A Generic Information Platform for Product Families

A Doctoral thesis

by

Gunilla Sivard
TRITA-IIP-00-5
ISSN 1650-1888

© Gunilla Sivard
Division of Computer Systems for Design and Manufacturing
Department of Production Engineering
Royal Institute of Technology
S-100 44 Stockholm, Sweden

Stockholm 2000. Universitetsservice US AB
The research work detailed in this dissertation relates to the computer representation of information which concerns product families and product platforms. Common to competitive companies today, is the quest of designing products and processes to meet a large variety of customer needs, in short time, and based on few resources. One way to succeed with this endeavor, is to plan for the variety and design a modular, or adaptive, product family based on a common platform of resources. To further increase the efficiency in delivering customized products in time, a computer processible model of the family is created, which is used to realize a customer specific product variant during the order phase.

The objective of this research is to define a generally applicable model of product family information for the purpose of supporting various applications, and for achieving an efficient utilization of information.

The approach is to define a model of the product family according to the theory of Axiomatic Design, which reflects the trace from various requirements to functions and different properties and components of the product. By representing information from design in a generally applicable format, this information can be reused when building the configuration models of the order phase. By adapting the model to an existing standard, information exchange between systems is supported, and access is provided to information concerning detailed physical parts as well as constructs addressing various use and version management.

Contributions include a description of a model architecture with reusable functional solutions, interfaces, structures and interrelations between platform solutions and product family. Further, it is described how to extend and model the domains and interrelations of axiomatic design in an information model, which is adapted to the product modeling standard of ISO 10303-214.

Keywords: Information modeling, product families, modularization, product platforms, product data management, configuration.
First of all, I would like to thank my supervisor Prof. Torsten Kjellberg for his visionary view and open minded comments. Without his ideas concerning product models and integrated information, this thesis would not have been.

My industrial supervisor, Techn. Lic. Lars Lindberg has provided invaluable support and insight into the industrial views concerning product data and structures. Apart from commenting on many drafts, Lars has supplied an invaluable connection to ‘reality’. Thank you for this!

My advisor from Chalmers University of Technology, Prof. Johan Malmqvist, has also provided many constructive comments and suggestions, I am very grateful and really appreciate the red ink!

Thanks go also to my colleagues at the Division of Computer Systems for Design and Manufacturing, KTH, for fun discussions and useful comments concerning the information models of STEP AP214 and other subjects of the universe. Especially, I would like to thank Svante Barthon for inspiring discussions and for providing insight into the thinking of an inventor.

The ENDREA research program has also brought many research fellows out of whom I cannot mention all, but I do want to thank Peter Cronemyr for stimulating discussions and for being such a nice friend, and Roger Stake for all our discussions concerning modularity and product families - and for letting me ‘reuse’ many of his nice pictures and formats.

Finally, innumerable thanks go to my wonderful and supportive family. To my husband Anders for sharing and, at times, maintaining the whole household, and to my lovely children Theresa, Rikard and Amanda for all their love and laughter. My parents, Ansa and Olle, also needs a praise for being such good grandparents, and my rock band for occasionally filling my brain with music.

This work was financially supported by the Royal Institute of Technology and the Swedish Foundation for Strategic Research through the ENDREA research program. This support is gratefully acknowledged.

Stockholm in December 2000
Gunilla Sivard
# CONTENTS

ABSTRACT .......................................................................................................................... 1

ACKNOWLEDGMENTS ................................................................................................. 1

CONTENTS ....................................................................................................................... 1

1 INTRODUCTION ........................................................................................................... 1
   1.1 BACKGROUND ........................................................................................................ 1
   1.2 PROBLEM DEFINITION ....................................................................................... 3
   1.3 GOAL ....................................................................................................................... 4
   1.4 HYPOTHESES ...................................................................................................... 5
   1.5 THE GENERIC INFORMATION PLATFORM (GIP) APPROACH ......................... 5
   1.6 OTHER APPROACHES ....................................................................................... 10
   1.7 SCIENTIFIC METHOD ....................................................................................... 11
   1.8 CONTRIBUTION ............................................................................................... 15
   1.9 DELIMITATION ............................................................................................... 17
   1.10 OUTLINE OF THESIS DOCUMENT ................................................................ 18

2 PLANNED VARIETY - A STRATEGIC USE OF RESOURCES ......................... 19
   2.1 PRODUCT PLATFORMS AND MASS CUSTOMIZATION ....................................... 19
   2.2 APPROACHES IN INDUSTRY ............................................................................ 20
   2.3 MEANS FOR VARIETY ....................................................................................... 32
   2.4 REUSING PLATFORM SOLUTIONS ..................................................................... 39
   2.5 DESIGNING PRODUCT FAMILIES .................................................................... 42
   2.6 CONCLUSION .................................................................................................. 46

3 COMPUTER SUPPORT IN PRODUCT DEVELOPMENT ..................................... 49
   3.1 SYSTEM OVERVIEW ......................................................................................... 49
   3.2 PDM AND ERP SYSTEMS ............................................................................... 50
   3.3 CONFIGURATION SYSTEMS ............................................................................ 57
   3.4 CONCLUSION .................................................................................................... 68

4 PRODUCT MODELING APPROACHES IN RESEARCH AND INDUSTRY ....... 71
   4.1 THEORIES FOR DESCRIBING PRODUCTS AND DESIGN PROCESSES ....... 71
   4.2 MODELING PRODUCTS ..................................................................................... 75
   4.3 MODELING THE PRODUCT DEVELOPMENT PROCESS ................................... 77
   4.4 MODELING PRODUCT FAMILIES .................................................................... 80
   4.5 PRODUCT MODELING STANDARDS ................................................................ 83
### 5 THE GENERIC INFORMATION PLATFORM

- **5.1 PRINCIPLES** .......................................................... 99
- **5.2 SCOPE** ........................................................................... 99
- **5.3 ARCHITECTURE** .......................................................... 101
- **5.4 MODELING FUNDAMENTS** ........................................ 103
- **5.5 GUIDELINES FOR RETRIEVING INFORMATION** ....... 105
- **5.6 CONCLUSION** ............................................................. 108

### 6 DESCRIBING PRODUCT FAMILIES IN THE AXIOMATIC DESIGN FRAMEWORK

- **6.1 DESCRIBING PRODUCTS IN AD** ........................................... 109
- **6.2 DESCRIBING PRODUCT FAMILIES IN AD** ....................... 117
- **6.3 DESIGNING PRODUCT FAMILIES IN AD** ....................... 126
- **6.4 CONCLUSION** ............................................................. 139

### 7 EXPRESS-G INFORMATION MODEL OF GIP

- **7.1 MODELING AXIOMATIC DESIGN DOMAINS** .................... 143
- **7.2 MODELING EXTENDED CONCEPTS** .............................. 150
- **7.3 VALIDITY RULES** .................................................... 159
- **7.4 CONCLUSION** ............................................................. 159

### 8 GIP INFORMATION MODEL ADAPTED TO AP214

- **8.1 ADAPTING THE GIP TO AP214** ........................................... 161
- **8.2 PRINCIPLE FOR RELATING THE MODELS** .................... 162
- **8.3 MAPPINGS OF CONCEPTS** ........................................... 168
- **8.4 CONCLUSION** ............................................................. 169

### 9 CONCLUSION

- **9.1 EVALUATION** ............................................................. 171
- **9.2 DISCUSSION** ............................................................. 177
- **9.3 RECOMMENDATIONS FOR FUTURE WORK** .................... 178

### REFERENCES

- ........................................................................... 181

### 10 APPENDIX

- **10.1 DEFINITIONS** ........................................................... 192
- **10.2 GENERIC INFORMATION PLATFORM INFORMATION MODELS** 198
- **10.3 THE MODULAR FUNCTION DEPLOYMENT (MFD) METHOD** 206
- **10.4 AXIOMATIC DESIGN COROLLARIES** .......................... 210
- **10.5 EXPRESS-G MODELING LANGUAGE** .......................... 212


1 INTRODUCTION

This chapter provides an overview of the purpose, and scientific method of this research. The research hypothesis is presented as well as the idea of a Generic Information Platform (GIP) which, if implemented, would serve to prove the hypothesis. A summary of the main architecture of the GIP is presented as a means to provide a focus to the following chapters. The chapter concludes with a description of the research method and declares some contributions and limitations of the work.

1.1 Background

Three industrial issues are in focus in this thesis. First, it is the increasing demand for products which are customized according to individual desires, secondly the task of shortening the lead times to meet the customers preferences, and thirdly the quest for decreasing costs and use of resources. Meeting these three goals of increased customization, short lead times, and low cost at once requires a holistic view of product development, cutting down the "fat", i.e. the mistakes and large margins resulting from separate processes and miscommunication. Thus the key to success is not increased resources, but rather the increased knowledge and accessible information of how to realize the actual needs in the leanest way.

In product family based design, a multitude of variants of a product are created based on essentially the same product family concept. Product development is divided into two phases: the product development phase, and the sales-delivery process, or order phase [Prodevent 1975](8).
A product family as a whole typically has a long life where the effort put into planning and developing the product architecture and realizing processes is repaid during the order phase by multiple reuse of information and other resources.

Though interdependent, approaches to managing variance put different emphasis on these two phases. Either the development phase and the design of the product and manufacturing processes for variety is in focus, or focus is set on the order phase and the realization of an efficient sales-delivery process. Companies with experience of order based design have long focused on managing the complexity of handling a multitude of variants and their documentation, and on enabling efficient processing of customer orders. Scania and ABB, for instance, have been successful in developing computer based product configuration and data management systems. Apart from managing variety, the problem for these companies often extends to that of communicating information when manufacturing one product at varying plants, or when out-sourcing components to many different suppliers. Though improving efficiency significantly, their computer systems suffer in varying degrees from problems concerning the modification and exchange of information and do not provide full support to the planning of product families.

On the other hand, modularization and product platform development methods provide means for achieving a high degree of reuse by designing products to be...
based on the same technical systems and manufacturing processes. These systems and processes may be reused within a product family of variants as well as between products over time.

Designing a good product family requires a deep understanding of the market, as well as of the product and the capabilities and resources of the company. Understanding the interdependencies between customer requirements, product characteristics and properties, and the realizing processes, what to reuse and what to vary, becomes key knowledge.

1.2 Problem definition

According to the reasoning above, knowledge of how to manage information concerning various markets, partnerships and resources, is core competence.

Many computer systems support either configuration or product data management, and there are some methods for designing product families. The problem is, that there is not yet a unified framework for supporting product family development and for communicating and integrating information which concerns development as well as order processing.

ARCHITECTURE DESIGN SUPPORT. There is a lack of computer support of the reasoning needed in the planning and modification of a product platform. This concerns initial design as well as the reuse of modules and of product structures and rules when creating new products.

ORDER SUPPORT. It is costly to develop an efficient order system due to a lack of guiding principles, methods and systems. One often time consuming task, is that of capturing information: acquiring it from its source and expressing it in a computer processible form. In the design of knowledge based systems the designer often has problems describing his or her knowledge in an explicit way, in a way that suits the system, is consistent etc. A knowledge engineer and/or programmer is needed for the task of elucidating the information and programming the system. This is still one of the problems when building configuration systems [Tacton 2000]: information that is actually formulated in the development phase needs to be re-acquired when building the configuration system.

MODIFICATION MANAGEMENT AND INFORMATION EXCHANGE. The computer support is costly to develop and update with changes in the preconditions due to poor exchange of information between the development and order phases.

I. During design as well as order processing, product data needs to be managed, updated, kept valid etc. These are typical functionalities ad-
dressed by PDM systems. Of specific interest to the management of product family data is the integration of variety concerning different alternative products, and version management. Though systems supporting design and change management are starting to get integrated with systems which configure variants, there is still a lack of principles for how to manage variety, segment product family models, represent and manage interfaces, assign different effective components depending on product etc.

II. Information needs to be exchanged if there are many users and applications involved with the development of product families. Information may be imported from suppliers for validation, or configuration, or from different factories for validation of manufacturing systems or Design for Manufacturing. Specifications of components or manufacturing documentation may be exported to suppliers for outsourced design or manufacturing. Such information exchange requires a common standard for expressing variation.

1.3 Goal

The goal of this research is to define principles for modeling information platforms which integrate the information created and used during the design phase of a product family with that used during the order process, e.g. by a configuration system.

Though such a platform could encompass process and resource models, this dissertation focuses on identifying the basic underlying framework and principles for describing product families. Specifically, an emphasis is on:

1) the description of product family architectures and relating different variants to different sets of requirements
2) the description of modules and interfaces and the integration of these into a product architecture

Objectives:

Supply information support for product family architecture design
Provide the information basis for developing configuration systems and other variant realization systems in the order phase
Provide product information flexibility to changes in preconditions
Support exchange of information between applications and between users
14 Hypotheses

It is possible to define an information model of the concepts of Axiomatic Design, which captures the fundamental information of a configurable product family model.

This information model can be adapted to the ISO 10303-214 standard, thus providing an information basis for supporting several applications during the development and order phases of customer order based design.

1.5 The Generic Information Platform (GIP) approach

The approach is to model a product family architecture, according to the semantics used in product development, prepared for the information needs of configuration.

Figure 2 Information Platform - serving as a common information repository in product development
Further, this model is adapted to the AP214 product modeling standard, thereby creating a standardized information platform covering the reasoning of development as well as order processing.

1.5.1 Main architecture

The architecture is based on the needs identified in documented industrial cases and scientific methods for modularization and platform development, presented later in this thesis (Chapter 2, 3, 4) as well as on the information contents in the information modeling standard protocol, ISO 10303-214 also called STEP AP214, presented in Chapter 4.5.

Following the requirements of designing product families with a high degree of commonality as well as designing several products around reusable components, the two main elements of the architecture are: 1) generic product specifications and 2) reusable solution libraries.

![Diagram showing basic elements of the GIP](image)

*Figure 3 The basic elements of the GIP*

Product architectures and component architectures are treated in a similar way, enabling a hierarchical structure of structures. Thus classes, or families, of components may be selected from the solution library and integrated into the framework.

**Generic product specification/product family model.** A product family is
modeled in terms of generic structures with alternative solutions. A design trace - a decision tree of mappings between requirements and solutions - is used as a means for selecting a product variant which meets the requirements. Customer desires for varying product properties are interpreted either as functional requirements or as constraints on property values. A large product family is typically segmented into a hierarchy of sub families or other classes of related variants. A solution, such as a motor, which is common to all variants of one product family may for example be one of several alternative motors within another family.

**Platform solution models.** These models describe physical solutions as well as conceptual models and specifications on e.g. interfaces and resource limitations. By representing conceptual models, e.g. only described in terms of their main function and interface, with detailed models, a natural integration and reuse of information on different levels of abstraction is achieved. A solution may be considered a core technology to the company, or just a common resource.

### 1.5.2 Product family model based on Axiomatic Design

This architecture is based on the concepts of axiomatic design, thereby providing a generic model of the design object integrated with the design process.

Differences between product variants are represented as different requirements or constraints, or alternative realizations of a requirement, thus describing the configured objects and their configuration rules in terms of the design rationale.

A product can be described in terms of its properties and architecture. A product family is a (possibly unlimited) set of products with many characteristics in common, a class of products. A class of products typically has the main functional characteristics in common, i.e. they share the structure of the upper part of the functional hierarchy. In Axiomatic Design (AD) [Suh 1990], the design of one product is described as a zigzagging between the functional, physical and process domains, hierarchically mapping requirements to solutions to new requirements to solutions to realizing processes etc., under the conditions of delimiting constraints.
The axiomatic framework is generic and holds for any type of product, and an integrated description of the design object and the design process is achieved. Apart from providing a generic description of the design object and process, the Axioms - the decision criteria of de-coupling and minimizing information contents - are very adequate when designing modular product families. In addition to the original axiomatic framework, alternative requirements and solutions are modeled, as well as the composition of design parameters into components. Reusability is addressed by explicitly modeling modules and interfaces, apart from the technical solutions in terms of AD concepts.

In sales configuration systems, used in the order phase, the knowledge of the design domain is often described in terms of objects and relations. Each object is identified by its set of properties, which characterize its function and performance. Properties fall into three semantic categories: attributes describing characteristics, ports describing interfaces, and resources describing supplies. In the Generic Information Platform, both conceptualizations are blended by relating interfaces and resource limitations to component properties and constraints.
1.5.3 Product family information model - merged with standard

The purpose with adapting the conceptual model to a standard is twofold: first the standard provides functionality and detailed information models, and secondly a standard format supports the exchange of information between applications and users.

An information model describes the information of an application in a form that is well defined and computer interpretable, though not dependent on the specific data format or system used in the application. EXPRESS-G, an object flavored modeling language, is used as a language for expressing the models while the standard protocol, ISO 10303-214 also called STEP AP214, provides a standard for modeling product information.

AP214 provides an extensive and solid platform for representing detailed product family, and is well prepared for representing generic structures and bills of material as well as the information that defines the specific component alternatives of a product variant. Rules for relating interdependent component alternatives are also possible to represent by using Boolean constraints.

The purpose of AP214 was not initially to support the design of evolving solutions, and models basically represent fully designed solutions. Interfaces may currently be represented as detailed geometries but not in a generic way, and all alternative components are explicitly represented and combined according to rules. Still, there exist constructs within the protocol for describing conceptual components and relations, e.g. describing the ‘realization’ of a product function by a conceptual product component or physical item.

Objective: One practical objective of this research is to define how to extend the AP214 model with the ability to represent and reason from requirements, thereby realizing a broad integrated information platform with the ability to support planning and conceptual design and change management.

In this thesis, focus is set on the issues that are particular to representing information of developing product family architectures. The general product modeling issues concerning detailed models and PDM (Product Data Management) functionality etc. are considered to be covered by the existing AP214 modeling protocol. In relation to AP214, there are parts of the Generic Information Platform which are modeled according to the
protocol, as well as additions and modifications, described in Chapter 7. (The sizes of the segments of $\mathbf{\pi}$ are not representative, but only serve the purpose of visualizing the idea of extending AP214).

Since the purpose of the platform is to provide flexible and exchangeable information, the representation format will probably not be operative for fast configuration. Thus the idea is that the configuration systems will draw information from this communication platform and compile it into a form that is more adapted to executing configuration.

1.6 Other approaches

In Industry: In today’s product information systems, such as PDM systems and ERP (Enterprise Resource Planning) systems, information concerning requirements, products, and resources is typically separated. Since the task of integrating information is becoming a recognized issue by industry, there are three main approaches of providing: 1) one integrated system, 2) a standard communication protocol, or 3) a standard information model.

Large systems vendors tend to integrate information from different applications within their own system, thereby providing their users with an "Integrated Solution". This may work as long as all users use the same system, and as long as the need of the user exactly fit the information support which is provided. Apart from forcing the users into one framework, this one framework quickly becomes "hacked up" and complicated to work with, requiring a lot of help from consultants.

Another approach is that of developing standard languages and interface protocols for communicating information, often based on XML (Extensible Markup Language) [XML 1998], with a focus on seamless transfer of information between applications. For the purpose of integrating information, not only exchanging it, a common language is necessary but not sufficient. A model which describes the framework for the information is needed, modeling principles which identifies how to fit the new piece of information into the existing model. The information needs to be described in such a way that it fits in the model, not only in a language that is understood. It is not enough to use the same words, (such as product, article, interface, requirement), they have to mean the same thing and be possible to relate in the same way. The document type declarations (DTD) of XML provide a means for describing this meaning - how to interpret the contents of the XML file. Still it has to be decided according to what principles, or ontology, to define the DTD concepts.
1 INTRODUCTION

AP214 provides such a standard for defining product information. What lacks is a well defined model for representing concepts used during the planning and conceptual design of product families and platforms - requirements, modules and interfaces.

**In research:** Common to several research approaches is the goal of defining product family information in a general way, close to general principles for product modeling, and implementing models in an object oriented fashion [Tiihonen et al. 1998], [Jiao 1998] [Erens 1996]. Since configuration and management of generic bills of materials are the most common computer supported product family applications, several approaches focus on these applications. The conceptualization of the configuration domain [Tiihonen et al 1998] is for example primarily intended for representing configuration models for the purpose of supporting management of a large number of product variants.

These approaches, which are described in chapter 4.4, do not really address the issue of designing product families. The principles relating the variety of the functional domain to technical solutions and physical domains are not described and implemented in detail. They are purely object oriented approaches, with no expressed meaning of the relations between a class object and a member object other than "membership"; no rationale in terms of functional specialization between a parent-child, or difference in specialization between two alternative children of a parent. Important principles for how to represent the underpinnings of product families are thus missing creating a gap between the general principles for how to describe a product family (including different views, commonality, modularity etc.), and the object oriented implementations.

1.7 **Scientific method**

Research is a creative process which just as little as an innovative design process can be described as a straight line from needs to result. The tasks of "Design", "Manufacturing" and "Marketing" in the design helix description of product development [Suh 1990] (1) could well be exchanged for "imagine", "write report" and "present ideas" in research, and coming up with an adequate set of requirements may require many turns in this spiral. Still, when describing the process with the purpose of verification of the result, and of achieving a deeper understanding of the reasoning behind, a straight top down description is probably adequate.
Figure 6 The design helix  [Suh 1990]

As this research belongs to what is considered as engineering science, an appropriate description of the research process is [Sohlemius 1990]:

- Analyze what is
- Imagine what should be
- Create what has never been
- Analyze the result
Analyze what is - learn about the problem area and analyze what the problem or possible improvement is. This is the stage where the problem is stated and requirements are identified.

I. Identified problem of managing information relating to customer order based design
   A. lead to licentiate thesis focusing on solving limited configuration problems by use of constraint based reasoning [Sivard 1994].
   B. lead to the conclusion that the main problem concerned communication and management of information rather than performing design automatically

II. Modularization and platform management
   A. Discovered a related area with the issue of facilitating the development of several variants based on a limited set of modules. In a majority of modularization cases, an existing product was restructured to provide an architecture that was adapted for creating variants. In this area there was a lack of support for analyzing large product architectures to provide understanding of the dependencies between different solutions and the requirements.

III. Problem statement:
   A. there is limited support for expressing product variants and their relations to customer requirements and realizing processes during product family development.
   B. there is no connection between the information concerning requirements and varying solutions, developed during modularization, and existing PDM and configuration systems.
   C. there is limited reuse of existing product data during the development of product families, this may cause redundant work and uninformed decisions.
   D. existing order based systems are difficult to modify when the dependencies between selected variants, subsystems and requirements are implicit.

In short, the needs identified in product family development were those of representing the relations between market requirements, product properties and
components as well as managing strategic aspects of modules and their interfaces.

Imagine what should be

I. Realized that there was a common issue of developing many products based on the same concept.

II. Imagined a common model of such a concept, useable during development as well as the order phase

III. Identified that axiomatic design provides a consistent description of mapping requirements to solutions, but it is not a modeling framework.

IV. Imagined to model the domains and relations of axiomatic design, such that the design information could be used to represent product family concepts.

V. Imagined to use the existing information modeling standard of ISO 10303-214 (AP214) and extend it to account for the concepts needed in product platform and family development.

Create what has never been

I. Developed an information model of the axiomatic domains and mappings between requirements and solutions

II. Described how to integrate solutions into components based on methods for modularization

III. Described how to represent components and interfaces in the axiomatic framework

IV. Described how to extend the axiomatic framework to represent variance

V. Designed an information model of a small product family described in the axiomatic terms, modeled in Express-G

Product family constructs such as reusable components, generic structures and design rules are represented, based on a framework of requirements, solutions and interrelations, as defined in Axiomatic Design. The design framework of Axiomatic design provides an integrated description of the design object together with the design process. Relations between requirements and solutions are described as a zigzagging of mappings between requirements to solutions to new requirements etc., under the limitations of constraints concerning e.g. required performance or limited resources.
Analyze the result

The resulting principles have primarily been analyzed by describing that a generic information platform can be defined, which represents the product family model as designed, and which could be reused for configuration purposes. Further it has been shown how this platform could be integrated with the information modeling standard of ISO 10303-214, identifying similarities and discrepancies. A small audio system family is used to exemplify the model, and in a limited sense, to verify the ideas.

Further, in Chapter 9.1, p 171, an attempt to evaluate the reliability and validity of the results based on the five criteria: Internal logic, Truth, Acceptance, Applicability and Novelty value, from [Olesen 1992] is presented.

1.8 Contribution

This research provides a contribution to the effort of defining how to represent product family information in a general way: providing the basis for an information platform - useable by different applications and systems.

While the modeling of single products is cohesively addressed in design science [Krause et al 1993], this work is part of an effort of describing families of products as opposed to single products. The key issue is not the detailed descriptions of individual product qualities or behavior, but rather the description of what is common and what varies between product variants, as well as relating a specific product variant to a specific set of requirements.

Scientific contribution

Scientifically, the contribution is the general models of a product family as designed, in particular of the design rules - information which concerns the selection of a product variant based on requirements. The product family model is based on a scientific model of the design object and process - Axiomatic Design, as a means for representing the semantics of alternatives and choices in a generally applicable way. Based on this model, it is described how the design history of mappings from requirements to solutions can be reused to express the design rules used for configuration of a variant during the order phase.

Since Axiomatic Design provides a consistent framework for relating general requirements to solutions, this theory is used as a basis for representing variants and their trace.

The contribution in more detail:
I. A specification of how to model product family information as developed during design

A. An information model in Express-G of the domains of Axiomatic Design: a cohesive model of requirements, constraints, components and their interrelations based on the theory of axiomatic design

B. An extension of the axiomatic design framework to cover alternative requirements and solutions, DP integration into component models and an explicit representation of interfaces

II. Information model adapted to the AP214 standard, enabling information exchange and detail information representation capabilities

Figure 7 Research Contribution

Industrial

Industrially, product development is geared towards modularization, designing product families etc., as a means for delivering customized products, in time at low cost. This leads to an increased use of computer support when configuring variants and managing information and product data throughout the whole process of product development. The generic information platform, suggested in this work, constitute the basis for a stepwise acquisition and elaboration of information in a well defined structure in a product database.

By defining how to structure information which is created during conceptual design, and how to interrelate this information with detailed part models, existing PDM systems can be extended to administrate modules, rules and interfaces - crucial product family information.
For companies following the AP214 standard, the benefit is principles for extending their existing information models to represent information concerning e.g. requirements, technical solutions, interfaces, and variety in these, thereby supporting conceptual design and development of configuration models.

1.9 Delimitation

**Information, not procedures:** The objective of the research is to define the information contents, while the procedures utilizing this information, or other functions required to realize an operational system, are not covered.

**Level of detail.** A high level conceptual description of structures and dependencies is attempted in the GIP and the level of detail is low and generally applicable to all types of design. Although constructs are defined for the purpose of connecting detailed models to the structure, no deep modeling of neither function, behavior nor geometry is attempted. Information modeling of geometry is covered by the STEP standard, and since the GIP connect this standard, these models can be reached.

**Manufacturing models:** Though the long term vision is to incorporate models of manufacturing resources and processes [Johansson 2000], apart from the existing process descriptions of AP214, these are not addressed in this thesis. Focus has been set on relating product solutions to requirements, viewing the existing resources as a constraining requirement in itself.

**Sales models:** There is no representation of time, cost or other factors relevant to realize an efficient order process. Moreover, statistics from sales is not incorporated in this model.

**AP214.** AP214 is a protocol developed for the automobile industry. Though it has been found to be very general and applicable to most mechanical products, it does not cover standards for representing electrical and software systems. Since this thesis is concerned with the general structures and does not cover detailed technological descriptions, this does however not become an issue.

**Parametric design.** Parametric design, though important and not at all ruled out from the generic model, is not addressed in detail.

