Abstract

Mass Customisation (MC) is relevant for many application areas, especially those where customisation is common such as for the engineer to order area. Very often, these enterprises must take on considerable changes in order to apply to MC.

MC most often leads to configuration, where individual products or components are defined from a configurable model – a model in which all possible variations are represented. Hence, modelling of such models is very important and this may sometimes be supported by computer based modelling tools. However, many of these tools are insufficient and many standard modelling systems have poorly defined modelling methodologies. Basically, an information model is required as a platform for development of software applications such as configurators.

In this paper, a rather simple modelling approach is presented. This approach is based on the theory of general systems and outlined in combination with the abstraction mechanisms classification and composition together with object-oriented analysis and design. Throughout the presentation, a previously developed generic model component is used to illustrate the approach and to prove, how it is based on a theoretical foundation.

Engineer to order companies must often accept market conditions, which imply long order horizons and many changes of the orders both before and after order acceptance. Therefore, it is necessary to concentrate on decisions, which are relatively invariant throughout order processing. By use of the presented modelling approach, it is shown, how modelling on multiple abstraction levels can be a solution to such challenges.

Keywords

Mass customisation, product configuration, information modelling, classification, composition, object-oriented analysis and design.
Introduction

Mass Customisation (MC) was initiated more than one decade ago as a research topic with Davis’ publication “From Future Perfect: Mass Customisation” [Davis, 1989], presenting how products and services could be realised as a one-of-a-kind manufacture on a large scale. Davis also presented the idea that the customisation could be done at various points in the supply chain. In 1993, Pine published a major contribution to the mass customisation literature: “Mass Customization: The new Frontier in Business Competition” [Pine, 1993], [Pine et al., 1993], which was an extensive study of how American enterprises during the seventies and eighties had been overrun by the efficient Japanese manufacturers, which could produce at lower costs and higher quality. Since its introduction, MC has called for a change of paradigm in manufacturing and several companies have recognised the need for mass customisation. Much effort has been put into identifying, which success factors are critical for an MC implementation and how different types of companies may benefit from it [Lampel and Mintzberg, 1996], [Gilmore and Pine, 1997], [Sabin, 1998], [Silveira et al., 2001], [Berman, 2002], [Silveira et al., 2001].

For obvious reasons, there are different strategies on how to implement MC most appropriately and it varies naturally also between different companies, markets and products. Because there is not a single generic strategy, it is important to look at the issue from different viewpoints. The fact that products must be easily customisable in order to achieve MC has been described comprehensively in the literature. [Berman, 2002] and [Pine, 1993] proposed that the use of modular product design combined with postponement of product differentiation would be an enabler to a successful MC implementation. This issue of course also relates to the question of readiness of the value chain.

Mass Customisation and Product Configuration

An often used approach for implementation of MC is product configuration, in which a series of products is defined by one single model – a product family model (see figure 1) [Jørgensen, 2003]. Hence, a product family can be viewed as the set of end products, which can be formed from a product family model [Jørgensen, 2003]. In the product family model, it is described, which modules are included in the product family model and how they can be combined [Faltlings, 1998]. The result of each configuration will be a model of the configured product, configured product model. From this model, the physical product can be produced (see figure 1).

From time to time, several different methods for defining product family models and product configurators have been proposed, each with their own advantages. A “Procedure for building product models” is described in [Hvam, 1999] based on [Hvam, 1994]. It is a rather practical approach with a seven step procedure, describing how to build a configuration system from process analysis and product analysis onto implementation and maintenance. For the product modelling purpose it uses the Product Variant Master method followed by object-oriented modelling to describe both classification and composition in a product family. The object-oriented approach is also applied by [Felfernig et al., 2001], who uses the Unified Modelling Language (UML) to describe a product family. This is done by using a UML meta model architecture, which can be automatically translated into an executable logical architecture. In contrast to [Hvam, 1999] this method focuses more on formulating the object-oriented product structure, rules and constraints most efficiently. The method also focuses on how the customers’ functional requirements can be translated into a selection of specific modules in the product family.
Most of the methods, which exist for product family modelling, focus on modelling of the solution space of a configuration process. This means that they describe the possible attributes of the products and the product structure. Hence they do typically not focus on additional information which goes beyond, what must be used to perform the configuration itself. This kind of information, which could include e.g. customer, market, logistics and manufacturing information, is according to [Reichwald et al., 2000] similarly important, since a successful implementation of MC must integrate all information flows in the so called “Information Cycle of Mass Customisation”.

In [Jiao et al., 1998], [Du et al., 2000] and [Männistö, 2001], mapping of functional requirements to specific modules is considered. Jiao proposes to use a triple-view representation scheme to describe a product family. The three views are the functional, the technical and structural view. The functional view is used to describe, typically the customers, functional requirements and the technical view is used to describe the design parameters in the physical domain. The structural view is used for performing the mapping between the functional and technical view as well as describing the rules of how a product may be configured. The description of this modelling approach is however rather conceptual, and does not easily implement in common configuration tools.
A product family model is often the basis for development of a product configurator, a tool, computer software, which can support users in the configuration process [Faltings, 1998], for instance by selecting modules to compose products. Hence, with a product configurator, it is possible to configure multiple individual solutions – perhaps a large set of products. Figure 2 shows a sample user interface of a product configurator by which users can select specific values in the presented fields for configuration of a computer.

**Application of Product Configuration**

Mass Customisation and product configuration is relevant for many enterprises and great benefits are normally found, where customisation is common and where the idea is introduced gradually. In general, however, the benefits depend very much on the product and the market. In the relationship between the manufacturer and the market or more precisely the product and the customer, the product configurator plays a major role.

A major distinction regarding markets/customers is between business-to-business (B2B) and business-to-consumers (B2C) and an important dimension here is the degree of personalisation. Personalisation is most relevant in relationship with B2C and a high degree of personalisation towards individual customers or small groups of customers generates special requirements to product configurators but, on the other hand, this also raises new opportunities for increased volume.

