Product family based coordination for design and production in the supply chain

H.J. Pels and K. Cheng
Section Information &Technology, Faculty of Technology Management, Technische Universiteit Eindhoven

Abstract

Product families are an important concept to manage increasing diversity in products. If coordination in the supply chain is to be based on product families, new complexities occur. Since product family is a rather abstract concept, a formal model is needed to describe it. This paper proposes object oriented data modelling as a tool to describe product data families. Also it uses a semi-formal database notation to express examples in a narrative form. The paper shows that this is an effective way to discuss abstract concepts in a precise way. A finding from the discussion is that, when partners in the supply chain establish supply contracts on product families, it is advisable that each partner has his own representation of that family, while the contract links the needed and the offered family and its parameters.

Keywords

Product family, data modelling, UML, design coordination.

INTRODUCTION

Product-family-based production is an effective approach to meet the diversified requirements from different customers. It enables to treat sets of different products as one planning item in order to simplify production planning, and it reduces the complexity of storing, maintaining and using product and production related data and thus enables more effective product data management. A typical approach of product family design is to configure different product variants around a common platform (Meyer & Lehnerd 1997), that largely decides the relevant technology. Platform-based production makes it possible to reuse the production line, equipment, resources, and knowledge inherited from the past generations of the product. Due to the attractiveness of this production paradigm, a number of research activities have been towards this direction (Erens 1996; Frank van der Linden & Muller 1995) and many research outputs and practical technological deployments have been achieved (Sabin and Weigel 1998; Hegge, 1995; Ulrich 1995; Barker & O'Connor 1989).

As we know, more and more products are manufactured by the joint-effort of different companies which are doing business in a supply chain. This type of virtual organization of such companies is also called extended, virtual or networked enterprise (Browne, *et al.* 1995). If product-family-based production is applied in

H. S. Jagdev et al. (eds.), *Collaborative Systems for Production Management*

such extended enterprises, then there are some special research issues considering the product information representation and coordination between the partner companies in the supply chain. Understanding relationships between evolving product models during product development is already difficult. Adding the additional abstraction level of product families (Mannisto, *et al.* 2001) and the additional complexity of sharing product data between partners in the supply chain, one touches on really complex material (Goossenaerts 1998; Cheng, *et al.* 2002, Zhang & Li 1999). Data models are a powerful means to maintain precision of reasoning in such complex situation. This paper approaches these issues, by analysing different situations of cooperation and expresses some relevant issues in a UML data model (Rumbaugh, *et al.* 1099, OMG 1999). This model will be built gradually in the course of the paper. The analysis will be illustrated with a running example, expressed in proper database contents.

CHARACTERISTICS OF PRODUCT FAMILIES

In normal repetitive production, product types are defined and series of identical products are produced from each product type. Even if different product types are quite similar, like shoes of the same model but different in size, the smallest difference in specification requires a specific product type with a complete product specification and, for production control, a specific bill of material. In a product family the differences between variants are characterized in parameters with defined value ranges, like for the possible sizes of shoes. Then for production planning and control a whole family of variants can be treated as a single planning item, while for each production order (batch size 1 ... n) the particular values for each parameter are specified. Each product family model will contain a set of constraints that define which combinations of parameter values are allowed (Hegge 1995). A product configurator can then be considered as a function that, for each product family model, maps every possible combination of parameter values on a specific bill of material and production process description.

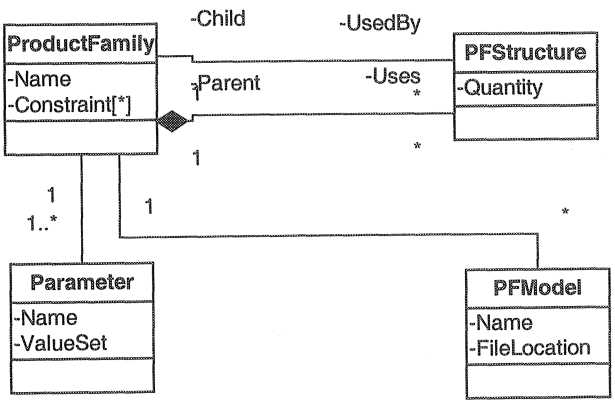

Figure 1 - Typical product family structure.