**Viewpoints.** The issue of different viewpoints is not treated in the conceptual design model, but rather one common model is suggested. Viewpoints on detailed components are supported in the AP214 framework, enabling different views of the information, depending on its role.
1.10 Outline of thesis document

In chapter 2, different approaches to product family planning and realization are described to provide an insight into aspects concerning product families and platform management. This chapter will also in more depth describe some principles concerning modular architectures.

Chapter 3 contains a presentation of computer methods and systems for supporting product development and variant configuration.

Chapter 4 contains a presentation of other approaches to representing and managing product family information.

Chapter 5 presents the main architecture and principles of the Generic Information Platform and serves as an umbrella for the three following chapters which describe the principles and models in more detail.

Chapter 6 contains one of the main contributions of this research: the description of product families according to the framework of Axiomatic Design.

Chapter 7 contains a description of how the product family can be modeled in the information modeling language Express-G, also a main contribution.

Chapter 8 describes how the model is related to the information modeling protocol of AP214, which constitutes a third contribution.

Chapter 9 concludes the thesis by discussing its validity and proposing recommendations for future work.

Definitions of some central terms are given in Appendix 10.1., and if no other notation is used, concepts are visualized using simple entity - relationship diagrams (π).

\[ \text{Figure 8 Legend of entity relationship diagrams used.} \]
2 PLANNED VARIETY - A STRATEGIC USE OF RESOURCES

In this chapter, different approaches to product family planning and realization are described to provide an insight into various important aspects concerning product families and platform management. This chapter will also in more depth describe some principles concerning modular architectures.

2.1 Product Platforms and Mass Customization

"Product platforms" and "mass customization" are popular terms for denoting the current efforts in industry of trying to efficiently design and use company resources - be it people, processes or components - to create various customized products in time, at low cost [Pine 1993], [Meyer & Lehnerd 1997]. Designing and efficiently using the full capacity of a product platform requires a broad understanding of how to use and develop resources for creating various products in different markets. Even with a moderately complex product, this means managing a large base of information about markets, technologies, processes, components, their alternatives and interrelations.

Common to different approaches to product family design is that many products are created based on essentially the same product concept, and the task is to reuse common resources as efficiently as possible. One principle of efficient reuse, of origin in combinatorial mathematics, is that of combining a small set of components in different ways to create multiple structures (if not constrained otherwise, the possible number of combinations of \(n\) modules (positions) each having \(m\) alternative realizations is \(m^n\)). This common principle is the basis for approaches based on modularity [Erixon 1998][Ulrich 1995](where the term "modular" refers to the characteristic that components can be combined in many ways), either the focus is on physical modules to be combined at time of assembly as in Mass Customization, or developing core technology modules based on which different products are based as in Platform Management. Even though modularity provides many benefits, in terms of cost savings as well as in strategic focus, there are cases where a more integrated approach is preferable or necessary. In the case of low volume, highly customized products, such as tailored suit jackets or large power transformers, it is for example not optimal to modularize at the physical solution level. Still,
there may be a high degree of reuse of design principles, rules and patterns (parametric models) as described in the case of *customer order based design* [Prodevent 1975].

### 2.2 Approaches in industry

In the following, some different aspects concerning designing and realizing product platforms and families will be described. Though in each case, and company, all these aspects may take part, one or two or the most apparent aspects will be in focus.

*Table 1 Different foci in product platform management*

<table>
<thead>
<tr>
<th>Product Focus</th>
<th>Process Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Family Architecture</td>
<td>Order based design/configuration systems</td>
</tr>
<tr>
<td>Core components</td>
<td>Flexible Manufacturing</td>
</tr>
</tbody>
</table>

Product Platform approaches focus on the core components in the modular system, while mass customization and product family approaches focus on the product family architecture and order processes.

#### 2.2.1 Planned innovation - Product Platform management

The evolution of an industry typically goes through phases characterized by different degrees of innovation concerning the product and manufacturing processes [Utterback 1994]. In the *Fluid Phase* (see *esseract*), the degree of product innovation is high, with a high diversity of products developed by many entrepreneurial firms. In the *Transitional Phase*, the main product architecture starts to stabilize and the development focus is on the innovation of the manufacturing processes. The third, *Specific Phase*, is characterized by mature markets, standardized products and often stagnation until the next stage of innovation.
Figure 9 The rate of innovation during the evolution of an industry [Utterback 1994]

In Product Platform Management, apart from designing one efficient platform, the goal is to maintain product innovation across multiple generations of a product family, continually introducing new technologies as they occur. The architecture and individual components of product families are planned so that a number of derivative products, addressing identified market segments, can be efficiently created from the foundation of a common core technology, called the product platform. "The product platform is a set of subsystems and interfaces that form a common structure from which a stream of derivative products can be efficiently developed and produced" [Meyer&Lehnerd 1997]. The main challenge when designing a product family is that of making a strategic trade-off between components standardization across a product family and the demand for distinct end products [Sundgren 1998].

The Power Tower method [Meyer&Lehnerd 1997], is one method for developing and continuously evolving product platforms, based on the principle of designing and efficiently using core modules in various products geared at various markets (*). The method emphasizes the importance of identifying the relations between the technologies, product components, and the markets currently addressed, to enable a strategic planning of the development and utilization of components. The five key steps, apart from that of assembling a multi-disciplined team, to guide the development of product families and platforms are:
1. Segment markets
   Identify major market segments and the price-performance tiers within them to construct a market grid.

2. Identify growth areas
   By gathering information concerning the current sales volume, expected growth rate and the driving customer needs the opportunities on the markets are identified.

3. Define current platforms
   Define where major product platforms "play" in the market, and the extent to which subsystems, interfaces, and manufacturing processes are shared between platforms.

4. Analyze competing products
   Within each market niche, index the functionality, cost and quality of competing products relative to the own products.

5. Consider future platform initiatives

   From the perspective of reuse, one key issue is to identify the core technologies and components within a company and to develop and refine these in dif-
ferent types of products. Sony, for example, has had great success in developing and evolving their HandyCam and Sony Walkman families on the American as well as Japanese and European markets [Sanderson & Uzumeri 1994]. Apart from standardizing their basic design elements and relying on flexible, automated manufacturing processes to keep the costs of the changes to a minimum, Sony controlled the costs of new models by building models around key modules and technologies such as their miniature motors. Sony succeeded in making 85% of their design changes based on minor rearrangements of existing features and cosmetic redesigns [Sanderson & Uzumeri 1994].

![Evolving product platforms at Sony]( Source: Sony HandyCam,)

Figure 11 Evolving product platforms at Sony [Sanderson & Uzumeri 1994]

In a similar fashion, Hewlett Packard adopted a platform strategy when developing their family of jet printers. These ink jet printers were developed around a platform of ink technology and print cartridges with locked interfaces. By continuously enhancing the platform, new families of printers with extended functionality or smaller size/lower cost, could be marketed [Meyer & Lehnerd 1997].

### 2.2.2 Product Architectures and Process flexibility - Mass customization

**Mass customization.** In Toyota’s effort of mass customization [Pine 1993], the goal was to make varied and often individually customized products at the cost of standardized, mass-produced goods. Mass customization initially focused on the processes of product realization, requiring "a dynamic network of processes and people that reconfigure perpetually" to produce exactly what customers want and need. This strategy has evolved into the use of flexible
manufacturing as well as designing product families and modular systems to satisfy customer requirements more fully.

Nippondenso Co. is Japan’s foremost manufacturer of automotive components. Basically through a modularized product structure and "assembly-driven" manufacturing, this company has succeeded in manufacturing a large variety of models at high volume with little changeover time between models [Whitney 1993].

"In fabrication-driven manufacturing, the order stream enters prior to fabrication, which makes a few complex parts to differentiate models of the product. In assembly-driven manufacturing, the order stream enters prior to assembly, which assembles a larger number of simpler parts to create different models. Parts are fabricated according to statistical trends in the order stream instead of the actual order stream. Fabrication-driven manufacturing is a low-bandwidth process compared to assembly-driven manufacturing since fabrication takes relatively longer than assembly, and changeover also takes longer." [Whitney 1993]
Configuration in Automotive Industry: One of the large truck companies in Sweden has a long history of standardization, modularization and efficient configuration of large volumes of alternative components. Based on a limited set of parts (for a truck cabin as in ‘ג’) a large variety of products can be configured.

Figure 12 Comparison of Fabrication-driven and Assembly-driven manufacturing [Whitney 1993]
To enable this, efficient support of information management is required: for configuring products based on the customer specification, for creating the manufacturing documentation, and for maintaining all documentation. This truck company represents an automotive industry with mature products having a high degree of standardization of components and information. An automobile or truck is usually co-developed by one OEM company and several suppliers, which emphasizes the need for a structured exchange of design information. Due to this need and the fact that different technical information systems (CAD, PDM) are based on native, incompatible, data formats, a protocol was developed within ISO 10303: STEP [STEP 1993]. ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. Typical data concerning parts and tools is: product definition data and configuration control data for managing variants of automotive products during the design phase; data describing the changes that have occurred during the design phase, including tracking of the versions of a product and of the data related to the documentation of the change process etc.

2.2.3 Process flexibility and information management - Order based design

In customer order based design, each product is designed based on one cus-
customer’s specification. With an unsystematic design, this has been identified to cause problems concerning:

- long and uncertain times to delivery
- problems to take in the whole picture of the assortment of components
- time pressure and problems with integration and use of resources

One suggested remedy to these problems is through a systematic and holistic view of the product development processes [Prodevent 1975]. In ABB:s design of power transformers, apart from the strategic planning of the product, focus was set on standardization of components, flexible manufacturing and computer based generation of manufacturing documentation [Sivard 1994]. To ensure a short development time of variants at customer order, using a minimum amount of resources, the work was divided into two phases; the development phase and the order phase.

![Diagram](image)

*Figure 14 The two phases in product development (same as Figure 1)*

In the development phase, a product concept is often developed and documented in paper format. Based on this concept, a computer processible product family model is developed consisting of detailed structure-, geometry, and NC templates, as well as of routines and rules for generating a specific product based on these templates [Lindberg 1991]. During the order execution phase, these templates are tailored and realized according to the specific customer needs.
In the order phase, a component may be selected in different ways [Sohlenius et al 1992]:

1. Ready made components brought from stock
2. Complete components bought for order
3. Manufactured components based on existing documentation (models or drawings). These components may be completely defined or differing in some dimension within a controlled range. The manufacturing documentation is prepared with the exception of a dimension i.e. the length of a pipe or angle of a bend.
4. Designed according to the product family model. Some calculation and design according to rules is required. Manufacturing is prepared according to rules within the model.
5. New design. These are components which fall outside of the defined concept, requiring new design solutions and manufacturing preparation.
Figure 15 Different ways of selecting components in order based design [Prodevent 1975]
**Order based design** is usually divided into four phases: process quotation, verification of the specification, order bound design and order bound process planning [Lindberg 1991]. The processing of a quotation is quite similar to the order bound design activities at large, except that the only planning activity in quotation is an estimation of the delivery time. To check the customer specification, make a design, and generate documentation are main activities for both. The major difference between quotation and order design is the level of detail. The purpose of the quotation process is to be able to make accurate cost and delivery time estimates and to generate sufficiently detailed documentation to the customer. The purpose of the order bound activities is to generate a product description and a manufacturing process description which are sufficiently detailed to be used for manufacturing of the product, as well as a documentation of the delivered product. Normally the same basic order handling system can be used for both quotations and orders.

The activity of order bound design is typically decomposed into several activities. First a calculation is made which often represents a mapping from the requirements on the primary functions to the key dimensions of the product. Second, the product structure is evaluated, generating a fully specified product structure including all parts and their relationships. Next, a detailed geometry may be created and calculations, e.g. stress analysis, are performed to validate the design. Finally, documentation is made, describing parts lists and drawings as well as discrepancies between the prepared product concept and the actual product design [Lindberg 1991].

**Order based configuration** is a type of order based design which implies the combining of pre-designed components. There are model-based and ‘open’
configurators e.g. Tacton [Tacton 2000] which are able to continuously delimit the range of alternative values and optimize a solution according to varying preferences. Other configurators are pre-determined, that is, based on a preset open architecture, only varying the components and often with only one possible solution to a given product specification. "Pure" configurable products can be defined as [Tiihonen et al 1998]:

1. Each delivered product is tailored to the individual needs of a customer
2. The product has been pre-designed to meet a given range of customer requirements
3. Each product individual is specified as a combination of pre-designed components or modules. Thus, there is no need to design new components as a part of the sales-delivery process.
4. The product has a pre-designed general structure.

The sales-delivery process of such a configurable product requires only systematic variant design, not adaptive or original design as defined by Pahl and Beitz [Pahl & Beitz 1988].

Order-based configuration is usually divided into two activities, one with the purpose of generating a valid product specification based on customer needs (requirements and preferences), and the other with the purpose of generating a detailed manufacturing documentation based on the product specification. For purely configurable products, these two activities are usually referred to as sales configuration and product, or engineering configuration.

**Sales configuration - going from requirements to design specification**

Task: From the requirements of one customer, create a specification (configuration description) of one product. Provide a description of the characteristics of the product and an estimate of the cost and time to delivery. This phase thus requires information for mapping from requirements as expressed by a customer, to the corresponding product properties. The configuration task is complex since it may require property modeling, or creating combinations of high numbers of alternatives that meet a certain set of requirements. Further, information concerning resource planning and time and cost estimates is required.

One difficulty in the quote or sales process is to identify what the customer wants and to transform these (possibly vague or incomplete) desires into appropriate technical parameter values, an activity sometimes referred to as "Specification Mapping" [Schwarze 1994]. Another view is to consider the whole process of sales configuration as part of acquiring a valid customer and product specification, since the configurator gives the customer the ability to
try out different specifications and learn about the resulting product properties [Tacton 2000].

**Product configuration - determining the manufacturing specification.**

Going from design specification to manufacturing description is an activity that requires detailed knowledge concerning the manufacturing processes and resources, and knowledge of how to describe the details of components for different processes. Another factor of complexity is that it requires the management of large sets of detailed, interconnected, information which may be continuously upgraded. Thus the issue of managing versions of configurations and their effectivity is of great importance.

Principles for configuration systems will be examined in more detail in Chapter 3.3.

### 2.3 Means for variety

Mass customization is efficiently realized either by using flexible processes, through the use of a modular product architecture, or both [Ulrich 1995]. Specific to a modularized decomposition is the goal to get the cost benefits of mass production by designing variants with a large commonality while realizing the variation by varying certain modules. Product structures are designed based on modules and interfaces which can be combined in different ways and thus create the diversity of products. Modules can be used as common components - one module used in many products, or variant components - different possible modules for one product.
2 PLANNED VARIETY - A STRATEGIC USE OF RESOURCES

Variety achieved by combinatorial assembly from relatively few component types
- Can assemble to order from component inventories
- Minimum order lead time dictated by final assembly process

May fabricate components to order as well as assemble to order
- May choose to carry component inventories to minimize order lead times
- Infinite variety possible when components are fabricated to order

Product Architecture

Modular
- High variety not economically feasible: would require high fixed costs (tooling), high set up costs, larger order lead times, and/or high inventory costs
- Variety can be achieved without relatively high inventory costs by fabricating components to order
- Minimum order lead times dictated by both component fabrication time and final assembly time
- Infinite variety is possible

Integral
- Low

Component process flexibility

Low

High

Figure 17 Product architecture and component process flexibility dictate the economies of producing variety [Ulrich 1995]

Adapting a modular architecture for a product family has many benefits concerning decreasing cost, while keeping short time to customer, and a high degree of customization. Still, depending on the strategy of the company, different activities will be in focus. In 2 some examples of different goals and how these can be achieved are listed.

Table 2 Different goals and strategies for product variety

<table>
<thead>
<tr>
<th>Goal</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrease number of part numbers</td>
<td>Standardization, integration, modularization</td>
</tr>
<tr>
<td>Minimize time to customer</td>
<td>Late point of differentiation, efficient sales-delivery process</td>
</tr>
<tr>
<td>Efficient variety</td>
<td>Portfolio planning, design of product architecture based on market, modularization</td>
</tr>
<tr>
<td>Minimize cost of a variant</td>
<td>Low change over cost in order process</td>
</tr>
<tr>
<td>Frequent introduction of new products</td>
<td>Flexible product architecture</td>
</tr>
</tbody>
</table>

In the following, aspects of processes and product architectures for variety will
be described in some more detail.

2.3.1 Processes for variety

Processes determine the cost and extra time requirements of realizing a variant as addressed in e.g. "Design For Variety" [Martin & Ishii 1997]. The processes of realizing the variety include the parts fabrication and assembly as well as the logistics of the order process. Sales-delivery activities are e.g. the specification of a product based on customer needs or getting a specific component in time, either it comes from stock or is out sourced to a sub supplier. A general rule for flexibility [Valckenaers 1993] states the benefits to LOW and LAte commitment. Translated into the design for variety, variety should be kept low and be introduced as late as possible in the development chain (to the lowest possible cost). Since variety of a product feature might be achieved in many different ways; by combining components or adapting a component feature, it has to be decided in the design phase in what way to vary the components in order to reach the desired variety at the least cost. Further, if the product family allows different product realizations depending on time and cost constraints, the choice of how to realize the variety can be decided at the time of order.

Based on the different identified activities in the development and order phases, a generalized development process of family based products can be described. Main activities during design are the component design, product architecture design, manufacturing process and system design, and order processing system design. During order, the activities comprise of the specification, manufacturing and delivery of a variant, and to some extent of the design of unique components that fall outside of the scope of the product family. A company may focus on one or several of these aspects depending on the nature of the product, the market, the strategy and competence of the company etc.
Common to all strategies is the specification of the product family and design of the common processes in a preparatory development phase. "Common components" may be physical components or the common characteristics of a parametric component or class of component alternatives as discussed earlier. The order in which these activities take place may differ; a new family architecture may be designed based on existing components or manufacturing resources, or without such constraints. Design of realizing resources and processes comprise the design of manufacturing resources as well as of the order based design/configuration system and other processes used in the sales-delivery chain. The manufacture of the common components may be performed before or after the point of sales - specification of the product variant. A decision concerning when to manufacture depends on the trade off between the cost of manufacturing ahead and keeping the component in stock, and the cost and time required of manufacturing or buying it at order. Often smaller batches are manufactured according to prognosis [Atlas Copco 2000].

If the product is complex, integrated and fully custom-made as in the case with large power transformers, most activities are performed at order and in principle inside one company, even though some components may be purchased or out-sourced to suppliers. On the other extreme, a company might focus on the activities of designing the product family architecture and realizing an efficient supply chain. In this case the design of realizing resources and processes comprise the design of the resources and processes in the supply chain.
A distributed design and manufacture is facilitated if the product can be defined and assembled by modules, as in the case of computer hardware systems. In that case the interfaces between modules are, and the main architecture is more or less, standardized. A high degree of outsourcing requires a close collaboration with suppliers [Dell 1999], and good market insight and knowledge of the changing technologies.

2.3.2 Product architectures for variety

Two interrelated aspects to the design of product families is how to design the product family structure and secondly how to design the components to make them reusable. For the purpose of enabling the reuse of physical components in different physical product structures, the product architecture has to be designed accordingly. A product architecture can be defined as the way in which the functional elements of a product are arranged into physical units, and the way in which these units interact [Ulrich & Eppinger 1995].

There are different distinctions between architecture types, such as that between a modular and an integral architecture. "A modular architecture includes a one-to-one mapping from functional elements in the function structure to the physical components of the product, and specifies decoupled interfaces between components. An integral architecture includes a complex (non one-to-one) mapping from functional elements to physical components and/or coupled interfaces between components" [Ulrich 1995 p. 422].

2.3.2.1 Modular architectures

The advantages of a modular architecture - using the combination of a small number of modules to realize a large number of products - are many. Direct improvements in cost reduction in lead time in assembly, work in process, assembly time, etc. are shown in many industrial studies [Östgren 1994]. In general, these studies show that modular design also is "a robust basis for
product renewal and concurrent development of the production system”, provides short feedback links due to the ability to test modules separately, and gives "positive effects to the total flow of information and materials” [Erixon 1998]. Main reasons for modularity can generally be said to be the increased ability to create variety, while utilizing commonalities and reducing complexities. [Miller&Elgaard 1999]. In addition to these reasons, the intent to prepare for future changes should also be part of the module drivers of flexible products [Erixon 1998] [Stake 1999].

2.3.2.2 Types of modular architectures

Modular architectures can be divided into three subtypes: slot, bus and sectional. These differ mainly in terms of their interfaces: in a slot architecture the structure is the same, with many variants but one type of interface for each ‘slot’. Each component is of a different type so that the various components cannot be interchanged (e.g. automobiles).

![Diagram of modular architectures](image)

*Figure 20 The different types of slot modularity from [Ulrich 1995]*

In a bus architecture, all components have the same physical interface and are only connected to one common element (e.g. electronics or roof racks). In a sectional architecture, on the other hand, all interfaces are also of the same type but there is no single element to which all the other components attach (e.g. sectional sofas or piping systems) [Ulrich 1995 p. 424].
Bus Modularity Sectional Modularity

Figure 21 Bus and sectional modularity from [Ulrich 1995]

Modules can differ in size or in some other aspects as in "cut-to fit" modularity, as long as they keep the same physical interface and realize the function in the pre defined way.

2.3.2.3 Module definitions

There are two essential characteristics of a module: its functionality and its interface. A commonly agreed upon definition of modules is not yet found, the main differences being the view of how the functionality of a product is realized within the modules and how to define a module interface. "A module is an essential and self-contained functional unit relative to the product of which it is part. The module has, relative to a system definition, standardized interfaces and interactions that allow composition of products by combination" [Miller and Elgaard 1999].

[Errens 1996] as well as [Ulrich 1995] expresses modules and architectures according to the mappings between functions and physical modules:

1:1 One function is allocated to one technology module. This is modular design.

1:N One function is mapped to several technology modules. This distribution of function over several modules results in an integrated design on that level.

M:1 Several functions are allocated to one technology module. Function sharing increases the level of integration.

M:N Several functions are allocated to several technology modules. Functions are distributed and shared, thereby further increasing the level of integration.

Most definitions are thus concerned with defining a functional module claiming that a module is a component which aggregates similarities concerning function. From a practical point of view, other dimensions of similarity have to be considered: the Modular Function Deployment (MFD) method, is a five
step product development method for designing modular products [Erixon 1998]. In the MFD method, described in Appendix 10.3, modularization is defined as: "Decomposition of a product into building blocks (modules) with specified interfaces, driven by company-specific reasons" [Erixon 1998]. Thus the goal of a modularization is to define a product structure that reflects not only the functional requirements of a product family, but also strategic considerations concerning e.g. manufacturability (commonality), customization (variety), flexibility (development) and life cycle aspects such as serviceability. To actually decide exactly how to compose modules requires the experience and knowledge of designers, and the method should be seen as a tool for supporting the reasoning required when modularizing, rather than as a complete algorithm. Tools based on cluster analysis from the statistical domain are suggested to further support designers using the MFD-method [Stake 2000]. Thus, are two types of modules: modules which are functionally pure, and ‘strategic’ modules which encompass a trade off between various reasons for establishing them as separate units. Common to both descriptions is the focus on robust interfaces, functional as well as physical, which facilitates (de-) connecting the modules. As the strategic module definition takes more aspects into account, this is the definition that seems most useful in practice.

2.4 Reusing platform solutions

2.4.1 Reusing modules

When determining how strictly to design and define the interface, the purpose and range of use of a module must be considered. It might be intended to be used for one specific product family or for several products within a company, or maybe as a standard unit useable by any company. Depending on the range of use, the robustness and documented definition of the interface is of different importance. Parallel to Ross’s guidelines for modeling: when designing a module, it has to be considered for what purpose, from which viewpoint and level of detail [Ross 1977].

Components/modules are reused for many different purposes and applications within mechanical as well as software and firmware design. Standardized machine elements for example offer one type of vast reuse of mechanical parts. In the design of integrated circuits, IC, there is an increasing effort to standardize different solutions to certain software problems (processors, memories etc.) for the purpose of building up libraries of intellectual properties IP:s or Virtual Components (VC:s). By selecting and combining different IP:s, different archi-
tectures of integrated circuits can be designed. To enable this, there is a great
effort in defining how to represent the functional interfaces of these IP:s [VSI
Alliance 2000].

2.4.2 Interfaces

Interfaces are thus in focus when designing modules, and interfaces must be
designed to be robust, and are preferably designed early [Blackenfelt 2000].
An interface is "a surface forming a common boundary of two bodies, spaces,
or phases" [Merriam-Webster 1997], "a pair of mating faces between two
elements" [Sellgren 1998], or "the place at which independent and often unre-
lated systems meet and act on or communicate with each other". Since the
interfaces represent the communication between components, they are vital to
define to be able to determine the behavioral and structural compatibility of
two components, as well as the behavior and assembly of the system.

Fit and communicate. In general, an interface describes physical and func-
tional aspects of a connection: physical - the geometry, functional - the type of
communication/transfer. The transfer may be classified as an energy transfer
realized in terms of material, signals and energy. Based on these four types of
interactions: Spatial (geometrical orientation), Energy (transfer), Information
(signals) and Material (exchange) and the strength of these interactions, the
interface between two elements can be described [Pimmler&Eppinger 1994].

In electronics, the geometric layout would correspond to the number and posi-
tioning of pins in a bus connection, and the type of transfer would indicate
what voltage the different pins are allowed to transfer. In software, there is no
physical layout, but layered protocols for communicating data. In mechanical
design, the layout might include geometry as well as the degrees of freedom in
movement between the faces in a dynamic connection. A model of interfaces is
suggested by Sellgren [Sellgren 1999] for the purpose of simulating the behav-
ior of a modular system. This representation is based on three basic types of
interfaces, or mating relationships, between physical design objects: rigid at-
tachment, constraint and contact [Roy and Liu 1988]. A fourth type, field
transfer, is added to treat system and environment relationships in a consistent
way. The field transfer could be a transfer of mechanical energy as well as of
electromagnetic fields or signals [Sellgren 1999].

Apart from these physical interfaces, the functional description of a component
might be seen as a functional interface. The functional interface describes all
that the interfacing components or system need to know about the component
in order to determine how it will behave and can be used, irrespective of its
internals. The functional interface could be anything from a total specification of the whole component to a black box description of an object as providing a certain function independent of context. The important thing being that if a component follows a standard, a commonly defined functional interface, a designer could use this component as part of a system without knowing the details of its internals; the simpler, more robust and common the interface description, the easier to just "plug in" the component.

2.4.3 Reusing patterns and rules

A modular architecture is one means for efficient variety, but integral architectures may also be varied and reused. The goal when designing product families is to reach a high degree of commonality, and to reuse common things over time, either it is physical modules or conceptual solutions such as parametric models and other design ‘patterns’ and rules. The interesting task is to identify how to represent a mixture of modules, flexible structures and conceptual solutions.

A cut-, or manufacture-, to fit module is a kind of hybrid between a module and a parametric component: as long as a component adheres to the defined interface and functionality, it is a module even when there are undefined dimensions. A conceptual module may be a parametric component such as a pipe with varying length, or a shape which varies but keeps the same interfacial boundary to connected components.