Development of computer based configurators provides a range of opportunities for adding new dimensions to the subject and configuration may also add more value to customers. Therefore, when a configurator is designed, a large number of design parameters must be considered and balanced decisions must be made. Many of the parameters are related to development of software systems, e.g. usability, reliability, flexibility and security. Some of these parameters will not be considered in detail in this paper. The primary focus will be on relationship between market/customer and the product and a new model with identification of customisation levels is developed and presented in the following.

**Customisation Levels**

In order to support the strategic decision making regarding customisation of products, the following new model for customisation has been developed (see figure 3). This model arranges customisation in four different levels of customisation, ranging from the structure level at the bottom, through the performance level and the value level to the learning level at the top. The model has a dual view on customers at one side and products at the other side and it is developed so that configurator designers must decide how far up in levels the configurator should reach.

Configuration on the structural level is a rather common view of configuration and is characterised as a matter of acquiring components, which can be used as building blocks. A well known example is the LEGO bricks. Important issues are the modularity and the interfaces of the modules. Modularity is in contrast to integrated modules. Further, different architectures of modularity is worth considering.
On the next level, the performance of products is essential. When products are installed in their user environment, they perform their functions – hopefully in the expected way. Therefore, considerations about the ability to perform the functions, which are required by the customer, are very important and should be addressed specifically. In this case, the focus of product configuration can be shifted to identification and definition of product properties/attributes instead of modules and components. This is particularly important in building construction where many changes have to be managed and where order horizons can be long.

Mapping of functional requirements to specific modules is considered in [Jiao et al., 1998], [Du et al., 2000] and [Männistö, 2001]. Jiao proposes to use a triple-view representation scheme. The three views are the functional, the technical and structural view. The functional view is used to describe, typically the customer's functional requirements and the technical view is used to describe the design parameters in the physical domain. The structural view, which corresponds to the structural level described above, includes the mapping between the functional and technical view as well as the rules of how a product may be configured. The description of this modelling approach is however rather conceptual, and does not easily implement in common configuration tools.

The two lower levels of customisation are rather common and widely used with many products and on all types of markets. Further levels of customisation will primarily relate to individual customers or homogeneity groups like couples and families.

The value level focuses on experience and emotions to the customer by providing the overall values of products, the more imaginary attributes. To be involved in a configuration process will for many customers result in a higher degree of satisfaction and the customer will likely feel a stronger attachment to the solution. In order to support this, it is important that the available options are matched properly with the customer needs. Otherwise, there may be a great risk that configuration will give the opposite effect. If the options are limited, it is important to be selective regarding customer segments. Means for good configurator support on the experience level are to provide good guidance to the user, to display consequences of choices and to set focus on expression of the basic values of the solution.
A further level of customisation can also be considered. In this, the configurator offers services that may result in **education of the involved customer**. In other words, the **customer becomes the product**. Such services will include a range of subjects that may give answers to the customer about complex questions. In case of buying a building, examples could be about requirements from authorities, financing, budgeting and maintenance. Adding such additional features also requires high quality in order not to give a negative effect.

The dependencies between attributes and module structure is illustrated in figure 4, which shows how underlying modules/components are determined on the basis of decisions regarding the chosen attributes. Five different alternatives are shown of which alternative number 1 is an exception from the general scheme.

![Figure 4 - Specification of modules directly or indirectly through functionalities.](image_url)

At selection no. 1, a specific attribute is not selected, because this is a case, where it is more natural to choose a module directly - typically **add-on modules**. An example of this is the sunroof of a car (provided that only one type of sunroof exists). Simply: Sunroof (Yes/No?).

At selection no. 2, attribute1 is equal to module 2. This can only be fulfilled in one way, and that is by including module no. 2. For instance, air conditioning equals an air condition module.

At selection no. 3, attribute no. 2 results in that both module no. 3 and 4 are selected. An example of this is that if the customer chooses the turbo car model, then both a turbo engine and ABS brakes must be selected.

Finally, selections no. 4 and 5 show a relatively usual case, where a module is determined by more than one attribute, i.e. attributes of the module. For instance, a seat can be specified from two attributes: the colour and whether there should be a headrest or not. When these two attributes are specified, then one module (a complete seat) can comply with both attributes.
3. Product Family Models

A product family model (see figure 1) [Jørgensen, 2003] has a set of open specifications, which have to be decided to determine or configure an individual product in the family. The product family model is a foundation for configuration and, in order to secure that only legal configurations can be selected, the family model should contain restrictions about what is feasible and not feasible. Hence, the product family is defined as the set of possible products, which satisfy the specifications of the product family model. The result of each configuration will be a model of the configured product, *configured product model*. From this model, the physical product can be produced (see figure 1). A *product configurator* can be defined as a tool, computer software, which is built on the basis of a product family model and which can support users in the configuration process [Faltings, 1998].

![Model of the structure with the three levels.](image)

Product configuration in the simplest form is a matter of combining a set of *modules* (see figure 5) so that the product model contains information about what modules and components are to be assembled. In this *compositional view*, a product consists of a number of components, which subsequently can consist of other components, etc. Modules are identified on a level above components from a configuration point of view whereas components usually are identified from a manufacturing point of view. Most often, the number of modules is smaller than the number of related components. Thus, in the *structural model* for configurable products, products consist of modules and modules consist of components.

![Three different categories in which two modules can be classified.](image)
In connection with identification of modules, it is important to analyse how modules interface with each other (see figure 6). Therefore, it is important also to look at the modules functional characteristics and secure that the modular structure is harmonised with the functional division of the product [Andreasen, 2003].

Besides structure, products have properties. It is essential for both the customer and the producer to focus on properties of the resulting product. For each configured product, the resulting properties are dependent of the selected components and structure of the product. In the product configuration process, algorithms must be available to estimate the resulting product properties. Some properties are simply the properties of the components, e.g. the colour of a car is normally defined as the colour of the car body. Other properties are computed from properties of the components. For example, the weight is simply the sum of the component’s weight. However, not all resulting properties are so easy to determine. For instance, the resulting performance of a pump is a non-linear function of certain component properties. Much more complicated examples could be mentioned.

