Modeling Products for Versatile E-Commerce Platforms - Essential Requirements and Generic Design Alternatives

Ulrich Frank

University of Koblenz, Rheinau 1, 56075 Koblenz
ulrich.frank@uni-koblenz.de

Abstract. Electronic commerce (e-commerce) covers a wide range of business transactions. Usually, a transactions includes products, i.e. goods or services. Often, e-commerce websites are intended to offer a large variety of products, most of which are not fully recognized at the time when the system is constructed. Nevertheless, there is need to describe these products in an elaborated way in order to support prospective customers in finding adequate products. Since new product types may have to be registered on a daily basis, changing a data base schema or program code is no satisfactory option. Instead there is the need to register new product types during the run time of a system. In addition, the representation of product variants and the construction of individual product configurations should be possible. Current approaches to represent products do not fulfill these requirements to a satisfactory degree. The design of appropriate conceptual models requires to thoroughly differentiate between various levels of abstraction. This includes decisions to be made between instantiation, specialization and "uses" relationships. Against this background, the paper presents three prototypical design alternatives for versatile product representations. They vary in terms of flexibility, reusability and ease of use. The work presented in this paper resulted from the design of a reference model for e-commerce platforms.

1 Introduction

While electronic data interchange (EDI) is restricted to the transmission of data within traditional business models, e-commerce allows for entirely new ways of doing business. Among other things, we witness the rise of new kinds of intermediaries that combine trading and brokerage with purely internet based, non-personal acquisition and sales, e.g. companies that offer platforms for electronic auctions or electronic marketplaces in general. We use the term "e-commerce platform" as an abstraction of information systems that support this variety of actual internet businesses. Unlike traditional inventory management, an e-commerce platform is aimed at covering the initiation and performing of business transactions on the Internet between participants that possibly do not know each other. Companies that run e-commerce platforms face a number of remarkable challenges, including the development of lasting customer relationships, the establishment of a competitive profile and becoming well known in relevant markets with a reasonable advertising budget. Additionally, a pivotal success factor for e-commerce platforms to perform economically is a powerful software architecture. A couple of years ago, a few fancy web pages, and impressive sales forecasts, were sufficient to qualify as a serious e-commerce contender - no matter whether there was only a quick and dirty implementation under the hood. Times have changed. For e-commerce platforms to be enablers of more efficient business transactions, they have to offer more than allowing customers to submit orders through web browsers. In general, they should promote the automation of business processes: even when e-commerce seems to focus at first sight on the generation of business over the Internet (and to which some current offers actually limit themselves), economic reasons suggest an integration and extensive automation of all involved processes, such as procurement, logistics and financial transactions. Also, an e-commerce platform should allow for representing a wide range of products. This is due to the fact that the diversity of products offered over e-commerce websites on the Internet is sometimes considerable; in some cases - for example in auction platforms – it is almost unlimited. In contrast to department stores or mail order companies that also often offer a wide array of products, the range cannot be planned in the medium term. Instead, it must be possible to make diverse, almost totally unpredictable changes to the product range on a daily basis; today, leather seats; tomorrow, helicopters. In order to support a tight integration with other systems in the supply chain, products must be described meaningfully with regard to corresponding use cases. Therefore, it is all about conceptualizing objects of which one has, up to that point, no knowledge. This paper is aimed at conceptual solutions to overcome this seeming paradox.

2 Requirements and State of the Art Solutions

To solve the problem outlined above, we need appropriate conceptual models of products. A conceptual product model does not only guide the implementation of software, it is also a blueprint for the specification of communication interfaces. Before we consider current approaches to represent products in information systems, we will have a more detailed look at the requirements a competitive e-commerce platform should fulfill.

1. In principle, it should be possible to represent any products (or more exactly: types of products).
2. The registration of new product types should not require a change of the program code or of the database schema, because in view of the general availability of the system and the frequent appearance of new product types this would not be acceptable.

Other requirements emphasize the customers viewpoint:

3. The objects stored on the basis of conceptual product models should support the customer when searching for suitable products.

