic values of the material. The engineer is however only interested in these results. The characteristic values are important for the work of other engineers in other domains. A multidisciplinary model does not replace single domain specialists but makes parts of their knowledge available to others in the design process. The goal of a multidisciplinary model is not to create an engineer that has full knowledge in all accessed domains. The key is to provide the information that influences the global design operations. This brings up two major issues:

- ⇀ identify parameters that are relevant for a multidisciplinary model
- ⇀ announce domains accessing a single parameter

The first issue can only be handled by modeling specialists defining the meta model for the multidisciplinary model in combination with single domain design experts. The modeling specialist needs to know about the requirements for several design tasks in single domain environments and their output and impact on the model. From this information he needs to extract common subsets and make them available in the holistic model.

Introducing a parameter to a model is however not enough. An engineer working on a model needs to know which design task does influence a parameter. Only in this way changes to parameters in the model are re-traceable, and only in this way an engineer is informed that his changes may influence other domains of the model.

Accessibility

As the model is built up as a multidisciplinary model it needs to be accessible for all domains contributing to it. At this point it must be seen that while the model should be set up by a modeling expert it is usually accessed by a design engineer coming from a single domain background. The design engineer reads data from the model, processes it and writes it back. Hence the model should be accessible as easy as possible. In a study [19] about this issue Fidel and Green find the most frequent accessibility factors for documentary sources as:

- Saves time
- Has the right format
- Is physically close
- ...

Time is always saved by storing data in a single model. This way the number of sources is kept small and information is easily retrievable. When additional information needs to be put into the model, a modular and fast infrastructure must be set up, that allows changes in the model quickly.

The right format for data is a requirement that is hard to establish as all pieces of information are transferred into the central information model. As the models' readability should be good, information should be accessible. Extensive data used by e.g. numerical simulations can be stored in the according format and then be accessed via links from the central model. Being in the right format is an attribute that can also be established by developing import/export functions that translate one information model into another.

The term physically close can be replaced with *at my desktop*. The model should be accessible from either local hard drives or via software that manages network storage. The author would also encourage an accessibility for all departments involved in the project, not only the design department. For example the sales and marketing departments are part of the product development at an early stage. These departments stay in contact to the customer, set up requirements and communicate development improvements. Access to simple geometry representations would enhance the work of these departments.

3.1.2. Transparency

The previous sections showed the holistic approach to an information model. It was said that all information regarding more than one domain should be stored in the model. Additionally the accessibility for all domains contributing to the model was

requested. This section enumerates requirements on how to order and represent the information in the model.

Visualization

The information stored in the model and the structure of it need to be represented in a way that allows easy navigation. Although the model is stored in a string based file (p21, stp, xml, xmi) the representation of the pure string based data would be far beyond readable. Chapter 4 shows some examples.

In some CAD Systems *Level of Detail* (LoD) can be used to only represent data that is currently under observation while other data from the model is not loaded on full detail. A similar display of information can be achieved by e.g. a tree representation. A designer that is faced with the full complexity of the model at once can hardly manage all pieces of information.

At this point it might also be a benefit to enable the view on the metamodel. As the metamodel only holds a reduced number of elements, it can provide a better overview of the data. Tools for visualizing information models as well as their metamodels are usually available as most languages contain a more or less detailed specification for graphic representations.

Visualization is only an issue for human interpretation. It may differ in some form from the stored model. One example for this representation can be seen for graphs. While all information can be stored in a tree structure, not all information can be displayed via a tree structure (bipartite graph). The machine and human representation must therefore not be identical.

Order

Transparency can not only be achieved by graphical representation of a model, but also by a good structure. Several abstraction methods are introduced in section 3.2. Hierarchy can be subdivided into the following three classes as elaborated by Mylopoulos in [42].

- **Ahierarchical Systems**:
  "Ahierarchical systems are independent combinations of information. The terms are coequal and without an inherent structure but they can be assorted e.g. alphabetically."
- **Partial-hierarchical systems**:
  "By adding order characteristics partial-hierarchical systems provide the opportunity to make a first grouping of objects according to content-related criteria.

So there is the possibility to select groups by needed properties. One example for this category of systems is a Thesaurus."

- **Hierarchical systems**:
  "Hierarchical systems possess a strict hierarchical structure. [...] This system is advantageous for a short, clear representation of information, but in consequence of the limitation of digits there is just a poor significance of number coding possible."

Another way of introducing order to the information model is by using semantic- or ontology-based systems. In these systems the focus is placed on modeling the relationships between entities. Proposals for these systems are made by Weber et al. for product data management in [61] and by Klein for ISO 10303 in [35].

Additionally, a possible breakdown for the order of information can be made by differing between top-down and bottom-up approaches. A top down approach starts by an upper element like e.g. the airplane and then models minor elements. Bottom-up systems are more related to biologic systems for example the build up of a organism evolving from a single cell.

The introduced classifications overlap in many cases. Semantics can easily be used to model hierarchic structures. Top-down and bottom-up systems guide the direction of the hierarchy. Nevertheless, most information models (and all models evaluated in this work) stick to a partial-hierarchical top down approach.

3.1.3. Reuseability

Modeling product data is a time and cost intensive task. As a result this task should not be repeated for every product or product derivate. In the automobile industry products are distributed in different configurations (coupe and convertible) as well as in the aerospace industry (long range or freighter). The created information goes through the process of: "specification, development, production, test, modification, use, storage, etc", as stated by Stark in [55]. Enabling the reuse of this information is a major task for the information model.

The creation of a detailed metamodel (e.g. well defined classes) allows the reuse of information. It is important to note that the information model should hold this information in form of applied modeling mechanisms and not in form of documentation. Documentation can encourage the understanding of a model but is often used to replace elements that could be modeled as well. Furthermore documentation may be ignored as it makes limitations to the model that can not be validated.

Rule-based systems can offer operations to build up a model. When the set of rules (production system) that was used to create a model is known, a reconstruction of the model or variants of it can easily be achieved by using a similar set of rules.

3.1.4. Version Control

The origins of version control for engineering information models lie within product data management. The idea is that the changes to a part or product can be retraced. The optimal solution allows a user to find out who and for what reasons applied changes to the model. Also the timeline can be reconstructed.

The version control implies an additional layer to the model. While contents of the model are stored in a partial hierarchical system, version control can add an additional dimension. But version control does not only enable the control of changes to a single model, but also to the development states of the metamodel.

Reuseablity that was demanded in the previous section can only be established when it is possible to retrace previous projects. Additionally it is important to track the versions of the metamodel, as tools targeting the model will only work for particular versions of the metamodel.

3.2. Abstraction Methods

The process of modeling is divided into two parts. On the one hand there is the specific or *instance* model of a product. This model carries detailed information about the product, in our case the airplane. For example this model holds information about the size of the wing as well as information about the number of seats in the business class section.

On the other hand the information that is detailed in the instance model is placed as abstract information in the metamodel. The metamodel describes the structure and types that can be used to instantiate the specific model. The metamodel describes the wingspan as an attribute of the wing and the number of seats as an attribute of the fuselage section. For the wingspan it allows positive values from type meter. The number of seats must be an integer value. In this work we examined languages for metamodels that are mostly based on object oriented mechanisms. In their paper [8] Bertino and Martino describe the basic concepts for such a model:

- each real world entity is modeled by an object
- each object has a set of instance attributes and methods
- the attribute values represent the object's status
- objects sharing the same structure and behavior are grouped into classes
- a class can be a specialized version of one or more classes

These concepts are lined out in the following sections to allow a better understanding of the modeling mechanisms used in the various information models examined in this

work. At some points the explanation exceeds the definition of Bertino and Martino by more modern concepts such as e.g. multi inheritance. Additional information on object oriented modeling is given in the language specific literature such as in [52] by Schenk and Wilson on EXPRESS or in [44] by Oestereich on the UML.

3.2.1. Classification

During the process of modeling, information is abstracted from an entity of reality. Only those aspects that are relevant for our model are described. When an airplane is described within a model for aerodynamic calculations the painting is not part of it. A modeled entity is called an object. After it is decided what information will be handled by the model, a class can be implemented. A class holds the structure and behavior of a set of objects. When a specific airplane needs to be modeled, an airplane object from the airplane class will be instantiated.

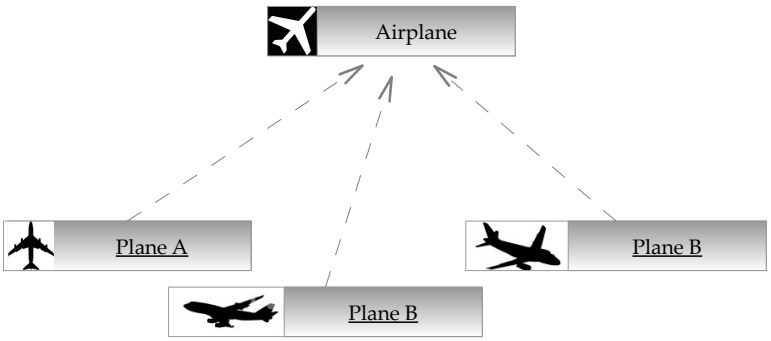


Figure 3.1.: Class and Object

Figure 3.1 shows a graphical representation for the above example. Several objects are derived from the airplane class. Objects are always underlined in a UML conform representation. In object oriented programming classes describe the behavior and structure of objects. The behavior is modeled via operations or methods. In our example this means that the class is equipped with an additional `fly()` method. For information modeling behavior is no longer considered.

The structure of a class consists of a set of attributes. Again this can be taken from the example as the wingspan and number of seats. The class describes the types, the object holds the values. The terms class, object, attribute, entity, element are used frequently in similar meanings in this work. The following table gives an overview of the language specific names in respect to the introduction given in this section.

It is important to note that each object should be marked using a unique identity. Objects may hold identical values for their attributes as well as an identical name. The example from above describes the modeling of an airplane. If there are two identical

term	EXPRESS	XML-Schema	UML
class	entity	element	class
attribute		attribute	attribute
relationship	relationship		association

Table 3.1.: Object Oriented terms in Information Modeling Languages

airplanes both named A 320 with a wing span of 34,1m and 20 business class seats, there must be a possibility to differ between the objects. This can either be established by introducing an additional attribute e.g. named id, or by declaring for example the name attribute to be of unique kind. The second method can of course only be deployed if the information modeling language knows a unique keyword.

The distinction of a suitable set of classes is important for a well modeled information model. Classification methods and schemes are elaborated in section 2.1.

3.2.2. Generalization

One powerful mechanism in object oriented structuring is generalization. Classes can be ordered in different systems (see section 3.1) hierarchically. A subclass will inherit all attributes and methods of the superclass. While inheriting preexisting elements the subclass can declare additional elements.

To follow the example from the above section, we will introduce a more general class than the airplane class. As an airplane is a means of transportation, the superclass will be called vehicle. A vehicle can hold business class seats but must not have a wingspan. A ferry can also offer business class seats but is defined through gross register tons. The two classes airplane and ferry will inherit the number of seats from the vehicle class.

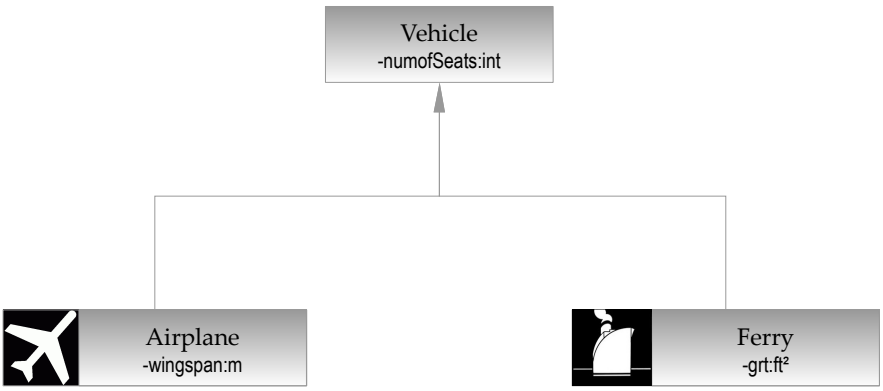


Figure 3.2.: Generalization Example

Figure 3.2 shows a graphical representation for the example above. It is possible to define the vehicle class from the example as an abstract class. Abstract classes can hold attributes and methods as well as all other classes, but they may not be instantiated. In some cases the inheriting subclass will overwrite elements from the superclass and redefine attributes or methods. This mechanism is called specialization. Multi-inheritance enables a subclass to inherit attributes and methods from two classes.

3.2.3. Association

One of the basic works for information models was the entity relationship model presented by Chen in [13]. Entities are introduced as classes in the object oriented modeling. Relationships are established via associations. In the terminology of the UML an association is created between classes, objects are connected via links.

Along with the association a multiplicity can be implemented. The multiplicity defines the number of objects that can be linked to each other. For example we can restrict the number of wings that may be attached to an airplane object to three. On the instance level the wing, the horizontal tail and the rudder can then be linked to the airplane object. Figure 3.3 shows an example. The left hand side shows the metamodel and on the right the instances are displayed.

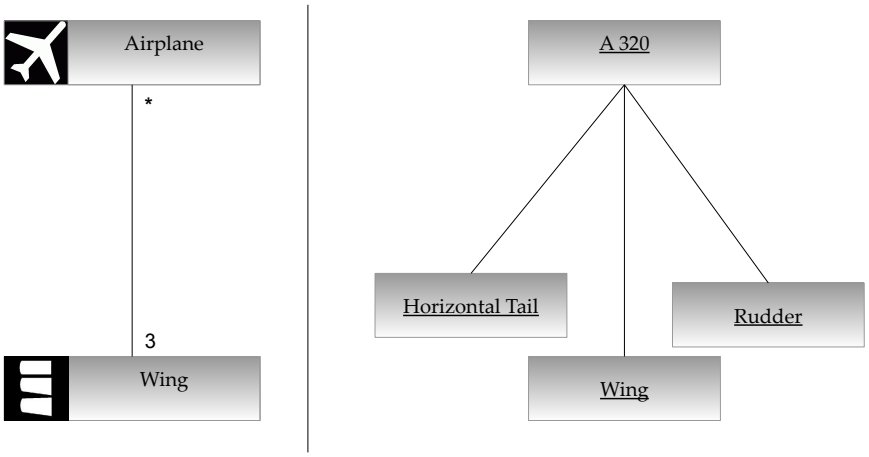


Figure 3.3.: Association Example

Association attributes and recursive associations are additional elements that can be used for modeling but are in this case outside of the scope of information modeling. Many of the named mechanisms have their origins in object oriented programming and make working with methods easier. However this is not relevant in all aspects to information modeling.

4. Information Models

In the previous sections the scope of information modeling in preliminary airplane design was outlined. It is important to note that at the DLR not more than a preliminary view on a product is interesting. The DLR is a major research institution and therefore curious about introducing new technologies into an airplane. This might lead as far as the construction of a prototype but not any further. Therefore handling product lifecycle data which is important for industrial users, is not an issue at the DLR. Section 2 described all the data that might come up in the information models and how it should be handled. The following section 3 lined out some requirements that the information models need to satisfy. The requirements section was split into the areas of modeling aspects and modeling mechanisms. This chapter introduces three information models and describes their benefits and shortcomings.

The oldest of the compared information models is the ISO 10303 also known as the **ST**andard for the **E**xchange of **P**roduct **D**ata (STEP). It is an international standard and widely spread throughout the industry. The models are built up by using the object flavored language EXPRESS. Instance files are stored either as .p21 or XML data. As STEP is a very extensive standard, only one application protocol is discussed. The reasons for choosing this application protocol along with a description of the other STEP parts can be found in section 4.1.

So far CPACS is a DLR intern standard. It is used in several multidisciplinary projects that handle the scope of preliminary airplane design as well as further aspects. The development came up in 2005. The standard is based on XML and benefits from a growing number of tools that are adapted to it. The CPACS related work can be found in section 4.2.

Although there is no standard for the modeling of preliminary aircraft data in UML yet, this chapter gives an outlook on some of the work done in this area. There are several projects running that work on these issues. The greatest common subset however is still the software specification of the UML, as outlined in section 4.3.

The following sections do show not only facts concerning the reviewed information models but also several hints that the techniques are in some way connected and that there are several intersections. Finally an analysis (section 4.4) is carried out that summarizes the evaluation made and gives hints about the the most valuable approach for future work at the DLR.

4.1. ISO 10303

The ISO 10303 Product data representation and exchange is also known as **ST**andard for the **E**xchange of **P**roduct **D**ata. It is a set of standards for the modeling of product data. The following sections give an outlook on the requirements that were set up during the development process of STEP. The history of STEP along with some of its ancestors is presented. Subsequently, the structure of the standard along with its operational area and some of the main tools that have implemented STEP is illustrated.

At the beginning of the STEP development several requirements were determined that describe the scope of the evolving standard. Fowler elaborates these requirements in [20] as:

- The creation of a single international standard, covering all aspects of CAD/CAM data exchange
- The implementation and acceptance of this standard by industry, superseding various national and *de facto* standards and specifications
- The standardization of a mechanism for describing product data, throughout the life of a product, and independent of any particular system
- The separation of the description of product data from its implementation, so that the standard would not only be suitable for neutral file exchange, but also provide the basis for shared product databases, and for long-term archiving

Ongoing from these requirements the development of STEP started within the International Organization for Standardization (ISO) in 1984. Other neutral file formats of the time did not comply with the challenges.