In the following, the term attribute will be used in the models corresponding to properties of physical products. Consequently, when a configuration is performed, the desired properties of the resulting product must be determined by defining values of attributes in the product family model. All relevant attributes of both the resulting product and the available modules must be specified and their optional values to be selected during configuration tasks must also be defined. In relation to this, it is important to notice that the selectable modules and components are sometimes substituted by one or more attributes. For instance, a computer can be ready for use (attribute) or the operating system is installed (module/component). Therefore, the configuration process can be considered as a mixture of attribute specification and selection of modules, which together can satisfy the required attribute values.

4. Fundamental Issues of Information Modelling

Methodologies for system development are often based on concepts derived from General Systems Theory. According to this theory, a system model is an intentionally simplified description of a system, fulfilling a certain purpose. Hence, the simplifications imply that some choices are made in order to select the most important properties, components and relationships. Thus, a system model can e.g. be suitable for communication between designers, because with the model, it will be possible to concentrate on the most important aspects of the system. Models are created by analysis, i.e. analysis models, or by synthesis, i.e. synthesis models (see figure 7). Analysis models are models of something existing, often physical objects and synthesis models are models created as a foundation for construction of something new, which eventually will become physical – an artefact [Jørgensen, 2002]. Hence, synthesis models are built from ideas, thoughts and imaginations and obtained in some kind of representation. Design by modelling is a development approach, where a synthesis model is designed as an intermediate result and the final result is an implementation of the model in the real world.
Computer-based models are fundamentally stored in computers as *data objects* and *data structures*, which can be manipulated by applications. Therefore, development of tools for modelling includes both development of a *data model* and a number of *applications* with relationships to the data model [Jazayeri, 2000]. One of the most important requirements for the data model is that it must be non-redundant so that no data value is stored more than once. In order to ensure that this requirement is fulfilled, the model representation has to be considered very carefully based on the meaning of data, the semantics. Therefore, the foundation for a data model is an *information model* ([Hammer 1978], [Rumbaugh et al. 1999] and [Halpin 2001]), created in combination with semantics from the domain, which the design model is addressing.

![Figure 7 - Models are created either by analysis or synthesis](image)

Computer-based models are fundamentally stored in computers as *data objects* and *data structures*, which can be manipulated by applications. Therefore, development of tools for modelling includes both development of a *data model* and a number of *applications* with relationships to the data model [Jazayeri, 2000]. One of the most important requirements for the data model is that it must be non-redundant so that no data value is stored more than once. In order to ensure that this requirement is fulfilled, the model representation has to be considered very carefully based on the meaning of data, the semantics. Therefore, the foundation for a data model is an *information model* ([Hammer 1978], [Rumbaugh et al. 1999] and [Halpin 2001]), created in combination with semantics from the domain, which the design model is addressing.

![Figure 8 - Sample composition structure of a computer](image)
An important fundamental issue of information modelling is abstraction mechanisms, which provide the means for identification and design of invariant components and structures ([Smith 1977a], [Smith 1977b], [Rosch 1978] and [Sowa 1984]). Two abstraction mechanisms are defined here: composition and classification [Jørgensen, 1998].
Composition focuses on the components and the relationships between the components. The most frequently used structure is the component structure, which shows *aggregation* versus *separation*. Such a structure is illustrated in figure 8 for a sample computer.

Classification focuses on identification of *classes/types* of components based on the *properties/attributes*, which characterise them. This can be illustrated in a diagram, termed *taxonomy* (see figure 9), where the relationships *generalisation* versus *specialisation* are shown. Often, a UML class diagram is used for the taxonomy ([Rumbaugh et al. 1999]).

The taxonomy in figure 9 shows that all types of computer components can be divided into the different types on the next level: 'Mass storage components', 'Print cards', 'Integrated circuits', etc. Further, the type 'Mass storage components' has the sub-types 'Hard disks', 'CD drives', and 'DVD drives'. Observe that the type 'Mass storage components' is different from the type 'Mass storage modules' because e.g. 'Hard disks' are sub-types of the first type but not a sub-type of the module type.

In information modelling, composition and classification together support identification of fundamental structures on a *type level* as the basis for generation of individual components on an *instance level* and they provide the means to set particular focus on the most invariant decisions. A classification process results in a basic structure of types and a composition process results in a basic structure of components.

Another important issue of information modelling is the *object-oriented paradigm*, which can be adopted in harmony with the abstraction mechanisms. In this paradigm, each model component is regarded as a living organism, which act and interact with other components. Thus, object-oriented components are equipped with behavioural attributes, which enable them to respond to requests and, consequently, even if a real world component is non-living, the corresponding model is created as an active component.

The two abstraction mechanisms are used in design tasks, but, as indicated in figure 10, classification is used first and composition afterwards. Classification primarily supports the identification of model components and the basic structure at the type level. Based on this, the structural considerations are identified by use of composition.

![Classification and composition hierarchies](image-url)
5. A Generic Information Model Component

In order to be able to create all sorts of models and to perform many different modelling processes, a conception of a generic model component is introduced. This component is inspired from general systems theory and from object-oriented modelling and can be regarded as a component that can be used for system models in general and for information modelling.

![Generic model component](image1.png)

Figure 11 - Generic model component

The generic component consists of a set of attributes and a structure of sub-components (see figure 11). Some attributes are factual attributes, defining the state of the component, and some attributes are behavioural attributes, defining the operations, which the component can carry out. An alternative division of attributes defines some attributes as visible attributes, which can be called from other components, and some are defined as hidden attributes. The structure establishes the relationships between the component itself and the sub-components.

![Model component type](image2.png)

Figure 12 – Model component type is the basis for generating objects (instances)

All structures can be represented by two kinds of relationships in the information model: references and collections. For the computer example, a reference could represent the
relationship e.g. between the keyboard and the computer. A collection could represent the relationship e.g. between the cpu board, the anchor, and multiple memory units, the members.