Figure 1 shows the typical data model for a product family. As an example we take a company that produces desktop PCs. A particular family of desktops is named PC2002. We denote such facts as quasi formal database statements of the form:

<pf1 is a ProductFamily with Name PC2002>

This expression states that the database contains an object pf1 that represents the product family named PC2002. This family has a disk drive family as component:

<pf2 is a ProductFamily with Name HD2002>,
<pfs1 is a PFStructure with Parent pf1, Child pf2 and Quantity 1>.

Disk drive family HD2002 has capacity as a parameter:

<prm1 is a parameter of pf2 with name capacity and value-set {10Gb, 20Gb, 40Gb}>.

The details of the product families are described in product-family models. The content of these models is supposed to be in computer files:

<pfm1 is a PFModel with name 'PC2002 3D model' and FileLocation C:
\design\pc2002\assy2002.dwg>
<pfm2 is a PFModel with name 'PC2002 assembly instructions' and FileLocation:
\design\pc2002\instr2002.doc>

The constraints of a product family restrict the allowed combinations of values for different parameters. Since this example so far has defined only one parameter, we cannot yet give realistic examples of constraints.

ORDER FROM CATALOGUE

Contracts on product families

Now in the case of extended enterprises problems may arise because of differences in parameter definitions and constraints between different partners in the supply chain. This means that the model must be extended with the possibility to assign product families to companies and to link these product families with purchase contracts. This extension is shown in Figure 2.

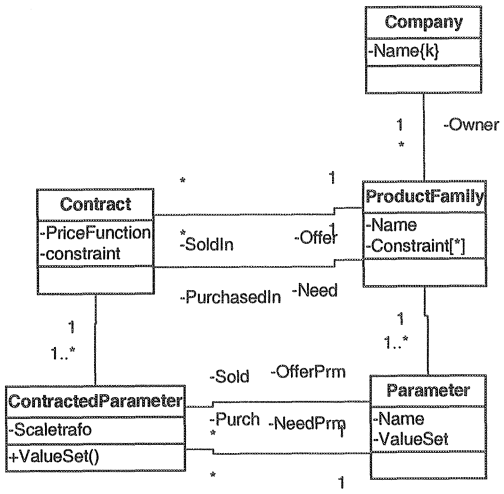

Figure 2 - Contracts between product families of different companies.

In Figure 2 each product family has an owner. A company defines product families for the products it offers to the market, as well as for the products it needs. Ownership in this context means that the owner is the company that decides about the definition of the product family:

<c1 is a Company with name A>,
<pf1 has owner c1>,

A second company named B produces a family of disk drives:

<c2 is a Company with name B>,
<pf3 is a Productfamily with Name HD001 and Owner c2>,
<prm 2 is a Parameter with Name capacity and Valueset {10Gb, 20Gb, 30Gb, 40Gb}>,
<pf3 has Parameter prm2>.

In the context of this paper we suppose that supply contracts between companies are on product families. A contract then is an agreement between a customer company and a supplier company that matches a product family as needed by the customer company with a product family as offered by the supplier company. This means that in the data model a contract relates two product families: the offered family and the needed family. It is essential that the customer and the supplier each define their own product family: one as needed and the other as to be offered: they each own their own view on the product family. A contract could now be agreed to supply the HD001 product family as fulfilment of the needed product family HD2002:

<ctr1 is a contract with Offer pf3 and Need pf2>.

Now the parameters of both product families must be matched explicitly, since parameter names probably will be different between companies. For this purpose the

model provides the object class ContractedParameter that links two parameters in relation to a specific contract:

<ctp1 is a ContractedParameter with Contract ctr1>,
<ctp1 has OfferPrm prm2 and NeedPrm prm1>.

The general meaning of this link is that the intersection of both value sets is the value set of the contract. This set, eventually further contrained by the constraint in the contract, becomes available as the result of the procedure ValueSet. Part of the contract is to specify the price of each possible variant. This is expressed in the attribute PriceFunction. During negotiation of this price function it appears that both companies cannot agree on the price of the 10Gb variant. Therefore A decides to find another supplier for that one. This is expressed as a constraint in the contract:

<ctr1 has constraint [10Gb ∉ self.ValueSet]>.

In their search for a supplier for the 10Gb variant company A finds company C

<c3 is a company named C>,

who supplies a product family of hard disks including a 10Gb variant:

<pf4 is a product family with name HDx01, owned by c3>
<pf4 has parameter prm3 with Name capacity and valueset {10Gb, 20Gb, 30Gb}>.