4. With this, the description of products from the point of view of the customers should have a sufficient content and be fairly set out for all their various needs in a detailed and clear form. This requires, for instance, that both a customer searching a particular type of furniture and another one searching for an excavator should be able to specify relevant features. Common search engines are usually not satisfactory. They operate on full text representations that do not include product descriptions as semantic structures. For example, imagine to search for a dining-table with a glass top and mahogany table-legs. While it would not be possible to express exactly what you are looking for, a search engine would retrieve pages that contain both glass and mahogany tables but maybe none that satisfies the relevant search criteria.

Besides that, there are a series of requirements that are connected to the automated use and care of product data:

5. The product descriptions should allow for the collection, over a period of time, of relevant marketing knowledge about the purchasing behavior of customers. This requires the analysis of buying preferences of customers - grouped by demographic variables - with respect to specific features of products.
6. The system should extensively shut out the registration of meaningless product descriptions. That implies to define products types that restrict the possible descriptions of corresponding instances.

The same is valid for the registration of customer orders:

7. It should be possible to represent product variants as such, since in this way not only can the analysis of purchasing habits mentioned above be supported, but also redundancy of data registration and use can be avoided: Features that have been described for a product type already do not have to be described again when it comes to specify corresponding variants.
8. The customer should be in the position to specify individual configurations, e.g. by assigning equipment to a car. To avoid orders that cannot be satisfied, it is essential to allow for correct configurations only.
9. Different forms of pricing and price assigning (for product types and single products) should be possible.
10. The product descriptions should be versatile enough to satisfy diverse communication interfaces (e.g. to customers, suppliers, banks, and logistical partners).

The representation of products is subject of a number of EDI protocols, such as UN-EDIFACT. Typically, these protocols are aimed at the specification of business documents like orders, invoices etc. They do, however, allow for the use of product identifiers only, thereby relying on the participating parties to share a common understanding of these identifiers. Current standardization initiatives that are aimed at fostering e-commerce transactions, like Open Buying on the Internet [OBI99] or RosettaNet [Rose99], also do without concepts of products. Instead, the XML document types they define, include tags for product identifiers only. This is the case, too, for document types that are propagated by companies which operate marketplaces on the Internet ([Arib00], [Comm00]). Probably the most common approach to describe product types are data structures that include features that all product types have in common, like Name, Description, Price, Picture etc. The differentiation between various product types follows solely through specific initialization of the generic features. While such flat descriptions are very flexible in the sense that they allow for the representation of any kind of product without the need for changing code (requirements 1, 2), their support for searching products is limited to text retrieval (requirements 3, 4). At the same time, it must be taken into account that a product concept of this kind does not support the differentiated analysis of consumer behavior, since the product types that are offered cannot be distinguished conceptually (requirement 5). Requirement 6 marks an inherent weakness of this concept, since in conceptual terms almost all nonsensical product descriptions are possible. Therefore it jeopardizes the integrity of a system. Specifying product variants is not feasible on a conceptual level since it is not possible to show which characteristics distinguish one variant from another (requirement 7). For similar reasons it is not possible to represent product configurations (requirement 8). Information on pricing and for logistical purposes (requirements 9, 10) could be added. Class hierarchies, where each class represents a specific product type, would allow for more elaborate descriptions of product types. For our purpose, however, it is not satisfactory because the introduction of a new product type would require to define a new specialized class, hence the modification of code (requirements 1, 2). There are a few approaches that allow for more meaningful, yet flexible definition of products. Vendors of e-commerce software, like BEA or Intershop, supplement flat descriptions of product types with entity types that are in fact meta types. They allow for the specification of additional attribute types (www.bea.com, www.intershop.com). BMEcat, an intended XML based standard protocol for B2B transactions, uses a similar approach ([HRS99], [SKP+01]). While all of these approaches allow to describe new product types without changing code, none of them allows for the specification of configurations or variants. Also, their focus is mainly on the definition of exchange protocols not on providing for a conceptual foundation of corresponding information systems. Also, none of these approaches is based on a thorough differentiation of the levels of abstraction to be taken into account.

3 Levels of Abstraction

To prepare for the representation of products, it is helpful to clarify the levels of abstraction one has to deal with - and to contrast them with the levels of abstraction that are available in state of the art software architectures.