From a design point of view, it is desirable to be able to reuse conceptual solutions, such as a shape that meets a certain function of opening a can, even when there is no existing physical realization. As an example, large power transformers, as opposed to medium sized, contain a high degree of complexity and uncertainty concerning how differing requirements are to be realized. Especially the constraints concerning electromagnetic fields and their interaction with physical components makes it difficult to define all possible solutions beforehand. In the case of designing a pipe connection for transporting oil for example, the routing of the pipes depends on the effect of the transformer. The desired power determines the size of the transformer core which determines the strength and spread of the electromagnetic fields outside of which some parts of the pipe connection has to be located. Still, the main components and topology is always the same. Only the lengths and the angles of the pipes vary, constrained by rules concerning how much they have to incline, how long they should be to be possible to paint etc. In this case, it would be appropriate to reuse the basic solution and the constraints and design rules, while leaving the exact implementation open [Sivard 1994].
2.4.4 Reasons for an integral architecture

**Technology shift.** A prerequisite for reaping the benefits of a modular architecture is that the structure is determined; that it has been decided how the functionality of the product is realized within different (physical) modules. Modules can be changed as long as they provide the intended function and adhere to the defined interfaces. But if the product isn’t mature, the structure may change due to increased understanding of the problem or how to optimize the design, and due to new technologies that require a different subdivision of functionality between modules [Whitney 1997].

In an environment where the technologies change fast, part of the task is to identify the changes and try to identify how these changes will affect the architecture of the product. Either it is possible to isolate the change to a part of the structure or it will require structural changes. In a well designed, decoupled product, it is easier to adapt and incorporate these changes, but as soon as a different solution is chosen on a high functional level, the underlying structure may need to be changed [Nordlund 1996].

**Modifications:** When the prerequisites change, the contents change and the decomposition into modules may change. Object orientation is good if the objects remain the same, but a configuration system based on objects suffer from the same problems as with physical modules when the decomposition is changed.

**Generality costs.** A modular structure introduce extra information which for each individual product is not necessary. Since it is designed to fit with several products it is not optimized for one product, but either comprise extra capacity not used by all products, or comprise a trade off which makes each module sub optimal.

**Interfaces cost.** A component or a feature which is introduced with the only reason to serve as an interface to other components is functionally not necessary for the single product. It will require extra resources, either physically in terms of weight and space, or functionally in terms of energy or time. [Boothroyd&Dewhurst 1987][Erixon 1998]. Especially in solutions in which some property needs to be optimized, such as the weight in fighter airplanes, a physically integrated solution may be the only possibility.

2.5 Designing product families

In the following, the process of designing a product family is described with a focus on the design of the product family architecture, as opposed to the de-
development of new technologies. The steps in the process description is based on established design theory concerning the development of one product [Pahl&Beitz 1988], [Pugh 1990] [Andreasen&Hein 1987], as well as on the cases and methods for product platforms described previously in this chapter, in particular on the MFD method [Erixon 1998] (presented in more detail in Appendix 10.1). Phases described are: Clarification of task, Concept design, Embodiment design, Detail design and Production Preparation. While these phases have slightly different names and delimitation in the different theories, and although the activities within the phases may be intermixed, they provide a clear serial presentation of the design activities.

2.5.1 Clarification of task (specification phase)

Identification of markets and customer needs, competition, and company resources. **Platform/family specific activities:** Manage variance - the customer end.

I. Identify the desired variety in various market niches

II. Structure customer requirements into alternative clusters, which partly may have the same basic requirements. Identify requirements which vary between clusters.

III. Relate customer requirements to a product’s functional properties

In this phase, it is crucial to identify properties which are important to vary from a marketing standpoint - the ”Customer Attributes” of the product that contribute to the customer satisfaction and are signs of its uniqueness [Ulrich 1995]. Apart from relating customer requirements to product properties and deciding whether properties should be made common or optional, alternative property values are identified.

2.5.2 Concept design - turning varying desired properties into technical solutions:

**Platform/family specific activities:**

I. Relate varying functional properties to physical properties and components

II. Strive for functionally independent solutions as a first step towards designing independent modules.

III. Identify location and desired characteristics of interfaces
IV. Decide what core platform components (logical and physical) and interfaces to reuse and design based on these.

V. Analyze degree of dependency between customer requirements and various solutions

VI. Analyze degree of functional modularity

In this phase, technical solutions are designed based on the varying product properties. This is what commonly is seen as the core of the design effort; the main difference between the design of one and a family of products is the focus on realizing the identified variance as cost efficiently as possible. Components or properties that are not perceived by any of the senses of the customer can be made common to all variants, manufacturing constraints concerning set-up, change-over time and other costs of variety are of specific concern [Martin&Ishii 1997]. Planned reuse of principle solutions as well as physical components and interfaces from the platform may be required.

Result from conceptual design of product family architecture: main architecture, conceptual modules and interfaces, system boundaries, design trace.

2.5.3 Embodiment design

**General activities**: Physical realization of principle solutions, development of layout in mechanical design etc.

**Platform/family specific activities**: Modularization of product family - specifying what parts to separate from the common structure and integrate into modules considering technical and strategic reasons. Avoid physical integration of solutions which should be functionally separated. Make functional tradeoff between optimizing a module for an individual product, and making it a common component, good enough for all products.

Specific to a modularized decomposition is the goal to design structures, modules and interfaces such that they can be combined in different ways and thus create the diversity of products. Modules can be used as common components - one module for many products, or variant components - different possible modules for one product.

Physical realization comprises the integration of principle solutions into components, as well as the aggregation of components into composite components, or modules. Aggregating a set of technical solutions into modules in principle means determining how to integrate a set of physical properties into a physi-
cally integrated component. Often components are already selected, more or less consciously, and this step comprise the aggregation of components into composite components, or modules. Later, methods for design for manufacture (DFMA) [Boothroyd&Dewhurst 1987] may be used to integrate sub components and get rid of unnecessary diversification. If modularization instead is seen as a two step activity of 1) relating functional properties to physical properties and components and further 2) integrating physical properties into components, and aggregating components into modules, these manufacturing constraints could be considered already at the conceptual stage.

2.5.4 Detail Design (Evaluation and modification)

**General activities:** Design components in detail, complete drawings and manufacturing documentation.

**Platform/family specific activities:** Detail all components, determining specific interfaces and property values. Design modules and interfaces for assembly, limiting number of interfaces and preparing modules to be assembled in a sequence or on a base module [Erixon 1998]. Design interfaces to be robust to variance i.e. so that modules easily can be interchanged and also upgraded [Blackenfelt 2000].

**Analysis of the family architecture.** Fast analysis is one of the keys to efficient product development [Suh 1990]. In this context, analysis of the product family entails determining the relations between customer requirements and the resulting product variation [MMAB 2000] as well as calculating the achieved commonality and possible product variation [Collier 1981].

By documenting the relations between modules and product families, it is possible to keep track of what product variants a module is part, and how a modification might change the interrelations: if the interfaces remain the same or if the surrounding architecture needs to be modified. When creating a new version of a product family for example, large parts of the old structure and rules might be kept, while adding, modifying or removing features or components. This requires keeping track of the effects of a change, tracing which product variants and corresponding markets will be affected and making sure that all adequate changes to dependent systems are performed.

**Detail design of Interfaces.** An interface is a mating relationship between two, or several, ports. The interface may be a known precondition, but other-

---

1 Often the term “interface” is used to denote only one face, the “interface” of a component.
wise, a component property does not become a *port*, a part of an interface until it is decided that the component should interact with the surrounding world through this property.

### 2.5.5 Production Preparation

Design manufacturing and sales-delivery processes for realizing the desired variety at low cost. Processes should enable a fast processing of each order, specifying and realizing a satisfactory product individual in time.

Design of order based design system for selecting a variant and specifying the manufacturing process based on a product family model, enable integration of information concerning bought components and component families.

Integrate out sourced modules with the internal product structure.

### 2.6 Conclusion

Product platform planning and product family design are of strategic interest to industry in order to achieve efficient ‘mass customization’. Following are some of the identified keys for efficient product family based design, keys of varying importance depending on the character of the product and product development:

**Core components, with standard interfaces, around which different product architectures are designed.**

The importance is to define the interfaces to enable continuos development of these core components, and to enable their use within different products as long as the interfaces are kept intact.

**Designing product family architectures for variety.**

Efficient customization requires a product family which is designed for the purpose of generating several variants, thereby utilizing the inherent commonality between the product variants. By employing a modular architecture, the advantages of combinatorics is gained, while planned reuse of conceptual solutions is a means for efficient variety in integrated systems. Moreover, if a configurable product family is defined broadly, that is, if it comprises a large possible variety in terms of its components’ values and possible combinations, the same product family definition can be used for configuring products that were not planned for from the beginning as in the Sony case. With a system that allows incremental addition of component alternatives or property values, these types of changes are facilitated.
Designing an flexible and efficient order process.

Whether the product architecture is modular and based on core components, or not, an efficient order process is key to achieve efficient customization. Order process issues include:

- Flexible manufacturing and sales-delivery process, to achieve a process in which variety is easy to realize.
- Efficient configuration of a variant - to support the specification of a particular variant, from customer need to manufacturing specification.
- Flexible architecture, processes and information management - to adapt to fast changes in markets and technologies.

Impact on research.

The impact on this research is mainly the identification of the following issues:

- It is crucial to support the design of a product for variety, which requires a holistic reasoning concerning the relationships between various markets and the product’s properties and components.
- It is crucial to support design and reuse of the core components, and interfaces, around which several product families are developed.
- To realize information efficiency in creating variants, information created during product family design should be reused during order processing, and the other way around.
This chapter describes some various systems and methods for supporting product design and the configuration of a variant. The goal is to provide an understanding of the purpose of these systems and what support capability they encompass, and more important, what they are lacking.

3.1 System overview

While the design phase involves creating and documenting solutions, the order phase is where this documentation is used to turn one customer’s needs into a specific product.

From an information management viewpoint, the order phase involves three basic steps: sales configuration - identifying the customer’s wishes and specifying a product based on these, product configuration - generating a manufacturing documentation based on the product specification, and resource and process planning - planning the total materials and resource flow in the manufacturing process based on batches of several orders.
Commercial systems supporting product development may mix different functionality, but often focus on one of the aspects in the following quick characterization:


II. **Product Data Management (PDM)**. Focuses on the management of the documentation of products - the process of creating and modifying documents. The basic product model that is used is the part decomposition, the "part_of", structure (Matrix: www.technia.se, Windchill: www.ptc.com, Smarteam: www.solide.se).

III. **Enterprise Resource Planning systems (ERP)**. Initial focus on supporting the manufacturing processes, extended to support all processes within an enterprise (SAP: www.sap.com, BAAN: www.baan.com). Manages information about materials, time, cost, etc.

IV. **Configuration systems**. Designed for flexible configuration of components according to specification and optimization criteria (Tacton: http://www.tacton.com, Xpertrule: http://www.attar.co.uk/, Configura: http://www.configura.se/). Manages design rules and variants.

### 3.2 PDM and ERP systems

A short but clear definition of PDM systems is that: "A PDM system links product data to the enterprise workflow by providing a toolbox for the modeling of product structures and engineering processes" [Pikosz 1997]. An ERP system, on the other hand, controls and manages the manufacturing facility, including not only production but also purchasing, finance, and engineering.
3.2.1 Purpose and background

The main purpose of a Product Data Management (PDM) system is thus to administrate the management of product data throughout the enterprise, ensuring that the right information is available for the right person at the right time and in the right form. It must support a product/process definition through the whole product life cycle, including design, review, rework, approval, release and revision. A PDM system should typically support the following product data management activities [CIMdata 1997] and [OMG 1999]:

I. Store and retrieve product "documents" (design geometry, engineering drawings, project plans, part files, assembly diagrams, product specifications, numerical control machine tool programs, analysis results, correspondence, bills of material, and many others)

II. Track assembly/component relationships, alternate and substitute components, provide bills of materials and perform "where used" and "where useable" searches.

III. Manage multiple part versions and their effectivities

IV. Provide overviews of the relationships among products, product specifications, process specifications, maintenance specifications, and the engineering and management tasks that create, use and modify them.

V. Provide workflow and process management, communication and notification - during design and engineering changes to control update and release

VI. Data transport and translation

Initially, the computer based creation and manipulation of geometrical data was performed in CAD systems, and the structure of products was represented in production administrative systems, ERP- (Enterprise Resource Planning) systems, evolving from earlier Material Requirements Planning (MRP) systems for inventory control and later Manufacturing Resource Planning (MRP II) technology for shop floor scheduling and coordination. These ERP systems were for a long time the only available support also for designers and others creating and using the product structure. By coordinating the manufacturing operation for peak efficiency, the ERP systems of today help reduce manufacturing time and cost as well as facilitate teamwork and collaboration, and have become an important tool for management.

When manufacturing a product, raw material is machined to produce components which are assembled to create different sub-assemblies, and finally the
complete product. Each step on the path from raw material to finished product is called a refining level, and this path of refining levels is called a manufacturing structure. The documentation of the product was earlier generally tied to these refining levels, by one parts list (including the raw material needed, its dimensions etc.) and possibly one detail or assembly drawing for each refining level. This hierarchical structure of documents, describing the manufacturing of the product was called the product structure.

### 3.2.2 System parts

The technical product data used during product development is primarily used by designers and production engineers. First, the data manipulated concerns the geometry - the shape and relative positions of the parts making up the product. Second, data concerns the structure - attributes of and the relations between parts [Lindberg 1991]. These types are strongly interrelated, since every physical part has both shape and attributes. A reoccurring issue in many applications is deciding which of the models to change and how to keep the consistency between the two models.

A PDM system has two main parts; the User Services and the Utility Services [CIMdata 1998]. **User Services** include: Data / Document Management, Process/work flow Management, Product Structure Management (Configuration), Program Management, Classification. **Utility Services** include: Communications, Data transport and translation, Image Services, Administration.

The traditional way of documenting a product is by parts list and drawings. A parts list, as the name indicates, is a description of the structure of a part. Every part which is contained in the list, is typically described by a part number, a name, a description (type, material, dimension etc.), and a quantity. The item number is an unique identification of each branch in the substructure of a part. The Bill Of Materials (BOM) is a list of the total set of parts needed to manufacture the product.
The process plan, often residing in the ERP system, consists of material and operation specifications for every parts list. The material part of the specification includes the raw material needed and its dimensions etc. The operations part of the specification holds data about each operation needed to manufacture the part, including workshop address (i.e. the machine to be used), the setup time and the cycle time. The parts lists are the major inputs here, while the drawings, at least on assembly levels of the product structure, normally are used only for reference.

The development of PDM and ERP systems is progressing towards managing the full product definition life cycle in an extended enterprise, including "collaborative product commerce" - Internet communication, and "enterprise applications integration" - integration of PDM and ERP systems [CIMdata 2000].

3.2.3 Product family management - needs

From a product documentation viewpoint, one problem when creating variants of a product with many parts is the proliferation of similar parts lists. For each product variant, maybe just differing in one component, whole new parts lists on all levels traditionally need to be generated.

Assuming that the structure is identical to all variants, and that for every variant it is only necessary to save the data needed to unambiguously recreate the manufacturing data, one solution is to create a documentation based on a generic variant of the product. This generic parts list would be similar to an ordinary list, but for each position where an unique variant is possible, there is an "opening" or statement that the actual part number will be specified in a separate document, the Order document. This generic, or open structure is thus not completely specified. With this method the number of documents needed to
specify all variants is in the magnitude of the sum (as opposed to the product) of variants for each component. Assume a product with two parts, part A with \( N_a \) variants and part B with \( N_b \) variants. The number of possible product variants equals \( N_a \times N_b \). If every variant is documented, this will result in \( N_a \times N_b + N_a + N_b \) documents, while if a generic structure is used, the number of documents is \( 1 + N_a + N_b \). The open generic structure differs from the closed generic structure in that the closed structure is limited, while the open structure includes parameters which are not enumerated, but in principle may take any values. STEP AP214, for instance, does not cover parametric values, but specifies a closed generic structure. The exemplifies an open structure of a transformer with the openings indicated with an *.

![Diagram](http://example.com/diagram.png)

**Figure 25 Example of open structure, open items marked with an * [Sivard 1994]**

The documentation needed is thus: a generic structure, the part numbers of the parts used to close each opening, and rules describing how to fill the openings (i.e. what parts to use).

This open documentation requires that the complete set of parts list is not needed by the ERP system, which however may be the case. This problem can be solved by creating in-house ERP systems as in ABB or Scania or by creating a generic bill of materials and processes (GBOMO) [Jiao 1999]. The GBOMO contains a mixture of product and manufacturing process data. The idea is to manage variety in products and manufacturing processes by defining which components to manufacture, and how, in the same framework - a bill of materials and processes which is generic to a whole product family. AP214 also specifies a structure with relations between parts and process documentation.

**Parametric models**
Many parts, such as a pump housing, come in part families, where the topology for all members are the same but the actual dimensions vary. To efficiently handle these families, different parametric functions are available which allow an instance of a part to be created based on a set of parameters. Many CAD systems provide associativity, i.e. the ability to create parametrized 3D models as a means for creating many variants of the same basic 3D model. These systems however often have problems describing minor variations of topology such as the addition of holes or cut-outs.

A parametric drawing is an open drawing, i.e. a not fully specified drawing which is closed through the evaluation of a set of parametric programs with a set of parameters. One question when implementing parametric drawings is where this evaluation should be made - in the geometry or the structure system. A possible solution is to tag the parameters with an additional attribute, specifying where it is to be evaluated, an approach used early on by e.g. ABB Transformers [Agerman 1991].

Managing changing information of complex markets

A product family may cover subsets of product variants which have more in common with each other than with other variants. A car model, for example, comes in a sedan and a hatch-back variant, with different equipment in different countries, varying slightly between year models etc. An automobile family model is thus typically segmented into several ‘subfamilies’, or ‘product_classes’ [AP214 1998]. Within one ‘subfamily’, one particular set of components may be common to all variants. A component may also vary, or be common most of the time but vary occasionally. In \( \mathcal{S} \), the sets refer to sets of components.
Figure 26 Sets of product family components, small circle denotes common components, large circle all components.

A classification of the role of a component would aid the configuration by enabling an automatic selection of components which are common to a subfamily when that subfamily is chosen, or the selection of default components if an option is not specified. The role may change during the life cycle of the component: a CD system, starting as a "DeLuxe" option may well end as a standard feature. A component may also have different roles in different product families, e.g. a CD might be standard in the "Sports Package", while optional in the "Family model".

**Maintaining systems**

Previous attempts to automate the product configuration task often failed, because of difficulties in maintaining the systems when the configuration knowledge was implemented in procedural programs [SICS 1996]. Product configuration systems need large amounts of knowledge, about the mapping from requirements to components, and about technical restrictions. This knowledge is subject to continuous change, and hard to validate. As an example of this problem, there is DEC's rule-based configurator, XCON/R1 [Winston 1984] that configures VAX-11/780 computer systems. Despite XCON having a reputation
for being well used and cost efficient, there were main flaws to the system. The domain knowledge - describing the components, and the control knowledge - the routines, were not separated. This makes the system very difficult to maintain and enhance, since the general and overall effect of a small change in one particular rule is sometimes difficult to capture. In 1989 XCON had over 31,000 components and approximately 17,500 rules incorporated [Stefik 1995]; the fact that knowledge-base changes were at the rate of 40% per year stresses the importance of the issue [Pechoucek 1996].

3.3 Configuration systems

Order based design means generating a product based on a customer’s desires, and the activity of specifying the product variant involves knowledge of how to elicit requirements from the customer and how to map these requirements to a specification of a variant under certain constraints and optimization criteria.

Configuration is a way to generate the decomposition structure and features of different products, based on sets of requirements and component alternatives. With different sets of requirements or building blocks, different products will be achieved. Configuration is used more and more commercially, for creating a product specification based on customer choices as in sales configuration, or creating manufacturing documentation based on these specifications as in product configuration. Moreover, configuration can be used in the planning of a product family architecture, simulating and comparing the different product variants that would be achieved based on different choices of decompositions and component alternatives [Optiwise 2000].

3.3.1 Variation in Product Families

A product family is a (possibly infinite) set of products variants with many characteristics in common. Seen as sets of variants, a product family model needs to describe the union of all the components (\( \forall \)), as well as rules for distinguishing each individual. The common set refers to everything that is common, i.e. not only to components, but also to the structure and specification in general.
In analog with the commonality goal of product family planning, this intersection in the model should be large while providing the ability to express the desired variation (seen as a “non” robust design problem [Phadke 1989], the task is to identify the product parameters of which a small variation will create a large sense of product variation).

3.3.2 Representing variety

Rather than flexibility and creativity, the product family models of the order phase are characterized by accuracy and efficiency. Systems for order based design need to accurately represent the variety of components as well as the rules for achieving this variety.

Strictly, a component is defined by its properties, which in turn may be real
ized by sub components. Variety may be achieved by changing properties directly, or through the variation of sub components or their interconnections. A component family thus incorporates information concerning the alternative sub components as well as rules for how these components may be connected. As an example, assume that our product is a simple block with varying height. If it is not a composite component, the only way to vary the height is by varying its own height-property as in example ‘A’ in Figure 29. If it is composed by a set of sub components, the height can be varied by varying the properties of the sub components, ‘B’, or by changing the way in which the components are connected - their interrelations, ‘C’. A fourth principal way, is by varying the number of sub components as in example ‘D’.

One difference between product families is their principle(s) for realizing the variety, and a difference between systems which configure variants is how these varying components and rules are represented. They may be represented in terms of:

I. a generic structure with a direct link between the product characteristic and the realizing components - as in rule based systems

II. components and their functional and physical properties, models of these properties, and generic modeling rules - as in model based configuration

III. parametric geometrical models, adapting a pre-defined geometric model according to chosen parameter values - as in parametric solid modeling CAD systems.

3.3.2.1 Generic structure

Since the physical component structure of all variants of a product family to a large extent is the same, one popular approach is to define one common generic structure [Prodevent 1975][Van Veen 1992]. Assume that we have a
product with three components which each come in 2 alternatives, then the
generic (assembly) structure is one structure with all alternative components at
the nodes: \{Hardtop, Fabric, Radio1, Radio2, Antenna1, Antenna2\} \(\gamma\).

If not constrained otherwise, any combination of these components could im-
plcitly be created, resulting in the union of all possible configurations. The
number of combinations is even larger than the number of alternatives \(\times\) num-
ber of component positions (in the order of the faculty of the number of com-
ponent alternatives), since all classes of components do not have to be part of
all product structures. A component class could be either a required or an op-
tional part of the family. The motor of a car, for example, is probably a re-
quired component while the radio and antenna might be optional.

### 3.3.2.2 Constrained union

**Functional and other constraints.** All combinations of the node components
are neither desired nor functionally feasible, feasibility being determined by
the functional structure and constraints concerning incompatible interfaces,
technical constraints, regulations, styling etc.
The functional structure, simply visualized in Figure 31 describes how to combine components to achieve functionality. In this case it defines that a radio system requires a radio and a antenna, thus ruling out combinations such as Hardtop and "Radio without antenna". Still, even if the solutions to required functions must be contained in the structure, there may be reasons for including a component even though its function is not required. An antenna may for example be included even though the car is bought without a radio system. This may be because the customer wants to buy the radio separately, or since it is less costly to equip all cars with antennas even when it is not a required functionality. Apart from the functional structure delimiting the set, there are constraints concerning e.g. incompatible interfaces, technical constraints, regulations or styling which further delimit the set of combinations. The feasible, constrained set is thus the generic structure with alternative combinations constrained by different limitations. Out of this set of combinations, though feasible, there may be only a part that is normally requested by the market.
**3.3.3 Rule based configuration**

The basis for a rule based configuration based on a generic structure, is to control the choice of alternative components of one structure with procedures or rules. This is a compact way of representing many similar structures, representing what is common once and then at each order only the differences for each variant. No functionality or interface description is required for configuration purposes, the knowledge of mapping from requirements to solutions is compactly represented in the rules (٪).

![Diagram of rule based configuration](image)

*Figure 33 Generic product structure with alternatives and configuration rules*
Problems with generic structures. An open structure defines the solution space in advance, that is, although the contents of the nodes may differ, the possible solutions are limited to those which follow the structure. If a whole new branch, which affects other branches, needs to be added, the whole structure may have to be modified. Thus, the specification of the characteristics of the product is closely linked to the physical components of a specific product family realization, making it difficult to change the way a function is realized. This also means that there is a close connection between the way customer requirements can be expressed and the physical components of the solution. There is no intermediate representation of the functional properties of the product.

3.3.4 Model based configuration - one type of knowledge based design

Selecting and arranging combinations of parts which satisfy given specifications is the core of a configuration task. An open, knowledge based design system, described in more depth in chapter 4.3, could in principle configure a product based on the specification, modules with well defined functionality and interface, and rules mapping from requirements on functionality to solutions.

Tools that use a problem-solving method that is specialized for product configuration tasks, with the application specific knowledge represented in a vocabulary suited for this task, are often called configurators. In this thesis, they are called model based configurators to separate them from the rule based systems. The specification of configuration tasks, in general, involves two distinct phases, the description of the domain knowledge and the specification of the desired product [Sabin 1999]. The domain knowledge describes the objects of the application and the relations among them. The specifications for an actual product describes the requirements that must be satisfied by the product and, possibly, optimizing criteria that should be used to guide the search for a solution. The solution has to produce the list of selected components and, as important, the structure and topology of the product.

A configuration task is well-structured and completely specified: the description of all the components is complete, and all the relations and constraints among different components are stated explicitly. Most of the complexity of solving a configuration problem lies in representing the domain knowledge. Modeling the domain knowledge is thus critical, and any configuration framework must address the issues of expressiveness and representational power, efficient knowledge application in a highly combinatorial context, and mechanisms for coping with the high rate at which knowledge changes.
Domain knowledge is typically represented as object classes available in the application domain and the relations among object instances: It is a common practice to organize application objects in an abstraction or generalization hierarchy that contains both abstract and physical entities. The physical entities are concrete parts-components available in the application domain. By generalizing the functions of a class of physical parts, abstract parts are developed. Each component is uniquely identified by its set of properties, which characterize its function and performance. Properties fall into three semantic categories:

- **attributes**
  specify descriptive features such as functional and technical characteristics. Each value can take values from within a pre defined range, discrete or continuous.

- **resources**
  specify characteristics that components in the system can either supply or use e.g. electrical power. Resources should be either balanced (flow in = flow out) or satisfied (enough fuel).

- **ports**
  are places through which an object connects and communicates with other objects in its environment.

In addition to the descriptive view, each object type has an inherent internal structure, described by a set of constituent objects and the interconnections among them.

Relation types that any configurer should be able to handle:

1. classification (is-a)
2. aggregation (part-of)
3. user defined relations among sets of components
4. local constraints (structure, arithmetic etc.)
5. global constraints (resource constraints, optimization criteria etc.)

In constraint based reasoning systems, described in Chapter 4.3.2, differentiation is made between local constraints which state a relation between a few, connected variables (structure, arithmetic etc.), and global constraints which state relations involving many variables spread over the constraint network (resource constraints, optimization criteria etc.).
3.3.5 Comparing rule based and model based configuration

Assume a simple example - a family of loudspeakers with two characteristics: its cost and its power. There are two basic kinds of speakers: simple speakers with only one element, and separate base speakers with one element for the medium and high frequencies, and a separate base horn, \( \text{horn} \). The membrane as well as the horn comes in two versions.

Assume that a customer can specify either cost or power, then one rule based way of describing the family structure is through a generic structure + selection rules and configuration constraints [Erens 1996]. Configuration constraints in this case are the rules relating power to cost - delimiting the customer choices from start through the dependency between these two characteristics.

Figure 34 Generic part structure of speaker family
In a model based configurator, the product family is modeled as components with properties and ports. Properties of the product family are modeled with 'open' values (e.g. a set, interval) and relations (rules, inequalities, equations) and in this way calculates a set of property values that correspond with the desired values. Relations are represented as rules concerning e.g. connections, or constraints which delimit property values. Together with a mechanism for propagating rules and selecting values, a configuration is generated.