The most famous ancestor of STEP is the Initial Graphics Exchange Specification (IGES). IGES has been developed in the 1970s by the IGES/PDES Organization and was published as ANSI standard. It is still used for the exchange of geometric information and most major CAD-Kernels offer translators for IGES although it has been officially replaced by STEP [20, 27].

Several lacks of IGES (e.g. no handling of non-geometric data [14]) were tried to capture within SET. SET stands for Standard D'Echange et de Transfert and was developed in France. Its first release [20] was in 1985.

Attempts for a neutral format for the exchange of surface data led to the development of VDA-FS (Verband der Automobilindustrie-Flächen-Schnittstelle). VDA-FS came up in the 1980s as well. The origins of the project lay within the german automobile industry. As some measuring systems output their data via VDA-FS it enables the control of CAD-Data. Due to internal recommendations by the VDA [5] the format was replaced by STEP. The list of neutral data formats could be continued and includes

formats such as DXF or Parasolid. These standards however neither manage the hole scope of product data nor are they connected to the history of STEP.

Until the first release of STEP in 1995 a lot of development was carried out. Within the release ISO published the first 12 parts of STEP. Over the years the standard has evolved to a much bigger information base for product data handling. Part of the development is the new information modeling language EXPRESS. EXPRESS is described further in section 4.1.2. As EXPRESS is just one of the many parts published until today the structure and organization of the different parts of the standard are outlined further in section 4.1.1. Some of the parts that are related to the work carried out are described in the following sections.

As already stated, STEP is developed and distributed by the ISO. Within the ISO the technical comittee 184 with the sub-commitee 4 is responsible for STEP. Along with STEP several other standards such as IIDEAS or PLIB are in the TC184/SC4's[1] liability.

As STEP is still mostly used to exchange geometric information, all major CAD-Kernels have implemented it. These tools such as Catia[2] or OpenCascade[3] however only cover a small bandwidth of the standard. The fact that no parametric information is translated by STEP is in someway appreciated by industrial customers [14]. In this way only pure geometric information *en bloc* is exchanged. This is common practice between OEMs and their suppliers. Additional features of STEP are implemented within some parts of Catia[2] and the software packages by LKSoft, namely JSDAI[4] and IDA-STEP[5]. A more extensive list of tools that use STEP can be found in the STEP Application Handbook [2]. All named tools are explained more precisely in section 5.1.

Due to the long and intense standardization process[6] of STEP its bandwidth is broad enough to attract the big companies in the aircraft industry. Boeing has started working with STEP early on AP 210 as stated by Smith in [54]. In 2004 the *Value Improvement through a Virtual Aeronautical Collaborative Enterprise*(VIVACE)[7] project started. The project is funded by the EU and Airbus. For the developed EDM framework STEP was also used.

Due to the continuing development of STEP a lot of scientific work is carried out. One of the best sources for STEP related projects is the NASA-ESA workshop for Product Data Exchange. The PDE[8] has been carried out for the 10th time in 2009.

1 http://www.tc184-sc4.org/
2 http://www.3ds.com/products/catia/
3 http://www.opencascade.org
4 http://www.jsdai.net
5 http://www.ida-step.de
6 This must not be an advantage. Solely, AP 214 holds 3500 pages. The information in this document however is not sufficient enough to create a geometric model.
7 http://www.vivaceproject.com
8 http://step.nasa.gov/pde2009/

4.1.1. Structuring in STEP

As already mentioned in the previous section, the **ST**andard for the **E**xchange of **P**roduct **D**ata is split into several parts. This section gives a rough outlook on the arrangement of the different parts. The line-up off the application protocols is described more precisely.

The standard is split into the different items:

- description methods
- implementation methods
- integrated resources
- application protocols
- conformance testing
- abstract test suites

All items hold a set of parts and all parts are published separately. Description methods hold information regarding the modeling aspects in STEP. In particular, in this area the EXPRESS language is described in several domains (EXPRESS, EXPRESS-I, EXPRESS-X). A review of Part 11: EXPRESS *language reference manual* is given in section 4.1.2.

The second item *implementation methods* includes several parts that allow the creation of STEP-related data as well as the output format of the data. Parts 21: STEP-*File Clear text encoding of the exchange structure* and 28: STEP-XML XML *representation for* EXPRESS-*driven data* describe the output as p21 file and xml. These parts come up in sections 4.1.3 and 4.1.4. Subsequently, several interfaces for programming languages are defined in other parts of this item. Part 22 is the *Standard Data Application Interface*. Its definition is used by JSDAI[4] which is one of the development tools elucidated in 5.1.

The item *integrated resources* is split up into the two areas of integrated generic resources and integrated application resources. As an example for an integrated generic resource Part 42: *Geometric and topological representation* can be quoted. This part holds several definitions for entities describing geometric elements. Integrated application resources describe parts that cover the areas of e.g. finite element analysis or kinematics. Application protocols bundle integrated resources to one specific set of entities.

Application protocols (AP) are the items of the standard that are best known to most users. Their task is to define the scope, context and information requirements for a domain as stated in [17]. The APs are built up from an application reference model and an application interpreted model. The idea behind this separation is that a user begins to conceptualize an application protocol with the terms and connections known in the domain the application protocol is defined for. This semantic information is put

into the application reference model. Via a mapping procedure these terms and connections are linked to entities and relationships already defined in the integrated resources in the application interpreted model. The application interpreted model therefore holds the syntax of the model. This part of the AP is the computer-interpretable model.

Several parts can be named that are well known APs. One of the more complex APs is Part 210: *Electronic assembly, interconnect and packaging design handling*. The Part 203: *Configuration controlled 3D designs of mechanical parts and assemblies* covers the application range for most work relating CAD. Most CAD-Kernels have implemented a STEP-processor that can handle AP 203 as well as AP 214: *Core data for automotive mechanical design processes*. AP 214 is outlined in section 4.1.5. At this point it can already be mentioned that APs can implement entities redundantly. For example, several entities from Part 42 are mapped in AP 203 as well as in AP 214.

Generally, all software tools that implement STEP APs can choose from a subset of conformance classes. A conformance class covers a set of entities from the AP that can be interpreted semantically. This however makes it difficult for a user who works with a certain AP to know if his work can be translated into different systems. Due to the ongoing development of STEP and its surrounding tools a milestone system seems to be an adequate measure, but the implementation level has to be found for each tool separately [2]. The structure of the named items is presented in fig. 4.1.

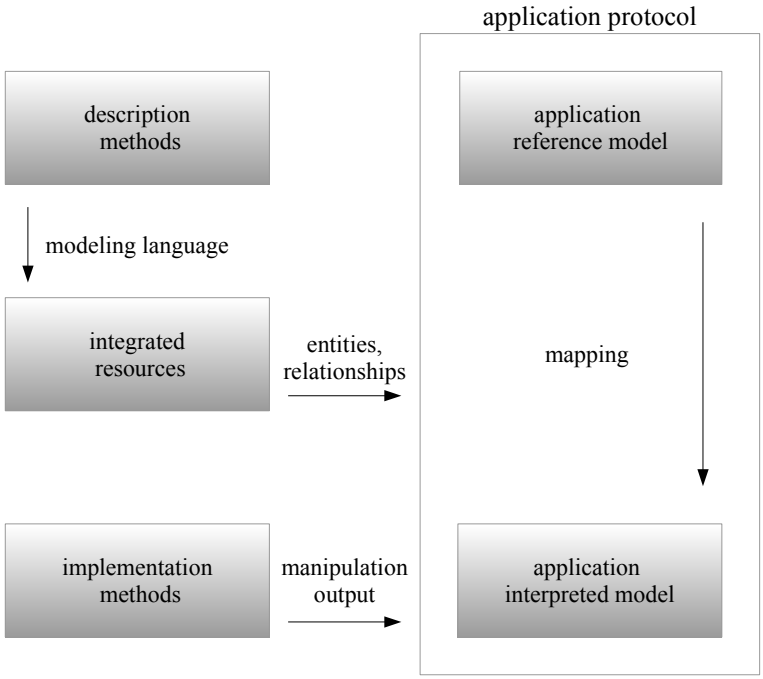


Figure 4.1.: Structure of STEP items

The question comes up whether the APs are a irreplacable part of STEP or not. The APs bundle sets of entities that are already described in the integrated resources. Hence, the integrated resources can be seen as the domain specific models (e.g. geometry, finite element analysis). APs only collect several pieces from these resources as they are application specific models. Most applications however cover identical domains. Ship, automobile and aerospace industry are all related to the named domains and use them in a more or less identical way. Due to the fact that all APs are based on an identical set of resources, the interchangeability between the APs should be given.

The last two items are not in the scope of this work and are therefore not covered entirely. The item *conformance testing* describes several parts that are applied to parts in development. Abstract test suites are a sub-item of conformance testing, namely Part 34. Originally it was planned to implement an abstract test suite for each AP. A full list of all parts can be found on the TC184/SC4 homepage[1].

4.1.2. EXPRESS language reference manual - Part 11

EXPRESS is a language for information modeling. It was developed during the first years of STEP after the initial work was undertaken at McDonnell Douglas Information Systems [30]. Although it has been a STEP project, the authors of the language, Schenk and Wilson [52], state that the language is used in several other cases. Ancestors of EXPRESS are the Natural language Information Analysis Method (NIAM) also known as Object Role Model (ORM) by Njissen [43] and the IDEF1X language developed by the U.S. Air Force [3]. By now, the language is accepted by the Object Management Group (OMG) and is made one of the OMG languages [18]. EXPRESS is part of the description methods in STEP. The language is used to define the entities in the integrated resources. The data itself is stored within the parts 21 and 28. These are explained in the following sections. This section gives examples for some of the modeling mechanisms that EXPRESS offers. As EXPRESS can be represented using the graphical form of the language EXPRESS-G, some of the examples are shown in figures, while others due to shortcomings of EXPRESS-G are shown in small code examples. The representation is compliant to EXPRESS-G in most, but not all cases. Of course, the EXPRESS language holds a broad variety of modeling mechanisms. Since not all of them are used in STEP and the whole description of the language would go beyond the scope of this work, interested readers are directed to the work of Schenk and Wilson. Some of the more known software tools for the work with EXPRESS are Expresso[9], CoOM[10] and the EXPRESS Compiler in JSDAI which are described further in section 5.1. The process of EXPRESS tool development is described by Goh et al. in [24].

9 http://exp-enginge.sourceforge.net
10 http://www.dik.tu-darmstadt.de/forschung_2/produkte/

An information model in Express is called a schema. Several schemas can be joined to one revolving schema. The APs of Step do not unite several schemes since the entities are simply copied from the integrated resources. The simplest case for the modeling of data is a parameter of a certain type. The starting point for a row of examples is a parameter x from the type Real. In the graphical representation of Express a base type is indicated by a box with an auxiliary line on the right side. This small example is displayed in figure 4.2.

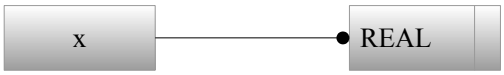


Figure 4.2.: Parameter and type in Express

Type definitions can be more complex in Express and go beyond the base types such as String, Boolean or Real. New types can be made of these base types for example a `length_unit` that is of the type Real. This way the interpretation of the parameter `x` is bound semantically to a length and can not be misinterpreted for e.g. a ratio. Enumerations and Selects can be used as well to model types. The following lines give an example for both commands.

```
1 TYPE unit = SELECT
2      (named_unit, derived_unit);
3  END_TYPE;
4
5 TYPE si_unit_name = ENUMERATION OF
6       (metre,gram,second,ampere,kelvin,mole,candela,radian,
7       steradian,hertz,newton,pascal,joule,watt,coulomb,volt,
8       farad,ohm,siemens,weber,tesla,henry,degree_celsius,
9       lumen,lux,becquerel,gray,sievert);
10 END_TYPE;
```

The Select command in line 1 enables the user to choose one element from a list of entities or types. `named_unit` and `derived_unit` are entities from part 41: *Fundamentals of product description and support*. An Enumeration holds a list of names. The names do not need to be declared any further. In this example several names are declared that can be used to specify the unit of an attribute. As Step only uses SI-units there is only a small amount available.

After a parameter or attribute is declared to be of a certain type it is connected to an entity. The term entity is used to describe an abstract form for a set of similar objects of the reality. Other object-oriented languages use the term class instead of entity.

As EXPRESS is not a programming language, its entities only hold attributes and no methods. In our case we declare an entity `point` that holds three attributes of type REAL and one attribute of type STRING. The entity is shown in figure 4.3.

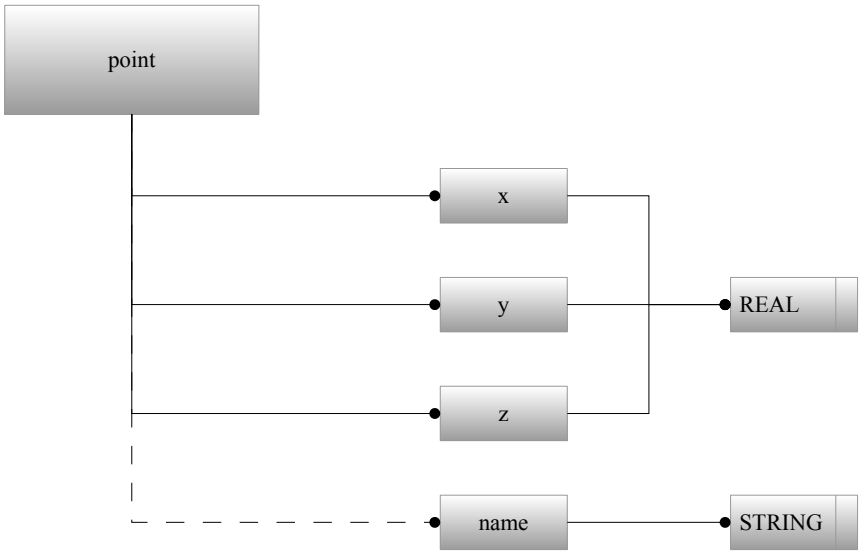


Figure 4.3.: Point Entity

The entities and attributes are connected via a line with a circle end. The name attribute is OPTIONAL and therefore only attached to the entity with an intersected line. The definition of IDs can be encouraged by an UNIQUE statement. Instances of an UNIQUE attribute may not hold identical values. In a EXPRESS-G conform representation the attributes would be displayed as text on the connecting lines and not as separated boxes. Relationships are established by declaring attributes that have an entity as a type. E.g. `name` could be an entity of its own, and hold several attributes such as a STRING for an identifier as well as an INTEGER for reference. The `point` entity could then be connected to the `name` by an attribute of the same type.

Generalization is a powerful mechanism for building models. One of the common application reasons for generalization is the description of a more determined mathematic model. The `point` example is elaborated further by introducing a subtype. The subtype is called `point on parabola`. The subtype inherits all attributes of the supertype. Two constraints are applied to the subtype using a WHERE statement. First the `x` and `y` attribute are combined using the parabola equation. Subsequently, the value of `z` is limited to values greater than zero. Note that the WHERE statement is used for validation and not calculation. Attributes that shall be calculated from symbolic equations can be declared via the DERIVE command, which is not used in STEP. Figure 4.4 shows the extended example.

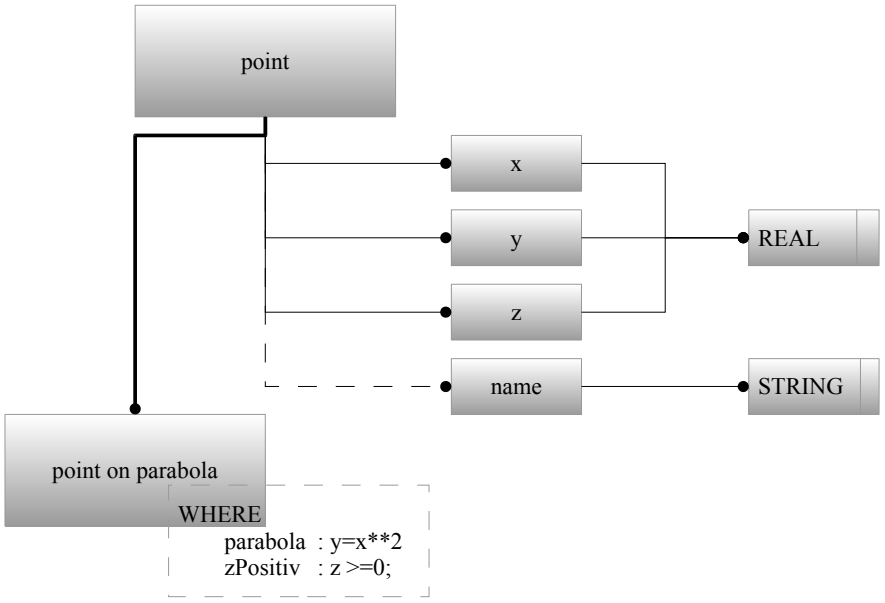


Figure 4.4.: Generalization in Express

Note that the generalization is represented as a thick line. The circle end is at the side of the inheriting entity. In Express-G there is no graphical representation for Constraints, Rules, Functions or Where statements. For this figure a simple box with an intersected edge was chosen.

When we look at the code for the generalization one problem in Express becomes obvious. In the following lines the code is displayed. One can see that the subtype **and** the supertype reference each other. This implies that the creation of a new subtype requires a modification of the supertype. This becomes a major problem with bigger information models[11]. Abstract supertypes can also be implemented in Express. This and other items not in the scope of this section are explained in [57].