When a synthesis information model is considered, a foundation for the components must be established by creating types of components. Component types are the primary content of information models and it is important to distinguish between modelling on the object level and modelling on the type level.

Each component type includes a specification of a set of attributes with name and data type. The classification abstraction mechanism is primary because, based on attributes, the component types can be classified and organised in a hierarchy, the taxonomy. Identification and specification of structures can also be included in the component types by creating the relations, which formulate the constraints regarding attributes and combinations of sub-components. The component type is a kind of template and, from each type, an indefinite number of components, instances, can be generated. The quality of these component types is the key basis to achieve an invariant information model foundation.

4 Product Family Modelling

There is a need for a methodology to describe and develop models of configurable products. Companies, who are implementing product configuration, need a comprehensive terminology and a systematic methodology in order to develop their modular products. It is of great importance to use well-defined terms and use the agreed terminology consistently in connection with a well-proven methodology, so that misunderstandings can be avoided and communication can be eased.

4.1 Attributes and Data Structures

As mentioned, products consist of properties, components and structure and similar contents goes for models of products and product families. In the following, the term attribute will be used in the models corresponding to properties of physical products. Consequently, when a configuration is performed, the desired properties of the resulting product must be determined by defining values of attributes in the product family model. All relevant attributes of both the resulting product and the available modules must be specified and their optional values to be selected during configuration tasks must also be defined. In relation to this, it is important to notice that the selectable modules and components are sometimes substituted by one or more attributes. For instance, a door can be lockable (attribute) or it can be equipped with a lock (module/component). Therefore, the configuration process can be considered as a mixture of attribute specification and selection of modules, which together can satisfy the required attribute values.

4.2 Development of Product Family Models

As stated above, product family models must be able to construct individual product models through a configuration task. Each product model must have sufficient data about attributes and structure to describe and manufacture the physical product. Consequently, the basic elements of product family models are the total set of attributes of the possible product
models and the set of identified modules, each with their internal attributes and data structures.

4.2.1 Module Types

The basic units of a product family model are module types. A module type is a model of the set of modules, which are interchangeable, perhaps with some restrictions. During configuration, individual modules of each type are determined. The attributes of the product models and the module types are selected on the basis of what is important and relevant.

In the following, the contents of product family models are illustrated by use of simple elements of a synthetic language. Furthermore, fractions of a simple example of a computers product family model are added to the illustration.

Each attribute in a module type is defined by a name and probably a data type (Boolean, Integer, Float, String, Currency, etc.).

This declarative statement shows the syntax for description of a module type:

type name {...}

Example:

type HardDisk {...};

The syntax of attribute declaration with data type is:

    name : data type;

Example:

type HardDisk
{
    Name : string(50);
    StorageCapacity : integer;
    AccessTime : float;
    Price : currency;
}

The available instances of a module type can be listed by a table with a column for each attribute and a row for each module.

<table>
<thead>
<tr>
<th>Name</th>
<th>StorageCapacity</th>
<th>AccessTime</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maxtor 10K-3</td>
<td>37 Gb</td>
<td>4,5 ms</td>
<td>1.375 DKK</td>
</tr>
<tr>
<td>Maxtor 10K-4</td>
<td>147 Gb</td>
<td>4,4 ms</td>
<td>4.055 DKK</td>
</tr>
<tr>
<td>Maxtor 10K-5</td>
<td>300 Gb</td>
<td>4,4 ms</td>
<td>8.975 DKK</td>
</tr>
</tbody>
</table>

Alternatively, module data can be extracted from a database.

Typically for product family models, some modules can be configured by selecting attribute values. In this case, each attribute is not defined by a data type but instead by a domain with the possible values. A domain can be a set of discrete values, an interval of integer values, string values or a list of identifiers.
The syntax of an attribute declaration with domain and a possible default value is:

name : {domain} [default value];

Example:

type HardDisk
{
    ....
    PreSet : {Master, Slave} default Master;
    OperatingSystem : {Non, WinXP, Win2000, WinMe} default WinXP;
    ....
}

When module data are specified in form of a table as shown above, the domain values can also be listed in columns to the table.

4.2.2 Relationships

Module types can have relationships. In general, there are four different forms of relationships, see figure 13.

![Figure 13 - Four forms of relationships between module types](image)

Relationship forms 3 and 4 are the simplest forms of relationships, which can be used to model derived values of different kinds.

Relationship forms 1 and 2 can be used to model various structures e.g. compositional structures and constraints, which are essential for product family models. Relationships of form 1 will preferably be identified by special attributes and, hence, be changed to form 2.

4.2.2.1 Derived values

The simplest form of relationship is relationships between attributes of a module type (type 4). Such relationships can be modelled as expressions and thereby define derived values. All normal arithmetic and boolean operators and standard functions may be used.

Example:

type HardDisk
{
    ....
    TaxedPrice : Price * TaxRate;
}


Such expressions may also include attributes of other module types (relationship type 3). Beside standard functions, some special functions may be defined as special algorithms. If just the name of a module type is included in such expressions, it has the meaning: "number of instances of the type".

Examples:

```plaintext
type Computer
{
    OperatingSystem : Boolean default true;
    Colour = Case.Colour;
    HardDisks = HardDisk;
    DiskMemory = Sum(HardDisk.StorageCapacity);
    Weight = SumWeight : Double { ... Specific algorithm ... }
    ..... 
}
```

4.2.2.2 Structures

Structures are represented by special kind of attributes.

The symbol -> represents a reference i.e. a one-to-many relationship

Example:

```plaintext
type Processor
{
    Name : string;
    ...
    ContainingBoard -> MotherBoard;
}
```

The primary purpose of references is to avoid redundant data and thereby ensure that every piece of information is only represented once.