Negotiation leads to a contract for this disk at a satisfactory price:

<ctr2 is a contract with offer pf4 and need pf2>
<ctp2 is a ContractedParameter with contract ctr2, OfferPrm prm3 and NeedPrm prm1>
<ctr2 has constraint [self.ValueSet = {10Gb}]>.

Completeness of contracts

For the buying company it is important that all variants of a bought product family are covered in one or more contracts. This can be expressed as the following database integrity constraint:

∀ pf ∈ ProductFamily
[pf.PurchasedIn ≠ ∅
⇒ ∀ prm ∈ pf.Parameter [prm.ValueSet =
∪{prm.Purch.ValueSet}
].

The contracted parameter also defines a function ScaleTrafo that maps the values as defined by the supplier on the values as defined by the customer. This is to cover situations where both parties use different units for the parameter values. In reality more issues will be addressed in the contract, like delivery time or critical product specifications, but these are left out of the example in order to keep it simple.

An order calls off the delivery of a number of copies of a specific variant, as defined by a specific choice of parameters. The data structure in Figure 3 shows how orders for variants of contracted product families are specified. Suppose that

company A want 100 20Gb disk drives to be delivered by the 10th of September 2002, then this would be registered in the database as:

<o1 is an order with Ordernr 100920021, contract ctr1, Quantity 100 and Duedate 2002/09/10>,
<prc1 is an ParChoice with value 20Gb, order o1 and ContractedParameter ctp1>.

Since orders are issued by the customer, the values are expressed in his unit of measure.

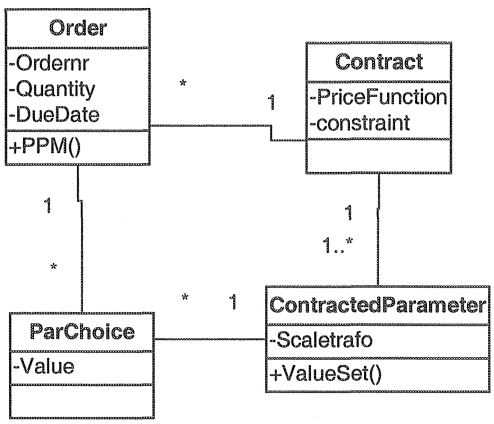

Figure 3 - Data structure for order on product family contract.

Above it was expressed in a database integrity constraint that for each purchased product family for every parameter all values must be covered in one ore more contracts. From the sellers' point of view, in a contract at least one parameter value must be specified for every parameter:

$\forall p \in$ ProductFamily
[pf.SoledIn $\neq \varnothing \Rightarrow \forall$ prm $\in$ pf.Parameter [prm.Sold $\neq \varnothing$]].

A practical solution can be to specify default values for unspecified parameters, but for simplicity reasons we leave this out of our model. An example where completeness of parameters is at stake is the situation where company B offers the choice between 5V and 12 V supplies:

<prm4 is a Parameter with ProductFamily pf3, Name voltage and ValueSet {5V, 12V}>.

For company A the Voltage is always 5V, so voltage is not recognised as a parameter. However, company B needs an explicit choice for voltage. Therefore in forming the contract it must be negotiated what value is required, so company A must add a single valued parameter voltage:

<prm5 is a Parameter with ProductFamily pf2, Name Voltage and Valueset {5V}>,

so that this parameter can be part of the contract:

<ctp3 is a ContractedParameter with Contract ctr1, OfferPrm prm4 and NeedPrm prm5>.

This makes the contract complete, even when some parameters are not relevant for the buyer.

CO-DESIGN.

Version and status

A more complicated case involves the "co-design" process. This kind of cooperation is not only about cost control related business units at both sides and engineering related business unit at the buying side, but involves also the design department at the supplier side. The typical problem in a co-design situation is that the product models in both companies are evolving and need therefore be versioned. In correspondence to normal engineering document life cycle management (Kenneth G.McIntosh 1995), versions must have a status attribute, to indicate what the quality of the model is and for what purposes it can be used. In our case, this new situation is depicted in Figure 4.