11 This is not a problem for users of Step, as the entities are predefined and can not be altered without violation of the standard

```
1 ENTITY point
2   SUPERTYPE OF point_on_parabola;
3 ...
4 END_ENTITY;
5
6 ENTITY point_on_parabola
7   SUBTYPE OF point;
8 ...
9 END_ENTITY;
```

The Express language holds the possibility to create complex entity. A complex entity is created by an entity that is constructed from two subentites with an identical superentity. In the common geometry definitions of Step a simple example can be found. The superentity in this case is the `named unit`. Two out of many subclasses are the `length unit` and the `si unit`. As all numerical information should be stored in SI-units a combination of these two entities is constructed. The complex entity holds the information off all three classes. See the following figure 4.5 for an example. The figure shows the metamodel as well as the appearance of the complex entity in a p21 instance file.

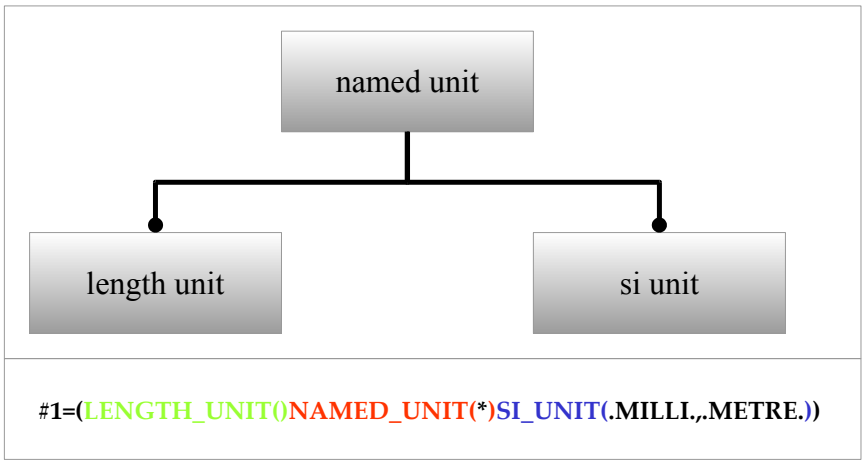


Figure 4.5.: Complex Entity

Feeney and Price promote in their talk [18] that Step should adapt to modern software mechanisms. One of the key requirements is the adoption of the Unified Modeling Language as replacement for Express. Several projects such as Engineering eXchange For Free[12] work on this issue and build a metamodel for the Express language.

12 http://www.exff.org/

4.1.3. STEP-File Clear text encoding of the exchange structure - Part 21

With the use of various tools, instances of STEP data can be output via the rules of part 21. Part 21 introduces files with the endings .p21, .stp and .step. Sovereign from the file ending, part 21 can be seen as the standard file format for STEP-data. All major software tools have implemented it.

In the following lines a short code example for a part 21 file is given. The example shows the code for a simple geometric file containing a cartesian point. Beginning and end of the file are declared by the ISO statements in lines 1 and 21. The file is divided into two parts. At first the header (ll. 2-16), containing various meta data, introduces information about the file, the name, the date of creation, et cetera. This information however is string based and not relevant for a computer interpretation. The schema (l. 14) gives a hint about the application protocol for which the file was created. Knowing the related schema enables a syntactic validation of the file. Subsequently, the instances are stored in a data segment. In this example the data segment holds only one instance (l. 19). The instance can be identified by the integer placed first in the line. After the identifier of the instance, the entity of the instance is named. In brackets the attributes of the entity are written.

```
1 ISO-10303-21;
2 HEADER;
3 FILE_DESCRIPTION(
4 /* description */ ('STEP Point Example'),
5 /* implementation_level */ '2;1');
6 FILE_NAME(
7 /* name */ ' ',
8 /* time_stamp */ '2009-05-27T15:33:06',
9 /* author */ (' '),
10 /* organization */ (' '),
11 /* preprocessor_version */ ' ',
12 /* originating_system */ ' ',
13 /* authorization */ ' ');
14 FILE_SCHEMA(('AP 214'));
15
16 ENDSEC;
17
18 DATA;
19 #1=CARTESIAN_POINT('Point.1',(1.,2.,3.)) ;
20 ENDSEC;
21 END-ISO-10303-21;
```

Relationships can be established via attributes. This is already shown in section 4.1.2. The identifier of an entity is put into the correlating attribute. The following code example shows the data segment from the previous extract. Additionally, there is an `axis_placement` related to the point placed in the model. The center of the axis is placed on the coordinates of the cartesian point. The dollar symbol (l. 3) stands for attribute values that are not assigned to a value.

```
1 DATA
2 #1=CARTESIAN_POINT('Point.1',(1.,2.,3.));
3 #2=AXIS2_PLACEMENT_3D(' ',#1,$,$);
4 ENDSEC;
```

The problem with complex entities in EXPRESS modeling has already been shown. These complex entities are parsed into the code in an additional set of brackets. The example for the length unit in part 21 file format is shown below.

```
1 DATA;
2 #1=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT($,.METRE.));
3 ENDSEC;
```

The files are saved ASCII coded and are therefore easily portable on multiple systems and via email. On the first sight, the part 21 files are human-readable and easy to interpret. The problem however is that at a certain filesize[13] the format gets unclear. There is no possibility to follow relationships and *jump* through the file. The partial hierarchic order is not apparent for the user. A special part 21 editor could solve these problems. Admittedly, the goal of part 21 was to create a file format that is exchangeable through organizations and is sovereign from specific tools.

4.1.4. STEP-XML representation for EXPRESS-driven data - Part 28

This part of STEP enables the user to output data via the Extended Markup Language (XML). The history of XML and its properties is discussed at full account in section 4.2.3. Part 28 was introduced as a *new item* to the TC184/SC4 committee in 1999. It is supposed to substitute part 21. The reasons to choose XML are expressed by Kimber in [33] as[14]:

- defines a generic, robust, and time-tested character syntax for representing structured data objects

13 The translated CPACS data contains approx. 6000 instances.
14 see page 12 of [33] for the full argument list

- facilities for the syntactic validation are available
- inherently extensible and flexible
- supported in Web browsers
- tied to an existing ISO standard, ISO 8879 (SGML)
- hot new technology that is attracting lots of attention

A similar list of arguments can be found from Peak et al. in [47]. The implementation of XML in STEP however is only limited to the instance level. In the following sections some examples of STEP Data saved in XML are shown. The examples outline the differences to the part 21 output.

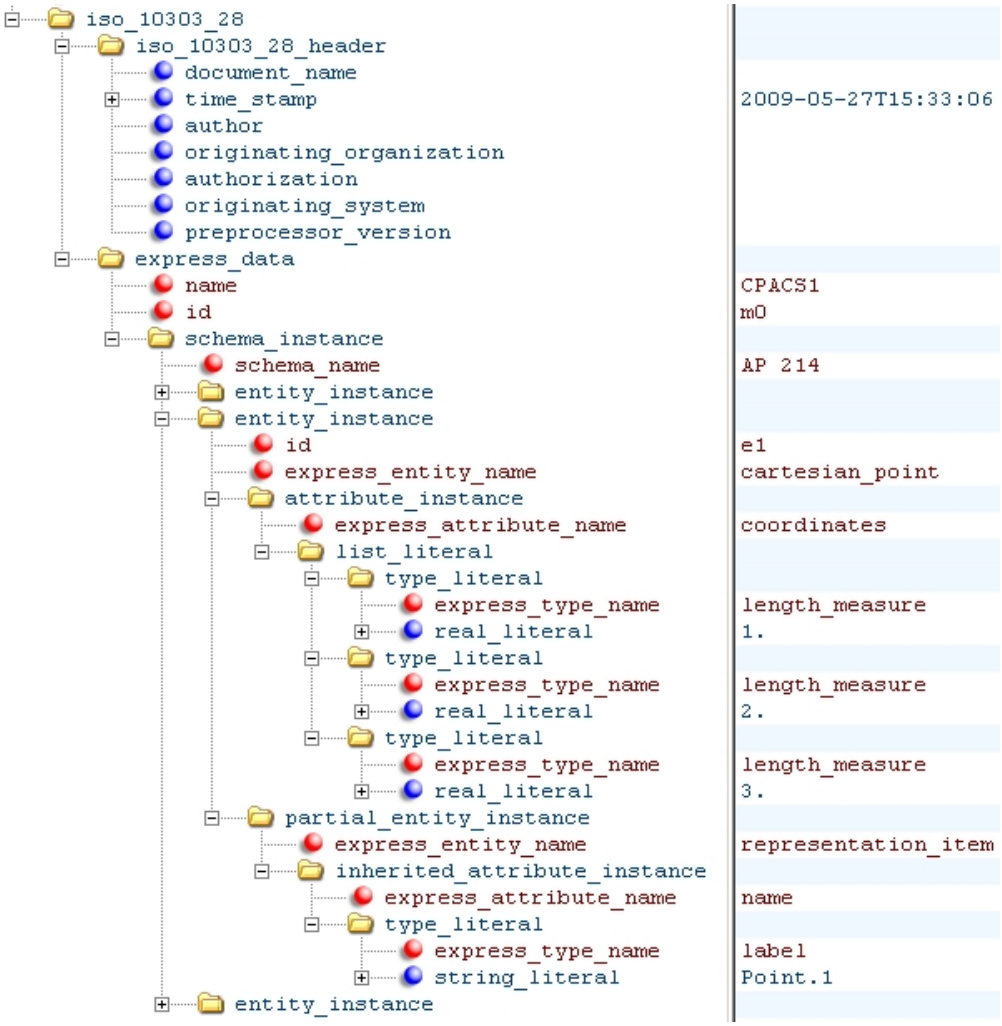


Figure 4.6.: Example for an Part 28 file

Figure 4.6 shows a set of STEP data equivalent to the first example from section 4.1.3. It can be seen that the tree-like structure of XML makes the presentation of the data more lucid[15]. With the use of modern XML tools, it is possible to collapse parts of the tree. On the left the tree is displayed. The right side of the figure shows the attribute data. In this example one can see the schema information on the top followed by the attributes.

Additionally to the attribute values, the type and in case of an inherited type, the supertype is displayed. This can be seen at the name of the point, where the name attribute is inherited from the representation item type. The following lines show the full code for the point entity of the part 28 file.

```
1  <entity_instance id="e1" express_entity_name="cartesian_point">
2    <attribute_instance express_attribute_name="coordinates">
3      <list_literal>
4        <type_literal express_type_name="length_measure">
5          <real_literal>1.</real_literal>
6        </type_literal>
7        <type_literal express_type_name="length_measure">
8          <real_literal>2.</real_literal>
9         </type_literal>
10       <type_literal express_type_name="length_measure">
11         <real_literal>3.</real_literal>
12       </type_literal>
13     </list_literal>
14   </attribute_instance>
15   <partial_entity_instance express_entity_name="representation_item">
16     <inherited_attribute_instance express_attribute_name="name">
17       <type_literal express_type_name="label">
18         <string_literal>Point.1</string_literal>
19       </type_literal>
20     </inherited_attribute_instance>
21   </partial_entity_instance>
22 </entity_instance>
```

The identifier (l. 1) from the part 21 file changed from the hash code #1 to *e1*. Due to the tree structure most references are not needed. They are only used to *jump* through the file. This not interpreted version is harder to read for a human, as much more information is displayed. This however enables a much better computer interpretation.

15 The displayed representation was generated using XML Notepad. The software is freeware and distributed by Microsoft. http://www.microsoft.com

4.1.5. Core data for automotive mechanical design processes - Part 214

AP 214 was designed to support the management of product data in the automobile industry. The development was initiated by the Verband Deutscher Automobilhersteller (VDA). Todays work on AP 214 is sponsored by the shareholders of the ProSTEP Association and the BMWi (german federal ministry of economy and technology). Some encouragement comes from members of the SASIG (STEP Automotive Special Interest Group) [23].

The development of AP 214 began during the 1990s, the latest version [28] was published in 2003. The scope of AP 214 holds 16 items. To allow a detailed analysis of the information models, the requirements of each model are shown. Therefor all 16 items are displayed, however not all in extend.

- ⇀ products of automotive manufacturers and of their suppliers
- ⇀ process plan information
- ⇀ product definition data and configuration control data
- ⇀ data describing the changes that have occurred during the design phase
- ⇀ identification of physically realized parts or of tools
- ⇀ identification of standard parts
- ⇀ release and approval data for product data
- ⇀ data that identify the supplier of a product and related contract information
- ⇀ eight types of representation of the shape of a part or of tool
- ⇀ data that pertains to the presentation of the shape of the product
- ⇀ representation of portions of the shape of a part or a tool by form features
- ⇀ product documentation represented on drawings
- ⇀ simulation data for the description of kinematic structures
- ⇀ properties of parts or of tools
- ⇀ data defining surface conditions
- ⇀ dimensional data and geometrical tolerance data

The first and most outstanding item defines the automobile industry as the key customer of the standard. There are no APs that are related to the aircraft industry explicitly. Some shipbuilding APs are released, but they lack the extent of AP 214. As long as no production related decomposition of the product is intended the approached tools can be dropped for preliminary airplane design. Interesting is item four, which enables the traceability of design changes. As a matter of course, the representation of the product data in various CAD formats (e.g. B-Rep, CSG) is part of the standard. The first chapter of AP 214 also defines several items that are not addressed in the standard. These are:

- ⇀ product definition data pertaining to any life cycle phase of a product not related to the design phase
- ⇀ business or financial data for the management of a design project

- ⇀ a general parametric representation of the shape of a part or of a tool
- ⇀ data describing the pneumatic, hydraulic, electric, or electronic functions of a product
- ⇀ continuous kinematics simulations over time
- ⇀ data describing the input or the results of finite element analysis

These limitations narrow the usefulness of AP 214 for preliminary airplane design. Not handling economic data may be suitable from an engineering point of view, especially in the early phases of product development. The life of a new airplane type usually starts by an offer for a new concept getting sold. Therefor a product model should hold at least some minor information regarding cost factors.

As already mentioned in the previous sections STEP does not support any parametric representation of modeling mechanisms. Due to several low-level attributes (e.g. radius of a circle) some kind of parametrization is still possible. Nevertheless, no symbolic equations are part of STEP. The fact, that numeric simulation data is at the moment not part of the AP is not fatal, as this data is usually extensive and is evacuated from the central model. This is even more true for unsteady simulations.

The named items are implemented and divided into several Conformance Classes (CC). These CCs can again be split into the different domains bundled in the model, as shown in table 4.1.

CC	description
1-5	subsets of CAD Representation
6-10	modeling of product data
11-13	process planing
14,15	feature-based construction
16,17	simulation and measurement data
18,19	data exchange via process-plans

Table 4.1.: Conformance Classes AP 214

For the investigations of STEP and CPACS it was evident, that we had to choose one AP. The decision to select AP 214 was made, because its scope is much wider than in AP 203. The Conformance Classes 1 and 2 of AP 214 are described as look a likes of AP 203 [4]. Additionally there have already been tries to implement other models into the existing AP [30]. The results from this work were used during the implementation of the converter tool (s. Section 5). Another key item of AP 214 is the scope on the easy handling of several variants of one product. This is important for the automobile industry (e.g. coupe and convertible of the same model) as well as for the aviation industry (e.g. several cabin layouts). Especially in preliminary airplane design several new concepts are tried, and should be easy to integrate into the central model.

4.2. CPACS

CPACS stands for **C**ommon **P**arametric **A**ircraft **C**onfiguration **S**cheme. The idea of CPACS came up during the first TIVA project at the German Aerospace Center (DLR)[16]. The DLR carries out major research in aerospace related technology. Due to its wide spread structure, many different tools are developed separately. These tools come from domain experts and vary in their level of detail from simple stochastic and analytic approaches up to full numeric simulations. While these tools have been tested in single domain projects, the future goal is the integration to reach more for a global maximum in multidisciplinary design workflows. Tools that need to be integrated are either under development or commercial (or tools with code that can not be accessed for other reason) tools. The number of interfaces can be decreased by introducing a central information model.

The goal of the TIVA project is to integrate these tools into one (or a set of) process chain. This led to the development of a single data structure called CPACS. The requirements that CPACS was designed for are:

- easy to handle
- hierarchic
- human readable and computer processable
- can be validated

The first point is probably the most important. Domain engineers first need to be convinced that a holistic approach to airplane design can result in major benefits for the product. The academic benefit from integrating a tool into a process chain in opposite is unquantifiable. Therefore the integration into and work with a new model should be as easy as possible. Note that the people working on the tools are in the first line engineers, mathematicians or other natural scientists. They are experts in their domains and inconclusively software engineers.

The requirement for a hierarchic and human readable format is standard in most information languages. The software-based interpretation is realized by a set of syntax rules. The possibility to validate the data is very important. However only a syntactic validation is possible. There is no opportunity to tell whether a value is reasonable or not in certain boundaries[17].

It was found, that the XML format offered the solution for the set off requirements. The format is widely spread and can be edited via a simple text editor. The tree structure

16 http://www.dlr.de

17 A negative value for a wing span is not sensible and can be caught by validation tools. A computer program can however not tell whether a wing span of 20m or 80m leads to correct results.

is lucid. By using the XML-Schema a validation could be introduced. These topics are elaborated in the following section more pervasive.