The symbol ->> represents a collection i.e. a one-to-many relationship. Collections are used to specify product structures. Here, it is described that a product/module (instance of a module type) consists of modules (instances of other module types), which eventually also consist of modules etc. until the component level is reached. In a module type, such a relation expresses the module types for possible sub modules.

Example:

```plaintext
type Cpu
{
    ...
    RelatedGraphicCards ->> GraphicsCard;
    RelatedIoCards ->> IoCard;
    ...
}
```

Collections may include multiple types
Example:

```plaintext
type Case
{
    ......
    Contents ->> { CPU; MassStorage; PowerSupply; PowerCable; }
    ......
}
```

As shown, a collection can only connect two levels of a composition structure (hierarchy). This means that collections must be defined for each branch of the structure.

4.2.2.3 Constraints

Collections may be constrained by a *multiplicity* specification to form a basic expression about the number of instances that can be included.

Multiplicities are formulated with the syntax:

```
from .. to
```

where 'from' is typically 0 or 1 and 'to' is typically a fixed number or any number greater than or equal to from. This is indicated by a *.

Examples:

```
3..5      from three to five
1..*      from one to many
0..*      from zero to many (meaning no constraint equal to no multiplicity)
```

Example:

```plaintext
type Cpu
{
    ......
    Contents ->> { 1..1 MotherBoard; 1..* Processor; 1..* MemoryUnit ; ...}
    ......
}
```

```plaintext
type MassStorage
{
    ......
    Contents ->> { 1..* HardDisk; 0..* CdDrive; 0..* DvdDrive; 1..* DiskCable;}
    ......
}
```

In general, constraints may be formulated by arithmetic or boolean expressions. Here, the ordinary arithmetic operators like addition, subtraction, multiplication and division can be used together with standard functions. The following arithmetic relation operators =, >=, <=, >, < and <> can also be used along with the boolean operators AND, OR, XOR, NOT, implication (⇒) and bi-implication (⇔). If the name of a module type is included in an expression, it means “number of instances of the type”.

17
Examples of relations with arithmetic and logical operators are:

```plaintext
type Cpu
{
    constraints
    {
        Cards : GraphicsCard + IoCard + TvTunerCard <= NbOfBusSlots;
        Processors : Processor <= ProcessorSlots;
        ....
    }
}

type Computer
{
    constraints
    {
        Monitors : Monitor <= 2;
        Cables : HardDisk + CdDrive + DvdDrive <= DiskCable * 2;
        Drives : CdDrive ⇔ not DvdDrive;
        ....
    }
}
```

### 4.2.3 Classification

As previously stated, the classification abstraction mechanism is fundamental for identification and definition of types; hence, the module types above are actually related to each other as indicated in figure 9.

The syntax of the relationships between super-types and sub-types is:

```plaintext
type name1 subtypeof name2 { ... };
```

Examples:

```plaintext
type ComputerComponent { ... };

type MassStorageComponent subtypeof ComputerComponent { ... };

type HardDisk subtypeof MassStorageComponent { ... };

type PrintCircuitBoard subtypeof ComputerComponent { ... };

type MotherBoard subtype of PrintCircuitBoard { ... };

type IntegratedCircuit subtypeof ComputerComponent { ... };

type Processor subtypeof IntegratedCircuit { ... };

type CompositeComponent subtypeof ComputerComponent { ... };

type CpuModule subtypeof CompositeComponent { ... };
```
type OtherComponent subtypeof ComputerComponent { ... };

type Computer subtypeof OtherComponent { ... };

With classification, it is defined that attributes in super-types are inherited to sub-types.

## Product Configurator Development

### 5.1 Rule- versus constraint-based methods

The algorithms in a configuration application have the purpose of dealing with the restrictions and other relations there exist between the modules. These algorithms are traditionally modelled according to a *rule-based method*. This method uses if-then constructions in the code. Another method is to use *constraints*. The primary advantages are that the constraint-based method is more elegant, needs less lines of code and has a better performance function. In figure 14, a further comparison is shown.

<table>
<thead>
<tr>
<th>Rule-based method</th>
<th>Constraint-based method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building and maintaining is very difficult and</td>
<td>Building and maintaining is simpler.</td>
</tr>
<tr>
<td>time-consuming.</td>
<td></td>
</tr>
<tr>
<td>The choice of modules/functionalities must be</td>
<td>Freedom of making the choices in any sequence.</td>
</tr>
<tr>
<td>performed in a predefined order.</td>
<td></td>
</tr>
<tr>
<td>‘Batch mode’, i.e. a number of choices are made as an</td>
<td>‘Interactive model’, i.e. consequences are derived immediately</td>
</tr>
<tr>
<td>input, after which the algorithm are executed with the</td>
<td>after each choice.</td>
</tr>
<tr>
<td>possible consequence, that illegal choices has been</td>
<td></td>
</tr>
<tr>
<td>made.</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 14* Rule versus constraint-based algorithms in product family models [salesPLUS, 1997].

### 5.2 Inference method

Use of constraints in the product family model is an alternative compared to *action rules*, i.e. if-then statements. Hence, this approach is in contrast to the earlier mentioned implementations in GDL objects and in the ISO PLIB standard. The primary advantages are the following: the constraint representation is more elegant, constraints bases are smaller than rule bases and inference algorithms have a much better performance. In addition, the inference algorithm allow decisions about values of the product attributes to be made in any order, see [Sabin, 1998; Soininen, 2000] for a further comparison. Research results regarding knowledge-based systems have resulted in inference algorithms with very good performance [Møller, 1995].

Implemented in product configurators, such algorithms can be activated each time the user makes a selection [Yu, 1998; Array, 2003]. For large models, the process may take too much time in the beginning, but, after some selections, the results can be presented within reasonable time. Referring to the constraints, this approach has the great advantage that the consequences for related attributes are shown before the next choice has to be made. This implies that, at each point of a configuration process, the already made decisions can
be checked for consistency and, furthermore, it is possible to freeze these decisions and transform the configurator to a new configurator with the remaining decisions to be made by others. In this way, a good balance can be maintained between proprietary product family models and open product family models.