3.1 Instantiation, Specialization and Re-use

The definition of new product types requires an appropriate language. In order to avoid definitions that are not compliant with an acceptable notion of product, the "language" can be restricted to a meta model that defines the set of all syntactically valid product type definitions. The meta model itself could be specified using the ER model or the UML. A simple meta model of this kind could, for instance, consist of two entity types as illustrated in fig. 1. Sometimes, it will not be satisfactory to define a product type from scratch by instantiating it from a meta model. Instead, it may be more appealing to specialize it from an existing product type.

Fig. 1 includes an example of a specialized product type. Notice, that in reality there is no fixed limit to the number of levels. Specialization seems to be well suited for any kind of is a relationships. However, it recommends a second thought with respect to the semantics of specialization or inheritance. While in natural language, there is no unique concept of specialization, one interpretation seems to be prevailing. It is based on an extensional notion of a class, which defines a class as a set of instances. Hence, a specialized class is a subset of its superclass. Database theory usually suggests a similar notion of specialization. This is, however, different with common object-oriented programming language that are usually the implementation language of choice. They feature an intensional notion of a class where a class is defined by a certain structure. A specialized class inherits this structure. Different from the extensional concept, an object can be an instance of one and only one class. Applying the intensional concept to product types reveals yet another challenge. Sometimes, the instances of a specialized type may have features with the same state as the instances of the corresponding supertype. Take, for instance, a special edition of a certain type of TV set. Its features (screen size, weight, etc.) have the same value as all the other instances of the type it is specialized from. In addition to these, it includes a computer system that allows for internet access. It is not possible to express this "re-use" of state through specialization that is based on an intensional concept of a class: you would have to instantiate a new instance of the specialized class without any inherited state (for a detailed discussion see [Fran00]). In fig. 1, the term cloned is used to denote this kind of relationship. For many products, there is no need to describe product instances. Sometimes, however, a customer may want to pick a particular instance. This is especially the case for used products. When it comes, for example, to consider the price of a used car, its particular state has to be taken into account. At first sight, this would suggest to represent particular products as instances of a certain product type. However, similar to the specialization into limited edition discussed above, instantiation is not satisfactory. Many of a used cars features will have the same state as all cars of the corresponding type. Using a class (or a type respectively) and its instances does not allow to express this information: Firstly, a class does not allow for storing states that apply to all instances (that would require meta classes). Secondly, the state of an instantiated object is on a different level of abstraction than the state of its class.