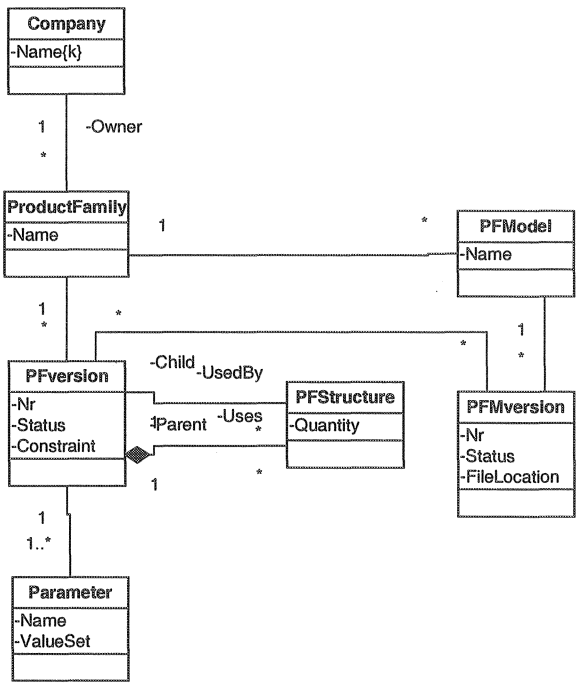

Figure 4 - Product Families with versions.

Product family and product family model have got versions which are represented by the object classes PFVersion and PFMVersion. Versions are characterised by a version number and a status. The contents, like the constraints of

the product family and the file location of the model, have moved to the version, because these contents may change between versions. Also the product family structure is now related to product family versions, because it must be precisely known what the composition of each version is.

Status is a very important attribute in the cooperation between the buying and the selling company. Before a model can be offered to the supplier, is must have been approved by the buyer. Before the regular production starts, the product family must have been released by seller. Status is assigned to version, because different versions may have different statuses. Different sequences of statuses are used in different companies. Such a sequence is also called a life cycle. In general it can be said that the more status values are distinguished, the better the life cycle can be controlled and synchronised between organisational units. This paper takes no position about what is the optimal life cycle in which situation. Just as an example the following status values are distinguished:

Product family version:

1. under development
2. for prototype
3. for pre-production
4. for regular production.

Product family model version:

1. in work
2. for review
3. for approval
4. released.

We will now consider the case where company A needs a product family of CRT monitors for its computers and wants a company D to design this. First company A will define the monitor and its first version:

<pf5 is a ProductFamily with Name M2002 and Owner c1>.

Then it will create the first version of this family with status 'under development':

<pfv51 is a PFVersion with ProductFamily pf5, Nr 1 and Status 1>.

Supposing that at this moment the PC is still in its first version:

<pfv11 is a PFVersion with ProducFamily pf1, Nr 1 and Status 1>,

the monitor can be made component of this product family version:

<pfs2 is a PFStructure with Parent pfv11, Child pfv51 and Quantity 1>.

The monitor has a parameter:

<prm6 is a Parameter with PFversion pfv51, Name diameter and Valueset {14Inch, 17Inch, 21Inch}>.

Note that, like product family structures, parameters are assigned to product family versions, because parameters might change between versions.

Process coordination

An issue in designing the monitor family is that the styling must be consistent with that of the housing and the keyboard. Two product models are defined: an interface specification and a design sketch:

<pfm3 is a PFModel with ProductFamily pf5 and Name 'M2002 Interface Specification'>
<pfm4 is a PFModel with ProductFamily pf5 and Name 'M2002 design outline'>.

Now the game is played as follows. In company A designers create first versions of the interface specification and the design outline and check them in into the local PDM system:

<pfmv31 is a PFMVersion with PFModel pfm3, PFVersion pfv51, Nr 1, Status 1 and FileLocation C: \design\pc2002\M2002intspec.doc>
<pfmv41 is a PFMVersion with PFModel pfm4, PFVersion pfv51, Nr 1, Status 1 and FileLocation C: \design\pc2002\M2002dsgoutl.dwg>

Both documents are submitted for review:

<pfmv31 change Status 2>, <pfm41 change Status 2>

and reviewed. The interface specification is accepted:

<pfmv41 change Status 3>.

The design outline is rejected by the reviewer and the designer creates a new version:

<pfmv42 is a PFMVersion with PFModel pfm4, PFVersion pfv51, Nr 2, Status 1 and FileLocation C: \design\pc2002\M2002dsgoutl2.dwg>.

In order to reduce time to market, company A has the strategy to involve suppliers as early as possible. Now the interface specification has been submitted for approval, this means that the design is considered to be sufficiently stable to use it in contacts with the supplier. Company D is approached and a contract is created. Since this contract does not yet imply a final agreement on delivery of the product, contracts in a co-design relationship must have a status, like product families and their model. Since agreements can change during the process, and decisions made on earlier agreements must remain traceable, contracts need versions too. This is depicted in the data structure in Figure 5.