Table 3 Class as selection rules and constraints

<table>
<thead>
<tr>
<th>Family</th>
<th>Variants</th>
<th>Selection Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loudspeaker</td>
<td>Low end speaker</td>
<td>power=25 or power=40</td>
</tr>
<tr>
<td>Constraints:</td>
<td>High end speaker</td>
<td>power=110 or</td>
</tr>
<tr>
<td>power=25-&gt;cost=10</td>
<td></td>
<td>power=520</td>
</tr>
<tr>
<td>power=40 -&gt; cost=20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low end speaker</td>
<td>Membrane A</td>
<td>power = 25</td>
</tr>
<tr>
<td>---- Membrane system</td>
<td>Membrane B</td>
<td>power = 40</td>
</tr>
<tr>
<td>High end speaker</td>
<td>HornA</td>
<td>power = 110</td>
</tr>
<tr>
<td>---- Base system</td>
<td>Horn B</td>
<td>power = 520</td>
</tr>
<tr>
<td>---- Membrane system</td>
<td>Membrane A</td>
<td>Base=HornA</td>
</tr>
<tr>
<td></td>
<td>Membrane B</td>
<td>Base = Horn B</td>
</tr>
</tbody>
</table>
While a generic structure configurator hides the rationale for selecting a certain component within the selection conditions, a model based configurator separates the model of the product family from the rules for selection. Thus, these two paradigms of "rule based" and "model based" configuration require different types of information and provide different flexibility. However, there may in both approaches exist a common part of the structure, and a part which varies controlled by rules.
The advantage with a rule based system is that it is easy to build if the rules are known, while property models may be difficult to elicit and model. The advantage with a model based approach is that it specifies the properties of a product without making all alternative solutions explicit. Models can be general and adapted in a way that was not anticipated as long as they are within the defined scope. This makes the model based approach especially attractive for sales configuration, while the rule based approach often is adequate for product configuration. Model based systems are flexible and powerful, but a large amount of computer represented design knowledge is required for this type of configuration, knowledge which especially in mechanical design might be complex to elicit as well as difficult to express formally.

Table 5 Characteristics of different types of design systems

<table>
<thead>
<tr>
<th>Component descriptions</th>
<th>Design Knowledge</th>
<th>Order processing system</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes of complete alternatives</td>
<td>Generic structure of classes + configuration rules</td>
<td>Rule based system</td>
<td>Efficient, simple, expressive</td>
</tr>
<tr>
<td>Modules with documented functionality and interface</td>
<td>Models of interrelations between properties</td>
<td>Model based system</td>
<td>Flexible, requires deep definition of components and rules</td>
</tr>
<tr>
<td>Parametric geometric components</td>
<td>mathematical calculations</td>
<td>Parametric modeling system</td>
<td>Focused on complex geometries</td>
</tr>
<tr>
<td>Incomplete components</td>
<td>Knowledge of how to complete components</td>
<td>Human</td>
<td>Very flexible, slow</td>
</tr>
</tbody>
</table>

3.4 Conclusion

Existing PDM systems provide extensive support to product development, ERP of the flow of resources during sales to delivery, and configuration systems support the configuration of variants. Recently, there are also an increased number of PDM and ERP systems with integrated configuration support. What still lacks is an integrated way of providing information to support the design and management of product family models, including representing requirements, supporting modularization, segmentation of the product family,
and managing dependencies between different versions of component alternatives.

It is concluded that there are two different kinds of configurators - rule based and model based. Rule based configurators describe the product in terms of one generic structure with alternative solutions and rules for selecting one variant, and is especially suitable for product configuration. A model based configurator defines the product in terms of property models, connections and generic problem solving, can handle most types of configuration, but is mainly used for sales configuration.
Common to many approaches to product modeling is that of trying to identify the definition, or “meta model” of products, defining what a product is, from different points of views, for different applications. Purposes of such meta-models are e.g. to facilitate the exchange of product data between different users as in STEP, to model the design history to facilitate verification and reuse, or supporting product configuration or platform design.

Thus, a model and systems architecture is often suggested, which is more or less founded on some explicit principle or theory for describing products. In this chapter, two of the main theories for describing the design process and artifact are presented, as well as product modeling approaches which to different extents are based on a product modeling theory.

4.1 Theories for describing products and design processes

4.1.1 The Theory of Technical Systems

In design science, the design process is often described as successive mappings between functions and the means for their realization [Andreasen 1992] [Hubka 1988] or as mappings between functional and physical domains [Suh 1990]. In the Theory of Technical Systems (TTS), a system is described in terms of its effects, functions, organs (technical solutions) and components [Hubka 1988]. An effect is the purpose of a system which is achieved by certain function structures. The TTS function is a property of the technical system, namely “the ability to convert an input measure into a required output measure under precisely given conditions”. A function is realized by an organ, or function carrier, which is a means for realizing the function, such as the concept ”room” of a house, or a ”bearing”. Thus an organ structure is an abstract model of a technical system [Hubka 1988], while the component structure is a description of a technical system which refers to manufacturable constructional elements. In addition to these descriptions, different views on the information concerning the product’s life cycle may be considered [Andreasen 1998][AP214 1998].
4.1.2 The Axiomatic Design framework

As opposed to the TTS main goal of systematically describing systems, the purpose of Axiomatic Design (AD) is to prescribe fundamental principles in the design of components (or systems) and processes. AD is general, applicable to all types of technology, and pleads "free and lean thinking" - the importance of identifying the needs independently of solution and then realizing these as simply as possible. These principles are defined in terms of two axioms (1) and a set of corollaries and theorems derived from these (listed in Appendix 10.1). Functional Requirements (FRs) are defined to be "the minimum set of independent requirements that completely characterize the design objective for a specific need" [Suh 1990 pp. 37-39]. By definition FR:s are independent, but it is certainly possible to define a set of dependent requirements, believing that they are not.

Table 6 The Axioms of Design [Suh 1990]

<table>
<thead>
<tr>
<th>Axiom 1, The Independence axiom</th>
<th>Maintain the independence of Functional Requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiom 2, The Information Axiom</td>
<td>Minimize the information content in the design</td>
</tr>
</tbody>
</table>

AD further provides a fundamental basis for describing decision making during design. It contains representations of the design object (a hierarchy of functional requirements, constraints, design parameters, and design matrices) and the design process (decomposition and zigzag mappings) combined with rules for decision making (the independence and information axioms). ∀ illustrates the zigzagging, with constraints delimiting the available "solution space" of the domains. Customer needs are not decomposed or treated in detail in AD, apart from pointing out that the establishment of independent FRs based on the customer needs is the most important step of design.

The FRs constitute a formalization of a recognized societal need, and thus cover all needs that a product satisfies, they are not limited to describe the requirements on the technical functions.
An FR is realized by a design parameter, DP, which may be a principle or physical property of a solution. A DP can be embodied in a component, but a DP can also be embodied in a geometrical feature of a component, a material property, a feature involving two or more components (an "organ"), a location of a feature, the number of features, etc. Likewise for software design, a DP can be a collection of data, an algorithm, a pointer, a function call etc. At higher levels, DPs may be considered as assemblies of components; at lower levels, they may be component properties, and the DPs at higher levels are not necessarily an aggregation of the DPs below them [Tate 1999].

In Axiomatic design, one component is thus designed by zigzagging between the functional, physical and process domains, hierarchically mapping requirements to solutions to new requirements to solutions to realizing processes etc., under the conditions of delimiting constraints. The goal is to define one solution in which the requirements can be realized independently (an uncoupled or de-coupled solution) and which is of minimal complexity. A functional coupling means that one design parameter partakes in realizing several functional requirements. If there is another DP that also partakes in realizing the same FRs in a way such that no order in which the FRs can be realized without having to "tune" the DPs trying to establish an equilibrium, the system is coupled. If an order can be established, the system is decoupled, and if all FRs can be realized independently of each other, the system is uncoupled.
The dependencies between FRs and DPs are represented in terms of a design matrix, DM, relating the vector of FRs: \{FR1, FR2, …\} with the DP vector \{DP1, DP2,…\}.

\[
\{FR\} = [DM]\{DP\}, \\
\text{where } [DM] \text{ is of the form:}
\]

\[
\begin{bmatrix}
A_{11} & A_{12} & \cdots & A_{1n} \\
A_{21} & A_{22} & \cdots & A_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
A_{m1} & A_{m2} & \cdots & A_{mn}
\end{bmatrix}
\]

In an uncoupled design, all off-diagonal elements are zero (as in the example of \(\gamma\)), and in a de-coupled design, the DM is triangular. By representing FR-DP relations in this way, matrix calculus can be used for determining the degree of coupling and for ‘de-coupling’ a coupled design.

\[
\begin{array}{c|c|c|c}
\text{FR} & \text{DP 1: fan} & \text{DP 2: movable chassis} & \text{DP 3: exchangeable bag} \\
\hline
\text{FR 1: create suction} & X & & \\
\text{FR 2: reach all spaces} & & X & \\
\text{FR 3: get rid of rubbish} & & & X
\end{array}
\]

*Figure 36 Design Matrix for vacuum cleaner (adapted from [Erixon et al 1996]).*
4.2 Modeling products

MODELING PRODUCTS ACCORDING TO THE CHROMOSOME MODEL

Johan Malmqvist [Malmqvist 1995]

For the purpose of capturing design history, a product model based on the functions-means tree model of the design process and the "chromosome" model for product modeling [Andreasen 1992] is suggested by Malmqvist [Malmqvist 1995]. The design characteristics of a design object in a chromosome model constitute of its process-, function-, organ- and component-structures. In this approach the functions and means in the tree are extended to represent any of these design object characteristics, and the capabilities of modeling constraints and objectives are also included. This provides a model which is capable of representing a very wide variety of design decisions, but it only models singular products, not product families.

"SYSTEMS-BASED PRODUCT MODELING"

Wayne Collier [Collier 1999]

This approach is mainly a way of organizing and relating models of products using defined interface objects. Technology models are related to manufacturing bills of material through "usage" relationships, and design and change management is facilitated by the interfaces which represent dependencies between models.

Architecture

Systems-Based Product Modeling propose an architecture which links product design content into networks using interface control objects (ICOs). These objects manage interactions and requirements, and explicitly define rules for change propagation. Two complementary views of the network, reflecting the product’s functional and manufacturing structure, product systems and manufacturing modules, are provided. Product Systems specify a set of technical solutions which in combination satisfy a given product function or set of functions. Product systems are specified by one or more engineering models, which in turn specify technical solutions independent of a given product. A product system maintains well-defined interfaces which both describe an acceptable product environment for that system, and specify the performance parameters necessary for its operation. Manufacturing Modules specify a set of physical, assembled parts which are delivered as a single entity to a subsequent assembly operation. A manufacturing module’s definition also includes the definition of a manufacturing process used to connect the module to any product
which incorporates it.

Modularity

Three types of modularity are defined:

1. Physical modularity applies to manufacturing modules, and groups as a single collection the parts added in a given assembly process step.

2. Technical modularity applies to engineering models of technical solutions which describes all the elements of a given technology required to fulfill a given function.

3. Functional modularity applies to systems which completely satisfy one or more functional objectives of a product. A functionally modular product system should also define the input/output interfaces and performance requirements under which it can provide that function in any product context.

Functional modularity builds on technical modularity, but since few products benefit from combining all three modularization types to achieve "pure" modularity (i.e. a decomposition with modules which are the same for each of the functional, technology and physical domains), physical modularity generally remains apart.

Interfaces

Interfaces govern the interaction between one system or module and any number of other systems or modules, and capture critical information describing how these interact in a total product context. Interactions among parts and models are distinguished from interactions among systems and modules using interface items and ICOs (Interface Control Objects). Interface items describe and manage one interface element of a specific type, e.g. a physical features, electrical connection, software calls, firmware communication, and may have tolerances attached to them. ICOs collect all interface items relevant to a given system or module, adding a set of performance requirements or assembly processes, respectively. Together these interfaces and their associated performance requirements describe how a system interacts in a given product, or how a module attaches to a total product. There are two principal classes of ICOs: Socket/Port ICOs for linking manufacturing modules, and Input/Output ICOs for linking product systems.
4.3 Modeling the product development process

4.3.1 Principles of knowledge based design

From the view of Artificial Intelligence research, Engineering Design has long been an interesting example of a problem solving process [Tong&Sriram 1992]. Within this field the effort has been to understand the principles of design for the purpose of making programs which perform more or less innovative design based on different levels of uncertainty. The design task is characterized along several dimensions including:

Available methods and knowledge - the extent to which it is known how to realize the design. If sufficient knowledge and methods are available for always directly (e.g. without problem-solving) generating the next point in the design space and for converging on an acceptable design with little or no search, the task is called a routine design task. If the available design task require some problem-solving or search, it is called innovative design. Finally, if a problem-solving process is required to construct the design space in the first place it is called creative design. If the design task is performed interactively, the common available knowledge determines the character of design, either it is encoded or tacit, recognized by designers but difficult to describe [Tomi- yama 1992].

Amount of unspecified structure. Design maps function onto (physical) structure (including components). A design task often provides part of the structure of the design, and it is useful to identify how much and what parts of the physical structure that is left to be determined as a measure of the design task complexity [Steinberg 1989].

Within different fields of AI, methods and principles have been developed for automatically solving problems, by efficient search routines or by using knowledge. Let's assume a problem: to plan a trip from point A to point B. We have a map and want to determine a good path, possibly optimizing length or fuel consumption. The following are the different basic AI principles:

I. Problem solving as search - search large solution spaces effectively. E.g.: quickly trying all possible roads on the map until finding a path

II. Knowledge as power - expert systems based on representing specific expertise. E.g. ask someone who has driven between the cities before and get a full path.

III. General problem solver - defining the general mechanisms for solving problems (e.g. generate -test -debug) and applying these to different
domain knowledge. E.g.: Find a basic road going in the right direction, find connections between the start and end points to the basic road. Avoid time consuming places and activities. Domain knowledge: Maps.

Apart from these principles, there are systems based on other principles and information processing mechanisms such as learning (induction), genetic algorithms or collaborative agents. A good overview of these systems are presented in [Váncza 1999].

As described above, there are many approaches to implementing a knowledge based system. One useful way to clarify the characteristics of a KBS is to view it according to different levels of abstraction [Tong 1992]:

1. knowledge level knowledge that is used by and embedded in the system (artifacts, domain theory, design problem space, design solution space)
2. algorithm level problem solving principles used to implement the knowledge (generators, controllers, constraint propagators, patchers etc.)
3. program level the implementation of these principles (schemes, objects, rules, logic expressions, equations, constraints)

In knowledge based design systems, the knowledge level consist of the design space and the knowledge base; and the algorithm level of the process that navigates through that space, guided by the knowledge in the knowledge base. The design space is composed by a set of points and relations between those points. These points are either specifications or implementations, and associated with them are their parts (subparts or parameters), constraints on it and its parts and information about how its parts are connected. The relations between the parts are

*decomposition:* P2 is part of P1
*refinement:* P2 is a refinement of P1 where P1 and P2 both are specifications
*implementation:* P2 is an implementation of P1, where P1 is a specification for and P2 is a description of an artifact in the target technology
*optimization:* P2 is an optimization of P1 (i.e. P2 is better than P1 with respect to some evaluation criterion)
*patch:* P2 is a patch of P1 (P1 contains a constraint violation that P2 does not)
Knowledge based routine design is accordingly typically comprised of different types of basic operations: refinement operations which generate new points, constraint processing which prune inconsistent alternatives, patching operations convert incorrect designs into correct, and optimization operations converts sub optimal designs into more optimal ones.

4.3.2 Constraint based reasoning - an algorithm

A constraint network is a fixed network of relations between parameters, much like an equation system if the constraints are of numeric nature. Constraint satisfaction is useful for finding a set of valid values out of a large set of parameters, possible values, and constraints between them. If no valid set of values is found based on the initial preconditions, the network is not satisfiable. If, on the other hand, the set of values is not fully determined, a choice is made on some parameter’s value and the constraints are checked again. If this value was OK, a new parameter is given a value and so on. If a chosen parameter value should cause an inconsistency, that value is retracted along with all its consequences, and a new value is assumed; a process called backtracking. The procedure of choosing what parameter to instantiate, and with what value, determines how fast the network will reach a solution. This search controlling procedure can be designed in many different ways, and may, in a heuristic search approach, contain a substantial amount of design knowledge.

Though the network is predetermined, the parameters at the nodes may have many possible values and interrelations and thus contain a large degree of freedom. A constraint network also has the quality of being undirected, that is, the order in which parameters are instantiated is not pre determined; any value may be chosen first, controlling other parameters.

Constraints are thus a way of representing equations and other relations between variables. Constraint satisfaction, or solving, is a mechanism for determining values of the variables that are consistent with all the relations. A system of constraints, together with a constraint solving mechanism, can be used for determining individual values as well as for maintaining the consistency of the whole network [Güsgen 1989]. Constraint Logic Programming (CLP) [Jaffar 1987] basically means constraint satisfaction implemented in a logic programming framework. CLP supplies a platform for quickly implementing efficient and maintainable programs for configuration tasks which contain a mixture of constraint satisfaction and logic reasoning [Sivard 1994]. In the CLP based commercial configurator Tacton [TACTON 2000] for example, requirements or choices can be added one by one to see how they will effect the product properties or the possible variants of other components.
4.4 Modeling product families

There is a growing effort in trying to define how to model product platform information according to general principles for product modeling. Common to these efforts [Erens 1996], [Jiao 1998], [Tiihonen et al. 1998], is the goal of defining a framework upon which models that support some part of the product family life cycle can be built.

"ARCHITECTURES FOR PRODUCT FAMILIES”
Freek Erens, Karel Verhulst [Erens 1996] [Erens&Verhulst 1997]

This research focuses on representing the product architectures when developing product families. An ideal product family is defined as a product with identical internal interfaces for all variants in all domains. Interfaces must be standardized in three domains to allow the full exchange of components. A product family is defined by its parameters, and families are decomposed into sub families which inherit some of these parameters in an object oriented manner. A variant is created based on specific parameter values and constraints on the combinations of values. Following the German representation of technical systems [Hubka 1988], three parallel product architectures, a functional, a technological and a physical, are suggested.

The functional domain is a description of the set of functions of a system. It comprises the hierarchical structure of functions and the interfaces between these functions. Functions are here used in the formal sense, expressing abstracted behavior (function) of a product in terms of nouns (qualitative, quantitative, concrete, conceptual) and verbs (transformation, control, generation). Functional interfaces are defined as the interdependencies (couplings) between different functions, and it is assumed that there are functional interfaces irrespective of the physical realization of the functions which does not follow the view of axiomatic design.

The technology domain, consisting of technology modules (solution principles), is a description of the application of technologies to ensure the operation of the system.

The physical domain describes the physical implementation of the technologies that are applied in a system and is strongly related to the construction of the product. A design object defines a product on one level of abstraction and may be of type product family or product variant. Decomposition entities define relationships between design objects on different abstraction levels. Interface entities model interfaces which may exist between all design objects, both families and variants.
The relationships between domains are modeled as *Interdomain Mapping* entities, which concern both the allocation of functions onto modules and the validation of modules against functions.

Each product family has a set of *parameters* with *parameter values* which are used to specify variants and subfamilies. Parameters may relate to functions or modules in the technology domain or assemblies in the physical domain. Boolean *selection conditions* hold between a variant, a family and a parameter value. *Selection constraints* prohibit combinations of parameter values.

Erens suggest a generalized Generic Bill Of Material, a generic product structuring (GPS) concept, which is meant to structure product families independent of a specific domain. Although there are three parallel structures and mappings between these, these mappings are not used for explicitly relating alternative requirements to alternative solutions. The relating of requirements (family parameters) to solutions (component parameters) is done implicitly using class inheritance and Boolean constraints.

**PRODUCT FAMILY ARCHITECTURE**


The goal of the Product Family Architecture (PFA) is to unify sales, marketing, design and manufacturing through an integrated product family architecture. "The PFA can perform as a unifying integration platform to synchronize market positioning, commonality employment and manufacturing scale of economy across the entire product realization process." Based on the PFA, different product family planning applications are described, as well as a GBOM approach to managing bills of material of variants.

The PFA is based on representing three parallel structures of the product: the functional, behavioral and structural views. These structures are related by mappings and an object oriented framework with variety represented in all three domains is suggested. The principles for the relations between the variety of the functional domain and that of the behavioral and structural views are not described in detail, though. The three structures are designed in parallel, which does not support a holistic reasoning concerning what principle to choose and how that will affect the following requirements, or how to integrate parameters into physical components. Thus, even though the mapping constructs are present, the semantics of the mappings, and the trace from varying customer needs to physical realization, is not clear. It is declared that functions are created by organs which are materialized by components as in the Chromosome model [Andreasen 1992], but then the principles for how to relate the views in detail, and implement the mappings between them, are not covered. The PFA "organ-
izes and represents a variety of objects in different views using class-member relationships”. It is a purely object oriented approach, and there is no expressed meaning of this relation between the class object and member object other than "membership”; no rationale in terms of functional specialization between a parent-child, or difference in specialization between two alternative children of a parent. All in all, the PFA basically share the goal with the GIP, is an interesting approach, well described and based on a large bulk of research, but there are important principles for how to represent the underpinnings of product families missing - a gap between the information used when designing a product family and the object oriented implementation.

"CONCEPTUALIZATION OF CONFIGURATION DOMAIN"

Juha Tiihonen et al. [Tiihonen et al 1998]

This research focuses on the sales-delivery process, but it is argued that the conceptualization can be used in the development phase as well. In the conceptualization of the configuration domain, a product family is equaled with a configurable product. Based on object oriented conceptualizations proposed in AI-oriented configuration, a generalized model adapted to real world products is described. Three categories of configuration knowledge are distinguished: configuration solution knowledge specifying an (possibly partial) individual configuration, configuration model knowledge: specifying all the entities and their dependencies, and requirements knowledge specifying the requirements on the configuration to be constructed. A configuration can contain individuals of four different types in the configuration model knowledge: component, port, resource and function. These types are organized into object classification hierarchies with inheritance of properties from supertypes to subtypes. Types can be concrete - for use in an unambiguous configuration, or abstract - providing partial information of an entity and used for creating classes.

A product structure is built up using part role relations describing how a component type is composed by sub components using part-relations, at the same time describing the function, or role, of the component in the part-relation. E.g. a component type ‘Lamp’ has part roles ‘Lampshade’ and ‘Stand’, which may be filled by a component individual ‘lbs1a456’. It is emphasized that things that are common to all variants should usually not be modeled. Connection interfaces, physical or logical, are modeled as ports. A port type has a compatibility definition that defines a set of port types whose port individuals can be connected. In addition, a port type defines a set of connection constraints which has to be satisfied to connect port individuals. Resources are used for modeling the production and use of some more or less abstract entity. This conceptualization, as opposed to basic global resource
mechanisms, adds a context mechanism that makes it possible to limit resource availability to some specific set of component individuals. Functions and function structures are defined for representing the functionality that the product individual provides to the customer. Constraints are used as a general mechanism for specifying the interdependencies of configuration related types in the configuration model. Although this conceptualization narrows the gap between object oriented configuration models and product definitions, there is no theoretic foundation describing the relations between requirements and solutions, or the rationale for classifying variants into type classes. In addition, the aspects of managing versions of these models during product development, and managing views and parallel functional, principal and physical structures, are not addressed.

4.5 Product modeling standards

4.5.1 Logical Platform

A ‘Logical Platform’ is under development by the Volvo Group, addressing the need of this global enterprise of facilitating the information exchange between its companies, and reducing the cost of implementing PDM systems in each individual company [Volvo 2000]. The aim of the PDM logical platform, is to define a common specification of an implementation independent PDM system for use in each of the Volvo product companies. The common logical platform constitute an information model which defines the required PDM information within Volvo, and which is mapped to the STEP AP214 standard application neutral and open format.

4.5.2 ISO 10303

The intent of the ISO Standard 10303, STEP [STEP 1993], is to develop standardized information formats for expressing product information for the purpose of exchanging data independent of system or application.

"The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system" [AP214 1998]. AP214 "specifies the use of the integrated resources necessary for the scope and information requirements for the exchange of information between the applications that support the development process of the mechanical aspects of automotive vehicles".

Figure 37 The objective of STEP and scope of AP214
STEP consists of common parts: the Integrated Generic Resources (IGR’s) and Integrated Application Resources (IAR’s), and application specific protocols (AP:s). STEP AP214 is one application protocol, developed primarily for automotive mechanical design processes. This protocol defines the context, scope, and information requirements for various development stages during the design of a vehicle and specifies the integrated resources necessary to satisfy these requirements. The AP addresses the requirements of the automotive industry covering cars, trucks, buses, and motorcycles. Even though this scope might seem narrow, it has turned out that the protocol is applicable to many types of products. Since configuration management is of utmost interest to the car industry, there are constructs for expressing variants and rules that provide a basis for expressing product platform information in general.

An application protocol is in principle first written independently of STEP using the terminology of the application area. The result is an Application Reference Model (ARM). This model is then implemented using the STEP concepts, that is, using both the Integrated Resources (IR), common to all protocols, and the extensions defined in the application protocol itself. The result is an Application Interpreted Model (AIM) which is the actual data model of the application protocol in STEP.

EXPRESS [Schenck 1994] is an object flavored information modeling language with a graphical representation which is based on entities and relations: EXPRESS-G. EXPRESS is Part 11 of the standard ISO 10303 and was developed for expressing the information models of STEP. A short description of EXPRESS-G is given in Appendix 10.5.

4.5.3 The AP214 modeling protocol

AP214 was developed as a standard for exchanging information relevant to vehicle design, and comprises extensive constructs for representing structures, their hierarchies, views etc. A class of products can be described from different points of view and levels of detail using three different types of entities; functional, logical and physical components, and their decompositions and interrelations.

A Product_function is a behavior or an action expected from a product. E.g. The Product_function ‘transportation’ with sub-functions ‘acceleration’ and ‘retardation’ are Product_function objects in a hierarchical decomposition structure. A product_component is a logical entity, a principle solution and an element in a conceptual product structure. Most Product_component objects are realized by parts, but there are exceptions such
as *trunk space* or *holes for wire harness* which are not. The *item*, or rather *item_version*, is the physical entity which realizes a *product_component*, and it serves as the collector of the data characterizing a physically realizable object in various application contexts.

![Diagram of component structure](image)

*Figure 38 The component structure ”motor” of AP214*

EXPRESS-G Legend:
- Rectangular boxes represent entities being defined in this schema
- Round boxes represent entities being defined elsewhere
- Thick lines denotes subtype - super type relationships with inheritance of all properties
- Lines with a circle are used to connect an object with its attributes
- Dotted lines are used to indicate optional attributes

Functional and conceptual product components and their alternative solutions are of type *complex_product*. Product functions and components are also, together with (physical) item instances of type *product_constituent*. These components can be related by a *product_structure_relationship* which are of different types such as *realization*, *decomposition*, or *specialization*. Structures of these components can be created and modeled in EXPRESS-G using the structure building ”motor”, shown in the diagram. Since the purpose of AP214 is to define components and subsystems of vehicles independent of companies and products, and since the decomposition and exact definition of components change, no general classes of components such as ”motors” or ”suspensions” are defined. This prevents the ability of directly representing
classes of components with inheritance, while providing generic information models of components that are flexible to changes.

4.5.4 The AP214 Product family model

Simply put, in AP214, a product family is modeled as one generic structure of component alternatives, common to all products + constructs for segmenting this union into different classes according to market needs. This segmentation is represented in terms of specifications describing the product characteristics which vary in each product class. Selection of choices is made by a direct mapping between desired characteristics and realizing components.

4.5.4.1 Union of alternatives - generic structure

In the following simplified models (indicated in the caption), to increase clarity, relations are only illustrated as small boxes, as versus the larger entity boxes.