TIVA I started in 2005. Evolving from this first project CPACS has been used and extended in the following projects:

- **TIVA I & II**
  Technology Integration for the Virtual Aircraft
- **CATS**
  Climate optimized Air Transport System
- **EVITA**
  EValuation of Innovative Turbine Engines
- **UCAV 2010**
  Unmanned Combat Air Vehicle

The scopes of these projects are spread from analysis of climate data over the preliminary design of airplanes to multidisciplinary approaches in jet engine design. Of course, the detail level in each project is different. While for example EVITA contains extensive data on jet engines, the engine data in TIVA is relatively simple. TIVA II ended mid 2009 and is followed by VAMP. This project continues the integration of tools and establishes further process chains in the preliminary analysis methods. A more detailed description of the project's scope and the relevance of CPACS can be found in [6] by Bachmann et al.

The following sections at first gives an outlook over the project structure of CPACS, section 4.2.1. Subsequently, the CPACS model is explained. The metamodel is declared as XML-Schema (section 4.2.2) whereas the instances are stored as XML (section 4.2.3). Finally, the two libraries that are used for the manipulation of CPACS data TIXI and TIGL are presented in sections 4.2.4 and 4.2.5 .

4.2.1. Structuring in CPACS

As already mentioned, tools are integrated into CPACS using different libraries. The following figure 4.7 shows the overall structure. The reason for the central data format can be at first explained by the lower number of interfaces. For a set of tools that need to interface each other the number of interfaces is $n(n-1)$. With a central data format the number of interfaces is reduced to $2n$. Another reason is that the tools to some extent already exist. Hence their connection to the data format should be as easy as possible.

The tools are integrated using a framework that is based on a Server/Client approach. A user chooses a CPACS data set and connects the data set via a process chain to several

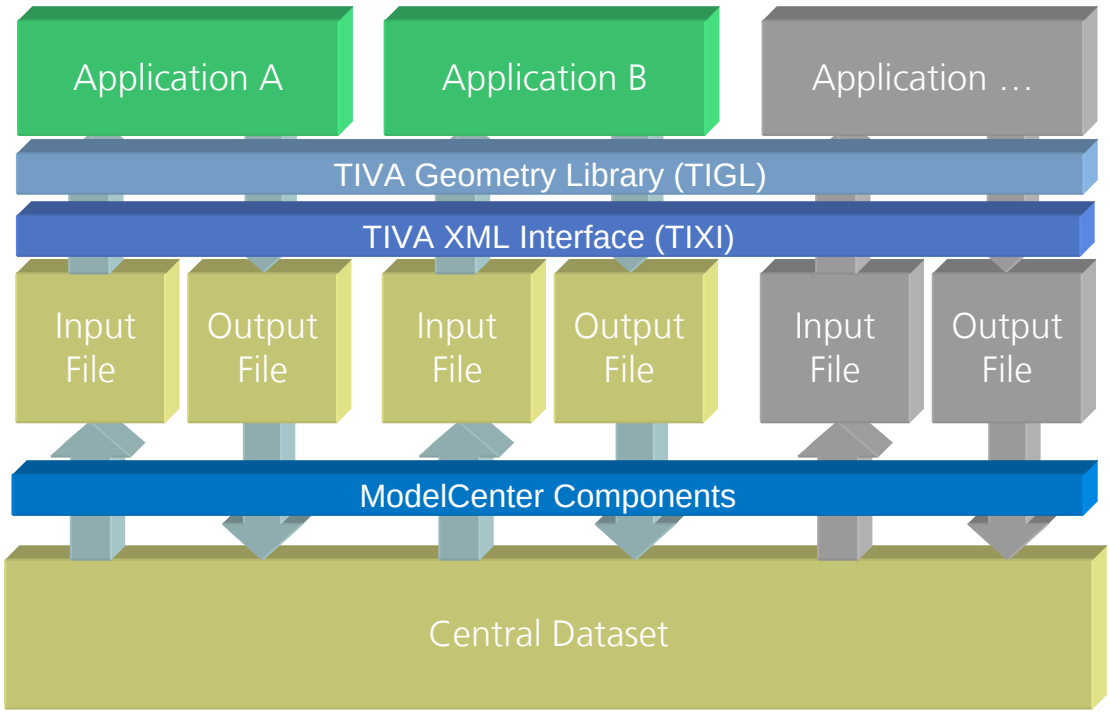


Figure 4.7.: CPACS Structure, from [37]

tools. The tools are running on servers that are set up at the different institutes of the DLR. While executing the process chain the data is transferred to and from the servers.

The integration of one specific tool is established by a mapping structure. This way a tool designer only has to build one XML-file for input and one for output. Via the mapping this data is linked to entities in the CPACS-file. A change in the CPACS-file only leads to a change in the mapping files. The tool itself does not need to be altered. The mapping is also established using XML-files. A short quote from a mapping file is shown in the following lines.

```
1 <map:mapping>
2 <map:source>/result/values</map:source>
3 <map:target>/configuration/common/values</map:target>
4 </map:mapping>
```

The example above shows a simple mapping. In this case an output from a tool is stored in a XML-File. The value from l. 2 is transfered to the value specified in l. 3. The mapping is probably a very powerful tool for the future work with CPACS. A new graphical editor for the creation of mappings is currently under development at

the DLR. One possible extension would be the execution of symbolic equations in the mapping rules.

It should be noted that the mapping is resolved to XSL Transformations[18] by the interpreting software. These transformations can be used to transfer one XML file into another. Alternatively to the shown mapping syntax above, a user can define mapping definitions in XSLT. The reason to develop an additional mapping syntax was to hide the difficult transformation language from the user.

As a framework currently ModelCenter[19] by Phoenix Integration is used. ModelCenter is the Client Software. One of the reasons to choose ModelCenter was its inbuilt optimization algorithms [32]. The tools run on *Analysis Servers*[20]. This way only a single version of a tool has to be administrated. The support of the tool is easier as well, as the servers are located at the institute that developed the code. Software is stored as simple executable programs that can be accessed through an API or command line interfaces. A change to a different framework is possible e.g. to an Eclipse[21] based system like the *Reconfigurable Computing Environment*(RCE)[22]. RCE is developed at the DLR as well. Another commerical tool for the integration of engineering tools and evaluation of process chains is iSight[23].

One of the problems with the current CPACS is the non-central administration. All named projects work within different domains. The EVITA project for example is based on jet engine design. The CPACS-version that is used in this project is therefore much more based on elements relevant for this scope. Other more holistic projects targeting airplane design might find a very different structure to organize jet engine data. The interchangeability of the process chains is therefor not given. Another problem from the decentral versioning is that some of the tools might not run in future versions of CPACS.

4.2.2. XML-Schema-Definition - XSD

The content-model of CPACS is created using the XML-Schema[24]. XML-Schema files are noted in XML format and stored via the .xsd extension. XML-Schema is explained by Duckett et al. in [16] and by van der Vlist in [60] as well as by the World Wide Web Consortium (W3C)[25]itself. XML-Schema files can be used to validate a set of XML data

18 http://www.w3.org/TR/xslt
19 http://www.phoenix-int.com/products/modelcenter.php
20 http://www.phoenix-int.com/products/analyzer.php
21 http://www.eclipse.org
22 http://www.rcenvironment.de/
23 http://www.simulia.com/products/isight.html
24 This section refers to the `Cpacs`-Version 0.9b from March 17, 2009
25 The W3C website (http://www.w3.org/) holds several recommendations for the work with XML-Schema. The most relevant documents are:
→ XML Schema Part 0: Primer Second Edition

in the same way as in STEP. One of the many tools for validation to be named is XSV[26]. Ancestors of XML-Schema are the Document Type Definition (DTD)[27] and the Regular Language Description for XML New Generation (RELAX NG)[28]. The following section gives an outlook on the mechanisms used to set up CPACS. Additionally, some of the entities in CPACS are described.

Similar to EXPRESS all definitions are saved in a `Schema` block surrounding all other implemented items. The following examples try to stick as near as possible to the introduction already given in section 4.1.2 on EXPRESS. At first a simple parameter of a base type is declared. The parameter is named `x` and is of a float type. In XML-Schema we use a DOUBLE type. The example is shown in the following line.

```
1 <xsd:element name="x" type="xsd:double">
```

Similar to EXPRESS, XML-Schema knows two different items. An `element` can be of any type and can be a leaf or node item in the XML-Tree. An `attribute` as it is noted in XML-Schema is always of a base type and may not carry any children items and is therefore always a leaf item. The `attribute` is equal to the same term in EXPRESS, so the `element` can be associated with the `entity`.

XML-Schema distinguishes between two types. The DOUBLE from the above example is a so called `SimpleType`. These types only restrict the text that is put into an attribute's instance. A valid input for the `x` attribute would be `3.142` but not `true` or `false`. `SimpleTypes` are used for attributes or text-only elements in XML-Schema. A `ComplexType` is used to describe an element. It declares the allowed sub elements and attributes of the element.

XML-Schema enables the user to create extensive documentation of the created data via annotations. The annotations can be used for the documentation of the saved data as well as annotations for applications processing the data. In this case the annotation is saved with an additional `appinfo` element. The following lines show a simple example for a documentation of the `x` attribute that has already been declared. The annotations can be used in a graphical form of the XML-Schema as it is shown later in this section.

→ XML Schema Part 1: Structures Second Edition
→ XML Schema Part 2: Datatypes Second Edition

26 http://www.w3.org/2001/03/webdata/xsv
27 http://www.w3.org/TR/REC-html40/sgml/dtd.html
28 http://relaxng.org

```
1 <xsd:annotation>
2   <xsd:documentation>X-Component</xsd:documentation>
3 </xsd:annotation>
```

Enumerations and lists are possible in XML-Schema but not in use in CPACS. Different from the modeling in EXPRESS where an attribute needs to be adressed by a value unless the attribute is optional, XML-Schema offers the introduction of a quantity of instances. The quantity can be restricted by `minoccur` and `maxoccur` elements. A `minoccur` value of zero is equivalent to the OPTIONAL keyword in EXPRESS.

```
1  <xsd:complexType name="pointType">
2    <xsd:annotation>
3      <xsd:documentation>
4        Point type, containing an xyz
5        data triplet with optional z component
6      </xsd:documentation>
7      <xsd:appinfo>&lt;point&gt;</xsd:appinfo>
8    </xsd:annotation>
9    <xsd:sequence>
10     <xsd:element name="x" type="xsd:double">
11       <xsd:annotation>
12         <xsd:documentation>X-Component</xsd:documentation>
13       </xsd:annotation>
14     </xsd:element>
15     <xsd:element name="y" type="xsd:double">
16         <xsd:annotation>
17         <xsd:documentation>Y-Component</xsd:documentation>
18       </xsd:annotation>
19     </xsd:element>
20     <xsd:element name="z" type="xsd:double" minOccurs="0">
21       <xsd:annotation>
22         <xsd:documentation>Z-Component</xsd:documentation>
23       </xsd:annotation>
24     </xsd:element>
25   </xsd:sequence>
26 </xsd:complexType>
```

The above example combines most of the items already named in this section. A `ComplexType` is used to model the `pointType` (ll. 1, 26). Annotations are used for `documentation` as well as for `appinfo` (l. 7). The `minoccur` (l.21) expresses the optional z-component of the point.

In XML-Schema inheritance mechanisms can be used. If a type inherits from another this type is called *derived* in XML-Schema. It is possible to derive `SimpleTypes` but for our example we stick to `ComplexTypes`. As there is no predefined element in XML-Schema that allows the modeling of symbolic equations such as in EXPRESS a different example for inheritance is chosen in this section. The equation from the `where` statement is dropped and a new `ComplexType` is created that inherits the `pointType` from the previous example. `ComplexTypes` can be derived either by extension or restriction. The new `ComplexType` is called `translationType` and needs to hold a data triplet as well. In the following lines the code[29] can be seen. For this case we used an extension. Usually with the use of extensions, further `attributes` or `elements` are added to the entity. Restrictions are used to insert further constraints to the entity. In the example the extension is used without adding any elements.

```
1 <xsd:complexType name="translationType">
2   <xsd:complexContent>
3     <xsd:extension base="pointType">
4     </xsd:extension>
5   </xsd:complexContent>
6 </xsd:complexType>
```

Additionally to the XML-Schema elements, the developers of CPACS included own mechanisms for the work with the content model. As already seen EXPRESS can model relationships easily. To allow the creation of these relationships in CPACS as well, a new element for identification was introduced. The `uIDs` are introduced as attributes. They enable the libraries to *jump* inside the model. The following line shows the code for a simple `uID`.

```
1 <xsd:attribute name="uID" type="xsd:string" use="optional"/>
```

An element like an airfoil for example is tagged by an `uID`. A wing segment can then reference the airfoil data by the `uID`. The translation element shown earlier can be used to alter the airfoil data properly for the wing segment. The structure of geometric data stored in CPACS is shown in section 4.2.5 more precisely. There is no mechanism in XML-Schema that allows the validation of `uIDs` in the way they are declared. They should obviously carry unique values. This is a potential source for errors, but is catched by the TIXI library.

29 This code does not originate from the CPACS XML-Schema. In the CPACS XML-Schema the `translationType` is a `complexType` that simply holds a element from `pointType`

An alternative declaration could be achieved using the unique keyword and Xpath[30]. This solution has one shortcoming compared to TIXI. The unique keyword can only be used on an instances of an element locally. The uIDs are unique globally throughout the document.

Including all data into one single XML-File could lead to very big files. This could cause trouble with the limitations of simple parsers as well as with the readability of the file. Therefore an `externaldata` node was introduced into CPACS. This element allows the import of data from other XML-files. The structure is redundant therefore huge XML-Trees can be constructed from simple files. Via an extension the following lines are added to each base type. So in all parts of the model external data can be attached to a leaf.

```
1 <xsd:attribute name="externalDataNodePath"
2   type="xsd:string" use="optional"/>
3 <xsd:attribute name="externalFileName"
4   type="xsd:string" use="optional"/>
5 <xsd:attribute name="externalDataDirectory"
6   type="xsd:string" use="optional"/>
```

Generally speaking, XML Schema offers a broad variety of modeling mechanisms. In depth these mechanisms show some weak points. The creation of relationships is not very lucid. Even though graphical representations of the model can be created, only the tree structure is recognizable. The inheritance mechanisms are suitable for easy modeling activities but e.g. do not enable multi-inheritance.

For future versions of CPACS a more extensive use of the inheritance methods should be made. The `translation` example shown previously in this section is a good example. In the current CPACS there is no inheritance from the `pointType`. As the `pointType` is also used for multiple other entities the interpretation of the data would be much easier with inheritance. A distinction between 2D and 3D points should be established as well. An entity should be restricted by modeling mechanisms and not only by documentation.

Although it is mentioned in the documentation of the CPACS XML-Schema that only SI-Units are allowed, these units should be modeled as well and get attached to the attributes. For example the attributes of the `pointType` should be from type [mm]. The `scalingType` which holds an element of `pointType` is without units since the scaling is based on dimensionless factors. This does not a support a sensible interpretation of the data. The author recommends to create a `dataTriplet` element that holds three dimensionless elements. Other elements such as `pointType`, `scalingType` or `translationType` should then inherit the `dataTriplet` and add units via extension.

30 http://www.w3.org/TR/xpath

Xml-Schema offers simple mechanisms for the creation of a content model. The standard is widely spread due to the extensive use in other application areas. Several tools, commercial and free, such as XmlSpy[31] or the XmlNotepad[32] can be used to work on Xml-Schema. With the use of stylesheets[33] graphical representations can be generated.

4.2.3. Extended Markup Language - XML

The probably most famous example for a markup language is the *HyperText Markup Language* (Html)[34]. The Html is used for displaying content in web browsers. Both Html and Xml have their origins in the *Standard Generalized Markup Language* SGML[35]. Detailed information on Xml can be found in the literature by Hunter et al. in [26] and on the already mentioned W3C homepage[36]. The tools named for the manipulation on Xml-Schema can be used for Xml as well.

It should be noted that Xml does not more than markup data. It does not help in any way to interpret or process the data as stated in [33]. Only putting data into Xml syntax can not result in a persistent data model. The creation of a metamodel (e.g. in Xml-Schema) is therefore an unreplaceable step. It can be found in [51] by Sachers, that there are no guidelines or examples how to use Xml in product data management yet. The current work on Cpacs may help to close this gap.

Comparing Xml and p21 files leads to one of the shortcomings of the format. There is no compatibility to any CAD-Kernels. The only way to access geometric data is through Tigl. The library is based on OpenCascade[3]. There is no exchangeability to other CAD-formats so far. The geometry information is extracted by Tigl, but it can not be processed to a representation without implicit steps. These calculation steps are not documented in the data itself. Hence a user can not generate a geometry representation by himself.

An example for a markup language that is used for geometry representation is the *Virtual Reality Modeling Language* (Vrml)[37]. This language is known for a very fast build up of geometry data. The fact that Vrml files are calculated for every view, makes them not useful for high-level representations. For fast geometry generation however Vrml might be an valuable extension[38] for the Cpacs data.