5.3 User interfaces

The design of product configurator user interfaces is very important and the great potential in using web technologies should not be underestimated. With this in mind, a large number of graphical components should be included where appropriate, i.e. drop-down lists, check boxes and radio buttons. In addition, automatic generation of visual effects in connection with the selectable options would be very helpful. For instance, when different colours are the option, this should be shown graphically and, when different modules can be selected directly or indirectly, they should be presented as graphic 3D images. Ultimately, the complete picture of the configured product should be generated and perhaps presented as a high quality rendered image in 3D stereo.

In order to include such graphical components in the user interface, it is important to include geometrical data in the product family model. This underlines the fact that a product model is more than a CAD model. When a product family model is the origin, it is obvious that geometrical data is only a part of the model. Attributes describing the product geometry may even be included as secondary attributes, where the values are derived through the internal constraints.

7. Abstraction by Classification

Regardless of whether the selection of modules is implicit or explicit, multiple abstraction levels can also be established by the use of classification. In a taxonomy over module types (see figure 9), the types towards the root are the most general types whereas the types towards the leaves are the most special types. Therefore, a selection of relatively general types represents a higher abstraction level compared to selection of relatively special types.

**Taxonomy:**

Computer components

... Print boards

... Sound boards

Surround

4.1 channels

5.1 channels

6.1 Channels

Stereo

Ordinary

Four Point

3D

...
Figure 15 shows a partial taxonomy as a further classification of a specific module type of figure 9 and reveals two additional levels of specialisation. Clearly, this example illustrates that a preliminary selection of a relatively general type is a way of postponement, i.e. some indications are given but further specifications can be submitted.

All module types have attributes, which can be included in the configuration process. Besides an obvious price attribute, further technical properties of the available modules can be represented as attributes of the module types. These attributes can be located at different levels of the taxonomy depending on how general or special they are. Consequently, a selection of a type results in a set of additional attributes, which can be used for further specification. However, if a specification of a specific attribute is required, a specialisation down to a certain level is implicitly made. If for instance something is required about attributes which are only relevant for stereo sound, then stereo sound boards are implicitly selected.

In general, classification is very much related to attributes. Besides what is already described, identification of sub-modules can be based on values of attributes. For instance, the sub-types of surround sound board could be identified by values of an attribute 'NoOfChannels'. In fact, this attribute could remove the need for classification at the lowest level. Hence, if multiple classifications of these sound boards were relevant, i.e. if multiple and equally important classification criteria exist, it will be more flexible to identify the corresponding attributes and their possible values.

8. Application of Product Family Modelling

Many observations indicate that implementation of Mass Customisation and product configuration in ETO companies must focus on product modelling in order to gain immediate economic results from saving resources for tendering and order processing. This top-down development approach is also important when different organisational units must be joined and different software applications and databases must be integrated. Therefore, a number of theoretical topics about system modelling, product modelling, modelling of product families, information modelling and data modelling must be utilised.

In this paper, it is proposed that modelling of product families should be performed in a way that multiple levels of abstraction can be identified and a top-down configuration approach with specification of attributes and structure. This is especially suitable for order processing over long time, where it is important to control the degree of freedom at different steps. It is necessary to postpone certain decision until enough requirements are available.

The proposed approach is currently under implementation at the Danish case company, Aalborg Industries. Here, the development of product family models and product configurators has been carried on for several years starting with a simple model for calculation of quotations. In later versions, data from the product configurator has been used as parameter input to other software applications for producing data sheets and drawings. This development has proved the necessity to set greater focus on product modelling on multiple abstraction levels.

The current version of the product configurator is web-based so that sales and tendering can take place everywhere around the world. This technology will also be used in the future and the company is now developing a more advanced product model and related product configurator software modules with the purpose of integrating more of the existing software.
applications and get more optimised order processing and production planning. Furthermore, supply chain management issues are taken into consideration so that decisions about selection of manufacturing locations and suppliers can be optimised. Especially, issues about interaction with ERP systems are important and require software modules for automatic interfacing.

As described for the case company, the order horizon can be rather long and many changes in the order specification occur. In addition, many modules can be purchased as products from multiple suppliers, which can deliver with a variety of properties for sizes, price, performance, quality, lead time, etc. Hence, for this company, it will be important to rise to a higher abstraction level by setting focus on specification of attributes and move away from the structural model of configuration.

Two examples from the case company can illustrate this. In the first example, alternative feed water pumps for boilers can be selected as illustrated in table 1.

<table>
<thead>
<tr>
<th></th>
<th>Delivery head Bar(gauge)</th>
<th>Capacity m³/h</th>
<th>Supply voltage V</th>
<th>Price €</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Requirements)</td>
<td>(&gt;= 22)</td>
<td>(&gt;= 25)</td>
<td>(3 x 330)</td>
<td></td>
</tr>
<tr>
<td>Product 1</td>
<td>23</td>
<td>25.5</td>
<td>3 x 330</td>
<td>1600</td>
</tr>
<tr>
<td>Product 2 *)</td>
<td>25</td>
<td>30</td>
<td>3 x 330</td>
<td>2000</td>
</tr>
<tr>
<td>Product 3 **)</td>
<td>24</td>
<td>25</td>
<td>3 x 330</td>
<td>1800</td>
</tr>
</tbody>
</table>

*) Has frequency converter drive, i.e. significantly lower power consumption  
**) Approved for running in explosion risky zones

Table 1 – Alternative feed water pumps specified with a set of attributes

The table shows that three sample requirements are specified and that three different pump products can satisfy the requirements. It also shows that additional attributes may be taken into consideration if further specifications have to be made.

In the second example, it is shown that alternative safety valves can be selected (see table 2).