It can be seen that the contents of the contract, like PriceFunction and Constraint, have moved to the ContractVersion. The life cycle as maintained for the contracts of company A is characterised by the following statuses:

1. Request for proposal
2. Request for offer
3. Order accepted

The new relationship is recorded in the database with:

<c4 is a Company with Name D>,
<ct3 is a Contract with Buyer c1, Seller c4 and ContractNr C2002003>.

Before company D can create the first contract version, it must create its own representation of the product family, including first version and contracted parameters:

<pf6 is a ProductFamily with Owner c4 and Name CRT002>,
<pfv61 is PFVersion with ProductFamily pf1, Nr 1 and Status 1>,
<ctv31 is a ContractVersion with Need pfv51, Offer pfv61, Nr 1 and Status 1>,
<prm7 is a Parameter with PFVersion pfv61, Name diameter and ValueSet {14Inch, 17Inch, 21Inch}>,
<ctp4 is a ContractedParameter with ContractVersion ctv31, OfferPrm prm7 and NeedPrm prm6>.

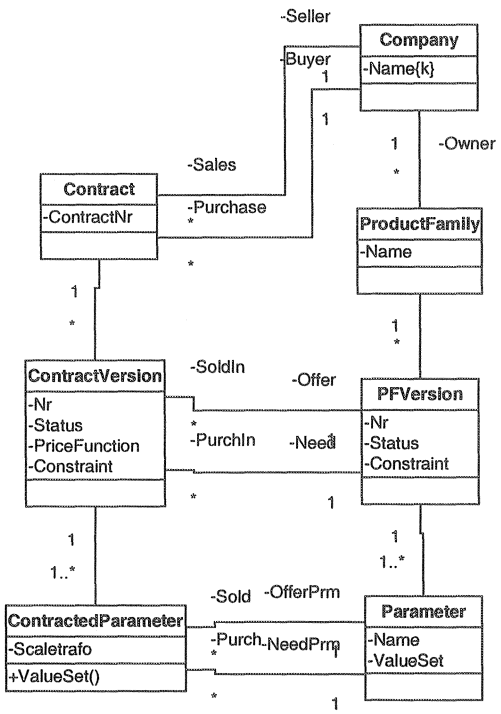

Figure 5 - Versioned contracts.

Now there is a contract, the engineers of company D can access version 1 of the M2002 product family and from there the product models that have status 3 or higher. This means that they get access to version 1 of the interface specification. They can start with the technical design of the monitor:

<pfm5 is a PFModel with ProductFamily pf6 and Name 'CRT002 technical design'>,
<pfmv51 is a PFMVersion with PFModel pfm5, PFVersion pfv61, Nr 1, Status 1 and FileLocation D: \design\C2002003\CRT002\techndsg.dwg>.

After some time the engineers in company A come to submit version 2 of the design outline for approval:

<pfmv41 change Status 3>.

Because the model version is promoted to status 3 the responsible engineer in company D is notified (either by a manually generated Email or by the shared PDM system) that new documentation is available, so he can start working on the physical design of the monitor housing:

<pfm6 is a PFModel with Name 'CRT002 housing geometry'>
<pfmv61 is a PFMVersion with PFModel pfm6, PFVersion pfv61, Nr 1, Status 1 and FileLocation D: \design\C2002003\CRT002\geometry.dwg>.

This design goes through a number of review cycles inside company D, until version 4 is promoted to status 3:

<pfmv64 is a PFMVersion with PFModel pfm6, PFVersion pfv61, Nr 4, Status 3 and FileLocation D: \design\C2002003\CRT002\geometry4.dwg>.

The proper engineer in company A is notified and he inspects the design. The are some comments but in order to keep the process going company A promotes the contract status to 'request for proposal', be it with some change requests attached:

<ctv31 change Status to 2>.

When the proposal, including price indications, is acceptable, the product family M2002 version status can be promoted by company A to level 2 (for prototype). In response also company D will promote the corresponding product family CRT002 version 1 status to level 2. Note that the product family version is still in version 1, while some of the models are already in version 5.