31 http://www.altova.com/xml-editor
32 http://www.microssoft.com
33 http://www.w3.org/TR/xml-stylesheet/
34 http://www.w3.org/TR/html/
35 http://www.w3.org/MarkUp/SGML/
36 http://www.w3.org/Xml/
37 http://www.w3.org/MarkUp/VRML/
38 In section 3 it was already mentioned that the access to geometric data should not be restricted to only the design department. As there are many free Vrml-players available, a low weight geometric

Dassault Systems, one of the major developers of CAD software, is currently working on a new file format named 3Dxml. A provisional version of a 3Dxml player[39] is already available. The format targets applications in the area of wide spread office software. In this way CAD data shall be made available in other company areas. Additionally, through a 3D printer information can be converted from OpenGl or DirectX applications. The sample data available is however not readable and can therefore not be analyzed.

Some simple examples for data modeled in XML following the CPACS XML-Schema is shown in the following lines. Primarily, a short example for a instance from `pointType` is demonstrated

```
1 <point>
2   <x>1</x>
3   <y>2</y>
4   <z>3</z>
5 </point>
```

The instance does not carry an identifier as in a STEP file. Although instances can be traced through the hierarchic structure of the document, an identifier like the `uID` or a xpath should be introduced into future versions. As already mentioned in the authors opinion, the attributes should be extended by a unit declaration. In this way the user could for example distinguish between data for rotation or translation.

One of the major benefits from the work with XML is graphical representation of the data. This has already been shown in section 4.1.4. The graphical representation is used to show an example of the header data for CPACS in figure 4.8.

Figure 4.8.: Header for a CPACS example

representation even in other departments (e.g. marketing) without CAD-equipment is possible. VRML would also offer the neat feature of a company specific background for the representation

39 http://www.3ds.com/de/products/3dvia/3d-xml/player/

As it can be seen, most data in the header is similar to STEP. Additionally, through the update data an in-file version control can be established. This enables the user to keep an overview over changes in the data. Yet there is no mechanism to create CPACS-files automatically. Because of this, the number of example data sets is low. A rule based system might help in the generation of valid CPACS data sets. This idea is currently under consideration at the DLR.

4.2.4. TIVA XML Interface - TIXI

One of the requirements during the development of CPACS was an easy integration of tools. This was one of the reasons to choose XML. Ongoing from this simple file format the TIXI library was designed to enable an easy access to data in XML files. In big XML files the tree structure grows large and referencing items becomes more time-consuming.

Additionally, the creation of data from simple types is easy. When writing data like arrays or vectors, these operations can become costly as well. The developer would have to create excessive and annoying string buffers for these operations.

The TIXI is based on the *lib*XML2[40]and offers an API for programmers using C, Fortran and Python. Through the API most operations to manipulate XML data can be triggered. The need for the different interfaces comes from two reasons. Primarily, this comes from the easy integration. The developers of the tools should be able to work in the programming languages they are familiar with. Secondarily, it is obvious that for the broad varsity of applications some languages are more suitable than others. For most numerical simulations Fortran is still used. Small scripts are written in Python.

If the software code of an analysis tool is not available, it is possible to introduce converter tools. These converter tools can then transform XML information into the native format of the analysis tool.

4.2.5. TIVA Geometry Library - TIGL

To get a better access and control geometric data in CPACS the TIGL was created. It calculates the geometry from the data in the XML file. Possible geometric entities so far are wings and fuselages. Through the TIGL-Viewer, which is based on OpenCascade[3], a graphical representation of the CPACS data is possible. An example for the GUI is given in figure 4.9. Note that this GUI comes from a testing toll named TIGLViewer. This tool is still under development and not an offical part of TIGL.

Additionally, TIGL can output data concerning the geometric structure. For example the number of airfoils in a wing, or the number of fuselage segments. The output is

40 http://www.xmlsoft.org/

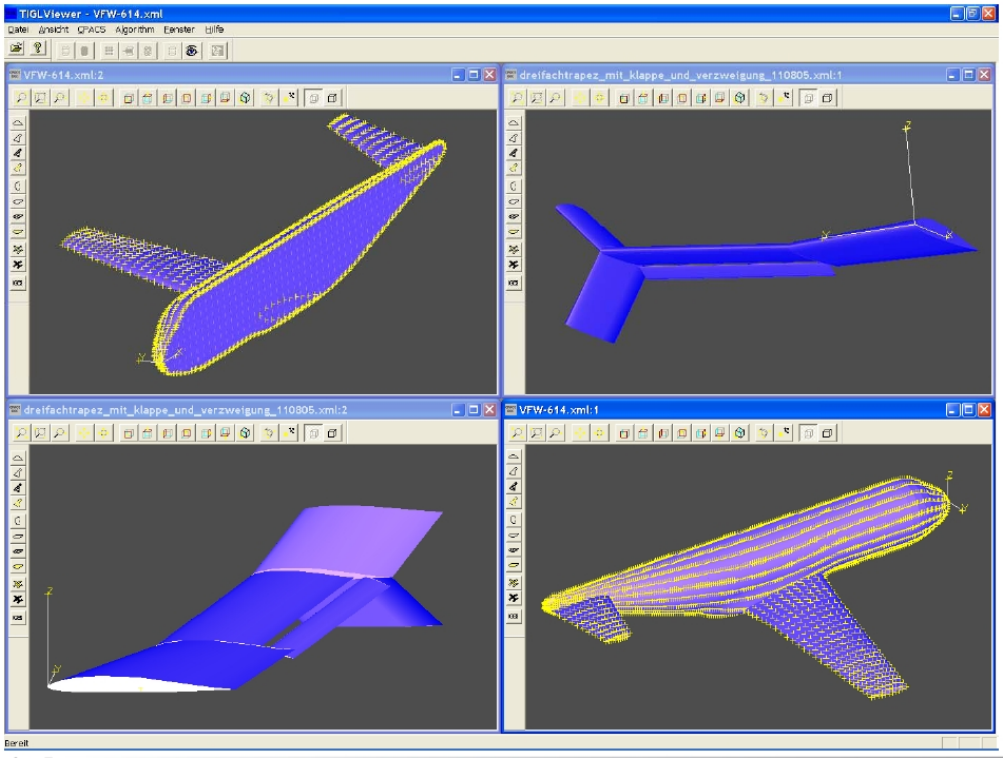


Figure 4.9.: TIGL-GUI, from [37]

however limited to theses numeric values. TIGL is not able to write geometric data. This issue has been identified as a problem and might be a target for future development. Alternately, it does enable the output as IGES or STL file. So far there are no tools in the process chains that modify the geometry. As soon as this is changed the issue becomes more important.

Although the existence of a geometric library offers great potential[41], this brings up problems for the model. As misleading information can easily be caught by the library, the risk to create misinterpretable models is great. This can be seen for the already mentioned `pointTypes`. In this case 2D information for airfoils is stored into the three dimensional element. For the calculation TIGL swaps values as they are not stored in the correct attributes. This operation can not be seen by the user. The model in this case is therefore false.

41 Commercial CAD Kernels can only be interfaced via APIs in many different and often aged languages. As TIGL uses OpenCascade theoretically a much wider application area is accessible.

4.3. UML

The development of the Unified Modeling Language began in 1990. The three authors Booch, Rumbaugh and Jacobsen joined their ideas for an object oriented modeling language in [10]. UML2 was published in 2005. The language is managed by the *Object Management Group*(OMG)[42]. Primarily, the UML has been developed to support the automatic generation and description of object oriented software. Through its graphic design layout it enables a lucid representation of contents. Currently the UML is extended to other domains such as systems engineering through the *Systems Modeling Language*(SysML)[43]. Further information regarding development using the UML can be found by Oestereich in [44]. Various tools for the UML can be found, either open source in the form of Eclipse[21] plugins or commercial tools such as Altova's UModel[44].

As well as a high bandwidth of modeling mechanisms, the UML offers several views on models. These views can come from different approaches, examples are a class or a use-case diagram. Through these views the model gets reduced to only some aspects. This mechanism allows a better overview, which is necessary in bigger models. Complexity can arise quickly in UML as Oestereich states.

The fact that future product models should use object oriented modeling has been stated by Stark in [55]. This section gives an outlook on some of the attempts to use the UML in engineering domains. As there is no major standard on how to use the UML in preliminary airplane design, most elements are explained by quoting from an airplane design language in the UML described in [9]. The author took part in the development of the design language.

Previously, some UML projects are introduced. Park and Sung [56] use the UML for the creation of a PDM system. All parts of the system are described in the UML. One of the requirements for the project was the web-based approach.

Lu proposes in her dissertation [38] requirements for a Next Generation Aircraft Conceptual Design Software Environment (NextADE). The work offers a detailed overview of modeling technique's and implementation method's history. The implemented version of NextADE is coded in Java with the help of UML. Several UML views are used for displaying the different development matters.

Van der Laan proposes the use of Knowledge Based Engineering (KBE) for airplane design in [59]. He describes the design process using KBE and outlines some tools that are used for this process. The step from an informal model to a formal is made using the UML.

42 http://www.uml.org/
43 http://www.sysml.org/
44 http://www.Altova.com/UModel

Oh and Yan [45] use the UML in combination with PDM Systems. In their approach the UML is used to create a mapping from product data stored in STEP files into a commercial PDM System. The transfered STEP is limited to CAD-Systems and conform to AP 203. STEP and UML are brought into relation more and more often. Besides the official Part 25: *EXPRESS to OMG XMI binding*, the exff[12] project has already be named. An overview on the connections between XML, UML and STEP is given by Peak et al. in [47]. In this work the UML is more seen as an implementation model. In the author's opinion the UML holds the most powerful modeling mechanisms of the presented languages. The following section gives an outlook.

4.3.1. Design Language for Airplane Geometries using the UML

The lack of a standard implementation[45] of product data in the UML leads to a more extensive presentation of a single approach for the preliminary airplane design. The following section outlines the work from a design language implementation that is described at full detail in [9]. The author contributed to this work and implemented the design language for airplane geometries during his secondary thesis at Stuttgart University. This could not have been accomplished without the expertise of Rudolph and Reichwein on design languages and the UML. The work is concentrated on the creation of geometries. Further work in this area extended the model for CFD-Analysis. More domains are to be accessed.

Therefore the design language can not handle data up to an extend such as CPACS or STEP. Further development might close the gap. It has however been shown that a holistic and multidisciplinary model could be developed. Additionally, through the methods of the design language, a fast and valid method for the creation of models is available for the user. This is an advantage over the other models.

Vocabulary

As mentioned before, the airplane design language vocabulary is expressed in UML through classes and instances. The implemented classes serve as templates for the generation of the instances. The objects are instantiated during the rule execution phase of the design compiler. The result forms a semantically interpretable model when combined in the correct way, using a set of rules in a production system.

The specific vocabulary is derived from the abstract intellectual decomposition of a future product concept. Since the Airplane Design Language is supposed to be used for the creation of geometries for an aerodynamic analysis using a computational fluid dynamics (CFD) software package, a geometrical and mathematical decomposition was

45 This does not imply that there is no interest in the industry for the UML. Some of the OMG members to be named are: IBM, HP, SUN, Oracle and Daimler Benz

chosen. Within a geometric decomposition scheme an object is split along its assembly boundaries. In many engineering disciplines information is bound to geometry. The mathematical decomposition is more function-oriented and uses typically mathematical operands and operators in the form of equations.

In the previous sections the point entity or pointType was used to elaborate some mechanisms of the language. Being the youngest of the languages for information modeling, the UML offers the biggest bandwidth of modeling mechanisms. In this section the point example is used as well. The following figure shows a point class and two classes inheriting from this class. The point class solely implements the three coordinates. Stereotypes tag the point class and its attributes to the adequate element in Catia V5[2] so that a output to CAD is possible. Using inheritance mechanisms and expressions new elements are created. The point on circle and point on ellipse classes are used within the design language for the modeling of fuselage ribs. Both classes hold new attributes and are determined through symbolic equations. Figure 4.10 shows the three classes along with the expression that hold the symbolic equations to determine the point on circle coordinates.

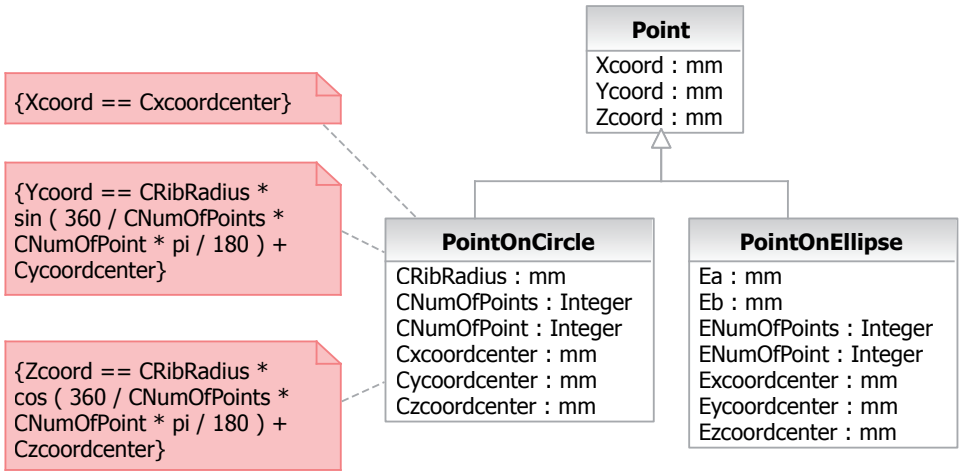


Figure 4.10.: Point Class, from [9]

With the introduced vocabulary several airplanes can be designed. Of course, a specific choice of a vocabulary set influences the future rule set. Since the rule set represents the design activity while the vocabulary represents the static entities, both choices influence each other and ennoble the conception of a "good" design language to an art, as long as no better, more systematic and theoretically sound way of doing this is known.

Classes are connected using generalizations and associations. Figure 4.11 shows the class diagram for the vocabulary in the Airplane Design Language. On top of the

class diagram is the class Airplane, similar to the tree hierarchy in Catia V5, where the airplane is the highest product instance. The classes on the next level are described in this section. They are the parts in Catia V5 holding geometric information about volume bodies. The class Profile is an abstract class. Only instances are created of classes inheriting the wing- or intersection-profile class. On the bottom of the class diagram in figure 4 several classes are shown that inherit the class point. Points are the geometrical and mathematical basis for the Airplane Design Language. The general point class links to a point in Catia V5. The more specific point classes own more mathematic constraints.

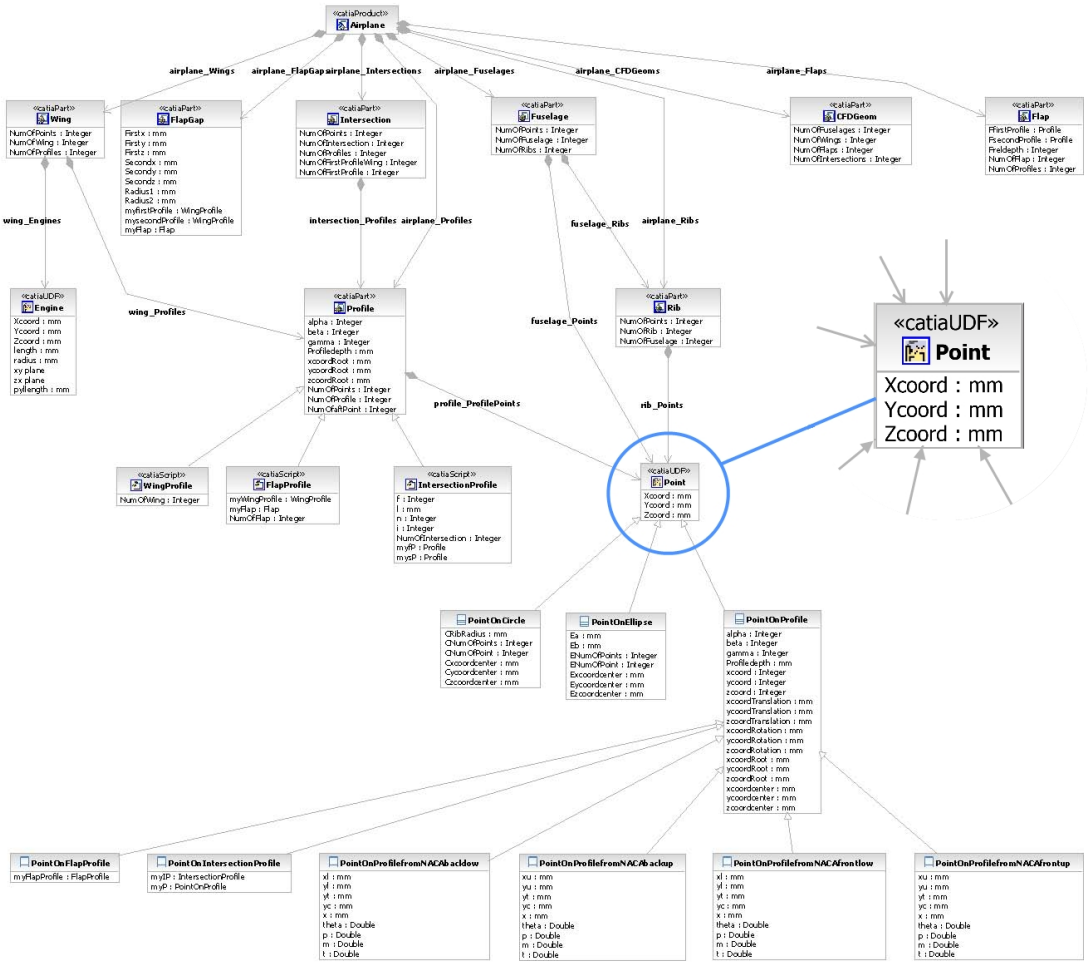


Figure 4.11.: Airplane Design Language Class Diagram, from [9]

Rules

Rules create instances out of classes. Using input parameters, they fill the instances with information to create semantic entities. In between instances, links must be set to connect corresponding elements. Links can only be built between instances, whose classes share an association. A fuselage only has a link to ribs belonging to the fuselage instance, not to all instances of the class rib. Using the object-oriented structure of the UML links and instance-specifications can be used to reference entities. A set of rules is called a production system. The production systems work through a sequential set of rules, chosen by the design engineer. The rules are written in Java code, a GUI for the selection of rules is currently under development.