The basis of a product family model is the common product decomposition structure, modeling the union of all alternatives:
The structure above describes a decomposition structure with conceptual components and models of the specific manufacturerable items which realize each component. This structure defines all manufacturable parts and is common to all versions of the product class ‘my auto’. It defines all parts and their structure, but no combinations of parts. These combinations are specified in a separate structure, soon to be described. But first, a more detailed description of the main structure will follow.
A `product_class` is realized by `product_components` which are decomposed into product components which may be the base_element of different alternative_solutions. An alternative_solution is the identification of one of potentially many mutually exclusive implementations of a Product_function or of a Product_component.

Figure 40: Alternative radio components (simplified illustration)
4.5.4.2 Classes of variants

PRODUCT CLASSES. Product classes are defined which cover different sets of the components.

Figure 41 Part of audio structure (simplified illustration)
A hierarchy of product classes, covering the whole enterprise or different product types or markets and years may be defined. According to AP214 there are 4 level types: Enterprise (all objects for an enterprise with several brands), platform (all models based on the same technical concept), car family (all products based on a fixed set of characteristics) and car type (products that are offered to the market, typically defined by the marketing department). Product classes may also come in versions, be replacing each other, or be derived from one another.
4 PRODUCT MODELING APPROACHES IN RESEARCH AND INDUSTRY

4.5.4.3 Specification of varying components

SPECIFICATION. A Specification is defined as a characteristic of a product which discriminates one product from other members of the same Product class, such as specifying that a car should have a sedan body or a "radio with CD player". The role of a specification with respect to a particular product class is described as an attribute of the relation between the product class and the specification.

A set of specifications sufficient to produce a product variant is associated with its Product class. A distinction is made between standard characteristics (e.g. number of doors) and alternatives that have to be chosen (e.g. outside color) or that are options to the customer (e.g., radio). The Product class entity may also be used to group characteristics together which belong to a whole enterprise, or to a technological platform upon which many product families are based.

SPECIFICATION ATTRIBUTES. The role of individual specifications as being default (belonging to the most wanted set of solutions) is described as
well as the role of a whole group of specifications as being required in a product family or not. If a component is common to all variants, it is not specified, otherwise specifications have the following types:

I. Common to some subset of the family:
   A. non replaceable standard
      The non replaceable standard is a characteristic of any product belonging to the product class and may describe a general constraint that will influence many components of the product, such as the climate zone for which the product is intended to be sold (‘South East Asia climate zone’ is a ‘non replaceable standard’ for a ‘Sunny’ with market context ‘Japan’).
   B. identification
      An identification is part of the identification of all sub classes of a particular product class such as the car body type ‘limousine’ which is identifying for product class ‘S40’.

II. Replaceable standard
    If nothing else is specified, this component will be chosen from the set of alternatives.

III. Option
    In AP214, an option replaces another, ‘replaceable standard’, component. Another interpretation of option is that is added to a structure, not replacing another component. In that case there is an option between something or nothing which might be difficult to handle in practice. In AP214 this is handled as specifying "default or default + option" instead of "no option or option".

IV. Availability
    Denoting that a component is one of several available alternatives, not saying whether it is default or an option.

Figure 44: Associating a product class with its characteristics

To connect a specification with something that realizes it, a configuration entity is used. A configuration entity associates a Class_specification_association with a certain component or process. The validity of the association may be limited by a time period through assigning an Effectivity object to it. This way different solutions can be used for the same specification in different product classes, or the effectivity of a solution may be different in different product classes (e.g. a component may be upgraded later in the American than in the European market). An Effectivity entity is "the identification of the valid use of an aspect of product data tracked by date or event." An effectivity entity can be assigned to a number of different entities and relations such as ‘product_class’, ‘class_specification_association’, ‘specification_expression’, ‘item’, and many more.
And, if the component decomposition is added to the picture:

![Diagram of component decomposition](image)

*Figure 46 Common decomposition structure + one specified characteristic and its realizing element (simplified illustration)*
In the audio example, it is specified that the analog radio is a default, or "replaceable_standard", with the digital radio and CD system as options. Since the radio needs an antenna, this component also has to be specified, which is done through the replaceable_standard "with antenna" and option "without antenna". The specifications concerning the audio equipment are put together in a specification_category. A Specification_category is the definition of a set of Specification objects serving the same purpose, such as the category ‘kind of drive’ for the Specification objects ‘front wheel drive’ and ‘rear wheel drive’ or the category ‘level of equipment’ for the Specification objects ‘standard’, ‘comfortable’, and ‘luxurious’. The implicit_exclusive_condition specifies whether the Specification objects within the Specification_category are mutually exclusive for the production of one particular product. In the audio case, the alternative specifications are not mutually exclusive.

A Specification_category may or may not be mandatory to specify when specifying the product from a product class, a quality which is represented in the relation class_category_association. A motor is for example mandatory to specify while a radio may be excluded.
The whole specification of the specified components of the example could look like this:

**Figure 47 Classification of the specifications of a product class**
4.5.4.4 Specification of relations between varying components - configuration rules

The validity of a specification may be conditioned by validity rules: specification_expressions or specification_inclusions. Specification_expressions are Boolean expressions relating specifications such as NOT (radio AND without_antenna).

A specification_inclusion declares that if one specification is chosen then another specification should be also included. It may be used to include one or many related specifications such as a specific "deluxe" package, or to exclude another specification. These validity rules are related to the product class by a class_condition_association, or a class_inclusion_association.

Figure 48 Configuration rules specifying that a radio requires an antenna

4.5.5 AP214 characteristics

No incomplete solutions. For example, it is not possible to express a condition that should hold between two class elements and have this condition be inherited to the children. General rules cannot be represented in AP214, such as: 'If A has an interfaceA and B has an interfaceB and interfaceA and interfaceB should be connected, then they should be compatible'. One reason for this is
that the protocol is developed for exchange and ambiguous models have to be avoided. Another reason is, that since the same entities are to be used by various companies, and companies represent their components in different ways, it is not possible to standardize the models of specific components (such as a class of motors).

If a constraint concerning the combination of two components is to be represented in AP214, it will be represented as a conditional rule between two component variants, with all alternative realizations explicitly modeled.

**Means for expressing variants and versions.** Even though the AP214 way of expressing product families is not adapted to the design of product family architectures, it comprises an efficient way of expressing large sets of varying structures and their configuration specifications. It is also efficient in keeping track of many various sub families and varying views of components, as well as various annotations (e.g. different languages for describing the same entity) - constructs that are truly useful when administrating large sets of data over many countries and applications.

### 4.6 Conclusion

Though these are interesting research approaches to modeling products, addressing aspects of realizing a common product model, none of them suggest a method for modeling product families in a way that captures the design rationale for selecting a variant. Although most of the methods represent functional as well as technical and physical structures, they do not, apart from [Malmqvist 1995] reflect the design decision process of interrelating requirements with technical solutions and physical components. These interrelations represent design decisions, which together with the requirements and solutions reflect the rationale for choosing one alternative solution for another.

Moreover, none of the research methods treat the issue of developing an integrated information platform, supporting the exchange of information between all phases of product family based product development. Although the AP214 standard protocol represents a large part of the product family and specifies how to represent information in an integrated way, e.g. providing support for managing versions and variants simultaneously, it does not fully cover design or sales configuration issues.
5 THE GENERIC INFORMATION PLATFORM

This chapter presents the fundaments of the Generic Information Platform, and serves as an umbrella for the forthcoming three chapters which describe the GIP models in more detail.

For definitions of central concepts, see Appendix 10.1 on page 192.

5.1 Principles

The fundamental principles and concepts of the GIP, which will be discussed in the following, are:

• the principle of defining a common model of information as a means for meeting the requirements on efficient use of information. Further, to standardize the model as a means for widening the range of use and exchange

• the principle of modeling information in a semantic model, which defines the meaning of each entity irrespective of application, and clearly separates entities with different semantic meaning

• the concept of acquiring information, concerning rules and conceptual solutions, during the design phase, to be reused during the order phase

• the concept of modeling conceptual solutions in a solution library for the purpose of reuse

5.2 Scope

The vision of efficient use of information in product family based product development can be formulated as two separate goals:

• enable systematic use of information during design to create a flexible and efficient product family by
  - enabling the reuse of resources in terms of concepts and technology in addition to the reuse of physical solutions
  - providing analysis support of characteristics of the product family

• enable reuse of information between applications and systems
The general solution proposed to meet these goals is by a common computer based model of product information - an information platform. The purpose of the Generic Information Platform is to define and describe the core of product family information - the constructs and principles for modeling product families. While one long term goal is to also represent resource-, process- and other information required when designing the product family and realizing resources, these models fall outside the scope of the current GIP.

5.2.1 Purpose

The overall purpose of a product family model is to serve as a basis for structuring and relating product family information in a way that a) supports the design of product families, and b) facilitates information exchange and reuse within and between applications and systems using this information.

Information from design should ideally be modeled in a way that requires a minimum of (ideally none) adjustment when interpreted by the applications in the order phase.

5.2.2 Viewpoints

The specific applications in focus are the support in the planning and conceptual design of product families, and support in the configuration of a specific variant to order. Two viewpoints are thus in focus: the product family architecture design view, and the configuration systems development view.

5.2.3 Detailing level and delimitation

A purely conceptual description is attempted in this thesis: a model which captures the structural aspects of the alternative solutions and their interrelations. The level of detail is low and generally applicable to all types of design. Although some simple property models are represented, and constructs are defined to relate detailed models to the structure, deeper modeling such as detailed models of function or detailed numerical calculations is not attempted.

Information modeling of geometry is covered by the STEP standard, and since the information platform connects to this standard, these models can be reached. Information is described irrespective of the specific requirements of operational implementations and applications. Aspects of efficiency are not treated. The representation of the (realization) process domain is not part of the description although it, as previously mentioned, is considered a natural part of a future version.
5.3 Architecture

Based on the goals of efficient reuse of solutions as well as providing support in the analysis of one product family, the Generic Information Platform is divided into two main parts: Platform Solution models and Product Family models.

![Diagram of Parts of the Generic Information Platform]

**Figure 49 Parts of the Generic Information Platform**

5.3.1 The Platform solution models

The platform solution models define a design library and consists of reusable models of resources which are shared between product families; some are common to all variants, while some vary. Further, there are pure standardized components as well as core technologies which hold strategic significance.

Platform Process models describe the processes by which components are realized, and, apart from the specific manufacturing descriptions provided by AP214, these are out of scope in this thesis.

Solution models describe components - physical components and abstractions such as composite systems, as well as incomplete patterns and technological principles. A model may also define a solution in terms of its function, interface, or boundary constraints, i.e. a function and interface which is common to
all variants, while the internal embodiment of the solution may vary. A solution may comprise variation by representing a whole class, or be a specific variant.

Table 7: Kinds of solutions in platform solution library

<table>
<thead>
<tr>
<th>Models in platform solution library</th>
</tr>
</thead>
<tbody>
<tr>
<td>fully defined components</td>
</tr>
<tr>
<td>component classes (Modules) with variants</td>
</tr>
<tr>
<td>parametric models</td>
</tr>
<tr>
<td>black box solutions, defined by their function, interface, and boundary constraints (e.g. geometrical envelope).</td>
</tr>
<tr>
<td>design patterns, defining main function and a (incomplete) trace from requirements to technical solutions</td>
</tr>
</tbody>
</table>

5.3.2 The product family model

The product family model can be viewed as a model of all members of the family, which is constrained by the individual needs of a specific customer. A family model is comprised of:

I. a model of all solutions of the family - (the trace from generic requirements to) generic solutions

II. a model which specifies the individual variants based on the requirements - the trace from specific requirements to individual variants

5.3.2.1 Model of all solutions

In the information platform approach presented here, functional requirements, technical and physical solution structures are represented to support the design, as well as the variant selection, processes. Varying views of the detailed physical structures, other than the functional and assembly views, are possible to represent in accordance with the AP214 protocol. In the conceptual model, views are not specifically addressed, other than the possibility to model view-specific decompositions of components.

5.3.2.2 Representation of the variant specification

The information platform approach is to define the choice of variants in terms of the requirements on properties because that 1) provides a flexible representation - specific solutions may be changed as long as the requirements are met
2) is based on the principle of following the design decisions and thus captures the rationale for the choice and 3) follows the principles of AP214 and thus enables integration with this standard.

5.4 Modeling Fundaments

To realize a common computer based information platform, it is necessary to:

- acquire the (right) information
- represent the information in a manner that makes it possible to use and modify for all (intended) applications and users

The fundamental idea is to represent product family information in a format which is generally applicable for various purposes. Apart from establishing a common information language, this requires that concepts with different semantic meaning (and use within applications) are separated within the model. Defining such a ‘general information model’ renders the possibility of:

1. reusing common information for different applications such as design and configuration
2. integrating information from different sources
3. exchanging information between different systems

Moreover, acquisition of information during design is facilitated if information reflects the conceptualization of the design process, and thus eliminates the need for entering information as a separate activity.

As a means for realizing such a model, the information of conceptual product family design is represented in terms of the domains and interrelations of Axiomatic Design, while the information of detailed design is represented in the standard information models of AP214. By integrating these two models, an integrated standard information model is achieved.

5.4.1 General Model of the conceptual design trace and result

It is generally accepted, that there are three fundamentally different design views of a product: its function, its technical solutions and its physical components. Requirements concerning function are realized by technical solutions, which are detailed into physical embodiments.

Since the entities of the functional, physical, and process domains of Axiomatic Design are inherently independent, separating concepts with different semantic meaning, axiomatic design provides a good basis for modeling in-
formation in a general way.

By developing a product family model based on the framework of Axiomatic Design, the following characteristics are achieved:

**Integrated model.** By modeling the design process according to Axiomatic Design, a model which interrelates the design process and object in a structured way is acquired. This model describes the trace between requirements and solutions as a mapping from requirements to the key design parameters, which are embodied in components.

**Semantic description.** In addition to describing functional requirements and design parameters and their interrelations, the theory defines a semantic description of constraints in relation to these entities.

**Axioms for product family design.** The axiomatic design framework provides design rules (axioms) that are adequate in the design of product families. Independent functional requirements are separated from constraints, and by modeling the relations between functional requirements and design parameters, the degree of interdependence within a solution can be analyzed and used as a guidance towards designing a more optimal product family. The main principles of axiomatic design are presented in chapter 4.1.2 and the product family model based on AD in chapter 5.

**Process models.** Though process models are not addressed in this thesis, the axiomatic principle of representing the process domain in integration with the functional requirement and design domains, is adequate in design for variety and thus another reason for choosing this framework.

### 5.4.2 Standard model of detailed product information

By adhering to a standard format, information can be exchanged between all users adhering to the standard, the inherent functionality of the protocol can be used, and the information already modeled in the protocol can be integrated with the model.

In the GIP, the family model is adapted to the information modeling standard of ISO 10303-214, which provides the ability to express additional structuring and modeling constructs to describe detailed parts, advanced product family segmentation and version management, and utilizing the advantages of a standard format. AP214 is described in chapter 4.5.3 and the adaptation to this protocol in chapter 7.
5.5 Guidelines for retrieving information

In the following, the types of ideas, facts and processes used in the three main product family information applications are presented together with a description of how this information is represented within the GIP model.
Table 8 Information in design

<table>
<thead>
<tr>
<th>Needed information</th>
<th>Representation in GIP model</th>
</tr>
</thead>
<tbody>
<tr>
<td>the trace from customer requirements to product properties, components and assembly modules</td>
<td>customer needs <em>interpreted as</em> functional requirements <em>realized by</em> design parameters <em>embodied by</em> components <em>aggregated into</em> composite components (assembly modules). customer needs are also <em>interpreted</em> as constraints which <em>delimit</em> design parameters and are <em>refined</em> into other constraints or functional requirements</td>
</tr>
<tr>
<td>the relation between functional requirements and design parameters (Axiomatic Design)</td>
<td>functional requirements are <em>realized by</em> design parameters which <em>requires</em> other functional requirements. FRs are also <em>dependent</em> on other DPs, forming a coupling between FRs.</td>
</tr>
<tr>
<td>connection between conceptual solutions to detailed models (supporting reuse and verification)</td>
<td>integration of GIP models with AP214. Specifically a GIP ‘component’ is modeled as a AP214 ‘product_component’, and the constructs of AP214 related to ‘product_component’ can be used, such as the relation <em>realization</em> between product_component and item_version which models parts.</td>
</tr>
<tr>
<td>ability to design, reuse and manage interfaces</td>
<td>representation of interfaces as separate entities, related to components, modeled in terms of descriptions of the interface interactions and connected ports, and indexed for search</td>
</tr>
<tr>
<td>ability to define and reuse components based on boundary constraints</td>
<td>constraints of axiomatic design correspond to system limitations such as required interfaces or boundaries, as well as to property constraints.</td>
</tr>
</tbody>
</table>
### Table 9  
**Sales configuration - model based configuration**

<table>
<thead>
<tr>
<th>Needed information</th>
<th>Representation in GIP model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. attributes - descriptive features such as functional and technical characteristics</td>
<td>1. partly inherent in the descriptions of the FRs and constraints which represent functional characteristics, and DPs and component properties which describe the technical characteristics</td>
</tr>
<tr>
<td>2. resources - property that a component either supplies or uses</td>
<td>2. represented as a constrained DP or component property</td>
</tr>
<tr>
<td>3. ports - places through which components connect and communicate</td>
<td>3. represented as component properties</td>
</tr>
<tr>
<td>1. classification, aggregation</td>
<td>1. ‘specialization’, ‘aggregation’</td>
</tr>
<tr>
<td>2. local constraints</td>
<td>2&amp;3. Information models of design constraints: An axiomatic constraint ‘impacts’ a technical solution, constraints are refined into more detailed constraints</td>
</tr>
<tr>
<td>3. global constraints</td>
<td>Operational models of some constraints: Boolean and equality constraints provided</td>
</tr>
<tr>
<td>value selection and optimization mechanism</td>
<td>FR-DP mappings provide functional to technical property relations, otherwise no mechanisms supported</td>
</tr>
</tbody>
</table>

### Table 10  
**Product configuration - product structure + configuration rules**

<table>
<thead>
<tr>
<th>Needed information</th>
<th>Representation in GIP model</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic structures with alternative components</td>
<td>generic physical structure</td>
</tr>
<tr>
<td>selection rules (from customer choice to selected value)</td>
<td>CN interpreted to Constraint which may be refined into lower level FRs or impact a DP to be a specific value</td>
</tr>
<tr>
<td>delimitation rules (internal model)</td>
<td>functional component structure (if not explicit, created by following the FR-DP-Component path) and constraints between components such as matching interface constraints.</td>
</tr>
</tbody>
</table>
### Table 11 Information management

<table>
<thead>
<tr>
<th>Needed information</th>
<th>Representation in GIP model</th>
</tr>
</thead>
<tbody>
<tr>
<td>varying views of components</td>
<td>components in AP214 may be assigned different views by an attribute of the entity</td>
</tr>
<tr>
<td>ability to manage many related product families with different characteristics</td>
<td>product class structure and configuration relations between product class and customer needs</td>
</tr>
<tr>
<td>ability to reuse components differently in different product families</td>
<td>specification roles and configuration relations which link a customer need to different constraints depending on product class</td>
</tr>
<tr>
<td>ability to integrate product families from one supplier as components of another product family of a system</td>
<td>representing components in analog with product families with exception for the relations to customer needs. A product family is integrated as a component by devising it as a DP which realizes an FR in the ‘parent’ family.</td>
</tr>
<tr>
<td>ability to relate parts to the way in which they are manufactured</td>
<td>component to item_instance to process relations</td>
</tr>
<tr>
<td>ability to manage different effectivities of parts</td>
<td>AP214 manages effectivities of parts</td>
</tr>
</tbody>
</table>

### 5.6 Conclusion

The GIP architecture, which is based on a solution library and a product family specification, represents information in a general way for the purpose of information exchange and reuse. To achieve this general model, the process and object of design is represented based on a general, recognized, design process - Axiomatic Design. This model captures information concerning the requirements and their realization within a product family, which can be reused for describing configuration models. By adapting this model to the ISO 10303-214 standard, a model is achieved which captures conceptual as well as detailed information concerning the product properties and parts, and their roles in the product family and development process. This way an information platform can be realized, which can be reused for various tasks, such as design, configuration and information maintenance, during customer order based design.
6 Describing product families in the Axiomatic Design framework

This chapter presents a major contribution of this thesis: the design and description of product families based on the concepts of Axiomatic Design (AD). Major contributions concern the description of variety in AD, and relating the information of the AD product family model to its use during design and order based design. Based on previous research and methods for AD, modularization and design for assembly, a way of integrating DPs into components adapted to product family design is also suggested. An example describing the design of audio equipment is used to clarify different concepts. Since one objective in this chapter is to focus on the principles of representing product families, the definition and use of models is limited, but treated in detail in Chapter 6.

6.1 Describing products in AD

Axiomatic Design was chosen as the basic framework for describing product families since it, apart from the design axioms, provides an integrated description of the design process together with the design object.

As described in Chapter 4.1.2, axiomatic design is a theory for designing products, describing one product as the physical realization of a set of requirements. In summary, the axiomatic way of describing the way customer needs (CNs) are turned into design parameters (DPs), can be described (based on [Suh 1990] and [Tate 1999]) as: Interpreting societal needs to independent functional requirements (FRs) and constraints. Zigzag mappings between functional requirements to realizing design parameters, delimiting the choice of design parameters according to the constraints. Constraints are refined into functional requirements and constraints at lower levels. Design parameters may introduce constraints on the following levels, since they define the system that the following design parameters have to reside in. This process, and the entities and relations of the theory will be described in more detail in the following.

6.1.1 Domains and mappings

In 2, the relations between Functional Requirements and Design Parameters
are shown in more detail in a function-means tree [Tjalve 1979][Svendsen&Thorp Hansen 1993], visualizing both domains in the same structure. A Functional Requirement is realized by a Design Parameter which in turn needs a set of new requirements to be satisfied in order to be realized. Constraints delimit the possible realizations of an FR, either by pruning possible alternatives from the solution tree or by delimiting some parameter of the chosen alternative. If for example the FR is to ”remove dust” and a constraint is that the solution could not use electricity, all DPs requiring electricity would have to be pruned from the tree. If, on the other hand, the constraint concerns delimiting the total allowed electric effect, it would be necessary to constrain the value of the total effect of the solution and correspondingly constrain the effect-properties of all sub design solutions, such as the electric motor of a vacuum cleaner. Constraints may thus, as mentioned previously, be part of the initial preconditions, or be introduced as a result of previous design choices.

**Figure 50 Tree with relations between FR:s and DP:**

Constraints represent the bounds on an acceptable solution. In the initial theory, they were classified as input constraints, which are constraints in design specifications, or system constraints, which are constraints imposed by the system in which the design solution must function. The input constraints are
usually expressed as bounds on size, weight, materials, and cost, whereas the system constraints are interfacial bounds such as geometric shape, capacity of machines and even the laws of nature.

*Table 12 Examples of constraints on DP realization*

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Constraints</td>
<td>effect &gt; 50hk, brand = 'Johnson', space limits stated by customer</td>
</tr>
<tr>
<td>System Constraints</td>
<td>interfacial boundaries (e.g. geometry, capacity of machines) derived constraints (from choices higher up in the hierarchy) &quot;the interface of connected components must match&quot;.</td>
</tr>
</tbody>
</table>

Constraints are different from FRs in that they do not have to be independent from each other or from the FRs, and that they normally do not have any tolerances associated with them, while FRs typically do [Suh 1990]. Notwithstanding these rules, it has been found difficult in practice to distinguish FRs from constraints. To remedy this, Tate, [Tate 1999], suggests a more detailed classification of constraints:

1. critical performance specifications: constraints imposed on the attributes of the top-level target objects or on the rate at which these transforms are performed.
2. interface constraints: constraints imposed on the inputs and outputs that the system must accept.
3. global object constraints: constraints with the potential to affect all DPs in the design (or some significant fraction) and which are broken down in an additive way.
4. project constraints: constraints on the development resources allowed for design or redesign, or on the decisions made across projects (standardization etc.)
5. feature constraints: constraints that apply to the choice of specific DPs within the system.

In constraint based reasoning (described in Chapter 4.3.2), constraints are often classified as being of a global or a local kind. Local constraints state a relation between a few, connected variables (structure, arithmetic etc.), and global constraints state relations involving many variables spread over the constraint network (resource constraints, optimization criteria etc.).
Based on these classifications, for the purpose of describing constraints on a product variant in a product family, constraints will be classified along two dimensions: extent of dependency introduced (local or global) and origin (customer input or system limitation):

Table 13 Classification of constraints

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>performance specifications</td>
<td>feature constraints</td>
</tr>
<tr>
<td><strong>System (initial &amp; derived)</strong></td>
<td>resource limitations (boundaries)</td>
<td>interface constraints feature constraints</td>
</tr>
</tbody>
</table>

### 6.1.2 Describing components and component structures in AD

The decomposition structure of DPs is a physical description which may contain principle solutions such as a kind of interface, as well as material and geometry. The embodiment of DPs into physical components and properties, and the description of a physical structure, is not treated in detail in the original description of axiomatic design, but has been addressed in more recent research [Tate 1999]. Based on that research and methods for modularization and design for assembly, a way of integrating DPs adapted to product family design is suggested in the following sections.

#### 6.1.2.1 Integrating DPs into physical components

When requirements are satisfied by DPs, these may or may not be parts of the same physical whole. The fact that two properties are part of the same whole could be considered as a realization of a constraint "hold together” which may be (more or less consciously) added by the application. Integration can occur in one or two steps: either several DPs are embodied into different properties of the same component, or DPs are embodied into separate components which are later aggregated into one composed component. The purpose of this discussion is to clarify the similarities between integrating DPs into components and aggregating components into composed components or modules.
6.1.2.2 Separation drivers - rules for integration

According to the Information Axiom, as long as the independence axiom is met, it is preferred to integrate and reuse parts: Corollary 3: "Integrate design features in a single physical part if FRs can be independently satisfied in the proposed solution". Corollary 4: "Use standardized or interchangeable parts if the use of these parts is consistent with the FRs and constraints". Based on this axiom, the starting objectives are thus 1) to integrate and 2) to reuse existing components as much as possible. The integration task is to make a trade off between these two objectives, and to identify what is necessary to separate and change.

There are different methods for making these principles operational [Vallhagen 1996], [Boothroyd&Dewhurst 1987] [Erixon 1998]. The MFD method (described in 10.1), introduces operative ‘module drivers’ that correspond to various reasons for integrating or separating components.

Separation drivers: Instead of module drivers, the notion of ‘separation drivers’ is now introduced since it is assumed that all DPs should be integrated into the same physical embodiment unless there are identified reasons for separation. The rationale for separating and changing is here divided into 3 main types of aspects - functional, strategic and life cycle process exemplified in Fig. 51.

Figure 51 Embodiment of DPs into components

6 DESCRIBING PRODUCT FAMILIES IN THE AXIOMATIC DESIGN FRAMEWORK
The strive for integrating two DPs or components, and then separating these from the rest, is a result of some kind of coupling between these two. Either functional, or strategic (e.g. they are both part of satisfying the same group of customer needs, and thus vary according to the same pattern) or process (e.g. their physical integration is common).

### 6.1.2.3 Functional versus physical structures

There are many views, required for different purposes, of how to structure
components to reflect their various relations to each other. Two important views are treated here, namely the physical component structure - organizing components according to their **aggregation** into assemblies, and the functional component structure - organizing components according to their functional **decomposition**.

DPs are embodied into properties and components which may be aggregated, forming an physical structure. The functional component structure is achieved from the DP decomposition structure (which in turn is a result of the FR-DP mappings) together with the DP-component embodiment relations. Illustrates a model of the FR-DP and DP-component relations which can be followed to achieve the implicitly captured functional component structure.