Conclusions for Co-design

The scenario above shows how version and status of product families, product models and contracts are upgraded according to well defined rules so that the development processes in both companies can proceed in parallel but with fine synchronization. In would be beyond the scope of this paper to describe the process in all detail until regular production is reached. However it can be concluded that coordination between development processes is very complex. Also the fact that development in product families in stead of in single product types, increases the complexity. It has not been discussed in the scenario above, but approval of a product family design means that possible conflicts in all possible variants must be detected and considered. Therefore it is not surprising that many companies choose to have contracts only on identified product types and to take ample time slack between these processes in order to reduce the control complexity. On the other hand it makes clear that in order to achieve real improvements in time to market for product family design in supply chains, further research on optimal life cycle definitions and coordination rules is necessary.

CONCLUSIONS

The exercise described in this paper illustrates in the first place the extreme complexity of coordination of purchase and design processes for product families between partners in a supply chain. The use of a data-model, illustrated with a case described in actual database contents is introduced as a means to have a very precise description of the case. The database comes in the place of a graphical illustration, that is difficult to use for rather abstract entities like product families. The use of separate database statements in place of tables as are normally used, enables to use the database in a running example and to fill the database in narrative form. Here it must be mentioned that we also used a MS Access database as a background recording of all statements to enable the researchers to keep overview. We do not print this database in the paper, since it provides no additional information, but researchers who are challenged to use this method in further discussions on product family issues, we advise to follow this practice. The database gets then the function of a 3D model of a complex product.

Another finding from this research is that it eases coordination between customer and supplier if both their representations of the product family are considered to be different objects with different identities. This separation is similar to the one promoted by PDM-professionals, to separate identities of products and their documents. This separation between offered and needed product family makes it possible for each partner to own its specific model, to optimise it for their specific needs and to use the contract to coordinate differences between these models. The separation provides also a possibility do discriminate between private knowledge of each partner and shared knowledge between them.

REFERENCES

[1] D.Sabin and R.Weigel. 1998. Product Configuration Frame-works-A Survey. *IEEE Intelligent Systems.* (July/August): 42 - 49.

[2] F.J.Erens. 1996. the synthesis of variety developing product families. PhD thesis. Eindhoven University of Technology

[3] Frank van der Linden and Jurgen K.Muller. 1995. composing product families from reusable components. *Proceedings of IEEE.* 35 - 40.

[4] Hegge, H. M. H. 1995. Intelligent Product Family Descriptions for Business Applications. PhD thesis. Eindhoven University of Technology

[5] J.Browne, P.J.Sackett, and J.C.Wortmann. 1995. Future manufacturing systems—Towards the extended enterprise. *Computers in Industry.* 25: 235 - 254.

[6] James Rumbaugh, Ivar Jacobson, and Grady Booch. 1999. the unified modelling language reference manual.

[7] Jan Goossenaerts. 1998. Product Configuration in the Framework of the Virtual Enterprise. Working paper. Eindhoven University of Technology

[8] K.Cheng, H.M.H.Hegge, J.C.Wortmann, and J.B.Goossenaerts. 7 - 27 - 2002. explore the knowledge distribution for making a family of products in an extended enterprise. (Proceedings of the 9th International Conference on Concurrent Engineering, Advances in Concurrent Engineering, Cranfield, UK): 633 - 643.LISSE/ABINGDON/EXTON(PA)/Tokyo: A.A. Balkema Publishers

[9] Karl T.Ulrich. 1995. The role of product architecture in the manufacturing firm. *Research Policy.* 24: 419 - 440.
[10] Kenneth G.McIntosh. 1995. Engineering data management. McGraw-Hill
[11] Mannisto, T., Peltonen, H., Soininen, T., and Sulonen, R. 2001. Multiple abstraction levels in modelling product structures. *Data & Knowledge Engineering.* 36(1): 55 - 78.
[12] Marc H.Meyer and Alvin P.Lehnerd. 1997. The Power of Product Platforms, Building Value and Cost Leadership. 35New York: The Free Press.
[13] V.E.Barker and D.E. OÇonnor. 1989. Expert Systems for Configuration at Digital XCON and Beyond, *communication of the ACM.* 32(3): 298 - 318.
[14] Zhang, W. J. and Li, Q. 1999. Information modelling for made-to-order virtual enterprise manufacturing systems. *Computer Aided Design.* 31(10): 611 - 619.
[15] OMG, 1999, "OMG Unified Modelling Language Specification", version 1.3, June 1999, Rational Software Corporation, www.rational.com/uml/documentation.