<table>
<thead>
<tr>
<th></th>
<th>Set pressure Bar(gauge)</th>
<th>Size</th>
<th>Production location</th>
<th>Delivery time</th>
<th>Price €</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Requirements)</td>
<td>(19)</td>
<td>(DN50)</td>
<td>(Deliv. location: Finland)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product 1</td>
<td>19</td>
<td>DN50</td>
<td>Germany</td>
<td>2 days</td>
<td>200</td>
</tr>
<tr>
<td>Product 2</td>
<td>19</td>
<td>DN50</td>
<td>China</td>
<td>30 days</td>
<td>130</td>
</tr>
</tbody>
</table>

Table 2 – Alternative feed water pumps specified with a set of attributes

Two valve products satisfy the requirements but, as shown, with great difference between the prices. A significant attribute is the delivery time, which may set serious limitations regarding the time for procurement. However, this is dependent on the production location so, if for instance the production location is changed to the East Asia, a dramatic reduction of delivery time and price can be reached.
Two examples of abstraction by classification can also be presented (see [Jørgensen, 2008] for description of the syntax). Example one is about oil fired boilers, where the module type 'OilfiredBoiler' is the super-type for two sub-types 'MissionOS' and 'MissionOL'. Two attributes show the decision making, 'BurnerType' and 'Capacity'.

```plaintext
type OilfiredBoiler subtypeof ...
{
  BurnerType : {KB,KBO,KBE,KBSA,KBSD};
  Capacity : [1.6 .. 15.5];
}

type MissionOS subtypeof OilfiredBoiler
{
  BurnerType : {KB,KBO} default KB;
  Capacity : [1.6 .. 6.5];
  ...
}

type MissionOL subtypeof OilfiredBoiler {...}
Etc.
```

For oil fired boilers, the burner type can be any of the listed values, while for mission OS boilers only a subset of burners is valid. The capacity for mission OS boilers is similarly narrowed compared to the oil fired boilers in total.

Example two regards feed water pump units, where there are two sub-types and where the regulation type differs.

```plaintext
type FeedWaterPumpUnit subtypeof ...
{
  RegulationType : {OnOff,Modulating};
}

type FeedWaterPumpUnitOnOff subtypeof FeedWaterPumpUnit
{
  RegulationType : {OnOff};
}

type FeedWaterPumpUnitModulating subtypeof FeedWaterPumpUnit
{
  RegulationType : {Modulating};
}
```

Both examples show that the super-type modules represent decisions on a higher abstraction level because selection of a general module type establish some degree of specification while remaining decisions are postponed. In contrast, sub-types represent decisions about more precise specifications. In the sales process, it will be possible to assist the customers with decisions about how specific they must be from the beginning. A balance must be obtained. Relatively specific decisions give more precise estimations (cost, required capacity, delivery, etc.) but are most likely subject to changes and, on the other hand, decisions on a more general level will lead to uncertainty about estimations. A key issue in relationship with configuration is to develop models for calculating estimations based on different levels of abstraction in decision making.

### 9. Product Family Modelling for Manufacturing

When a product family model is represented as an information model, all possible data can be included. Often, the models focus on data, which are necessary only to perform the configuration process and provide an overview over the resulting product structure like the
one in figure 8. Sometimes, additional data are presented, e.g. selected attributes of the product or even graphic data, which can be shown in viewers.

Obviously, product family models can potentially be utilised much more, for instance as the foundation for manufacturing. This means that, when a product has been configured, the specific model for this configuration should have valuable data for efficient and effective planning and execution of manufacturing tasks. According to figure 1, the manufacturing data should be included in the product family model and carried over to the product model as a result of the configuration. In the following, these considerations are only limited to planning data but similar results can probably be shown for other applications. 

Based on such requirements for the product family model, a number of modelling issues must be considered. In the simplest form, data may be added to the existing components of the model and easily presented from these. Often, however, additional components must be created and it may be necessary to form new structures of the model. Consequently, the information modelling process may be more complicated.

6.1 Manufacturing Structures

As already mentioned, the product structure is an often occurred description of a product and usually this structure is formed as a result of configuration (see figure 8). However, such a structure can be created in many ways dependant on the purpose. This means that structures, which are suitable for users involved in configuration (by use of a configurator) may not be useful for manufacturing. Even in this context, multiple structures may be preferred.

For operational planning, the primary focus is on the flow of components and the operations performed on the components. To describe this, a rather deep tree structure may appear because each branch will represent an operation, where one or more components are taken as input and a new component is delivered as the output. For each input item, a quantity is specified. In the following, this general view is simplified so that a number of operations are collected into one operation (see figure 16).

![Figure 16 – General composition structure for manufacturing](image)

In connection with a larger product structure, this general structure can be applied to all levels, i.e. from components to components, from components to modules, from modules to modules and from modules to end-products. Attributes, necessary for description of the operations can be included in the types of the OutItem.

For the computer example, a suitable manufacturing structure different from the one in figure 8 could be the structure shown in figure 17. The operations are not shown but they
are assumed at each transition from one level upwards to the next. The example shows only the major assembly structure and not many of the underlying sub-components and their related operations. On the other hand, the assembly operations are normally the primary operations in connection with product configurations.

Observe that some types, e.g. 'Case basis' and 'Case frame', are additional types, which must be added to the taxonomy but they are assumed to have no interest for the customer's configuration process and instances will only occur in the manufacturing structure.

Design of the manufacturing structure may involve decisions about important issues like modularity, platforms and postponement and how they should be handled in connection with configuration. With reference to figure 17, for instance, the Case Basis could be characterised as a platform, which could be used equally for many (perhaps all) computers. Next, print circuit boards are good examples of modules, i.e. a result of modular product design. In contrast, processors, memory units, print circuit boards, and mass storage modules are often selected differently by individual customers in the configuration process and, consequently, they will be candidates for postponement and are placed on the same level late in the assembly operations. The lock is in the example assembled on a lower level and should perhaps be moved upwards.
To make it possible to generate the manufacturing structure and perhaps other structures for all possible configurations and transfer it to the model of the configured product requires that the product family model must have the necessary data and specifications. As stated, structures can be described by collections and an example suitable for configuration could be

Contents ->> { 1..1 CPU; 1..1 MassStorage; 1..1 PowerSupply; 0..1 Lock }

The corresponding collections describing the manufacturing structure in figure 17 for the computer 'Case' and the 'Case basis' can then be defined as this

```plaintext
type CaseBasis subtypeof OtherComponent
{
    .....  
    Manufacture ->> {CaseFrame; MotherBoard; PowerSupply; Lock;}
    .....  
}
type Case subtypeof OtherComponent
{
    .....  
    Manufacture ->> { 1..1 CaseBasis; Processor; MemoryUnit; 2..2 Panel; ... }
    .....  
}
```

Observe that it is only necessary to specify multiplicity clauses for types, which are not constrained by the configuration collections. These constraints are assume to take effect first and will rule the existence of the affected components. Moreover, the specifications in the product family model will also regulate the resulting manufacturing structure.