4.4. Analysis/Comparison

In the previous sections three different information models were elaborated. The goal of this work is to evaluate these models in respect to preliminary airplane design. The chapters 2 and 3 line out some benchmarks. This section will analyze the introduced models in respect to the benchmarks.

Information in STEP is mostly classified as product structure elements. Additionally a classification relating geometric elements can be found. Apart from the geometric elements the classification is rough. CPACS is more detailed and holds an airplane related classification with clear structure. Additionally, CPACS can handle tool specific data. CPACS is therefore very powerful for preliminary airplane design, but the specific classification may result in conflicts when the data needs to be transfered. As it is already stated in the UML there is no standard for preliminary airplane design. This section can only rely on the approaches in the literature [38, 59, 9] outlined in section 4.3. The approach from the airplane design language uses a purely geometric classification. To reach the extent of STEP or CPACS it still needs to be enlarged.

Information Objects can be expressed via entities, elements, classes and attributes etc. in the three information models. Express and UML are more powerful when it comes to more complex elements like arrays or vectors. Also these two languages offer more possibilities for e.g. symbolic information. The current .p21 files of STEP however turn the structure from EXPRESS into hardly readable notations. The way to XML files for STEP data can improve the transparency of the information. XML is still limited on some of the listed issues. As a workaround CPACS allows external files, either coded in XML or some other file format.

Geometry was identified as key information object. STEP is by far the most prevailing geometry format in the industry. It can be processed by all major CAD Kernels. Parameters can not be modeled via STEP. CPACS offers a geometric approach using

the TIGL library and OpenCascade. OpenCascade offers for example mechanisms to output STEP data. Much of the geometry is however produced by TIGL, the implicit steps are not retraceable in CPACS. The only implicit geometry modeling is done in the airplane design language interfacing Catia. The macro parametric approach is still limited to one CAD-Kernel. As the geometry is modeled via symbolic equations, changes can be introduced easily.

All introduced models fulfill in some respect the requirement of holism. STEP is the most extensive model. An arising question is, if holism can be kept up with a rising number of entities. STEP can therefore be accessed by many tools. Version Control is not a big issue. The standard is published by the ISO. Changes are made rarely once an AP is published. CPACS introduces various disciplines on different detail levels into the design process. The mapping structure also enables the creation of further information. A rulebased mechanism to create models would increase the reuseability even further. A version control system is identified as possible future development. Again the UML model is not as voluminous as the other models, but the stereotype mechanism is useful for identifying domain specific elements. UML is not as widespread as XML and therefore the accessibility is rather low. In return the reuseablity is high.

EXPRESS is an advanced object flavored modeling language. The development started in the 1980s. Some of the mechanisms are therefore outdated (e.g. inheritance). Adopting the language as a OMG language and creating links to the UML and SysML are the current steps of development. XML-Schema is not an information modeling language by definition. It serves well for validation, but lacks several modeling mechanisms. The UML is the most advanced modeling language introduced in this work. Nevertheless, working with the UML needs some preparation and may be restricted to modeling experts. This does not imply, that the UML is a too complex language. On respect to the fact, that the barrier for a designer needs to be low the work with a instance model in the UML is more complex than working in plain XML.

The instance model of STEP was initially stored via p21 files. These files are hard to read and will be replaced by XML notations. The CPACS model is stored in a xml file. XML can be processed easily and is readable for everyone. The barrier for design experts starting their work on multidisciplinary models is therefore low. UML holds the instances in the same model. However UML data is saved as XMI and is therefore near XML.

Conclusion

Before a conclusion can be drawn it must be said, that the target information model needs to work in a relatively small environment. The key design processes are carried out by engineers at the DLR and are reduced to the implementation of new technologies. Hence the integration of domain experts is important. The conclusion drawn in this subsection must not be suitable for a major producer of airplanes. These organizations might have different requirements for their models. For the work on multidisciplinary models in preliminary airplane design at the DLR the author suggests a combination of UML or SysML as metamodel and XML as instance model for the following reasons:

1. **Why UML / SysML?**
   UML holds the most powerful modeling mechanisms. It originates from Software development and enables an easy integration of tools. SysML is a dialect of the UML specified for engineering systems. STEP is evolving into UML and SysML
2. **Why XML?**
   XML is *easy* - there is only a low barrier for domain experts. This the most important requirement. STEP is evolving into XML as well. XML can revert to existing libraries like TIGL and TIXI.
3. **Why not STEP?**
   The scope of STEP is large and at some points too abstract. The benefits of STEP (shareable, accessible) can be gained through a CPACS to STEP converter (see chapter 5)

For the development of a future modeling environment some questions are still to be answered. While SysML and UML are very similar both bring different advantages. On the one hand more software is available for UML as its origins lie within software development. On the other hand SysML is not as complex as UML and may be better suited for the description of product data.

Subsequently, several ways are possible to connect the metamodel, whether UML or SysML, and the instance model. One possible solution is to create a metamodel to the XML-Schema. The UML can map elements from XML-Schema. The intermediate schema can for example be used for validation. Alternatively, a direct link between UML and XML can be established. How much coding must be done for this approach, is a question for future research. Since UML data is saved as XMI it is even possible to extract instance data from it and store it in the known CPACS syntax in a XML file. Nevertheless, the list of possible approaches makes no claim to be complete and only future research can lead the way to the best solution.

5. Converter for Information Models

Although the CPACS standard is spread more and more through the DLR and it is written in XML, one of the most wide spread data exchange formats, one of the goals of this work was to increase its compatibility. STEP is a standard that can be interpreted by all more advanced CAD-Kernels. It was our goal to write a converter tool that enabled us to put out all data from CPACS to STEP. Besides the transformation of geometric data the question came up whether it is possible to:

- ⇀ reuse existing tools for straightforward development
- ⇀ transfer all CPACS data without data loss
- ⇀ put all other product and tool data into STEP
- ⇀ rebuild the part-hierarchic structure in STEP
- ⇀ develop a modular software that can be adapted to new CPACS Versions easily

For STEP as well as for XML there are several software packages available as free ware or under open source licenses such as the GPL. Commercial software is used mostly in the area of CAD. The tools used to develop the converter as well as the tools to test the results are outlined in section 5.1.

Some tries to implement additional data into STEP have already been published. Most of the connections are established via an additional mapping from external data to the ARM of the STEP data. The methodology of these approaches has been adopted to create the converter. The structure of the converter and the approach behind it are explained in section 5.2.

With the development of software, quality assurance is important. This is even more important for extensive amounts of data such as in CPACS. Several validation methods were applied to the created STEP data. These methods along with shortcomings of the translation can be found in section 5.3

5.1. Development Tools

5.1.1. Eclipse 3.4

The development of a converter tool forces the designer to work with various different computer languages. At first the languages of the two data formats which are to be converted into each other must be handled. Secondly, and very important in this particular case, the metamodels and their respective languages need to be interpreted to allow the handling of classes and make validations possible. Afterwards a central programming language must be chosen that can handle the named models as well as it offers a structure in which the converter tool can be written.

In modern software development Eclipse[21] is one of the most widespread development platforms. Through its plugin structure it can be adopted to various development environments. This applies to general decisions (which programming language) as for specializations (integration of e.g. STEP specific tools). Eclipse evolved from the IBM Visual Age for Java 4.0. Today it is developed further by the Eclipse Foundation. Although Eclipse is open source now, its development is still supported by IBM. Eclipse is described more precisely by Bernette in [12]. The converter tool is developed under Eclipse version 3.4.2.

Eclipse is based on Java[46] technology. Several other programming languages can be used with Eclipse, but we decided to stick with Java. First of all Java is a widespread and accepted modern programming language. Additionally most plugins (see other sections in this chapter) are written for Java code. Java is a modern object oriented programming language. It was important that the language for programming the converter tool at least owns all the modeling mechanisms the translated languages know. This way the superior programming language provides all the methodologies natively.

Java is a platform independent language. This ensures that the converter can be used in several environments. The language is developed and trademark of Sun Microsystems[47]. As Sun Microsystems was overtaken by Oracle[48] the future of Java is currently unpredictable. Information for the work with Java can be found by Sierra et al. in [53]. Java runs under version 1.6.

As the development of CPACS and the surrounding libraries is an ongoing process, a versioning system for the developed software was desired. The standard that comes with Eclipse is the *Concurrent Versions System* (CVS). At the DLR instead of CVS another versioning system named Subversion[49] is used.

46 http://java.com/de/
47 http://de.sun.com/
48 http://www.oracle.com/
49 http://www.eclipse.org/subversive/

5.1.2. JSDAI 4.2

With the development of STEP, an application programming interface was developed. It ensures that EXPRESS schemas can be created and manipulated. It also enables the output and validation of p21 and XML files. The interface is described in part 22: *SDAI Standard data access interface specification*.

LKSoft is a company that develops many STEP related tools. One of these products is JSDAI[4]. This Eclipse plugin enables the work with SDAI while using Java under Eclipse. JSDAI is published under the GNU APGL license. For this work we used version 4.2.0 of the JSDAI Eclipse plugin. Further reading about JSDAI can be found by Klein (Head of LKSoft) in [34].

Several packages from JSDAI can be used within Eclipse. EXPRESS related work is done using the EXPRESS compiler. For the work with the compiler an EXPRESS project needs to be created first. During the development of the converter tool a new schema was developed from existing entities, rules, types and functions from AP 214. This information is stored in a simple .exp file. Using the compiler creates a .jar library that can be used in other Java applications.

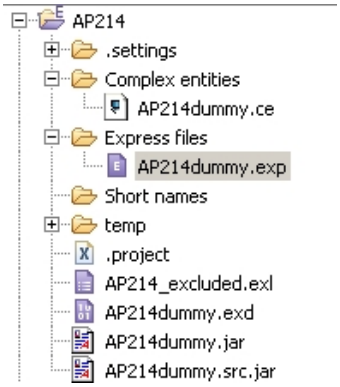


Figure 5.1.: Express Project in Eclipse Navigator

To compile complex entitiesm, they have to be declared separately in a complex entity (.ce) file. In the current work this was made for the already named unit example. The complex entities are declared in a plain file and are combined using a + sign as follows:

```
 1 length_unit+si_unit
 2 plane_angle_unit+si_unit
 3 si_unit+solid_angle_unit
 4 geometric_representation_context+ )
 5 global_uncertainty_assigned_context+ )
 6 global_unit_assigned_context
 7 geometric_representation_context+ )
 8 global_unit_assigned_context
 9 global_uncertainty_assigned_context+ )
10 global_unit_assigned_context
```

The created .jar file can be used within Java applications. It contains constructors as well as setter/getter methods for the compiled entities. For practical reasons a separate class was written that bundles these methods again. JSDAI holds several methods to create and manipulate data repositories. The repositories can be output via part 21 and 28. A validation for the created data is possible as well. For more information on the work with JSDAI help can be found on the projects tutorial[50] website.

5.1.3. JAXB 2.0

As shown in the previous section, for the handling of STEP related data JSDAI was chosen. Another package is needed that allows us to work with the metamodel and the data of CPACS. The CPACS data is stored via XML files and modeled in XML-Schema definitions. As these standards are popular, it is easy to find appropriate software. For the work with Java, the JAXB[51] package is the most suitable. JAXB is available as an Eclipse plugin as well.

At first, the metadata of the model has to be transfered into Java. For this purpose the XJC Plugin[52] is used. It is also part of the JAXB project. XJC allows the compilation of a .xsd file. Unlike JSDAI, it does create a separate Java class for each element in the .xsd file. These classes contain all information of the schema and are equipped with additional setter and getter methods. The following lines show a short example for a Java class, generated from XML Schema using XJC. The full example holding all XML annotations and Java methods can be found in the appendix.

50 http://www.jsdai.net/support/tutorials
51 https://jaxb.dev.java.net/
52 https://jaxb-workshop.dev.java.net/plugins/eclipse/xjc-plugin.html

```
1 public class PointType
2     extends ComplexBaseType{
3       protected DoubleBaseType x;
4       protected DoubleBaseType y;
5       protected DoubleBaseType z;
6       @XmlAttribute(name = "uID")
7       protected String uid;}
```

This example shows that the superior programming language should provide all (or more) modeling mechanisms as handled languages. Therefore the models can be integrated easily. The classes were integrated into the converter tool using a design pattern. Extensive information for design patterns is given by Freeman et al. in [21]. To extend the generated classes the *decorater* pattern is used. See the following lines for an example. Again the example is conform to the point element.

```
1 public class PointTypeC extends PointType{
2   PointType myPointType;
3
4   public PointTypeC (PointType pointType){
5     this.myPointType = pointType;}
6
7   /**Method to process Object Elements to STEP*/
8   public void process2STEP(...){
9     ...
10    }}
```

A new class is defined (l. 1) that extends the class that is generated via XJC. This class holds an attribute of type PointType (l. 2). The constructor (ll. 4,5) than accesses the attribute. Finally, a new method (ll. 7.10) is introduced that holds the constructor specific methods. These methods are explained further in section 5.2.

Besides the handling of XML Schema information, JAXB can be used to read out XML data and transfer it into a set of objects. These objects are instances of the classes previously generated via the XJC plugin. As all classes can be generated easily, the software can adapt to new versions of CPACS without inconveniences. Further reading for JAXB can be found by Michaelis and Schmiesing in [40].

5.1.4. Catia V5 R17

Catia² is one of the major CAD-tools that is in commercial use nowadays. The short form stands for *Computer Aided Three-Dimensional Interactive Application*. The version

5 is available since 1999. New retails are published on a regular basis. Catia is developed and distributed by Dassault Systems and IBM. Catia V6 has been published recently, but has not made its way to become an industrial reference yet. Catia V5 is an appropriate tool for testing the translated data as it is used widely in the aerospace industry (e.g. Boeing and Airbus). The STEP processor of the current version can handle the Conformance Classes 2,3,4 and 6 from AP 203 and 1 and 2 (assembly and 3D geometry / topology management) from AP 214.

5.2. Converter Structure

This section explains the approach that was taken to create a CPACS to STEP converter. In the previous section the used tools were described in more detail. We show how the information is processed through the different models.

As a first step, the converter reads a CPACS XML file from a named location. The named file is processed via the JAXB plugin. Its content is loaded into the class structure that was generated previously using the XJC compiler.

As the class structure is of course given, it is not known whether any or how many objects are instantiated in the CPACS tree. Changes can come up for every CPACS file. A fuselage element can for example carry fifty or hundred points. In this case the method must be modular so that it can act with an arbitrary number of objects. Another issue might come up with changes in the CPACS schema[53]. Updates from the schema are followed by changes in the compiled classes. As the compilation can be automated, the method should adapt to these changes by itself. The method that accesses the CPACS objects is therefore recursive and reflexive.

A recursive method calls itself as long as no break condition becomes true. The method designed to process the CPACS data starts at the top-level CPACS element and checks all lower classes. If a class holds an object it calls a *process2Step* method that converts the object. Afterwards the method calls itself and processes all sub elements of the converted class. The break condition becomes true if it finds an object of one of the base types.

While recursion is a term well known in software development, reflexion came up with the use of object oriented programming languages. Reflexion allows to gather knowledge about classes and objects during runtime. This way the central code of the converter can be kept modular. Even if the CPACS version changes, meaning that the compiled class structure changes, the methods still work because the class structure is not hard coded. The method already described therefore runs through the whole CPACS data and picks out all classes and objects while processing them. This way it

53 The current version of the converter tool works with CPACS version 0.9b. The release of CPACS version 1.0 is due to the end of July.

can be guaranteed that all information stored in the model is found. The converter tool knows two different sorts of classes:

1. elements and attributes that are translated generally, because there is no semantic equivalent in STEP
2. elements and attributes that need to be translated semantically correct, because they are interpreted by STEP processors

A very simple approach for the conversion of the first sort of classes is described by Jaroš in [30]. In his dissertation he describes the mapping to the simple elements `item`, `specific item relationship`. These elements are however part of the ARM. In the AIM the equivalent entities are `product` and `alternate product relationship`. The idea behind this mapping method is that elements and attributes are written to the `product` entities and the tree structure is reconstructed using `alternate product relationship` entities.

For the second set of classes specific methods have to be created that process the data to STEP. Section 5.1.3 shows how the classes generated from XJC are extended via the decorator pattern. Each new class owns a method that processes objects of this class to STEP. These methods are of course specific for each class. The second type of classes only comes up with geometric entities that need to be processed from CPACS to STEP. Figure 5.2 shows a graphical representation of the translation process.