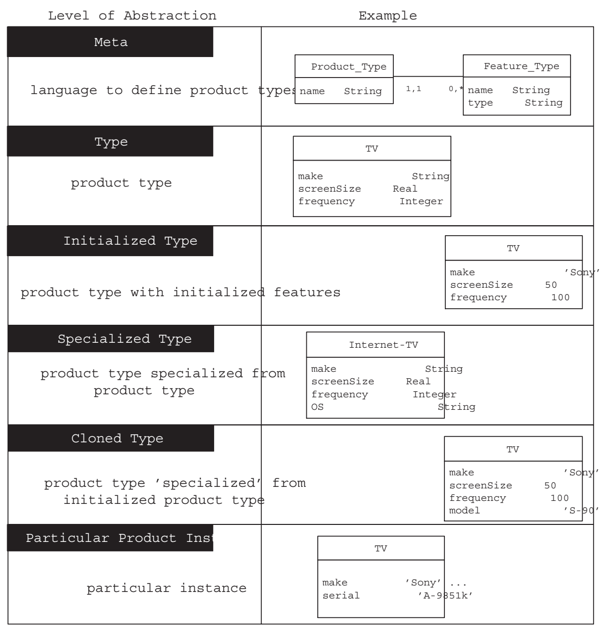


Fig. 1. Levels of abstraction concerning the representation of products

3.2 Layers in Common Software Architectures

So far, we have seen that there are many levels of abstraction that have to be taken into account with respect to the representation of products. Also, the relationships between the various levels cannot always be clearly classified as instantiation or specialization. To make things even worse, we have to consider the limitations of common software architectures to represent the abstractions we have identified so far. Usually there are two main levels of abstraction in addition to the implicit language layer. The "meta layer" allows for the definition of concepts such as classes or relation types. The "object layer" serves to store the corresponding instances. Avoiding the modification of code for the purpose of introducing a new product type implies to reserve the meta layer for the definition of a meta model of product types. Product types would then actually be instances on the object layer. This has two implications. Firstly, there is no way to directly use specialization relationships between product types because they can be applied to classes only and not to instances. Secondly, it is not possible to instantiate objects representing particular product instances from product types because an instance cannot be instantiated from an instance. Our brief discussion of the abstractions to be taken into account as well as the concepts available in state of the art software architectures shows that a versatile representation has to face two essential challenges. On the one hand, it is necessary to clarify the levels of abstraction and the corresponding levels of re-use (re-use of concept, re-use of state) that are needed to represent a certain population of products. On the other hand, these abstractions have to be reconstructed using the restricted set of concepts provided with common software development environments.

4 Three prototypical Design Alternatives

It would be nice if there was one representation of products that would fulfill all requirements (see 2) to a satisfactory degree. However, our work on the design of an object-oriented reference model for e-commerce platforms suggests a differentiated approach. Goals like flexibility and re-usability are in part conflicting and come with additional trade-offs like compromising integrity and ease of (re-) use. Depending on the products to be represented for an e-commerce platform, the relevance of these goals varies. Therefore we introduce three prototypical product models each of which is suited for a set of products that meets certain conditions. Emphasis on Versatile Definitions of Product Types and Configurations Some product types offered on e-commerce platforms allow for individual configurations or come in different variants. The object model shown in fig. 2 allows for a fairly flexible specification of configuration options. At the same time it is limited with respect to the re-use of product types (conceptual re-use) and instantiated type features (instance level re-use). Hence, it is suited for domains where similarity of product types is of no concern - because product types do not have features in common or it would be to expensive to consider similarity when introducing a new product type. To take advantage of the additional flexibility offered by meta concepts within common architectures, it is necessary to overload layers with more than one abstraction. Some of the concepts shown in fig. 2 are part of a product meta model (e.g. Standard_Feature_Type, Optional_Feature_Type). Others are directly related to certain product types (e.g. Feature_Value, Configuration_Type). As a consequence, the corresponding instances cover two levels of abstraction, too. This model is valid for all conceivable product types. Product_Type serves to represent a particular product type such as BMW 325 i or IBM Thinkpad T21. To introduce a new product type, an object of Product_Type has to be instantiated first. Then one would assign the necessary number of feature types either instances of Standard_Feature_Type or Optional_Feature_Type. Features may be available in various form (color, size, etc.). The range of available feature forms can be specified by assigning corresponding instances of Feature_Value. Associations between feature types (replaces, requires) allow one to express configuration constraints. All the requirements formulated in section 2 are fulfilled using this approach, whereby pricing mechanisms are abstracted. However, one must consider that only a part of all conceivable configuration constraints can be expressed (through the association between Optional_Feature_Type and Feature_Type). For instance, it would not be possible to specify that a sun roof is available only for black cars, because that would require one to refer to the state of a Feature_Value, not just to the existence of another feature. However, a limitation of this kind is not a serious drawback in view of the intended scope of application. It would not be economic anyway if a trading site reproduces the detailed configuration possibilities