Figure 52 DP-FR mappings with each DP embodied by a component

To make the functional structure, as well as the DP decompositions, explicit, ‘decomposition’ relations are added.

Figure 53 DP decomposition structure and explicit component design structure

The aggregation of components into assemblies are modeled as aggregation relations, forming a physical component structure as illustrated in 27.
If the design matrix is diagonal, and each DP is embodied into a component following the DP-hierarchy, and these components are aggregated following the same DP-hierarchy, the functional requirements, and the functional and physical component structures will be equal, realizing a ‘pure’ modular structure [Ulrich 1995]. Within modules, DPs may be integrated without causing couplings on the higher level.

If, on the other hand, a DP is integrated with DPs of another function, a physically coupled design is achieved even though it may remain functionally decoupled. Reasons for such a ‘non-modular’ design may be that systems are physically distributed e.g. a radio system with the antenna in the ‘roof module’.
6 Describing Product Families in the Axiomatic Design Framework

Another reason is that there are strong strategic reasons for integrating components of different functionality, e.g. a radio which is integrated with the control module since a radio is almost always required and it is more cost efficient to equip all units with a radio even if it is not required.

And if all DPs are embodied into the same component, there is no structure at all of individual, physically separated, components, since one component realizes all the different functional requirements. As illustrated in Figure 57, abstract embodiments of DPs are not necessarily modeled as components, but the functional component structure mainly consists of DPs with components in the leaf nodes only.

6.2 Describing product families in AD

A product family can be seen as a set of products meeting somewhat differing but essentially the same, functional requirements, and satisfying some different constraints (on e.g. different sizes, styles, performance) or optimization crite-
ria. One key to designing product families in the framework of axiomatic design is that of formulating generic functional requirements for the whole product family, enabling a high degree of commonality [Acclaro 2000].

6.2.1 Product family definition

Two principle ways of describing a product family in AD are suggested:

I. as one solution to generic family requirements which encompass the whole variety

II. as the union of all alternative solutions which meet common and alternative requirements

Depending on which principle is used, solutions are designed somewhat differently. In the first case, with the whole family regarded as one solution, the union of customer needs is interpreted to ‘family’ FRs and constraints which encompass the whole variety. E.g.

FR1: ‘requirement A and B and C’ \rightarrow DP1 ‘system with A, B, and C’
FR1.1 realize A \rightarrow DP1.1 A
and FR1.2 realize B \rightarrow DP1.2 B
and FR1.3 realize C \rightarrow DP1.3 C.

Since the whole family is one solution, only the original AD relations, as described above, are needed. Example:
In the second case, the family is described as common FRs and constraints, which are realized by alternative DPs, delimited by alternative constraints.

FR1: ‘generic requirement on (implicitly) A or B or C’

\[ \rightarrow \text{DP1 A} \]
\[ \text{or DP1 B} \]
\[ \text{or DP1 C}. \]

This description requires the ability to express alternative solutions and requirements. Example:
In this case, the constraint on ‘common interface’ is assumed to come from a higher level decision of designing families with interchangeable modules.

In the following it is assumed that both ways of describing a product family should be supported. The ‘family requirements’ approach since it is considered an interesting way of designing a whole family from start as if it was a product, following the concepts of the original AD framework. The ‘union of alternatives’ approach is supported since it follows the principle of focusing on one specific requirement at a time, viewing the solutions as alternatives, and is useful when adding variants to an existing model, originally developed for one product variant.

6.2.2 Relating a variant to specific requirements

Apart from the generic solution which encompasses all variants, the way in which a specific variant realizes the needs of one specific customer must be
described. Following the principle that the product family defines the union of all required solutions, it is not possible to add independent functional requirements to this union, since the union then in fact did not define the whole family. Rather, this union is constrained by the individual needs.

Customer Needs may directly be interpreted as constraints (on performance, weight etc.) or concern a variation of an existing function, alternative functions on a ‘low level’ in the solution tree. Relating CNs to FRs can be done either by interpreting CNs to constraints on the high level, which are refined into FRs on a lower level, or by referring CNs directly to the low level FRs. A choice is made here to follow the principle of only relating CNs to the highest level requirements of a common solution, and then ‘take care’ of the requirement refinements and solution mappings in the product definition, thereby allowing a higher degree of freedom concerning the way CNs are interpreted - ‘decoupling’ the CNs from the product definition model.

Figure 60 Interpreting Customer Needs into Functional Requirements and constraints
From here on, the union of alternatives approach of representing the product family is in focus, since it requires some additional concepts for expressing alternatives.
6.2.3 Alternative mappings

Alternative FRs are treated in the design of large systems [Suh 1995b]. A large system has characteristic of ‘having to satisfy varying sets of FRs over its lifetime, some of which are not known a priori’. Thus an ‘open’ product family, which can adapt to FRs which are not stated explicitly in the model, could be considered a ‘large system’. Whether a product family is ‘open’ or not, it needs to satisfy varying sets of FRs, where each FR may have a set of alternative DPs. One main difference between one system that varies and a product family, is that the FRs of a family vary in space (different product variants) rather than in time.

![Diagram of product family design](image)

*Figure 62: Domains of product family design*

During initial design, customer needs, product ideas and technological capabilities are elaborated to create a definition of the product [Ullman 1992]. The customer domain is then allowed to be open ended in order not to delimit the ability to identify new needs and imagine new ideas. But, during the order phase, as much as possible of the customer domain should be systematized to provide an efficient order processing. Decomposition of customer need is not treated in AD, and generally, requirement management, i.e. the process of creating, disseminating, maintaining and verifying requirements [Fiksel&Hayes-Roth 1993], is an area that has not been addressed well in research. One interesting approach, though, is the FR pattern creation and reuse [Tseng&Jiao 1998]. In that approach, based on customer needs, a decomposition hierarchy
of functional requirements is created, serving as a basis for the further design. In the framework presented here, such FR pattern would more or less correspond to a CN decomposition hierarchy plus the interpretation of customer needs into functional requirements.

The function-means trees for two different products which share most of the main functionality and components could be represented as one common structure with some varying nodes:

![Diagram of function-means trees](image)

**Figure 63 Common main structure**

**Create and constrain.** The differences between alternative variants of the family lies in the different or additional requirements or constraints. An FR might be mapped to many different DPs out of which most of them are ruled out due to constraints concerning e.g. how they adapt to the surrounding system or how much it is estimated to cost to manufacture the solution. Thus the FR is initially mapped to a larger solution space which is delimited by constraints (דס). If two products vary in terms of the constraints on the solution, e.g. an electric motor for 110V or 220V, these two products could be viewed as meeting the same functional requirements on transforming energy, but being delimited by different constraints on the voltage.
Figure 64 The process of generating and discarding solutions

Once a solution is selected, this will constrain the realization of solutions further down in the hierarchy. The selection of a component with one type of interface will for example constrain all connected components to match that interface.

**Same requirements - different realizations.** Variation between two products can thus be described as a variation of design parameters due to a variation of functional requirements or constraints. If not all constraints are made explicit, two individuals of a family could include different realizations of the same explicit requirements and constraints, and a selection may be made based on optimization concerning e.g. time to delivery or cost, constraints which were not initially stated. This reasoning assumes that all possible needs that a product might meet are considered as requirements (FRs and constraints), and specifically that the ability to appeal to human senses is a need which can be expressed in terms of measurable functional requirements and constraints. A product that satisfies a requirement posed by no one or nothing is considered redundant and could in theory be exchanged by a product which satisfies all
but this requirement. If this alternate product could be realized without changing anything else, this product would contain less and thus, according to Axiom 2, be preferred. But, there might be other reasons for redundancy, such as the manufacturing and logistic benefits of choosing a standard component although it has excessive performance. Another reason for redundancy is when it adds to flexibility by satisfying requirements not stated in the current version, but which are likely to become important in future versions. This reasoning and balancing of redundancy and leanness is essential in product family design.

6.3 Designing product families in AD

6.3.1 General principles related to product family design

The aim of product family planning of meeting the "right" customer requirements with a minimum variety between the product variants is a direct implementation of the axioms of design: the independence axiom for achieving independent modules, and the information axiom for achieving high commonality.

De-coupling: the basis for modularity: Striving for low coupling between FRs is an axiom of all design efforts. In product family design, achieving functionally decoupled solutions is a first step towards a "strict" physically and functionally modular architecture [Ulrich 1995]. In such architecture the functional and physical component structures are identical since each physical module realizes exactly one function. In Axiomatic Design, a decoupled solution, uncoupled in all mappings FR-DP-PV, may still not be physically modular. Though functionally modular, many DPs may be aggregated into one physical module. Even if the product family is not modular, the achievement of decoupled solutions is of particular interest since few couplings means simpler interfaces causing a smaller impact of a change.

Low information: a principle for commonality: Some of the corresponding corollaries, or design rules, are: Corollary 2: Minimize the number of FRs and constraints, Corollary 4: Use standardized or interchangeable parts if the use of these parts is consistent with the FRs and constraints, Corollary 7: Seek an uncoupled design that requires less information than coupled designs in satisfying a set of FRs.

That is, for a single product, the principles are to 1) design with as few functional requirements as possible (don’t do anything unnecessary, it only complicates things), 2) to keep the functional requirements as uncoupled as possible
in the solution, and 3) to choose the simplest of two satisfying solutions. For a set of products, this can be interpreted as: 1) identify the properties of the product that are important to the market to vary and only vary those, 2) find solutions (modules) that meet the FRs independently of each other since they then are easier to interchange 3) reuse as much as possible between two variants.

### 6.3.2 Design of a product family

If a product family is designed from scratch, the process may simply be described as that of defining DPs in one step, and then in a consecutive step to integrate DPs into components, properties and their interconnections. All this under the constraints of, among others, the realizing processes (PVs). The design process described in Chapter 2.5 will here be rephrased to reflect the AD concepts.

#### 6.3.2.1 Clarification of task (specification phase)

Identification of markets and customer needs, competition, and company resources. Establishment of FRs and Cs.

**Platform/family specific activities:**

1. Identify the desired variety in customer needs in various market niches
2. Structure CNs into alternative clusters, identify CNs which vary between clusters, ∈

![Figure 65: Clusters of customer needs](image)

*Figure 65: Clusters of customer needs*
3. Establish FRs and constraints for the whole family based on common and varying customer needs (זס). Identify which FRs and Cs to vary to achieve the desired variation in CNs. Identify whether to express varying requirements as alternative FRs or constraints.

Figure 66 Varying requirements on same base function expressed as constraints

6.3.2.2 Conceptual design:

**General activities:** Establishment of independent solutions - mappings of FRs to DPs under delimitation of constraints. Analysis of couplings in DM and de-coupling if necessary. Refinement of constraints into sub FRs or sub constraints on lower level realizations.

**Platform/family specific activities:**

1. Establish generic FR-DP-FR mappings for the whole family (זס) based on common customer needs. Establish alternative mappings based on varying FRs and constraints.

2. If product family is designed from start - select common interfaces as principle solutions.

3. Analyze couplings and de-couple.

4. Refine alternative constraints into alternative DPs and FRs (טס and ע).
Assume that a ‘Stereo equipment’ is chosen as a DP to the main audio FR (၀၂၇၀), and sub FRs are established. The task is now to identify how to refine the alternative constraints concerning functionality, cost and power. The choice of whether to refine a constraint into sub constraints or FRs may be difficult to make, but concerns deciding if it is possible to state an independent FR or not. Assume that it is decided that the main function that contributes to the power is the one that transforms the signal, then the power constraint mainly will impact the selection of a solution to this FR.

In further decomposition, the constraint may be refined into delimiting the selection of sub DPs (၀၂၇၁).
If a constraint concerns functionality, it is rather refined into FRs lower down in the FR hierarchy (2).

Figure 69 Refining constraints into sub constraints

Figure 70 Refining constraints into sub FRs:
Two varying constraints in the same diagram:

**Constraint**
- 'cost < 20'
- 'sound effect >= 100W'

**FR**
- 'transform signal'
- 'transform low frequencies'
- 'transform signal'
- 'transform low/med/high frequencies'

**DP**
- 'base horn'
- 'separate transformation of low frequencies'
- 'transform all frequencies together'

Figure 71 Different solutions for varying customer needs
6.3.2.3 Embodiment

**General activities:** Integration of DPs into components, avoiding physical integration of solutions which should be functionally separated, and considering constraints concerning physical integration emanating from e.g. manufacturing. Conceptual design of interfaces - determine connections and general type of interface.

**Platform/family specific activities:** In product family design, apart from considering use and process constraints, strategic drivers for separation are considered as described in Chapter 6.1.

In the audio case, the two sub DPs ‘signal membrane’ and ‘base horn’ result from alternative choices due to requirements concerning power and cost which vary between customers. According to the separation driver ‘varying specification’, they should then not be integrated. But, the high level DPs ‘transform all frequencies together’ and ‘transform low and med/high frequencies separately’ could have embodiments based on the same component, ‘mid/high frequency element’ (if not the same physical item). This use of the same component does not cause a coupling since these two DPs are solutions to FRs which are separate in space (different product variants). So as solutions to the varying requirements, we now have two loudspeaker components: 1) the ‘mid/high frequency element’, and 2) the ‘multi-element loudspeaker’, containing the ‘mid/high frequency element’.

![Diagram of Embodiments of DPs into components](image)

*Figure 72 Embodiments of DPs into components*
Designing physical modules: The selection of two DPs to address each of the constraints corresponds to the selection of a set of solutions which meet the same main FR, while the members of the set meet different constraints. A module with variants is a set of solutions which meet the same main function, have the same interface, but which may vary in the exact implementation. To support the design of modules, the set of solutions which meet a function is made explicit in a generic DP as in \( \text{בע} \). Specializations of the speaker principle are speakers which ‘transform all frequencies together’ and those that ‘transform low and medium/high frequencies separately’.

Figure 73 Selecting a set of solutions

For this speaker principle class to be a physical module as well as a functional, the two alternative modules in the class need to follow the same physical interface. Such a requirement is a interface constraint impacting the solutions that realize the different FRs (\( \text{בע} \)).
6.3.2.4  **Detail Design**

Components are designed in detail. Interfaces, derived properties and resulting property values are determined based on the conceptual specification. Models of components’ behavioral or geometric characteristics and other properties are developed and analyzed, leading to possible refinement of the architecture. A broad meaning of the concept ‘model’ is intended: any representation of an object that answers a question, e.g. a mental model as well as a physical prototype is considered a model.
6.3.3 Reuse of product platform models

Throughout design, an existing platform solution - a DP principle, interface specification or physical component, may be selected as a solution. Design based on an existing platform of components may be divided into three cases:

1. Design initially constrained to use existing component to meet certain function
2. Design initially constrained to conform to specified interface when meeting a certain function
3. General preferred reuse of platform components

In the first case, there is an explicit requirement stating that certain functional modules should be used. Such design based on a component is expressed as an input feature constraint (see *) on the solution. This initial feature constraint thus states that a certain solution should be used, which is a parallel with the case when an end customer specifies the use of a component from a specific supplier.

If, on the other hand, the focus is on preserving physical interfaces but the exact realization of the component may differ as long as it has the same function and interface, a input interface constraint expresses this requirement. Resource constraints may also be specified in this way (\( \forall \)).

Figure 75 Reuse of component
Concerning reuse in general, models of functional requirements and principle technologies as well as physical components could be reused.

The same FR is normally satisfied by a number of physical solutions, and one physical solution has many functional aspects. The selection of an existing DP could be aided by use of a library of DPs, with DPs described in terms of their functional properties (the FRs that they meet, see \( \forall \pi \)). If this description was
computer represented and could be mapped to the way the desired FR was described, the search for a matching DP could be automated [Suh 1995].

Figure 78 Finding and choosing a solution corresponding to an FR

When reusing an existing physical component (or the description of it), there are several aspects to bear in mind:

1. it may realize or affect other functional requirements. Either this is a desired feature, facilitating the following design work, or it creates a redundancy, or coupling, with already made solutions.

2. it may introduce constraints on following solutions which may be unnecessarily limiting, leading the designer into selecting a less optimal solution. The reuse of specific solutions to an FR in a new design may also institutionalize old and noncompetitive designs [Tseng & Jiao 1998].

The selection of an existing component may be modeled as an embodiment of a DP which introduces constraints and into which other DPs in the hierarchy may be embodied.

Since the main goal of this research is to identify how to model product families, devising guidelines for the design and reuse of components is not treated further at this point.
6.3.4 Order system preparation - relating a configurable model to the AD description

Based on a product concept developed during design, modeled in terms of the domains and interrelations, a model of the product family, adapted for configuration and/or variant management in the order phase, can be built. The configuration task - determining the structure, connections and values of a product variant based on a generic model and a set of requirements, comprises of two tasks which are often separated:

I. Mapping from general requirements to the characteristics and basic system structure of a variant. Based on a model of the product characteristics and a specific value of one characteristic, the values of other characteristics are delimited. Information needs:

A. Model of product properties and their possible values
   FRs and constraints pose requirements on properties, and DPs are solutions which capture properties or relations between properties. Thus property information is partly inherent in the FRs and constraints as well as in the DPs.

B. Interrelations between properties, and relations between product properties and basic system structure.
   DPs which represent property relations directly captures the interrelations. FR-DP mappings and Constraint - DP impact relations provide property-system relations, property interdependencies, and the way in which constraints affect the solution.

II. Mapping from the basic system structure, and its desired characteristics, to detailed parts structure

A. Model of generic component structure with component alternatives:
   Physical Component structure (the component aggregation relations)

B. Rules for delimiting possible combinations of alternatives:
   Functional component structure (if not explicit, created by following the FR-DP-Component path). Local constraints between components and global constraints on product family properties.

C. Rules for selecting component alternative depending on de-
sired characteristic:

Constraint - DP impact relation and the following DP-
Component functional structure

6.4 Conclusion

It has been shown how to represent the process of designing a product family which realizes alternative customer requirements by using extended concepts of Axiomatic Design. It was also described how the information acquired during this process could be used as a basis for building configuration models.

6.4.1 Summary of product family description

The main information described in the AD product family model is:

Table 15 Configuration information in AD model

<table>
<thead>
<tr>
<th>Information category</th>
<th>Representation in the AD Product Family Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Alternatives</td>
<td>Models of components with related properties</td>
</tr>
<tr>
<td>Delimiting Relations</td>
<td>Functional and physical component structures and constraints.</td>
</tr>
<tr>
<td>Selection Rules</td>
<td>Interpretation of customer needs into main functional requirements and constraints.</td>
</tr>
</tbody>
</table>

Together with the delimiting relations and component alternatives, which in effect defines the product family, the interpretation of customer needs as requirements and constraints defines the path from customer needs to specific components and property values.

In some cases, a solution is not fully defined based on these mappings, but there is room left for an exact configuration. Such an ‘open’ solution may be delimited by just considering it finished and defining exactly how it is realized by parts, or by constraining it further using e.g. interface or resource constraints. In the latter case, a search or optimization mechanism is required to generate an unambiguous configuration based on these constraints and other design domain knowledge in the AD model.

6.4.1.1 Components

DPs represent a parameter of a physical solution such as a component, property or interface where components may refer to abstract systems which realize
a certain technology, as well as to concrete parts. The approach taken here is to separate the representation of design parameters from the representation of how they are embodied, and models of those entities will be presented in Chapter 6.

6.4.1.2 Delimiting relations

The trace of decisions, the relations between FRs and DPs (which are visualized in a Design Matrix) together with the constraints, result in a functional component structure. This structure shows which components and property values to select to realize the intended function, and the physical structure, shows how to compose these components into assemblies. These structures models the realization of the whole class of customer needs, the recipe of the product family, which in effect delimits the possibility of selecting values and combining components.

6.4.1.3 Selection rules

The interpretation of a class of customer needs into a main functional requirement, and of specific customer needs into constraints on the realization of this function, constitute the selection rules.

The clustering of CNs and mapping from CNs to FRs and Constraints corresponds to a market based segmentation of the total product family - each subgroup of customer needs delimit the range of the product family. Clusters guide how to combine the varying FRs and Cs for specific customer groups or whole market segments. CNs may be annotated as ‘defaults’ to decrease the amount of specifications required by the customer.

6.4.2 Additions to the original AD framework

The original framework defines the existence of the following objects: Customer Needs, Functional Requirements, Design Parameters, CN to FR or Constraint interpretation, FR-DP mappings, DP-FR mappings, DP-FR dependency causing FR-FR coupling, DP-Constraint introduction, Constraint delimiting possible DPs. Apart from this the GIP models:

I. Explicit decomposition information as described by [Tate 1999]:

   Constraint refinement, DP embodiment

II. Additions in GIP:

6.4.3 Delimitation

While reasoning concerning dynamic behavior may well be part of the design process itself, no representation of this is included in the model, since models of dynamic behavior is not normally part of a configuration model.

As mentioned previously, by formulating generic functional requirements for the whole product family, a higher degree of commonality is acquired. This thesis, however, does not claim to describe how to best design product families using Axiomatic Design. Apart from relating the axioms to the qualities of a product family, it focuses on the task of identifying the entities, relations and rules needed for describing these product families irrespective of how well designed they are.

When deciding how to integrate DPs and aggregate components into assemblies, a trade off between the role of a component within one family and within the main family (union of sub-families) may have to be considered. It may, for example, be more beneficial to integrate a component with the architecture of a sub family A, even though the same component is an option in other sub families, if the family A is produced in much higher series. This type of reasoning, and the analysis required to evaluate the benefit of different architectures, is not covered in this thesis, but is interesting to pursue in future research.
7 EXPRESS-G INFORMATION MODEL OF GIP

This chapter covers how the product family description of Chapter 1 could be modeled in the visual information modeling language EXPRESS-G. Major contributions concern the information modeling of AD domains and interrelations, and the modeling of interfaces, components and the DP integration into components.

7.1 Modeling Axiomatic Design domains

An overview of all entity and relation models is presented in Appendix 10.1, and a short description of Express-G can be found in Appendix 10.5. Following the notation of AP214, a structure relation points to its related entities through the pointers ‘relating’ and ‘related’, describing the hierarchy between the ‘relating’ parent object and the ‘related’ child object.

Any object (market, function, technology, component, interface) could represent a core focus of the company, a characteristic captured in an optional attribute ‘strategic_role’. This attribute is useful when using the platform solution library.

7.1.1 Customer Domain

Entities: Customer Needs (CNs) constitute all needs and restrictions that a solution is supposed to satisfy, emanating from e.g. the end customer, rules and regulations, or the processes within the company. Customer Needs are represented as entities with identification number, names, a weight indicating the importance of the need, and an optional textual description of the requirement.

Relations: ‘decomposition’, ‘trade_off’. Customer Needs which belong together such as e.g. market segments or optional equipment packages, are grouped together into composite customer need entities by use of the ‘decomposition’ relation. A trade off between two CNs can be explicitly modeled by the ‘trade_off’ relation.
7.1.2 Function domain

**Entities:** Functional Requirements (FR), Constraints (Cs)

**Relations:** decomposition, refinement

**Functional requirements** are represented as entities with an optional textual description of the FR. An FR always has a name, and a value and tolerance attribute specified if needed.
To support the search for a certain (reusable) functionality, an optional attribute ‘classification’ can be used to describe the type of function that is required.

In the case of reasoning about function, a formal representation of functionality [Stone 1999] [Szykman 1999][Hubka 1988] could relate to the functional_requirement, but the formal representation of function falls out of the scope of this report. Information models for enabling the representation of some operational definitions are provided on page 150 7.2

**Constraints** (Cs). Constraints delimit the available design space. (Constraints may be feature, resource, performance or interface constraints, but this distinction is not made at this point). Constraints are represented as descriptive entities, with the ability to associate an operational definition (useable to model rules for actively relating and delimiting the space of solutions.)

**Refinement**: Constraint -> Constraint or FR refinement. Constraints on a high level are refined into constraints or FRs on lower levels.

---

**Figure 81** Definition of Functional requirement

**Figure 82** Information Model of constraint entity
In general, operational constraint rules could either be local - relating connected entities, or global - limiting a resource, and are not normally directed. Constraint rules are represented as entities with an operator and one or several operands. Constraints rules are subtypes of an abstract entity ‘operational_definition’ and will be modeled in Chapter 0.

7.1.3 Physical Domain

Entities: DPs

Relations: decomposition, specialization

Design Parameters (DPs). Design Parameters define a physical characteristic of a solution which would satisfy a Functional Requirement and the impacting constraints. A DP is thus a characteristic which might, or might not, be physically integrated with other DPs. A DP may refer to one solution variant or a class of solutions which all meet the FR and imposed constraints. The components and other extended concepts of the physical domain are modeled in chapter 7.2.

Specialization and decomposition. One DP may be a specialization of a more general DP. A structure of DPs can be modeled with the decomposition relation.

![Figure 83 Information model of design parameter with relations](image)

7.1.4 Process domain

In analog with the design domain, process parameters can be defined as a process or a property of a process. STEP AP214 provides the ability to relate physical components - items - to process descriptions, but the issue of representing processes is not addressed in this thesis.

7.1.5 Interrelations between domains

Interpretation CN -> Constraint or FR. A CN may be interpreted as a con-
Customer needs are interpreted as functional requirements and constraints, based on which a solution can be designed and evaluated. The relation ‘interpretation’ can optionally be given a value denoting the strength of the relation. Thus an information model of the weighted relations between CNs and FRs of the first house of quality of Quality Function Deployment (QFD) [Clausing 1994] is represented if desired.

**Mapping relations.** The zigzag mappings between Functional Requirements and Design Parameters are modeled as mapping_relations, which can be of type ‘realization’, ‘alternative_realization’, ‘alternative_requirement’ or ‘requirement’.

**Table 16 List of the different mapping relations**

<table>
<thead>
<tr>
<th>Mapping relation types:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Realization: FR1-&gt; DP1 mapping</td>
<td>The realization of a functional requirement by a design parameter. The ‘Realization’ mapping relation entity describes two relations: FR1 is realized by DP1, and DP1 realizes FR1.</td>
</tr>
<tr>
<td>Alternative Realization</td>
<td>An FR could be realized by different DPs due to different constraints.</td>
</tr>
<tr>
<td>Requirement</td>
<td>DP1-&gt; FR1.1 mapping</td>
</tr>
<tr>
<td>Alternative Requirement</td>
<td>A DP could require two alternative FRs (which may have different tolerance).</td>
</tr>
</tbody>
</table>
Modeling an FR as being realized by a DP with several specialized DPs, is the same as modeling an alternative_realization relation between the FR and each of the specialized DPs.

Dependency The coupling between FRs is modeled with a dependency relation between DPs and FRs (which may have an attribute indicating the strength). The coupling between two FRs can then be acquired by following the trace of relations between FRs and DPs e.g. FR1 is coupled with FR2 if there following relations exist: FR1 - realization - DP1 - dependency - FR2.
7 EXPRESS-G INFORMATION MODEL OF GIP

Figure 86 Example of mapping hierarchy

Constraint relations:

**Impact:** Constraint impact on (the selection of) a solution. Constraints are related to the DP selection that they impact.

**Introduction** DP - Constraint. A chosen DP may cause the introduction of a new constraint.

Figure 87 Constraint - DP relations
7.2 Modeling extended concepts

While the DPs and the mappings between FRs and DPs represent conceptual solutions in a decision process, components represent physical embodiments, abstract as well as concrete.

7.2.1 Components

A component is an entity which may be part of a physical structure and is defined by its properties and sub components. At some point in the design process, when all conceptual decisions are made, an abstract component turns into a concrete, realizable component. An abstract component may represent a conceptual system, or groups of components, while a concrete component represents one or a type of physically realizable parts or assemblies and needs to be available, maintained and documented in a more rigorous way.

A component can have any number of associated properties.