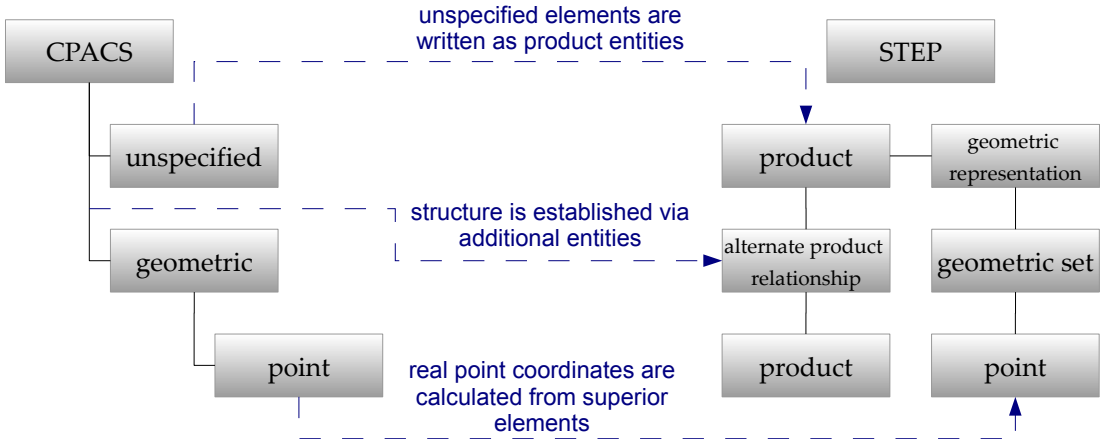


Figure 5.2.: Converter Structure

In STEP basic geometric data is bound to `point` entities. The `point` entities are processed directly and translated into cartesian coordinates. In CPACS however the `point` elements can hold all kinds of data. Generally speaking there are two sets of `point` elements. At first, there is a `point` element that is used to describe fuselage ribs and

airfoil data[54]. Secondly, `point` elements exist in the data that are used to modify the location from the previously named points. These elements enable rotation, scaling and translation. Only the first class `point` elements can be translated automatically to STEP. Anyway else a misinterpretation of the named elements from the second set of points might occur. Therefore this is one source of data loss during the translation process.

5.3. Validation

Although the developed converter tool is only a prototype, a testing procedure is introduced in this section. Two validation methods are possible. The first approach compares the number of elements and attributes from the XML data in CPACS with the number of entities that are created in STEP. As it is already mentioned in the previous section, data losses occur during the translation process. One simple example is all information stored in pointTypes that can not be processed to STEP. For example data stored in a translation element. There is no semantic equivalent to this element. Hence a comparison of the entity count is quite complex and does not reflect the quality of the tool.

Another validation method relates to the information that can be translated semantically correct. This concerns all geometry information. For this purpose the main fuselage section from the Attas was chosen. The model is stored in CPACS version 0.9c. The fuselage section is build up from 51 ribs. Each rib, except the first, holds 102 points. The volume of all the main fuselage sections can be calculated by TIGL.

Additionally, the converted version of the model is imported to Catia. In Catia scripts are written in VisualBasic code. These scripts create polylines and connect the points from the converted version. This way sections for the creation of a volume body are created. The creation of scripts in Catia is described by Ziethen in [63]. Using the measure tool of Catia the volume of the main fuselage section can be found.

	TIGL	Catia	Δ
	m^3	m^3	%
Volume	98,69	97,03	1,7

Table 5.1.: Validation by Volume Control

The above table shows the calculated values for the volume of the main fuselage section. It can be seen that the Δ for the overall fuselage volume is low. A more detailed listing can be found in the appendix A.

54 Additional `point` elements are used to locate e.g. the position of a micro for acoustic calculation. This data is however not in the scope of the geometric considerations

6. Summary

The goal of this work was to analyze the three different information models namely STEP, CPACS and UML in respect to preliminary airplane design. Additionally, the scope of this work included the concept and prototype for a CPACS to STEP converter tool

In this chapter the results of the work carried out are described briefly. A discussion will follow which analyses the usefulness of the suggested solution as well as the quality of the converter tool. Finally, a short outlook will be given on possible future development tasks concerning CPACS and the converter tool.

6.1. Results

For the analysis of different information models the lexical definitions for some of the major terms are introduced. Classification schemes that can be found in the literature are elaborated. The scheme of Rudolph is chosen and the named information objects are set into context with preliminary airplane design. After describing the information terms some important aspects for information models are defined. This list is however not complete and only outlines some of the major requirements. The common modeling mechanisms from object oriented languages are explained.

The main part of this work contributes to the analysis of the information models. Therefor all models are introduced extensively. The history and development of STEP are described along with the structure of it. The metamodel from part 11 and the instance model in part 21 are introduced. Some major parts are elaborated in more detail. Special interest is focused on AP 214. The origins of CPACS at the DLR are elaborated. CPACS is based on XML-Schema and XML. These standards are shown. TIXI and TIGL as the supporting libraries are outlined as well. For the UML several approaches for modeling in preliminary airplane design from the literature are introduced. On the example of the airplane design language some details are elaborated further.

In chapter 4 an analysis from the information models is drawn. While the size and abstraction of STEP make it inappropriate for a use at the DLR the other models show worthwhile approaches. On the one hand UML offers the most advanced modeling mechanisms and its usefulness for preliminary airplane design is shown in many cases. On the other hand XML is a wide spread and easy to understand language. It guarantees accessibility and allows exchange with other formats. The author there-

fore suggests a combination of these models for preliminary airplane design. The metamodel should be created using the UML or SysML. The SysML is a derivate of the UML and is specialized on Systems Engineering. Instances should be stored in XML. This also allows the continuous use of the already developed libraries.

This work introduces a scheme for a CPACS to STEP converter tool. The tool is created using open source software. Using Eclipse as development framework enabled the use of various plugins that are discussed in more detail. This way the code could be kept small and simple. The converter tool can be used to process information from CPACS to STEP. The geometry information can be accessed by standard STEP tools.

6.2. Discussion

During the first three chapters of this work several definitions are given. It can be seen at some points that terms used in the context of information modeling can not be hard defined. This is most obvious for *information*. Several definitions quoted from the literature are introduced in this work. Neither a precise description for the term is available nor can a uniform classification be declared. This vagueness makes it hard to define a holistic model. Even the requirement of a *holistic* information model is overqualified. A really holistic product description can hardly be established. Hence this work is focused on preliminary airplane design and the scope of technology integration at the DLR.

While examining the different information models it became obvious that in some cases the responsible organizations try to use synergy effects.

- EXPRESS is adopted as an OMG language
- STEP develops parts for systems engineering similar to SysML
- exff maps EXPRESS to the UML
- STEP can use XML as a new file format

In the recommendation for a future information model at the DLR it is tried to create a model that reflects these synergy effects and is similar to a future information model that might evolve in the industry. The choice of UML / SysML and XML can therefore also be justified by trends in information modeling.

The approach for a converter tool is outlined. A prototype is developed and the output to STEP geometry is established. A validation was carried out as well. Nevertheless, the converter tool can be improved in future versions. More details are given in the following section.

6.3. Outlook

In this section a short outlook is given on information modeling in respect to CPACS. The CPACS format is used more and more throughout the DLR. This is mostly because the related tools are generating benefits for distributed work and development. However, this results in a rapidly growing metamodel that handles data from various project horizons. Generally speaking, a growing metamodel is to be welcomed and more projects working with CPACS enable ongoing development. For the future boundaries should be set up between different CPACS subsets. The EVITA project for example handles extensive data on jet engines. The jet engine data should be stored in a subset of CPACS that is compatible to a subset for an airplane. The airplane subset is again compatible to a subset that handles climate data and relies on many airplane configurations. The technical possibilities for this are already given. The goal is to keep the CPACS data more lucid so that future projects can start easily with a subset of CPACS and not all information ever handled by CPACS.

For the future a combination of a UML metamodel with XML is suggested. For a possible implementation some questions are still to be answered. Use UML or SysML for the metamodel? Is a direct mapping from UML to XML possible? Or will there be an intermediate step using a generated XML-Schema?

The creation of CPACS data is slow. The current geometry model of the DLR's ATTAS holds more than fifty fuselage ribs. Each rib consists of more than hundred points. There is no automatic mechanism available to create this data yet. One possible approach for the creation of CPACS models is the use of a rule based system.

Also mentioned during the analysis of the models, version control is a central requirement for information modeling. A future CPACS maintenance could try to establish a version control, that handles changes in CPACS as well as changes in the involved tools.

There are certainly many possible fields of improvement for information models. Another approach could lead to the modeling not only of information but knowledge. This approach however will introduce a new field of research. Looking back at the introduction where information was defined as *data placed in context* a future approach needs to put both information and data into context.

Bibliography

Please note that a lot of research was made using the www. In this biblography section only print literature is listed. All information from the www is indicated via footnotes in the text. Footmarks are recurring, footnotes are set only once in the text. The websites were accessible during July 2009.

[1] *Oxford English Dictionary*. Oxford University Press, 2006.

[2] *STEP Application Handbook ISO 10303*. SCRA, 2006.

[3] AFWAL /MLTC. Integrated Information Support System(IISS), Common Data Model Subsystem, Part 4: Information Modeling Manual - IDEF1X. *AFWAL-TR-86-4006 5* (1985).

[4] Anderl, R., and Trippner, D., Eds. *STEP- Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303*. B.G. Teubner Stuttgart, 2000.

[5] Arbeitskreis "CAD/CAM". Umfang und Qualität von CAD/CAM-Daten. VDA-Empfehlung 4955/2, September 1999.

[6] Bachmann, A., Kunde, M., Litz, M., and Schreiber, A. A Dynamic Data Integration Approach to Build Scientific Workflow Systems. *Grid and Pervasive Computing Conference 0* (2009), 27–33.

[7] Batenburg, R., Helms, R., and Versendaal, J. PLM roadmap: stepwise PLM implementation based on the concepts of maturity and alignment. *Int. J. Product Lifecycle Management 1*, 4 (2006), 333–351.

[8] Bertino, E., and Martino, L. Object-oriented database management systems: Concepts and issues. *Computer 24*, 4 (1991), 33–47.

[9] Boehnke, D., Reichwein, A., and Rudolph, S. Design Language for Airplane Geometries using the Unified Modeling Language. In *ASME Int. Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE)* (2009).

[10] Booch, G., Rumbaugh, J., and Jacobsen, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[11] Brockhaus. *Brockhaus Enzyklopädie in 30 Bänden*, 21 ed. F.A. Brockhaus GmbH, Leipzig, Bibliografisches Institut und F.A. Brockhaus AG, Mannheim, 2006.

[12] Burnette, E. *Eclipse IDE - kurz &gut*. O'Reilly, 2006.

[13] Chen, P. P.-S. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (1976), 9–36.

[14] Choi, G., Mun, D., and Han, S. Exchange of CAD Part Models Based on the Macro-Parametric Approach. *Int. J. of CAD/CAM 1* (2002), 13–21.

[15] Codd, E. F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (1970), 377–387.

[16] Duckett, J., Griffitn, O., Mohr, S., Norton, F., Stokes-Rees, I., Williams, K., Cagle, K., Ozu, N., and Tennison, J. *Professional XML Schemas*. Wrox Press Ltd., 2001.

[17] Falkman, P., Nielsen, J., Lennartson, B., and von Euler-Chelpin, A. Generation of STEP AP214 Models From Discrete Event Systems for Process Planning and Control. *IEEE Transactions on automation science and engineering 5* (2008), 113–126.

[18] Feeney, A., and Price, D. FutureSTEP Project. Presentation, May 2009. 11th NASA/ESA Workshop on Product Data Exchange.

[19] Fidel, R., and Green, M. The many faces of accessibility: engineers' perception of information sources. *Information Processing and Management 40*, 3 (2004), 563 – 581.

[20] Fowler, J. *STEP for Data Management, Exchange and Sharing*. Technology Appraisals Ltd., 1995.

[21] Freeman, E., Freeman, E., Bates, B., and Sierra, K. *Head First Design Patterns*. O'Reilly, October 2004.

[22] Gao, J., and Aziz, H. Application of Product Data Management Technologies for Enterprise Integration. *Int. J. of Computer Integrated Manufacturing 16* (2003), 491–500.

[23] Geissen, M. Einen STEP vorraus. *Digital Engineering Magazin 2* (2005), 36–37.

[24] Goh, A., Hui, S., and Song, B. An integrated enviroment for product development using Step / Express. *Computers in Industry 31* (1996), 305–313.

[25] Hoofman, J., Mulyar, N., and Posta, L. Coupling Simulink and UML Models. *Formal Methods for Automation and Safety in Railway and Automotive Systems 1* (2004), 304–311.

[26] Hunter, D., Watt, A., Rafter, J., Duckett, J., Ayers, D., Chase, N., Fawcett, J., Gaven, T., and Patterson, B. *Beginning XML*. Wiley Publishing Inc., 2004.

[27] IGES/PDES Organization. *Initial Graphics Exchange Specification 5.3*. 1996.

[28] International Organization for Standardization. Industrial automation

systems and integration — Product data representation and exchange, Part 214: Application protocol: Core data for automotive mechanical design processes. ISO 10303-214, 12 2003. Second edition.

[29] Jarke, M., Mylopoulos, J., Schmidt, J. W., and Vassiliou, Y. DAIDA: an environment for evolving information systems. *ACM Trans. Inf. Syst. 10*, 1 (1992), 1–50.

[30] Jaroš, M. *Integration des STEP-Produktmodells in den Getriebeentwicklungsprozess*. PhD thesis, 2006.

[31] Katzenbach, A. Informationstechnik und Wissensverarbeitung in der Produktentwicklung. Lecture notes at Stuttgart University, 11 2008.

[32] Kersken, H.-P., Litz, M., Cornelsen, H., and Schreiber, A. A Software Environment for multi-disciplinary Simulation in Aircraft Predesign. *to be published*.

[33] Kimber, W. XML Representation Methods for EXPRESS-Driven Data. U.S. Department of Commerce, National Institute of Standards and Technology, 1999. NIST GCR 99-781.

[34] Klein, L. A Component oriented Software Architecture for Cross STEP-AP Implementations. Presentation, 2008. 10th NASA-ESA Workshop on Product Data Exchange.

[35] Klein, L. Linking ISO 10303 with the Semantic Web. Presentation, 2008. 10th NASA-ESA Workshop on Product Data Exchange.

[36] Lee, Y. T. Information Modeling from Design to Implementation. Manufacturing Systems Integration Division, National Institute of Standards and Technology (NIST).

[37] Litz, M., Cornelsen, H., and Kersken, H. Software Tools and Data Formats for Data Exchange in Airplane Predesign. Conference, 2008. PDE 2008.

[38] Lu, Z. *Data Management in an Object-Oriented Distributed Aircraft conceptual Design Enviroment*. PhD thesis, School of Aerospace Engineering, Georgia Institute of Technology, 2007.

[39] Meier, A. CAD-Datenaustausch - sicher und ohne Blindleistungsverluste. *Konstruktion 2* (2009), 27–28.

[40] Michaelis, S., and Schmiesing, W. *JAXB 2.0: Ein Programmiertutorial für die Java Architecture for XML Binding*. Hanser, 2006.

[41] Mun, D., Han, S., Kim, J., and Oh, Y. A set of standard modeling commands for the history-based parametric approach. *Computer Aided Design 35* (2003), 1171–1179.

[42] Mylopoulos, J. Information Modeling in the Time of the Revolution. *Information*

Systems 3-4 (1998).

[43] Nijssen, G., and Halpin, T. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. Prentice Hall, 1989.

[44] Oestereich, B. *Analyse und Design mit UML 2.0 - Objektorientierte Softwareentwicklung*. Oldenburg, 2005.

[45] Oh, Y., Han, S., and Suh, H. Mapping product structures between CAD and PDM systems using UML. *Computer Aided Design 33* (2001), 521–529.

[46] Pahl, G., Beitz, W., Feldhausen, J., and Grote, K. *Engineering Design*. Springer Berlin, 2006.

[47] Peak, R. S., Lubell, J., Srinivasan, V., and Waterbury, S. C. STEP, XML, and UML: Complementary Technologies. *Journal of Computing and Information Science in Engineering 4*, 4 (2004), 379–390.

[48] Reichwein, A., and Hertkorn, P. On a model driven approach to engineering design. *International Conference on Engineering Design* (2007).

[49] Ritter, J. *Historisches Wörterbuch der Philosophie : 13 Bände ; 1971 - 2007*. Schwabe, Basel, 1971. CD-ROM.

[50] Rudolph, S. *A Methodology for the Systematic Evaluation of Engineering Design Objects*. PhD thesis, Stuttgart University, 1994.

[51] Sachers, M. White Paper for PDM-Integration of OEM and Supplier in the Automotive Industry. White Paper, 5 2003.

[52] Schenk, D., and Wilson, P. *Information Modeling: The EXPRESS Way*. Oxford University Press, Inc., 1994.

[53] Sierra, K., and Bates, B. *Head First Java, 2nd Edition*. O'Reilly Media, February 2005.

[54] Smith, G. Utilization of STEP AP 210 at the Boeing Company. *Computer-Aided Design 34* (2002), 1055–1062.

[55] Stark, J. *Product Lifecycle Management , 21st Century Paradigm for Product Realisation*. Springer London, 2005.

[56] Sung, C., and Park, S. A component-based product data management system. *Int. J. for Advanced Manufacturing Technology 33* (2007), 614–626.