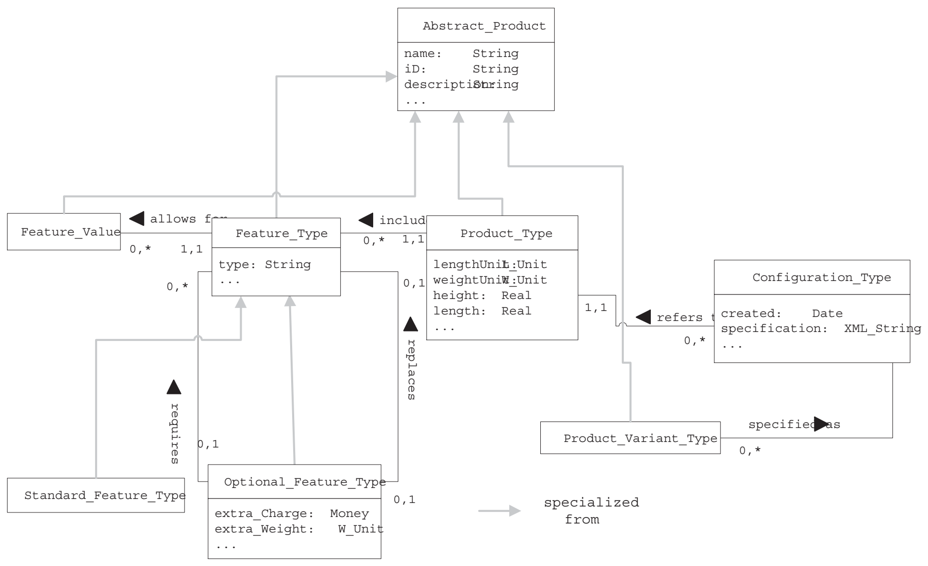
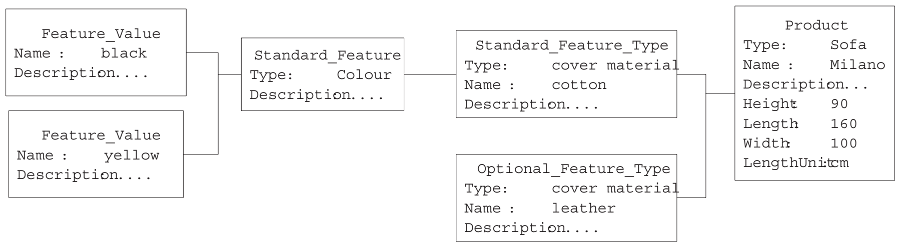


Fig. 2. Prototypical product model #1

of a complex product (a car, for example) in the same detail as the respective manufacturer. Therefore, for complex models the model is suited to describe subsets of all possible configurations. All individual configurations are managed as instances of Configuration_Type, whereby the attribute specification serves to describe an individual configuration in a suitable language (for example as an instance of a corresponding XML document type). Figure 3 clarifies this fact with an example. It represents standard and optional features of a particular sofa type and thereby expresses the set of possible configurations. In addition to that it shows an excerpt of a particular configuration that is represented as an XML string, which would be stored with an instance of the class Configuration_Type. Even when this approach supports a high degree of flexibility and concurrently a high level of integrity, it also has drawbacks. They are mainly related to the lack of support for re-usability. On the concept level, it is not possible to express that a product type is a specialization of an existing type. On the instance level, too, it may be helpful to re-use existing instances. Take, for example, certain makes of Feature_Type, like a particular car stereo make. While this is an important part of a product description, it could not be represented in an information system that is based on the model in fig. 2. The general feature type car stereo would be represented by an instance of one of the subclasses of Feature_Type. From a conceptual viewpoint, a particular make would be an instance of this instance - which is not possible. As an alternative, one could use instances of Feature_Type to represent particular makes. Unfortunately, this approach would result in a large amount of conceptual redundancy.



```
<Configuration>
<Product_Type> Sofa </Product_Type>
<Optional_Feature0>
        <Type> Cover Material </Type>
                <Name> Leather </Name>
        <Standard_Feature>
                <Type> Colour </Type>
                        <Feature_Value>
                                <Name> black </Name>
                        </Feature_Value>
```

Fig. 3. Exemplary instancing of the model with corresponding specifications of a particular configuration using XML