![Information model of component with properties](image)

**Property**. A property describes any quality of a component, be it a functional characteristic such as ‘maximum speed’, or a physical attribute such as ‘inner diameter’. A property may have one or several mutually exclusive alternative property values. A property is a kind of property_type, which defines the general aspects of the kind of property.

**Specialization, decomposition**. A component class may have many alternative
versions, and one alternative component is considered a specialization of the class of components to which it belongs. A number of components may be composed into a composite component.

![Diagram of component relations](image)

*Figure 89* Information model of structural relations of component

### 7.2.2 Operational definitions.

For the purpose of providing information needed when building configuration systems which are model based, e.g. based on models of the component properties, a separate entity, operational definition, is used to describe delimiting or defining relations between properties.

![Diagram of operational constraint rule](image)

*Figure 90* Information Model of operational constraint rule

Constraint rules are one type of operational definition and an expression is generated by forming constraint rules with constraint rules as their operands. The representation of the constraint rule depends on what type of relation it represents, described by the attribute `definition_type`. A specific representation of constraints such as equality constraints, inequalities, Boolean inclusion/exclusion, is provided, and the number of types may be extended.
In a Boolean rule, the constraint_type is ‘Boolean’ and the operator may take the values ‘and’, ‘or’ or ‘not’.

In an inequality rule, the constraint_type is ‘inequality’ and the operator may take the values ‘≥’, ‘≤’, ‘<’ or ‘>’. In an equality rule, the constraint_type is ‘equality’ and the operator is ‘=’.

A relation ‘operational_definition_association’ is used to relate an entity with this operational form. The described_element may be an FR, Constraint, DP, Component or Property, and the associated_definition is an operational_definition.
A component is described by its properties, which are of a certain type (material, weight, color etc.). Each type of property is defined in terms of the dimension it is measured in etc. A component’s properties are interrelated in some way, thereby defining the component. Such interrelations are the basis for models of the component, describing its various characteristics.

A cylinder, for example, may be described in terms of its height, diameter, and volume properties, where the cylindrical shape defines the relation between the properties. The operational model of this equation (the operator) could be modeled as an arithmetic constraint, or another program, operating on the property values (37). The information model of a definition of a cylinder is not defined within this thesis, but is just used to exemplify the idea of adding extended operational definitions to the model.

![Figure 94 Example: model of cylindrical component](image)

### 7.2.3 Interfaces

Interfaces define the characteristics of component connections, and are described for the purpose of reasoning about the system behavior or assembly.
Interfaces are also used to define the actual or desired characteristics of the interfacing ports as a manufacturing specification or as a design requirement to match connected components. An interface defines how components interact in a connection, but may in itself not be separable as a physical entity.

To capture the difference, and enable relations between, the desired interface and the actual interface, two separate entities are modeled: interface_type (0), describing a kind of interface and interface_occurrence (0), describing one specific connection. For the purpose of associating a component with a kind of interface, a component_interface_association is used.

![Figure 95 Definition of interface type entity](image)

Both interface entities are modeled with two ports and several descriptions of the interaction. Ports are differentiated as being a ‘master’ or a ‘slave’ port. An interaction is an entity which is described by a ‘type’ attribute denoting e.g. if it is an energy, geometrical or signal interface, and a description which may be used to describe detailed qualities of the interface. The difference between the interface entities is that an interface_type describes the desired characteristics of the ports (faces) using the entity port_description, while the occurrence describes the actual component properties. A port is modeled as a property of a component, a property which may not have been designed as a port, but may turn into one as an effect of the design, thus it is modeled as any other property.
Figure 96 Definition of interface entity

The port_descriptions and the properties could be related by an entity describing that they should match, a **port_property_relation**. An **interface_occurrence** interfaces components, or the ports of connected components, depending on the level of detail available/required in the model.

竺 exemplifies a simple way of modeling a geometric connection, while assador exemplifies a more detailed model of a geometric and electric connection.

Figure 97 Simple way of modeling a connection between two components
7.2.4 Embodying DPs

Embodiment: A DP may be embodied by a component, a property of a component, or an interface_type. The embodiment relation models how a DP is embodied and a set of embodiment relations implicitly describe how several DPs are integrated into one component. A DP may be embodied by a component which is part of embodying alternative DPs on a higher level. The embodiment of a DP does not always need to be explicitly modeled, since a DP may require further decomposition to be described in terms of components and their properties.
The embodiment relation has an optional attribute ‘rationale’ to enable adding information concerning the rationale, driver, for embodying the DP by that component.

**Aggregation** Apart from embodying several DPs by the same component, two components may be aggregated into a composite component as a separate activity e.g. during modularization. The resulting composition is modeled by the **aggregation** relation.
7.2.5 Configuration rules

There are two ways in which configuration rules are represented in the GIP. The main way is through the design history path as represented in the CN-FR-DP-component selection structures. The other way is through the definition of the resulting product, as represented in the operational definitions.

Configuration rules for directly selecting a component or property value based on a specification, are not represented explicitly, apart from the constraints which delimit interfaces or prescribe specific components.

7.2.6 Structures: functional, technical, and physical

Functional requirements structure

As defined in the axiomatic theory, an FR may be decomposed into several sub FRs, different depending on the DP chosen to realize it. The hierarchy of decomposed FRs is a functional hierarchy which reflects the structure of requirements to be realized by the structure of design parameters. If such a decomposition is represented explicitly, validity needs to be checked since it would introduce a redundancy with the mapping relations, (Chapter 0).

![Figure 102 FR decomposition](image)

Technical solution structure

Design Parameters are, through the mapping process, decomposed into design parameters on the next level, forming a technical solution structure. The decomposition relation is thus implicit in a FR-DP mapping hierarchy, but may also be added separately, while checking validity in analog with the FR hierarchy. The technical solutions and functional requirements hierarchies should have an identical structure according to Axiom 1. DPs are embodied into prop-
properties and components which may be aggregated into assemblies. As described in chapter 6.1.2.2, components may be structured according to how they are related to fulfill a function - functional component structure, or how they are aggregated together in an assembly - the physical component structure.

7.3 Validity rules

Apart from checking the correspondence between decomposition structures and the FR-DP mappings, rules must be added if the model should be consistent with the axiomatic design framework.

**Consistency with axiomatic design principles.**

Each FR should be mapped to one DP. No FRs may be decomposed into sub FRs without an intermediate DP. If there is a physical law relating two FRs, that law should be modeled by a DP, e.g. with a FR on the force and a constraint on preserving the mass, a DP ‘the law of force: \( F = m \times a \)’ should be modeled to relate the FR on force to an FR on acceleration.

An FR may have several alternative DP-mappings, out of which only one is valid in each individual variant. A DP could be mapped to many FRs.

**Validity when not adhering to the AD principles.**

This model could also be used even though axiomatic design is not followed. In that case, FRs may be decomposed directly, as long as there is a trace between Customer Needs and Components.

If operational definitions are modeled in the framework, these relate several properties. These properties may also embody different DPs, and then the FR couplings are in effect implicit in these models. There is thus a redundancy which has to be managed: if consistency of the product family model is required, the couplings implicit in the property definitions and the explicit dependency relations need to be matched.

7.4 Conclusion

The description of the extended axiomatic design framework is possible to model in Express-G. Together with models of interfaces, constraint rules and components, this model constitutes a product family model with integrated views of requirements, technical solutions, and components. The views are related by mappings which reflect the design decisions of selecting a solution to meet a certain requirement. Views are also related by operational definitions which delimit possible realizations.
Apart from modeling the entities discussed in Chapter 6, this chapter has introduced information models of interfaces and their relations to components. These models support an explicit reasoning concerning interfaces, such as e.g. searching for interfaces in a solution library during design, or analyzing how components are connected. They also capture port and connection information needed for model based configuration.

Following is an overview of the models in the GIP. All constructs are not visualized, and relations are not visualized with their entity boxes, but only as connecting lines. Thus, this is not an Express model in itself, only an illustration.

![Figure 103 Overview of GIP models](image)

*Figure 103 Overview of GIP models*
This chapter will describe how the GIP information model is adapted to the AP214 framework. The description of the way variety is represented in AP214, provided in chapter 4.5, is used as a reference. It is concluded that there are many overlapping concepts and some differing concepts and additions are suggested. A listing of the information models of the GIP, adapted to AP214, is found in Appendix 10.1.

8.1 Adapting the GIP to AP214

The purpose of integrating the GIP with the AP214 model is twofold. First the goal is to gain access to the concepts for representing detailed components, as well as the concepts for managing versions and effectivities. Secondly, since a standard format provides a means for supporting the exchange of information between users and applications, by adapting the GIP to a standard, the advantage of such a format is gained.

The objective at this point is to show the feasibility of an integration, the implementation of a detailed and consistent integration of entities and mechanisms is not attempted.

A product family model, with its relations between the requirements, product decompositions and properties, is used in four different ways during product family based development:

1. Development phase - Systems design, designing product family architecture and variants and acquiring information

2. Order Phase - Sales configuration with preliminary calculations to identify desired product characteristics and configuration of alternatives on a fairly
3. Order Phase - Product configuration, creating a manufacturing specification based on the product specification

4. Development and Order Phases - Change management

While the GIP primarily addresses system design aspects - designing product family architectures (1), AP214 addresses the aspect of data exchange between applications which support design. AP214 covers constructs for representing product configuration rules and their mappings to one unambiguous result (3), and constructs which support change management (4). It does not cover the models and relations required for making preliminary calculations of a product’s desired property values based on a customers needs (2), and it does not cover a representation of how a product is designed to meet a set of requirements (1).

Thus, AP214 provides the detailed constructs for representing and managing physically realizable models, while the GIP provides constructs for representing requirements, interfaces, and the design decisions in terms of mappings from alternative requirements to alternative solutions. In the following, some examples are presented concerning how an AP214 structure comprising of detailed constructs could be developed based on a GIP design.

The two models will be related along three lines, by:

1. identifying different conceptualizations and suggesting how to manage these
2. identifying similar concepts with the same information modeling entities
3. suggesting how to model concepts that do not exist in AP214

### 8.2 Principle for relating the models

In principle, the GIP concepts are used to model the decision process and the mappings from requirements to conceptual components, while the AP214 concepts model the mappings from conceptual components to detailed parts and processes.
From ‘product_component’ and on to more detailed items, the AP214 protocol is applied as is. Thus the entity ‘product_component’ embodies the interface between the two models. In essence, the GIP constructs replaces the ‘specification’ entity and its related mechanisms.

8.2.1 Functions - purpose or behavior?
Function has different meaning in GIP and AP214. Where the focus in the GIP is to describe the purpose - the functional requirement, the AP214 entity “Product_function” is used to model the functional solution. A functional solution would in the GIP framework correspond to a general DP with no physical realization, e.g. a DP ‘electrical signal transmission’ as a solution to the FR ‘transmit signal’. Still, the entity product_function in AP214 could be used to represent the entity ”functional_requirement” if the definition of the meaning of product_function was consistently changed in some context.

8.2.2 DPs and Components
In Axiomatic Design, as well as in the Theory of Technical Systems, a distinction is made between the technical principles (DPs and Organs) which realize a desired function, and the components which are the physical embodiments of the technical principle. This distinction is followed by the GIP by modeling DPs which are embodied in components (systems) or component properties. That is, a DP does not directly correspond to a component.

In AP214, a product_component is ‘an element in a conceptual product structure’, ‘a mechanism for collecting the knowledge a company has accumulated during the design of similar products’. Most Product_component objects are realized by parts, but there are exceptions such as trunk space or holes for wire harness which are not. If a space or hole is considered a physical object, the entity ‘product_component’ is thus only used to represent abstracted physical objects as opposed to technological principles. Thus, though a product_component entity could be used to model DPs which refer to abstract components (e.g. systems that realize some technical principle), a product_component in AP214 is not used to model a property or an interface.
In the GIP, no distinction is made between ‘abstract’ components, representing a conceptual system or groups of components, and ‘concrete’ components, representing physically realizable units or assemblies. Apart from varying abstraction levels, a GIP component represents many aspects of a physical embodiment. A component may be a type class of variants (nuts with dimension M6, M8, M10), as well as one (variant) type (nuts with dimension M8), as well as one instance of a component type (one nut with dimension M8). A component may also be an instance of a type class (one specific nut which may have the dimension M6 or M8).

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Technical</th>
<th>Abstract</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DP: a technical solution with alternative realizations in platform</td>
<td>Product_component with alternative decompositions in platform</td>
<td></td>
</tr>
<tr>
<td>Variant Type</td>
<td>DP: one technical solution in platform</td>
<td>One product_component, without variation, in platform</td>
<td>Item in platform</td>
</tr>
<tr>
<td>Class Instance</td>
<td>DP: a technical solution with alternative realizations in product family</td>
<td>Product_component with alternative decompositions in product</td>
<td></td>
</tr>
<tr>
<td>Variant Instance</td>
<td>DP: one technical solution in product family</td>
<td>One product_component, in product</td>
<td>item_instance and item_version in product</td>
</tr>
</tbody>
</table>

Table 17 Different types of solutions in the model

A DP is embodied by a property or by a product_component, and the entity ‘product_component’ is thus the common denominator between the two models. A product_component is a subtype of ‘complex_product’ and has in AP214 the following associated data:
Table 18 Attribute data of a product_component

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Specifies the identifier of the product_component, inherited from ‘complex_product’</td>
</tr>
<tr>
<td>version_id</td>
<td>Specifies the identification of a particular version of a product_component, optional. Inherited from ‘complex_product’</td>
</tr>
<tr>
<td>Description</td>
<td>The description specifies additional information about the product_component</td>
</tr>
<tr>
<td>instance_required</td>
<td>The instance_required specifies if the existence of a corresponding item_instance is required</td>
</tr>
<tr>
<td>is_influenced_by</td>
<td>The is_influenced_by specifies the Specification category objects that impact the design of a solution for the product_component in the context of the Product_class. When used together with a GIP, this attribute should not be used, i.e. it should come in 0 instances.</td>
</tr>
<tr>
<td>Is_relevant_for</td>
<td>Specifies the Application_context where the product_component has to be considered E.g. the type of application (assembly_study, preliminary_design etc.) and life cycle stage (design, manufacturing, recycling)</td>
</tr>
<tr>
<td>name</td>
<td>The name specifies the word or group of words by which the product_component is referred, optional.</td>
</tr>
</tbody>
</table>

Product_component in AP214 is a type of Complex_product and Product_constituent, and as such it is part of hierarchical structures relating it directly to product_functions and product_classes. A GIP component is an embodiment of a DP, and is not directly related to requirements or product classes. Thus ‘upward’ in conceptualization, apart from decomposition between product_components, the structure relationships of the AP214 product_component are not used, but replaced by the GIP ‘embodiment’ relation. ‘Downward’, relating the product_component to item_instances, the AP214 constructs of the ‘structure motor’ are used.
8.2.3 Alternative solutions

In AP214, all alternative realizations of a component are made explicit by use of the entity ‘alternative_solution’. "An alternative_solution is the identification of one of potentially many mutually exclusive implementations of a Product_function or of a Product_component." In the GIP, an explicit component alternative is modeled by a relation ‘specialization’ between the class component and its alternatives.

8.2.4 Selection and delimitation rules

A variant is in AP214 defined in terms of a common product structure + a specification of how varying characteristics are realized. Characteristics are related to their realizations by direct connections between a specified characteristic and an unambiguous solution.

The GIP relates a requirement to a detailed solution by following the designed path of CN-FR-DP decompositions and constraint refinements, thus the AP214 specification corresponds to short circuiting the design path into a feature constraint, directly constraining which physical solution to choose.

Delimitation rules are in the GIP modeled, not as ‘specification_expression’ rules, binding one solution to another, but in terms of the functional structures, stating that two components belong together since they are part of realizing the same requirement. Delimitations are also represented as constraints on components or properties. These constraints are made operational by expressing them as ‘configuration_rules’. For Boolean rules, ‘specification_expression’ is an
AP214 mechanism which provides an information model. Information models of inequalities are added, useable in model based configuration systems.

### 8.2.5 Product families

AP214 is strong in structuring the product family into subfamilies and relating the variation to these subfamilies - thus facilitating the task of managing information concerning many markets with varying needs. It is not strong in defining the semantics of the variation - the rationale for subdividing the family into subfamilies in terms of varying requirements or preconditions.

The GIP ‘customer_need’ entity corresponds to the AP214 ‘specification’ entity. As a means for utilizing the product_class and configuration constructs, the ‘product_class’ entity is used as it is, and customer_needs are associated with a product_class through a ‘class_need_association’, in analog with the ‘class_specification_association’ between a ‘specification’ and a product_class. The customer needs are decomposed using ‘decomposition’, a relation to some extent replacing the specification_inclusion. A product_class is related to the customer needs which vary between classes (with different roles for different classes).

The product_class entity is used to describe a set of similar products, i.e. a product family, and the relationships between product_classes is a way of classifying relations concerning consecutive versions of product families or to define a hierarchy of specifications. Thus product_classes may differ both in terms of the customer requirements, markets and years. They are in the GIP not related directly to product_components or processes, but only to customer_needs, which in turn are interpreted into configurable functional_requirements or constraints.

Configuration control is suggested to be slightly different from AP214. In AP214, a ‘configuration’ entity is used to describe that a component or process is not always part of a structure, but is controlled by a configuration defining the valid usage of it in the context of a product_class.

Instead, a ‘configuration’ is suggested to define the valid interpretation of a customer_need with respect to a certain product_class. That is, it is the relationship between a need and its functional_requirement/constraint, that is possible to control by use of the configuration entity. This is a way to describe that a customer need could be interpreted and realized in different ways depending on product_class.
8.3 Mappings of concepts

To adhere as close as possible to AP214, one objective is to preferably represent GIP entities as existing AP214 entities, possibly with specifying attribute values. New entities are preferably modeled as subtypes of existing ones.

The models and mappings of GIP entities to AP214 entities are presented in Appendix 10.2: ‘Generic Information Platform information models’, and the following is just a quick overview of these mappings:

**Similar concepts:** functional_requirements, design_parameters, and components are all represented in terms of existing complex_product entities. Property and property_values are represented using the property_value_representation entity.

**New entities as subtypes concepts:** Customer_need is added as a new type of complex_product. and class_need_associations and relation_configurations are modeled as subtypes of existing entities.

**New Concepts:** New models are added to model Embodiment, operational_definition, and the entities and relations used to model interfaces.
8.4 Conclusion

In this chapter, it was shown that an adaptation of the GIP to AP214 is possible, and it was suggested how this could be done in an implementation. The GIP provides new constructs to represent requirements and the mappings to conceptual solutions, while AP214 represents conceptual solutions and mappings to detailed models of physically realizable part versions, with control of the effective version and time. A natural ‘interface’ between the GIP and AP214 is the entity ‘product_component’ which represents a conceptual component in both models. Moreover, AP214 provides a means for managing information complexity concerning product families with many sub-classes, individual interpretations of the requirements and the means by which these should be realized. By making it possible to relate ‘Customer_needs’ to different product_classes with different roles (option, standard etc.) and interpretation, this AP214 expressiveness can be used as well. \( \text{\textsection} \) illustrates a summary of the implementation of the main axiomatic entities.

---

**Figure 106 Illustration of the information model of related entities**
9 CONCLUSION

To conclude, this chapter contains a short description of the result, and an evaluation of this dissertation as well as a discussion concerning the usefulness of this work, concluded by some directions for future research. The evaluation part starts with a discussion concerning the reliability of the result, followed by an argument concerning the validity of the hypothesis, and of the ability of the GIP model to answer relevant questions.

9.1 Evaluation

9.1.1 Evaluation of the results

As an attempt to evaluate the reliability and validity of the results presented in this dissertation, the five criteria from [Olesen 1992] have been used:

I. Internal logic - the results are based on known and accepted theories and there is a connection between the starting point, hypotheses and the result.

The research is based on: 1) Axiomatic Design which is a globally recognized design theory, 2) Modular Function Deployment which is a scientifically accepted and industrially used method, and 3) STEP AP214 which is an ISO standard information protocol developed by a large international community and implemented by several PDM companies.

The connection between the starting point: the need for better information management in product family based development, the hypothesis saying that required information can be acquired in design and reused during configuration, and the result: the information model of requirements, solutions and their interrelations representing design knowledge as well as the basis for configuration systems, is considered to follow an internal logic. A more detailed description of how the hypothesis is implied by the result is given in chapter 9.1.2.

II. Truth - the theoretical and practical result can be used to explain ‘real’ phenomena.

In this dissertation, one principle and contribution is that of binding
together the information needs of several areas of interest:

Design of product families and product platforms

The need for documenting modules as well as the trace between requirements, product properties and components is expressed by users of the recognized MFD. Need for connecting to detailed product data expressed by the same companies.

Configuration of customer order based products

The need for connecting configuration control to PDM systems is expressed by many companies in industry. The need for supporting the acquisition of information upon which configuration systems are based is recognized by configuration systems developers, e.g. the Tacton developers. Moreover, the problems with updating complex configuration systems is also recognized.

Information management support of product development

For the purpose of broad support of product data management and exchange of information between systems and users, the ISO standard AP214 was developed. This protocol is used in Swedish industry, but, admitted by representatives from industry, requires extensions when it comes to expressing concepts used in the early phases of product development.

Though the need for extended computer based information support is recognized in each of these areas, the vision of binding the information from conceptual design together with that of the order phase through a common information model, is not explicitly expressed. Thus this research should be viewed as visionary, or ‘technology driven’ rather than requirements driven, resulting in an innovation that has to mature and evolve to find its final form.

III. Acceptance - other researchers accept the theories used in the project and professionals use the tools based on the theory.

Within this research, several papers have been accepted and presented at international conferences and workshops [Sivard 1999] [Sivard1998] [Sivard&Barthon 1998]. More recently,
the results have been presented at seminars with researchers and industrial participants. The interest in the approach is growing as the need for and understanding of information management increases.

IV. **Applicability** - *the use of the tools allows the probability for success to increase with repeated use. It will not necessarily lead to success every time, but over a period of time give better result than if not used.*

An efficient use of information is a recognized means for increasing the probability of success in industry. By using a common information platform, a higher degree of reuse of information, as well as ease of managing the information, is gained. By allowing a way to model information during design, which is needed during the realization of a variant in the order phase, support of the development as well as maintenance of configuration systems is achieved.

V. **Novelty value** - *new solutions are presented, or new ways of looking at a problem are introduced.*

The principle of creating an information model during product family design, which is reused by the configuration systems in the order phase, is in itself new. As parts of this principle, several new solutions are presented:

- the modeling of the domains and interrelations of Axiomatic Design in an computer processible information model, as well as the extensions of the model to incorporate interfaces and alternative mappings
- the principle of expressing the semantics of configuration rules in terms of the structures and constraints of AD
- the principle of defining an integrated information platform for several phases of customer order based design based on the AP214 standard
9.1.2 Internal Logic

Hypothesis:

It is possible to define an information model of the concepts of Axiomatic Design, which captures the information of a configurable product family model.

This information model can be adapted to the ISO 10303-214 standard, thus providing an information basis for supporting several applications during the development and order phases of customer order based design.

The intuitive reasoning behind the hypothesis is, that the main concept needed for specifying a product based on an order, is the trace from requirements to solution. Further, that this trace in axiomatic design is described in terms of the interrelations between the Customer, Functional, Physical and Process domains. Thus it should be possible to define an information model of configurable product families based on the concepts of the theory of axiomatic design. A more formal reasoning is found in the following. Definitions of concepts can be found in chapter 2.

If the concepts of the theory of Axiomatic Design can be used to describe any member of a set of product individuals and it is possible to formally represent these concepts in an computer processible information model then a configurable product family model can be developed based on an information model of the concepts of Axiomatic Design further,

If the AD information model can be adapted to the ISO 10303-214 standard resulting in an extended model and fundamental information used by applications during product family development and order based design is represented in that extended model then an integrated information platform is achieved, which provides a basis for the support of some applications during the phases of product family development as well as order based design.

How the concepts of the theory of Axiomatic Design are used to specify any member of a set of product individuals: An AD product family model is described in terms of a model of the whole family - ‘a set of product individuals’ - plus a specification of the variance, ‘specifying any member of the set’. The whole family model is described by the trace from one main, generic, functional requirement down to the variety of solutions, and the variant specification is described in terms of the varying customer needs and their relations
from to specific constraints on the way the function is realized. This is de-
scribed in more detail in Chapter 1 and summarized in Chapter 6.4, page 139.

**How AD concepts are formally described in an computer processible in-
formation model:** Express-G is a language for modeling computer readable
information models by use of entities and attributes, which also represents
relations. The customer needs, functional requirements, constraints, design
parameters and their interrelations can be modeled as entities and relations in
Express-G as described in Chapter 1, page 143. Apart from modeling the origi-
nal AD concepts, components, properties and interfaces are modeled, as well
as alternative mappings, DP embodiment relations, and constraint refinement,
impact and introduction relations.

**The adaptation of the AD model to AP214** is described in Chapter 1, p. 161.

**Fundamental information used by applications during product family
development and order based design:** The applications in focus are: configu-
ration, information management and product family design. Chapter 5.5 page
105, presents how the information used in these applications are represented in
the axiomatic product family model. To further exemplify the use of the in-
formation during design, describes how a set of questions, considered im-
portant to platform design, could be answered by the model if implemented.
<table>
<thead>
<tr>
<th>Questions:</th>
<th>How to answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>What customer requirements or other prerequisites do we have, and how are these related to the product properties?</td>
<td>Mappings CN - FR or Constraint</td>
</tr>
<tr>
<td>How are the product properties related to the components (modules)?</td>
<td>FR + Cs - DP - Component (property)</td>
</tr>
<tr>
<td>What parts of the product platform can be reused in this family?</td>
<td>Matching the description in terms of FRs and constraints of the platform solutions with those of the family.</td>
</tr>
<tr>
<td>What parts of the product platform are used in this family?</td>
<td>Relation between solution type in platform, and solution instance in family</td>
</tr>
<tr>
<td>What is the range of variety measured in: Parts? Modules? Technical solutions? Functional characteristics?</td>
<td>All entities in branches with no specializations or alternative mappings are common. Else, it is suggested that the degree of variety/commonality of any type of entity may be estimated based on the percentage of alternatives at that level.</td>
</tr>
<tr>
<td>A) What customer requirements are affected by a change in this component? B) How much?</td>
<td>A) Component -DP - FR - CN</td>
</tr>
<tr>
<td>B) Not part of model</td>
<td></td>
</tr>
<tr>
<td>How does a architectural decision (e.g. introducing a variation high up or low down in the structure) impact the commonality of the family?</td>
<td>Comparison of how many functional branches differ, on what level and depth (number of FR-DP-FR mappings in branch)</td>
</tr>
<tr>
<td>If we make a component part of the platform: would its interface fit, and would it be used as a common component or variant?</td>
<td>Interfaces are modeled, and constraints on interfaces may be expressed. Common or variant may vary depending on product class, modeled as a relation between the product class and the component instance.</td>
</tr>
<tr>
<td>What kind of interfaces do we have in this product? Which modules take part in these interfaces?</td>
<td>Interface types are modeled and identified and components can be related to their type of interface and be stored in, and retrieved from, the platform solution library based on the interface.</td>
</tr>
</tbody>
</table>
Thus, the product family information model, together with AP214, captures information considered important to access during the design of a product family as well as the realization of a variant, but how this information is implemented and made operational is up to the interpreter.

9.2 Discussion

Criticism could always be raised towards the research presented in this dissertation concerning the suggested principles and details of capturing design information.