### 6.2 Manufacturing Planning

Data for manufacturing planning may be represented in different ways but most often, they are included in the component types as attributes and, for each specific configuration, they can subsequently be retrieved by use of the manufacturing structure. An obvious example is to create a complete overview over all the components included in the product by performing a recursive search through each branch.

Calculations can also be performed by use of the defined structures. If cost price attributes are available in each component, the total material cost can be calculated. Such a simple calculation may be formulated like this

```
Case.CostPrice = Case.CostPriceAdd + CostPriceSummation(Manufacture);
```

The attribute 'Case.CostPriceAdd' represents any extra cost, e.g. cost of extra materials such as screws, and the 'CostPriceSummation' method sum up the cost of the assembled components defined by the 'Manufacture' collection, i.e. case basis, processors, memory units, etc.

Also, to each component, additional specifications like required extra equipment (ex. screw driver), labour, and operation time can be specified. The assembly operation time for the computer case may be formulated in a similar way

```
Case.OpTime = Case.SetUpTime + OpTimeAggregate(Manufacture);
```
The attribute 'Case.SetUpTime' represents the operation time for any preparation of the assemble operation and the 'OpTimeAggregate' method calculates the assembly operation time based on contributions from individual components. To do this, it may be necessary to add specifications of possible parallel operations. Furthermore, it may be necessary to add constraints, which specify certain cross-going relationships between components. For instance, a specific assembly sequence may be required. Such requirements must also be represented in a general form, which can be handled for all possible configurations. A priority attribute e.g. could be added to the component types and the manufacturing collection could be iterated to form an ordered list according to the values of this attribute.

As stated, product family models are used for developing product configurators and the result of configurations is often just a simple list of the included components. Based on a product family model with much more data aimed for manufacturing planning, more precise structures and planning data estimates may be provided and included in models of the configured end-products. Typically, such data are submitted to the Enterprise Resource Planning (ERP) system for generation of inventory requests and job schedules. However, if there is a clear identification of components, which are manufactured independent of configuration, then the manufacturing planning of these components are traditional and suitable for the ERP system. Related to the computer example, all components except the assembly of the computer case must be manufactured independent of configuration and could be planned in the usual way by an ERP system.

In contrast, planning of the manufacturing – often just the assembly operations – may be handled much more suitable by the configurator. In this case, it would be obvious to separate the planning functionalities from the configuration functionalities and perhaps perform the configuration in two steps; first the sales configuration and afterwards the technical configuration with the manufacturing planning. Of course, exchange of data between the configurator and the ERP system will be necessary.

10. Conclusion

In this paper, it is underlined that there are some fundamental issues of information modelling, which can be applied to product family modelling. For Product family models, it is important to identify the attributes in the model of the end-products and, because some attributes in models of product families will be assigned values during the configuration process, they must be defined with optional values i.e. domains. It is also characteristic for product family models that relations/constraints must be defined between attributes of the possible end-products and the attributes of the identified modules/components.

As a basis for development of detailed information models, a generic model component is presented. Likewise, a generic component type is introduced as the basis for creation of information models. According to this type, the basic content of product family models is proposed in form of a module type and a simple synthetic language is presented. The use of this module type is illustrated by a number of examples.

In the paper, special focus is set on how to develop product family models, which can support product configuration on multiple abstraction levels – suitable for some engineer-to-order companies with long order horizons. First of all, it is proposed that configuration is performed by specification of attributes instead of selection of modules. This means that the structure of end-products is defined indirectly based on the values of attributes. Thereby, configuration is also oriented towards customer needs because attributes are essential in connection with the functional demands from customers. Further, it is proposed that, when
modules are selected, it is important to develop classifications of module types and form a taxonomy. Such a structure is well suitable for identification of multiple abstraction levels by classification, where specifications can range from a general level to a more specific level.

The aim of developing product family models is that they can be used as a foundation for development of specific product configurator software and the proposed methodology, included in this paper, is for the moment being used by a particular ETO company, which intend to develop an advanced product family model and a product configurator that can support many organisational functions in the company world wide. Especially, the top-down approach with modelling on multiple abstraction levels are followed very closely and considerable amount of specially designed software modules are being developed.

References

[Array 2003]  

[Andreasen 2003]  

[Berman 2002]  

[Bosch 2000]  

[Davis 1989]  

[Du et al. 2000]  

[Faltings 1998]  

[Felfernig et al. 2001]  

[Gilmore and Pine 1997]  

[Gilmore and Pine 2007]  

[Hammer 1978]

[Hvam 1994]

[Hvam 1999]

[Jazayeri 2000]

[Jiao et al. 1998]

[Jørgensen 1998]

[Jørgensen 2002]

[Jørgensen 2003]

[Lampel and Mintzberg 1996]

[Männistö 2001]

[Møller 1995]

[Pine 1993]

[Pine et al. 1993]

[Pine and Gilmore 1999]

[Reichwald et al. 2000]

[Rosch 1978]

[Sabin 1998]

[Silveira et al. 2001]

[Smith 1977a]

[Smith 1977b]

[Soininen 2000]

[Sowa 1984]

[Sulonen 1998]

[Yu 1998]