4.1 Emphasis on Re-Use

In cases where the shortcomings of the previous model are not acceptable, the representation of products should allow for expressing re-usability associations - both on a conceptual and an instance level. For this purpose, the object model in fig. 4 introduces additional concepts. They allow to represent product types as well as feature types on a higher level of abstraction. The model presented in fig. 4 requires to describe product types on a specific level, e.g. BMW 320i, BMW 325i etc. This is also the case with feature types, e.g. the particular make of a car stereo. As outlined in 2, a more abstract representation would offer better chances for re-use. One would, for instance, introduce a product type Car which could then be specialized into BMW 3 series and further on into the concrete product type BMW 325i. The class Concrete_Product_Type is intended to represent concrete product types. Similarly, one would represent feature types on a more abstract level, for instance Car Stereo in general. A particular make of car stereo could then be instantiated from a corresponding type. The subclasses of the abstract class Feature serve to represent concrete features. Since many instances of Concrete_Product_Type may be assigned to an instance of Product_Type, particular features (instances of Standard_Feature and Optional_Feature) have to be assigned to an instance of Concrete_Product_Type. Unfortunately, the model would still allow to assign features to a product that are not assigned to feature types of the corresponding product type. Therefore additional constraints are required (see fig. 4). Notice that there is no need to link a feature to every feature type. Sometimes, the description of a feature type will be sufficient. Take, for instance, an air conditioning system in a car. To support conceptual re-use, the model includes a recursive association with Product_Type that indicates a specialization relationship. At the same time the description of a concrete feature may require a more elaborate specification. For instance: the size of wheels. Since the relevant attributes of feature types may vary, there is need for an additional meta abstraction. It is provided with a meta attribute, i.e. an attribute that serves to specify a structure. The meta attribute feature_Details is specified by the class Attribute_Spec. It contains two tuples each of which contains the denominator of an object level attribute and the type of this attribute. The default value of this attribute contains the tuples (Name, String), (Description, String) and (Photo, String), which can be used to describe a particular feature (like an alloy wheel) on the object level. Since the structure of a feature type may vary, it is not possible to specify the structure of a particular feature in advance. Therefore a particular feature, stored as an instance of Standard_Feature or Optional_Feature, is specified as an XML document within the attribute StrucDesc. The structure of this document is defined with the corresponding instance of a subclass of Feature_Type as an XML document type definition (attribute DTD in fig. 4). Notice that an instance of Standard_Feature_Type or of Optional_Feature_Type may be associated with many instances of a corresponding subclass of Feature to express that there are options for specifying individual configurations. A product type consists of