[57] Tamburini, D., and Peak, R. Overview of Information Modeling Using STEP EXPRESS, EXPRESS-G, and Part 21 Models. COA/CS/ME 6754, 2002. Georiga Tech.

[58] Tsichritzis, D., and Klug, A. C. The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Dabatase Management Systems. *Inf. Syst. 3*, 3 (1978), 173–191.

[59] van der Laan, T. *Knowledge based engineering support for aircraft component design*. PhD thesis, 2008.

[60] van der Vlist, E. *XML Schema: The W3C's Object-Oriented Descriptions for XML*. O'Reilly, 2002.

[61] Weber, C., Deubel, T., Köhler, C., Wanke, S., and Conrad, J. Comparison of Knowledge Representation in PDM and by Semantic Networks. *Int. Conf. on Engineering Design* (2007).

[62] Weber, C., Werner, H., and Deubel, T. A different view on Product Data Management/Product Life-Cycle Management and its future potentials. *Journal of Engineering Design 14* (2003), 447–464.

[63] Ziethen, D. R. *CATIA V5 Makroprogrammierung mit Visual Basic script*. Hanser Fachbuchverlag, Berlin, 2006.

A. Section Volumes ATTAS VFW 614

Section	TIGL	Catia	Δ	Section	TIGL	Catia	Δ
	10^{-1} m^3	10^{-1} m^3	%		10^{-1} m^3	10^{-1} m^3	%
1	1,43	1,35	-6,02	26	32,05	29,86	-7,33
2	5,83	5,74	-1,54	27	31,74	29,85	-6,32
3	9,85	9,69	-1,69	28	31,25	29,67	-5,31
4	14,56	14,32	-1,66	29	29,35	28,75	-2,09
5	19,32	19,02	-1,58	30	28,56	28,59	0,12
6	22,86	22,51	-1,54	31	27,54	27,11	-1,59
7	24,82	24,45	-1,50	32	25,59	25,98	1,52
8	25,39	24,99	-1,59	33	23,24	24,59	5,47
9	25,41	25,00	-1,65	34	21,49	23,05	6,77
10	25,41	25,00	-1,65	35	19,83	21,48	7,66
11	25,41	25,00	-1,65	36	18,14	19,87	8,73
12	25,41	25,00	-1,65	37	16,40	18,16	9,71
13	25,41	24,99	-1,69	38	14,53	16,37	11,25
14	25,41	25,00	-1,65	39	12,73	14,55	12,49
15	25,41	25,00	-1,65	40	10,97	12,75	13,97
16	25,37	25,10	-1,07	41	9,36	10,22	8,39
17	26,09	25,67	-1,63	42	7,82	7,84	0,26
18	27,96	26,55	-5,30	43	6,45	6,46	0,17
19	29,59	28,02	-5,60	44	5,22	5,22	0,04
20	30,35	28,42	-6,81	45	4,13	4,13	-0,04
21	30,98	28,99	-6,87	46	3,20	3,19	-0,36
22	31,86	28,99	-9,91	47	2,39	2,39	0,25
23	32,39	28,99	-11,72	48	1,72	1,73	0,18
24	32,61	29,42	-10,84	49	1,16	1,19	2,43
25	32,47	29,71	-9,29	50	0,45	0,47	5,43
				Total	986,91	970,45	-1,70

Table A.1.: Section Volumes

Note that the average root mean square (RMS) of the delta value is 5,81. The following figure A.1 shows the delta value per section. It can be seen that the delta is big at sections where the cross section changes. This can be seen at the front and aft of the fairing as well as at the nose and tail of the fuselage section. These deltas are due to

different volume body creations. The testing method is based on polylines without guides and therefore implies variances.

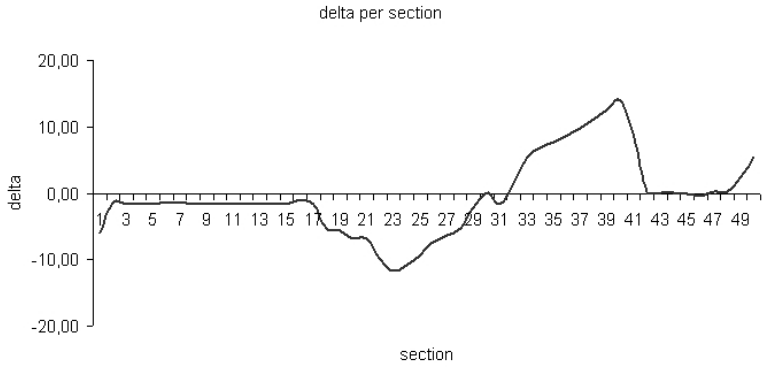


Figure A.1.: Delta per Section

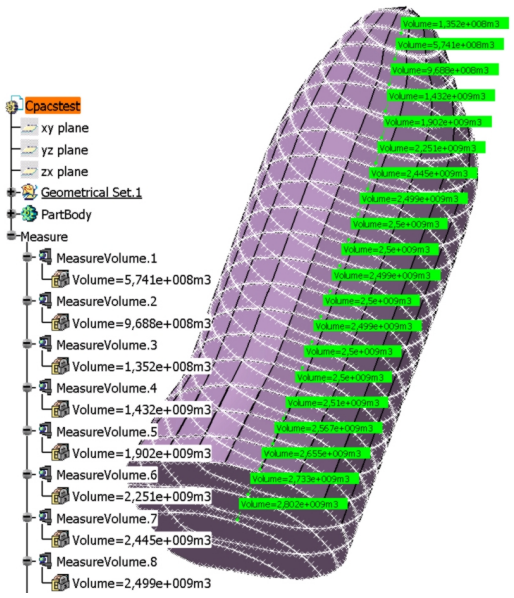


Figure A.2.: Volume Measuring in Catia

B. Setup for Development Framework

As already mentioned in chapter 5.1 the development environment is based on the Eclipse framework. Eclipse is available in various versions and can be configured via numerous plugins. For the work carried out the standard Java version is chosen. See fig. B.1. The software can be downloaded from `www.eclipse.org`.

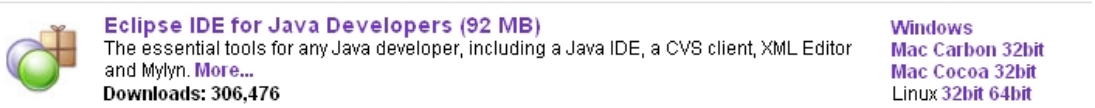


Figure B.1.: Eclipse for Java

Several plugins for the visualization and validation of XML are available for Eclipse. During the current development XMLBUDDY is used. The plugin can be found on the Eclipse webpage as well and is downloadable from the plugin section.

For further work with XML the XJC plugin from the JAXB project is used. The plugin can be found on the web at `jaxb-workshop.dev.java.net/plugins/eclipse/-xjc-plugin.html`. The downloaded files need to be extracted to the plugins folder of the Eclipse installation. Further information is given on the homepage.

The work with EXPRESS and the STEP is carried out under JSDAI. This plugin is available as a direct download via Eclipse. In the help section of Eclipse a software update register is available. As a new address `eclipse.jsdai.net` needs to be added. The JSDAI can then be integrated into Eclipse. The final build path is shown in fig B.2.

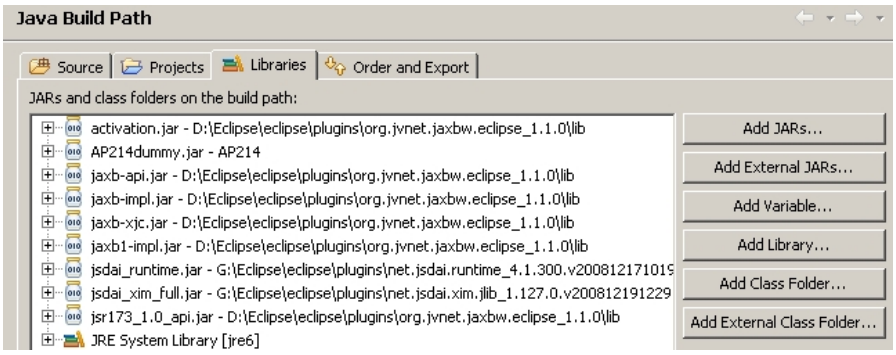


Figure B.2.: Libraries in Converter Project

C. Java Code for Point Entities

The following sections show Java code generated from Eclipse plugins. Similar to the structure in the text the point entity is used as an example. The first section shows the Java code generated by the XJC plugin. The interpreted entity comes from the CPACS XML-Schema in version 0.9b. In the second example the various classes and interfaces generated by Jsdai from a cartesian point in EXPRESS are shown.

Generated by XJC from XML-Schema

```
1  /**
2   * Point type, containing an xyz data triplet
3   *
4   * <p>Java class for pointType complex type.
5   *
6   * <p>The following schema fragment specifies the expected content
7  contained within this class.
8   *
9   * <pre>
10  * &lt;complexType name="pointType">
11  *     &lt;complexContent>
12  *         &lt;extension base="{}complexBaseType">
13  *             &lt;sequence>
14  *                 &lt;element name="x" type="{}doubleBaseType" minOccurs="0"/>
15  *                 &lt;element name="y" type="{}doubleBaseType" minOccurs="0"/>
16  *                 &lt;element name="z" type="{}doubleBaseType" minOccurs="0"/>
17  *             &lt;/sequence>
18  *             &lt;attribute name="uID" type="{http://www.w3.org/2001/XMLSchema}
19 string" />
20  *         &lt;/extension>
21  *     &lt;/complexContent>
22  * &lt;/complexType>
23  * </pre>
24  *
25  *
26  */
```

```
27 @XmlAccessorType(XmlAccessType.FIELD)
28 @XmlType(name = "pointType", propOrder = {
29     "x",
30     "y",
31     "z"
32 })
33 public class PointType
34     extends ComplexBaseType
35 {
36
37     protected DoubleBaseType x;
38     protected DoubleBaseType y;
39     protected DoubleBaseType z;
40     @XmlAttribute(name = "uID")
41     protected String uid;
42
43     /**
44      * Gets the value of the x property.
45      *
46      * @return
47      *         possible object is
48      *         {@link DoubleBaseType }
49      *
50      */
51     public DoubleBaseType getX() {
52         return x;
53     }
54
55     /**
56      * Sets the value of the x property.
57      *
58      * @param value
59      *         allowed object is
60      *         {@link DoubleBaseType }
61      *
62      */
63     public void setX(DoubleBaseType value) {
64         this.x = value;
65     }
66
67     /**
```

```
68          * Gets the value of the y property.
69          *
70          * @return
71          *         possible object is
72          *         {@link DoubleBaseType }
73          *
74          */
75         public DoubleBaseType getY() {
76             return y;
77         }
78
79         /**
80          * Sets the value of the y property.
81          *
82          * @param value
83          *         allowed object is
84          *         {@link DoubleBaseType }
85          *
86          */
87         public void setY(DoubleBaseType value) {
88             this.y = value;
89         }
90
91         /**
92          * Gets the value of the z property.
93          *
94          * @return
95          *         possible object is
96          *         {@link DoubleBaseType }
97          *
98          */
99         public DoubleBaseType getZ() {
100            return z;
101        }
102
103        /**
104         * Sets the value of the z property.
105         *
106         * @param value
107         *         allowed object is
108         *         {@link DoubleBaseType }
```

```
109         *
110         */
111        public void setZ(DoubleBaseType value) {
112            this.z = value;
113        }
114
115        /**
116         * Gets the value of the uid property.
117         *
118         * @return
119         *         possible object is
120         *         {@link String }
121         *
122         */
123        public String getUID() {
124            return uid;
125        }
126
127        /**
128         * Sets the value of the uid property.
129         *
130         * @param value
131         *         allowed object is
132         *         {@link String }
133         *
134         */
135        public void setUID(String value) {
136            this.uid = value;
137        }
138
139    }
```

Generated by JSDAI from EXPRESS

```
1  public interface ECartesian_point extends EPoint {
2
3      // generateExplicitAttributeMethodDeclarations: 1
4      // methods for attribute: coordinates, base type: LIST OF REAL
5      public boolean testCoordinates(ECartesian_point type) throws
6      SdaiException;
```

```
7       public A_double getCoordinates(ECartesian_point type) throws ↩
8       SdaiException;
9       public A_double createCoordinates(ECartesian_point type) throws ↩
10      SdaiException;
11      public void unsetCoordinates(ECartesian_point type) throws ↩
12      SdaiException;
13  }
14
15
16  public class ACartesian_point extends AEntity {
17      public ECartesian_point getByIndex(int index) throws SdaiException {
18          return (ECartesian_point)getByIndexEntity(index);
19      }
20      public ECartesian_point getCurrentMember(SdaiIterator iter) throws ↩
21      SdaiException {
22          return (ECartesian_point)getCurrentMemberObject(iter);
23      }
24  }
25
26
27  public class CCartesian_point extends CPoint implements ECartesian_point {
28      public static final jsdai.dictionary.CEntity_definition definition = ↩
29      initEntityDefinition(CCartesian_point.class, SAp214dummy.ss);
30
31      /*---------------------------- Attributes ----------*/
32
33  /*
34      // name: protected String ao;      name - java inheritance - STRING
35      protected A_double a1; // coordinates - current entity - LIST OF REAL
36      protected static final jsdai.dictionary.CExplicit_attribute a1$ = ↩
37      CEntity.initExplicitAttribute(definition, 1);
38  */
39
40      /*---------------------------- Attributes (new version) ----------*/
41
42      // name - explicit - java inheritance
43      // protected static final jsdai.dictionary.CExplicit_attribute ao$ = ↩
44      CEntity.initExplicitAttribute(definition, 0);
45      // protected String ao;
46      // coordinates - explicit - current entity
47      protected static final jsdai.dictionary.CExplicit_attribute a1$ = ↩
48      CEntity.initExplicitAttribute(definition, 1);
49      protected A_double a1;
50
51      public jsdai.dictionary.EEntity_definition getInstanceType() {
52          return definition;
53      }
```

```
54
55 /* *** old implementation ***
56
57      protected void changeReferences(InverseEntity old, InverseEntity newer) ↩
58      throws SdaiException {
59          super.changeReferences(old, newer);
60      }
61 */
62
63
64      protected void changeReferences(InverseEntity old, InverseEntity newer) ↩
65      throws SdaiException {
66          super.changeReferences(old, newer);
67      }
68
69      /*---------- Methods for attribute access ----------*/
70
71
72      /*---------- Methods for attribute access (new)----------*/
73
74      //going through all the attributes: #2635=EXPLICIT_ATTRIBUTE('name', ↩
75      #2633,0,#2677,$,.F.);
76      //<01> generating methods for consolidated attribute:    name
77      //<01-1> supertype, java inheritance
78      //<01-1-0> explicit - generateExplicitSupertypeJavaInheritedMethodsX()
79      //going through all the attributes: #2504=EXPLICIT_ATTRIBUTE( ↩
80      'coordinates',#2502,0,#2719,$,.F.);
81      //<01> generating methods for consolidated attribute:    coordinates
82      //<01-0> current entity
83      //<01-0-0> explicit attribute - generateExplicitCurrentEntityMethodsX()
84      // methods for attribute: coordinates, base type: LIST OF REAL
85      public boolean testCoordinates(ECartesian_point type) throws ↩
86      SdaiException {
87          return test_aggregate(a1);
88      }
89      public A_double getCoordinates(ECartesian_point type) throws ↩
90      SdaiException {
91          return (A_double)get_aggregate(a1);
92      }
93      public A_double createCoordinates(ECartesian_point type) throws ↩
94      SdaiException {
95          a1 = create_aggregate_double(a1, a1$, 0);
96          return a1;
97      }
98      public void unsetCoordinates(ECartesian_point type) throws ↩
99      SdaiException {
100         unset_aggregate(a1);
```

```
101         a1 = null;
102     }
103     public static jsdai.dictionary.EAttribute attributeCoordinates( ↵
104     ECartesian_point type) throws SdaiException {
105         return a1$;
106     }
107 
108 
109     /*---------------------- setAll() --------------------*/
110 
111 /* *** old implementation ***
112     protected void setAll(ComplexEntityValue av) throws SdaiException {
113         if (av == null) {
114             ao = null;
115             return;
116         }
117         a1 = av.entityValues[0].getDoubleAggregate(0, a1$, this);
118         ao = av.entityValues[3].getString(0);
119     }
120 */
121 
122     protected void setAll(ComplexEntityValue av) throws SdaiException {
123         if (av == null) {
124             a0 = null;
125             return;
126         }
127         a1 = av.entityValues[0].getDoubleAggregate(0, a1$, this);
128         a0 = av.entityValues[3].getString(0);
129     }
130 
131     /*---------------------- getAll() --------------------*/
132 
133 /* *** old implementation ***
134     protected void getAll(ComplexEntityValue av) throws SdaiException {
135         // partial entity: cartesian_point
136         av.entityValues[0].setDoubleAggregate(0, a1);
137         // partial entity: geometric_representation_item
138         // partial entity: point
139         // partial entity: representation_item
140         av.entityValues[3].setString(0, ao);
141     }
142 */
143 
144     protected void getAll(ComplexEntityValue av) throws SdaiException {
145         // partial entity: cartesian_point
146         av.entityValues[0].setDoubleAggregate(0, a1);
147         // partial entity: geometric_representation_item
```

```
148             // partial entity: point
149             // partial entity: representation_item
150             av.entityValues[3].setString(0, a0);
151         }
152 }
```