One of the main objectives has been to identify a ‘generic’ product family model - a model of the semantics of a product family with as few application-adapted concepts as possible. Axiomatic Design provides a transparent way of relating requirements to solutions, and it is considered a process which yields good designs. The problem is that most designers do not use Axiomatic Design when designing their products. Still it may be possible to document the decision process by using the zigzag mapping relations. If this is done after the solution is designed, it has been found easier in practice [Erixon 1999] to make this mapping in reverse, i.e. relating the already designed components to the requirements that they were part of realizing. Even though the principle of modeling the mappings between domains is not operational today, it should be seen as part of identifying how to define products in a way that captures their essential meaning in terms of the relations between the needs they meet and the solutions that they encompass. It is strongly believed that such a model will be necessary for the efficient management of product information in future computer systems.

Product development and order based configuration are different activities, using different concepts while reasoning about essentially the same thing: turning needs into means - requirements into products. In this work, the main goal has been to identify and suggest some principles for tying the different concepts together, as a means for a better utilization of information during product family based design (development and order). As a means for identifying the principles, a description of

1. how product families can be described based on Axiomatic Design
2. and modeled in Express-G
3. and adapted to the AP214 standard

was provided. The task of defining this model in detail is large, and the result presented herein should rather be seen as a rough map which could serve as a
starting point when refining the details. In each of the areas there is room for improvements.

### 9.3 Recommendations for future work

A future GIP is envisioned to support the representation of models of the processes, and realizing resources, as a means for further support the design of an efficient product family. These models could then be utilized for adapting the product family design to the manufacturing system, as well as adapting the manufacturing system to the product.

Based on the GIP suggested, there are several areas to refine:

I. The task of designing product families in Axiomatic Design could be covered with more depth and rigor. For example, one issue concerns whether DPs concerning e.g. quality should be represented as solutions to FRs of different tolerance, or as alternative solutions to the same FR, selected by different constraints. Information models need to be adapted to cover possible new AD constructs. Another issue concerns whether requirements on alternative property values should be represented as several alternative FRs or as a generic requirement to ‘provide alternative values’.

II. Extend the process to cover the mappings between Design Parameters and Product Variables, and include the ability to express information concerning the flexibility of the manufacturing processes and the way these constrain the product architecture.

III. Extend the capabilities of expressing and managing customer needs and other requirements.

IV. Define how to make information models of all types of required constraint rules (e.g. equations).

V. Define in more detail how to relate the property models of model based configuration to the FRs, constraints and component properties.

VI. Configuration - A hypothesis that was not tested is, that if a decoupled product is designed and modeled, the modeled FR-DP couplings could be used to make the order in which choices are made, avoiding the need for backtracking and facilitating configuration. E.g. If an uncoupled solution is designed, each alternative choice can be made independently of the other choices. If the design is decoupled, choices can be made according to the order of a sorted diagonal design matrix,
without the need of reiteration. And if the solution is coupled, reiteration is required in order to achieve a valid configuration. But if the solution is coupled, it should be redesigned.

VII. Exchange - The GIP captures the decisions made during the design process, information that may not be desired to exchange for strategic reasons. Aspects concerning what information to exchange and what not to exchange and how to meet different requirements concerning these needs to be investigated.

VIII. When deciding how to integrate DPs and aggregate components into assemblies, a trade off between the role of a component within one family and within the main family (union of sub- families) may have to be considered. It may, for example, be more beneficial to integrate a component with the architecture of a sub family A, even though the same component is an option in other sub families, if the family A is produced in much higher series. This type of reasoning, and the analysis required to evaluate the benefit of different architectures, would be interesting to pursue in future research.

Apart from theoretical work, one important task is to implement the GIP and apply it in a real design situation to get further indications of how to evolve and refine the models.
REFERENCES

[Acclaro 2000] Personal communication with representative for AcclaroTM Software, a software system supporting Axio-

matic Design.


[Atlas Copco 2000] Personal communication with representatives from industry


[MMAB 2000] Personal communication with Modular Management AB, consultants on the MFD method.


REFERENCES


[Sivard 1990] Sivard G., "Managing different types of design requirements - design is to juggle constraints", Tech Report, Royal Institute of Technology, 1990

REFERENCES


Svendsen K-H; Thorp Hansen C; "Decomposition of mechanical system and breakdown of specifications"; In: ICED’93, 1993.

R.B. Stake; "Using cluster analysis to support the generation of modular concepts in the MFD-method", International CIRP Manufacturing Systems Seminar, Stockholm Sweden, June 2000


http://www.tacton.se and conversation with Tacton creator.


J. Tiihanen et al., "State-of-the-practice in product configuration - a survey of 10 cases in the Finnish industry”


E. Tjalve; "A short course in industrial design”, Newnes-Butterworth, London 1979
REFERENCES


10 APPENDIX

10.1 Definitions

This chapter summarizes how some central terms are defined and used within this thesis. Partly, the purpose is to avoid misunderstanding since words such as ‘property’, ‘characteristic’, and ‘product platform’ are used in many different ways in literature as well as in industry. Even more important is, that since different usage of these terms often reflect different views of their meaning, it is of specific importance to define their meaning in this context. Especially the research field of describing product families and product platforms is relatively young, and there are many definitions of these two concepts in the scientific domain as well as in industry. It should be noted that this is by no means an exhaustive list, but the selection is made based on what seemed necessary to define during the thesis work.

References to ‘AP214’ are meant as references to [AP214 1998].

An architecture is ”The scheme by which the functional elements of the product are arranged into physical chunks and by which the chunks interact” [Ulrich&Eppinger 1995]. An architecture is typically characterized as being modular or integrated and, as mentioned, describe the structure and sub components and connections of a system.

A characteristic is ”a distinguishing trait, quality, or property”, an aspect of a component that distinguishes it from other components. Thus a characteristic is a "property of a property" which indicates whether it is a distinguishing property or just any property. Characteristics which varies between components are typically used for selecting one component for another. In AP214, such characteristics of a product class are called specification.

A component refers to an object, as opposed to a property, a process or principle. A component may be indivisible or composed by two or several interrelated components - sub components. A component entity may describe one single component or an abstraction or class of components.

A component class is a component with several possible property values. A parametric component is a kind of component class in which one property value may vary, causing other dependent properties to change. Typically, ver-
sions (component variants) of a component class are realized by more involved changes: e.g. by combining varying sub components differently, or by requiring different realizing (manufacturing) processes.

A **configurable product family model** is a model which defines how any member of a family can be specified.

A **configuration** as a noun, denotes the relative arrangement of components of a chosen product variant (of a product family). As a verb, *configuration* denotes the process of generating one product configuration based on an individual specification and a configuration model. In AP214, a ‘configuration’ is an element which associates a specified characteristic of a variant with a specific component or process, in order to define a valid usage of this component or process in the context of a certain product type.

A **configuration model** is a computer interpretable model which can be used for configuring any product variant based on an individual specification at order. In some other definitions, ”configuration model”, implies the configuration of fixed components in a predetermined structure [Tiihonen et al 1998].

A **core** entity is loosely defined as an entity which is of strategic interest to the company.

A **feature** is a ‘prominent part or characteristic’ and ‘relating to structure or form’ [Merriam-Webster 1997]. A feature is in this thesis considered as one of many properties of a component.

A **functional property** describes any functional aspect of a component. Aspects such as fuel consumption or sound quality, sometimes referred to as characteristics, are considered functional properties, as well as functionality such as ‘transmit rotary motion’.

A **generic information platform** is an information model of the types of ideas, facts and processes of an information platform, based upon which individual information platforms can be built.

**Information** is ”knowledge of facts, processes or ideas” [Schenck 1994], or data put in context.

An **information model** is ”a formal description of types of ideas, facts and processes.” [Schenck 1994]. (If an information model is written in a computer sensible representation, it has the additional quality of being computer processible.)”, [Ross 1977] divides the modeling of a subject into the modeling of data and activities. Data refers to ”the things” and activities to ”the happenings” of the subject being modeled. Thus according to Ross, an information
model = data model + activities model. According to Schenck, a data model is
normally geared towards a specific system while an information model reflects
the information irrespective of how the model is going to be implemented in a
system. Thus, an information model is a specification of the data model which
is a realization of the information model according to specific interpretation
rules, technology or system.

An information model is a formal description of types of ideas, facts and
processes, which if it is written in a computer sensible representation, has the
quality of being computer processible

An information platform in general contains information based on which
many applications can be developed. In this context, an information platform
constitutes the information part of a product platform.

An item or part is a type of physically realizable objects, item version in
AP214. It differs from a component in that it is described in enough detail to
be realized.

A model is description of something else, often simplified in some sense. The
model has a purpose of answering questions concerning certain aspects (ap-
pearance, behavior, decomposition etc.) of the entity which is being modeled
[Kjellberg 1992]. A model should preferably have a defined purpose, view-
point and detailing level [Ross 1977].

A ‘pure’ module is a component which realizes one major function and of
which all variants adhere to the same physical interface. ”A module is an es-
sential and self-contained functional unit relative to the product of which is
part. The module has, relative to a system definition, standardized interfaces
and interactions that allow composition of products by combination”
[Miller&Elgaard 1999].

A ‘strategic’ module is not necessarily functionally ‘pure’, and encompass a
trade off between various reasons for establishing it as a separate unit. Com-
mon to both descriptions is the focus on robust interfaces, functional as well as
physical, which facilitates (de-) connecting the modules. As the strategic mod-
ule definition takes more aspects into account, this is the definition that seems
most useful in practice.

A Modular Product Family is a product family based on physical modules

A Modular Product Platform is a product platform based on modules.

A platform solution library a set of reusable solutions, similar to the modular
system (Baukasten in German) which is the total set of modules creating the
product variety [Pahl&Beitz 1988]. One main difference between a modular system and a solution library is that the solutions upon which a Product Family is based might be other than modules. Components/modules can be members of several Product Family specific Solution Libraries.

A principle is defined as ‘a comprehensive and fundamental law’, ‘a rule or code of conduct’ ‘an ingredient that exhibits or imparts a characteristic quality’ ‘the laws and facts of nature underlying the working of an artificial device’ [Merriam-Webster 1997]. The description ‘principle solution’ is often used to mean a ‘conceptual solution’ e.g. not detailed, but describing ‘an abstract or generic idea generalized from particular instances’ [Merriam-Webster 1997]. In this thesis the term principle often refers to the less formal definitions of being an ‘ingredient...that imparts a characteristic...’, or ‘a fact... underlying the working...’ or ‘an abstract idea’.

A product family based product development means product development which is based on a product concept which encompass variety. This type of development is not necessarily customer order based, but also include products which are planned to offer a ‘static’ variety to the market.

A product family is a set of products with the main function and many characteristics in common. This set may be infinite, and define members which may not exist or which never will be physically realized. The concept of a product family is thus used to refer to all possible variants, not only to the physically existing products. Some differing definitions in research: [Erens 1996]: ”A Product Family is a product concept that is designed for a market but caters for the individual wishes of customers by introducing a variety within a defined product architecture and within a defined manufacturing process.” [Tiihonen et al 1998]: "A configurable product can be considered a product family since in effect it defines a set of variants.” [Jiao 1998]: "A product family is defined as a set of products that share common technology and address a related set of market applications (Meyer et al 1997)”. In AP214, the entity product_class is used to model the same concept. "Product_class is the identification of a set of similar products to be offered to the market.”

A product family model is a part of the information platform, which captures the definition of a family of products. This model contains a definition of the whole product family, as well as of how any member of the family can be specified. When the configuration of variants is in focus, it may also be called a configurable product family model.

A product is defined as a component which is offered to a customer other than the surrounding system. This definition differs slightly from AP214, where
‘product’ is not defined, but a ‘Product class is the identification of a set of similar products to be offered to the market’, in that the customers could be internal to the company. In general, a product may be a service or a process, e.g. anything which is offered to the market, but in this context, a product is a physical entity of some kind - including e.g. electronic, software, or mechanical components. The distinction between products and components may be hard to make since what is sold as a product by one supplier may become a component of a subsystem of another company. Normally components and products are documented in different ways, due to different needs for information.

A **product platform** contains ‘things’ (physical components, processes, information) based on which many products can be developed. A product platform may be segmented into a set of things which are common to all products based on the platform, and a (segmented) set of those that are used only by some products. Some other definitions: "The product platform is a set of subsystems and interfaces that form a common structure from which a stream of derivative products can be efficiently developed and produced” [Meyer&Lehnerd 1997]. [Erens 1996]: An architectural concept comprising interface definitions and key-components, addressing a market and being a base for deriving different product families. [Jiao 1998]: "generations of underlying product architectures” - [Wheelwright&Clark 1992], "The technological foundation of product families (Meyer 1997), "a platform is the physical implementation of a technical design that serves as the base architecture for a series of derivative products. The platform also embraces manufacturing technologies and processes employed in production.”

**Product Platform Management** concerns the tasks of planning, developing and exploiting the core systems and processes upon which a company can realize various products for various markets. Since the tasks of planning the product platform and the product families are related, this term is sometimes used to denote the management of product platforms as well as of product families.

A **product variant** is an instance of a product class/family.

A **property** can describe any aspect of a component. Properties may be physical or functional.

A **structure** is a description of the organization of an entity into sub entities, which in turn may be composed by sub entities. Depending on the view of the product, entities may be of functional, logical or physical art and the structures correspondingly describe the product’s functional, logical or physical decomposition [Hubka 1988]. The term ‘structure’ is used to refer to a structure of
entities, with the entities included, or to the ‘bare’ structure, without entities. In this thesis it is mostly the ‘bare’ structure that is referenced.

A system is defined in the dictionary as "a regularly interacting or interdependent group of items forming a unified whole" and a component as "a constituent part" of something [Merriam-Webster 1997]. In the Theory of Technical Systems [Hubka 1988], a system designates "all technical means (Heidegger) by which human beings achieves its ends", that is, one indivisible component may be a system. In this report, systems are considered as interacting components which often are physically distributed e.g. a stereo system or power train of a car. Systems are not explicitly modeled, but abstract systems and components are all designated as Components.
10.2 Generic Information Platform information models

In the following, the information models of the GIP, as adapted to AP214, will be described. New entities are not fully explained here, but more detailed descriptions can be found in Chapter 6. Figure 107 provides an overview of the models in the AP214-adapted GIP. For clarity in this figure, all constructs are not visualized, and relations are not visualized with their entity boxes, but only as connecting lines. Thus, this is not an Express model in itself, only an illustration.

Figure 107 Overview of GIP, adapted to AP214
10.2.1 structure_relationship

This slightly modified AP214 modeling construct represent most of the relations between the entities in the axiomatic domains of design. Most domain entities are also defined in this model.

![Diagram showing relationships between entities]

Table 20 The relations modeled by the structure motor

<table>
<thead>
<tr>
<th>GIP Relations</th>
<th>AP214 Relations</th>
<th>Description</th>
<th>Validity rules in Axiomatic Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>realization</td>
<td>product_structure_relationship</td>
<td>Values of relationship attribute ‘relation_type’</td>
<td>only FR-DP</td>
</tr>
<tr>
<td>requirement</td>
<td></td>
<td>realization exists</td>
<td>only DP-FR</td>
</tr>
<tr>
<td>specialization</td>
<td></td>
<td>functionality exists</td>
<td></td>
</tr>
<tr>
<td>aggregation</td>
<td></td>
<td>specialization exists</td>
<td></td>
</tr>
<tr>
<td>alternative_realization</td>
<td>alternative_realization new</td>
<td>aggregation new</td>
<td>only component - component</td>
</tr>
<tr>
<td>alternative_requirement</td>
<td>alternative_functionality new</td>
<td>dependency new</td>
<td>only FR-DP</td>
</tr>
<tr>
<td>dependency</td>
<td></td>
<td></td>
<td>only DP-FR</td>
</tr>
</tbody>
</table>
Table 21 The entities modeled by the structure motor:

<table>
<thead>
<tr>
<th>GIP entity</th>
<th>AP214 entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_need</td>
<td>New</td>
<td>subtype of complex_product</td>
</tr>
<tr>
<td>functional_requirement</td>
<td>product_function</td>
<td>A product_function with the ‘is_relevant’ attribute associated with the entity ‘application_context’ with the attribute ‘application_domain’ value ‘Axiomatic Design’</td>
</tr>
<tr>
<td>design_parameter</td>
<td>product_component</td>
<td>A product_component with the ‘description’ attribute value ‘DP’ and the ‘is_relevant’ attribute associated with the entity ‘application_context’ with the attribute ‘application_domain’ value ‘Axiomatic Design’</td>
</tr>
<tr>
<td>Component</td>
<td>product_component</td>
<td>A product_component with the ‘description’ attribute value ‘component’. This specification is needed since a component may not be part of the same types of structures as a product_component. The ‘is_relevant’ attribute is associated with the entity ‘application_context’ with the attribute ‘application_domain’ value ‘Axiomatic Design’</td>
</tr>
</tbody>
</table>

embodiment:

The ‘Embodiment’ relation is modeled separately from the structure relations.

A product_component which represents a DP may be embodied by another product_component, representing a component, and interface_type, or a property_value_representation.
10.2.2 Properties

The AP214 ‘item_property_association’ relation is a specialization of the relation class ‘property_value_association’ which is used for relating either items or processes with their properties. An property_value_association can be given a certain validity_context to differentiate e.g. the cost property of a component depending on supplier. Thus, properties and attributes are modeled in the following manner:

<table>
<thead>
<tr>
<th>GIP entity</th>
<th>AP214 entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>property_value_representation</td>
<td>A property_value_representation is the representation of a property, which defines a special quality</td>
</tr>
<tr>
<td>property_value</td>
<td>specified_value</td>
<td>specified_value attribute of property_value_representation</td>
</tr>
<tr>
<td>(attribute)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GIP Relation</th>
<th>AP214 relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity to property relation</td>
<td>item_property_association</td>
<td></td>
</tr>
<tr>
<td>tolerance and value of</td>
<td>property_value_representation</td>
<td>the value, and tolerance attributes of a functional_requirement are represented as a property_value_representation which is related to the product_function with the context ‘axiomatic design’ using an item_property_association</td>
</tr>
<tr>
<td>functional_requirement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10.2.3 Constraints

Constraints are represented as the existing AP214 entity ‘design_constraint’ with corresponding relations.

<table>
<thead>
<tr>
<th>GIP Relation</th>
<th>AP214 relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint</td>
<td>design_constraint</td>
<td>A ‘design_constraint’ is a requirement that has to be considered in the design process of a ‘complex_product’, where a complex_product can be a conceptual solution.</td>
</tr>
<tr>
<td>impact</td>
<td>design_constraint_ association</td>
<td>A ‘design_constraint_association’ is a mechanism to associate a ‘design_constraint’ with an object that is subject to the constraint indicated. The data associated with a design_constraint_association are: is_based_on (the constraint); is_constraining (the object); name.</td>
</tr>
<tr>
<td>Refinement</td>
<td>design_constraint_ relationship</td>
<td>A design_constraint_relationship is a relationship between two Design_constraint objects. The AP214 entity has to be extended to relate to product_function as well as to design_constraint</td>
</tr>
</tbody>
</table>

![Diagram of design_constraint_association and design_constraint_relationship]

![Diagram of design_constraint_association and design_constraint_relationship]

![Diagram of design_constraint_association and design_constraint_relationship]
Some additions are needed to model the relations of a constraint to the operational definition and to the concept of a DP introducing a new constraint.

10.2.4 Relating customer needs to the product structures

Interpretation is a relation which connects a customer need with a functional requirement or constraint. The relation entity has two attributes: 1) ‘associated_requirement’ which specifies the interpretation object - the functional_requirement or design_constraint and 2) ‘associated_need’ which specifies the interpreted customer_need.

A class_need_association is a kind of class_specification_association which associates a product_class with a customer_need instead of a specification. A relation_configuration is a kind of configuration entity which relates a product_class with a customer_need and its interpretation into a constraint or FR.
10.2.5 Operational definition

An operational definition is an information model of an equation or constraint, operational_definition_association relates any type of complex_product to an operational definition.
10.2.6 Interface models

An interface is modeled using several entities and relations.

Entities: interface_type, interface_occurrence, interaction, port_description,
Relations: component_interface_association
10.3 The Modular Function Deployment (MFD) method

The Modular Function Deployment (MFD) method, is a five step product development method for designing modular products [Erixon 1998]. The 5 steps:

1. Clarify customer requirements
2. Select technical solutions (conceptual and embodiment design)
3. Generate concepts (turn into assembly modules)
4. Evaluate concepts (analyze interfaces)
5. Improve modules (Design for Assembly etc.)

In the third, modularization step, each component is related to one or several reasons for being in a separate module. The reasons for modularity are thus made operational as 12 kinds of ‘module drivers’, as described in . Apart from manufacturing reasons, module drivers indicate various strategic reasons concerning commonality/variety, and reuse for separating a component into a module.

Figure 108 The 5 steps of the MFD method [Erixon 1998]
Table 22 The 12 module drivers [Erixon 1998]

<table>
<thead>
<tr>
<th>Module Drivers</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGN AND DEVELOPMENT</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Carry over</td>
<td>Are there reasons that this technical solution should be a separate module because the new design can be carried over to coming product generations?</td>
</tr>
<tr>
<td>Technology push</td>
<td>Is it likely that this part will go through a technology shift during the product life cycle?</td>
</tr>
<tr>
<td>Planned design changes (Product plan)</td>
<td>Are there reasons why this part should be a separate module since it is the carrier of changing attributes that will be changed according a product plan?</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Technical specification</td>
<td>Is this part influenced by varying requirements?</td>
</tr>
<tr>
<td>Styling</td>
<td>Is this part influenced by trends and fashion in such a way that form and/or color has to be altered, or should it be tied to a trademark?</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Common unit</td>
<td>Can this function have the same physical form in the product variants?</td>
</tr>
<tr>
<td>Process/ Organization</td>
<td>Are there reasons why this part should be a separate module because:</td>
</tr>
<tr>
<td></td>
<td>- a specific or specialized process is needed?, it has a suitable work content for a group?, a pedagogical assembly can be formed?, the lead time will differ extraordinary?</td>
</tr>
<tr>
<td>QUALITY</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Separate testing</td>
<td>Are there reasons why this part should be a separate module because its function can be tested separately?</td>
</tr>
<tr>
<td>PURCHASE</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Black-box-engineering</td>
<td>Are there reasons for which this part should be a separate module because:</td>
</tr>
<tr>
<td></td>
<td>- there are specialists that can deliver the part as a black box?, the logistics cost can be reduced?, the production and development capacity can be balanced?</td>
</tr>
<tr>
<td>AFTER SALES</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Service/ maintenance</td>
<td>Is it possible that the service repair will be easier if this part is easy detachable?</td>
</tr>
<tr>
<td>Upgrading</td>
<td>Can the future upgrading be simplified if this part is easy to change?</td>
</tr>
<tr>
<td>Recycling</td>
<td>Is it possible to keep the highly polluting material or easy recyclable material in this part (material purity)?</td>
</tr>
</tbody>
</table>
By creating a Module Indication Matrix (MIM), that relates components to these drivers, patterns can be discerned, and the rule is that components with similar patterns should be aggregated into modules (Figure 3). To actually decide exactly how to compose modules requires the experience and knowledge of designers.

<table>
<thead>
<tr>
<th>Module driver</th>
<th>Sub-function 1</th>
<th>Sub-function 2</th>
<th>Sub-function 3</th>
<th>Sub-function 4</th>
<th>Sub-function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development and Design</td>
<td>Carry-over</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Product plan</td>
<td>Technology push</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Styling</td>
<td>Styling</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Proc.</td>
<td>Common unit</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Quality</td>
<td>Separate test</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Purchase</td>
<td>Black-box eng.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Sub-function (techn. solution) = Medium driver (3)  
● = Strong driver (9)  
○ = Some driver (1)

Figure 3: Module Indication Matrix [Erixon 1998]

In the extended MFD method [MMAB 1999], which comprises a deeper focus on variety, the varying customer requirements within different market segments are identified, ordered and related to the properties of an existing basic product idea by use of several QFD matrices. Common and varying product properties of different importance are identified and related to the technical solutions of the system as a basis for determining a strategic modular architecture.
Figure 110 The extended MFD method correlating market segments with modules [MMAB 2000]

Functional reasons are treated separately, since the aim of the method primarily was to emphasize other than functional reasoning. Evidently the functional reasoning is crucial, and is considered in the second step, but not within the MIM itself. Attempts have been made to improve the method by integrating the functional and strategic drivers in a common module indication matrix [Lanner 1996].
10.4 Axiomatic design corollaries

I. (Decoupling of Coupled Design)
   A. Decoupling of separate parts or aspects of a solution if FRs are coupled or become interdependent in the designs proposed.

II. (Minimization of FRs)
   A. Minimize the number of FRs and constraints. *As the number of FRs and constraints increases, the system becomes more complex and thus raises the information content.*

III. (Integration of Physical Parts)
   A. Integrate design features in a single physical part if FRs can be independently satisfied in the proposed solution. *As long as the FRs are not coupled by the physical integration of parts, the integration strategy should be followed if it reduces the information content of the design.*

IV. (Use of Standardization)
   A. Use standardized or interchangeable parts if the use of these parts is consistent with the FRs and constraints. *In order to reduce inventory and minimize the information required for manufacture and assembly, special parts should not be used if standard parts can fulfill the FRs. Furthermore, the number of standard parts should be minimized so as to decrease the inventory costs and simplify inventory management, per Corollary 3. Interchangeable parts allow for the reduction of inventory, as well as the simplification of manufacturing and service operations; that is, they reduce the information content. This is even more the case if the design permits generous tolerances.*

V. (Use of Symmetry)
   A. Use symmetrical shapes and/or arrangements if they are consistent with the FRs and constraints. *Symmetrical parts require less information to manufacture and to orient in assembly. Not only should the shape be symmetrical whenever possible, but the hole locations and other features should be placed symmetrically to minimize the information required during manufacture and use.*
VI. (Largest Tolerance)
   A. Specify the largest allowable tolerance in stating FRs. *The specification of tolerances should be made as large as possible, but should remain consistent with the likelihood of producing functionally acceptable parts.*

VII. (Uncoupled Design with Less Information)
    A. Seek an uncoupled design that requires less information than coupled designs in satisfying a set of FRs. *There is always an uncoupled design that involves less information than a coupled design.*
**10.5 Express-G modeling language**

EXPRESS-G is a graphical representation of the lexical object flavored language EXPRESS, which was developed for the purpose of describing the information required to design, build and maintain mechanical products.

Although Express provides a object representation capability including subtype/super type inheritance and constraints on object attributes, only part of this is represented in EXPRESS-G. Entities and attributes are the main EXPRESS-G constructs, relations are represented as entities.

**EXPRESS-G Legend:**
- Rectangular boxes represent entities being defined in this schema
- Round boxes represent entities being defined elsewhere
- Narrow lines with a circle are used to connect an entity with its attributes.
- Dotted lines are used to indicate optional attributes
- Thick lines are used to indicate a hierarchical relationship, where the supertype "points to" the subtype with the circle. Subtypes inherit all attributes and constraints (visual or written in lexical Express) from the supertype.

A supertype may either be possible to instantiate in itself, or may be an abstract type requiring the existence of a subtype. An abstract supertype is labeled with the symbol (ABS) before the name of the supertype.

An attribute may be of different data types, such as string, Boolean, or real, where ‘string’ is the default, or refer to another entity or a selection between entities or data types.
As mentioned before, relations are modeled as entities in themselves. As an example, the decomposition structure between two components ‘Component 2 is a subpart of Component 1’ could be modeled using an explicit relation entity ‘decomposition’ which relates the component entities. Following the notation of AP214, a relation points to related entities through the pointers ‘relating’ and ‘related’, describing the hierarchy that the ‘relating’ object is the parent of the ‘related’ one.

To differentiate between information models which define the constructs, and models which are based on these definition, a diagonal line in the corner is used in instantiated models.

![Diagram](image-url)

*Figure 111 Instantiated model of the decomposition of a component*