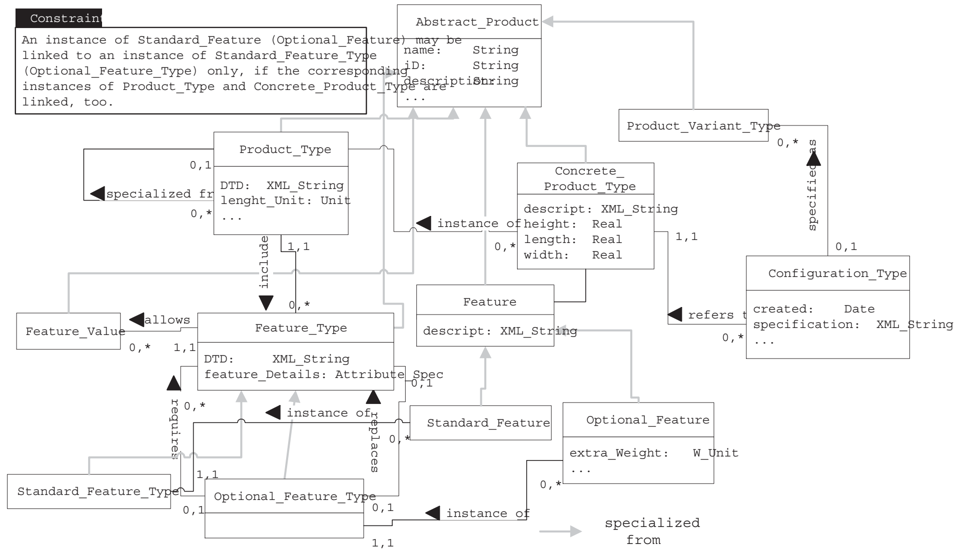


Fig. 4. Prototypical product model #2

feature types. With the specific structure of a feature type being defined in a corresponding DTD, the overall structure of a product type can be expressed as a DTD, too. It is composed of the DTDs of the associated feature types. A concrete product type, instance of Concrete_Product_Type, is then described by a number of general attributes, like Width, Height etc. and an XML string that contains a description of all features that are available with this concrete product type. Similar to the previous models, a particular configuration would be specified by an XML string that specifies all selected features of a corresponding concrete product type. Notice that we do not allow for the specialization of feature types, because this could result in the notorious covariance problem ([Meye97], pp. 621). This model fosters a higher level of re-use than the previous one. Nevertheless, it comes with two disadvantages. The first one is related to its complexity that is a threat to a systems integrity for two reasons. The implementation of additional contraints increases the chance of software bugs. At the same time, using it in appropriately demands abstraction skills that may possibly not be expected of system users. While some of this complexity may be hidden from users, the second disadvantage is to accept only, if the similar product types share the same standard and optional features. Otherwise, it is not possible to take advantage of the specialization association. Cars, for instance, will usually not fulfill this condition: a feature that is standard for one product type may be an optional feature for another type of the same kind.

4.2 Compromising Flexibility for Re-Usability: A Pragmatic Approach

The main problem we encountered with the previous model is caused by the variance of product types concerning possible configurations, i.e. their standard and optional features. The following model offers a pragmatic solution to the obvious conflict between flexibility (in terms of expressing configuration rules) and re-usability. Our analysis of products has shown that for many product types limited means to define configurations are sufficient. Therefore, the model compromises flexibility for re-usability. In contrast to the previous models, it does not distinguish between optional and standard feature types. Instead it includes only one Feature_Type (see fig. 5), which contains an attribute default that allows for expressing whether a feature type is mandatory (like wheels). Additionally, it contains the meta attribute Instance_Extra that is specified by the class Attribute_Spec (see 4.2). It contains two tuples each of which serve to store the denominator of an object level attribute and the type of this attribute. For example, (Extra, Boolean) indicates that the object level attribute Extra is intended to store a boolean value that specifies whether the corresponding feature is optional or standard. To increase the level of re-use, an instance of Feature is not linked to an instance of Concrete_Product_Type. Instead, the instances of Feature to be used with a particular instance of Concrete_Product_Type have to be specified within the XML string (attribute descript in Concrete_Product_Type). Similar to the previous model, the structure of a product type as it can be deferred from the corresponding feature types is stored as an XML DTD with the attribute DTD of Product_Type. Also, the structure of the corresponding product type is stored as an instance of this DTD. This model allows for a high level of re-use both on the conceptual level and on the instance level. The first being fostered through the recursive specialization association with Product_Type, the second

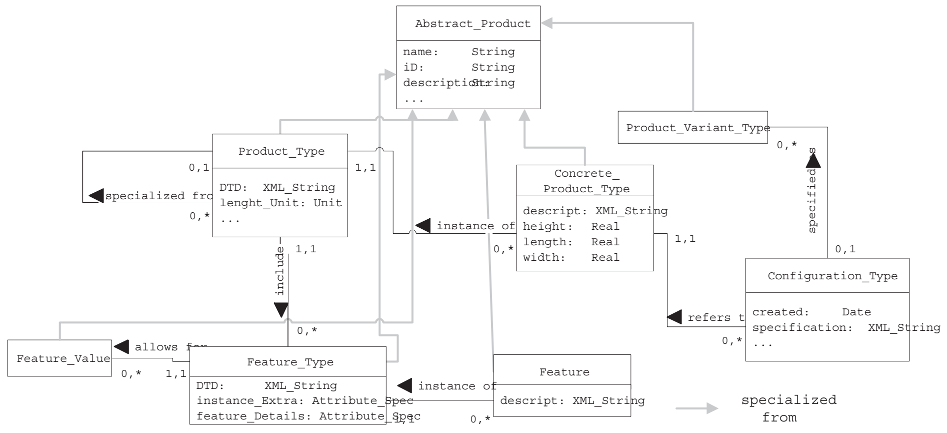


Fig. 5. Prototypical product model #3

by introducing meta attributes and corresponding XML strings. In comparison with the previous model, this model offers limited ways to describe configurations. While this may not be a serious disadvantage for many e-commerce platforms, the introduction of meta attributes and XML strings has its price: it certainly increases the complexity of a database that is instantiated from this model. Firstly, for a user to enter data, it is demanding to understand the various levels of abstraction. Secondly, for retrieving product types with certain features, using standard database search procedures is not sufficient. In addition to that, it is necessary to parse XML strings, too. While the first disadvantage can be compensated for to some degree by a user interface that hides most of the complexity, the second disadvantage can be reduced by using standard XML tools.

5 Conclusions and Future Work

In this paper we presented three prototypical conceptual models to represent products for e-commerce platforms. While all of them fulfill the requirements discussed in 2, each of them comes with specific strengths and drawbacks. Table 1 presents a comparison of the three models. Varying configuration rules refers to a models ability to handle varying configuration rules within the concrete product types that are instances of a product type the configuration rules were defined for.

The concepts for product modeling presented in this article were created during the development of a reference model for trading platforms on the Internet. The reference model exists currently as an UML object model with approximately 100 classes that is accompanied by numerous models of corresponding business processes. The representation of products within the reference model is similar

Table 1. Comparison of Models

	model #1	model #2	model #3
generalisation of product types	-	+	+
rich definition of configuration rules	+	+	o
re-use of feature types	-	-	o
re-use of features	-	+	+
varying configuration rules	-	+	+

to the last prototypical model. It covers, however, more aspects, like pricing, composite objects (to represent e.g. a computer system consisting of a PC, a screen and a printer) and packages. Our work on conceptual product modeling is part of the research project ECOMOD (Enterprise Modeling for E-Commerce) that is funded by the German National Science Foundation. Apart from this application area we encountered similar problems with abstractions required for the design of knowledge management systems ([3]) as well as for the design of modeling tools. Therefore our future research is also aimed at investigating more general design patterns for dealing with multiple levels of abstraction.

References

[1] Ariba: *cXML Users Guide. Vers. 1.1* Boston, London: Artech House (1998)

[2] Commerce One: *cXML Users Guide. Vers. 1.1* (2000) (www.cXML.org)

[3] Frank, U.; Fraunholz, B.; Schauer, H.: *A Multi Layer Architecture for Integrated Project Memory and Management Systems.* In: Khoshrow-Pour, M. (Ed.): Managing Information Technology in a Global Economy: Proceedings of the 2001 Information Resources Management Association International Conference. Hershey; London; Melbourne; Singapore: Idea Group Publishing (2001) 336–340

[4] Frank, U. *Delegation: An Important Concept for the Appropriate Design of Object Models. In: Journal of Object-Oriented Programming.* Journal of Object-Oriented Programming. Vol. 13, No. 3 (2000) 13–18

[5] Frank, U. *Applying the MEMO-OML: Guidelines and Examples.* Arbeitsberichte des Institut fr Wirtschaftsinformatik der Universitt Koblenz-Landau, No. 11, Universitt Koblenz-Landau (1999)

[6] Frank, U. *Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modelling.* Arbeitsberichte des Institut fr Wirtschaftsinformatik der Universitt Koblenz-Landau, No. 4, Universitt Koblenz-Landau (1997)

[7] Hmpel, C.; Renner, T.; Schmitz, V.: *BMECat Specification. Version 1.01.* Technical Report, Universitt Essen (1999)

[8] Keller, A.; Genesereth, M.: *Using Infomaster to Create a Housewares Virtual Catalogs.* The International Journal of Electronic Commerce and Business Media, Vol. 7, No. 4 (1997) 41–44

[9] Meyer, B.: *Object-oriented Software Construction.* 2. Ed., Prentice Hall (1997)

[10] OASIS, UN/CEFACT: *Electronic business XML (ebXML).* Technical Architecture Specification. Draft v 0.6.5 (www.ebxml.org/working/project_teams/technical_arch/) (2000)

[11] OBI Consortium: *Open Buying on the Internet*. Draft v 0.6.5 (www.ebxml.org/working/project_teams/technical_arch/) (1999)

[12] Schmitz, V.; Kelkar, O.; Pastoors, T.; Renner, T.; Hmpel, C.: *Specification BMEcat: Version 1.2* Technical Report, Universitt Essen (2001)