INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 05 MELBOURNE, AUGUST 15 – 18, 2005

SEMANTICS AND VALIDATION OF LARGE INTER-COMPANY PRODUCT MODELS

Sascha Opletal, Dieter Roller

Abstract

A highly dynamic industry needs to use synergetic effects to reduce production cost and the time to market. A common production scheme in e.g. the automotive industry is the support of the main manufacturer by several part suppliers. Usually those part suppliers have their own product model which represents their intellectual property. Some of them provide fixed standard components; others provide configurable designs and hand out abstract black box configuration systems for the main manufacturer. As development progresses, the product models are changed and subject to an evolving process. An integrated model of all parts and their inner details would make the intellectual property of a part supplier available to the main manufacturer and the part supplier would be obsolete as knowledge provider. In this paper we propose an integrated product model through well defined reasoning interfaces. Those interfaces are announced on a blackboard and can be adjusted to only release information which is green-lighted by the original part manufacturer.

Keywords: intellectual property, knowledge management, product models, reasoning, CAD

1. Introduction

The global market demand for shorter product cycles requires a fundamental change in the involved manufacturing and planning processes. In recent years it was attempted to make the product development more efficient by viewing it as an integrated effort. The organisation of the information flow of a company is researched and optimized within the CIM (Computer Integrated Manufacturing). In CIM the focus lies in the interactions between CAD/CAM and the processes involved in Production Planning and Control [1]. The knowledge which is shared in a CIM environment has direct effect on the design process and all involved work units of the production process. The focus of this paper is the exchange of product model information between companies and ensuring that vital company knowledge is kept secret while preserving the ability to reason over the product model. For a successful integration of a component in a greater design project, it is necessary to have access to the relevant information without getting access to restricted knowledge.

A variety of methods have been researched on how to express and automate certain aspects of the design process [2]. First generation systems used fixed geometric primitives to build models. The evolution to parametric models introduced the constraints and form features and enabled the efficient generation of product variants. Often the main focus for a designer is not the creation of new products from scratch but the adaptation of an existing product in order to evolve it to the next product generation. Classic methods to describe parametric object dependencies include constraints and form features. First the constraints were introduced to

express ground, dimensional, geometric and algebraic relations. The concept of form features was later added to provide a kind of template to predefine certain aspects of a functional module [2]. The form features allow for a greater expression of valid objects. However the relationships between disjoint objects and models can be difficult, if they don't use the same structural expression. Typical problems which are considered in this paper are the interaction between isolated components distributed over several companies and the integration into a global product model. It should be possible to validate the model, regardless of prohibited access to knowledge which should stay in the company that is supplying the component.
As an example scenario where rich product knowledge is captured within a product model is the "mass customization" of products, which can be done efficiently by using a product configurator. The objective of product configurators is to provide a CAD system for time efficient variant generation and validation [3]. This is achieved by integrating enough design knowledge into the system in order to have the system propose obvious and intended design steps or even execute them automatically, thus generating valid product variants.
Unfortunately, most product configurators are standalone applications which will not allow the import and export of product data from other configurators, as this would mean giving out all the companies' vital product knowledge. Product configurators use a set of rules to predetermine the possible characteristics of a product. By solving the rules on a given input, the machine can decide if it is possible to build a product or not and also reach a specified product quality. Learning heuristics can help to solve the configuration problem [4].

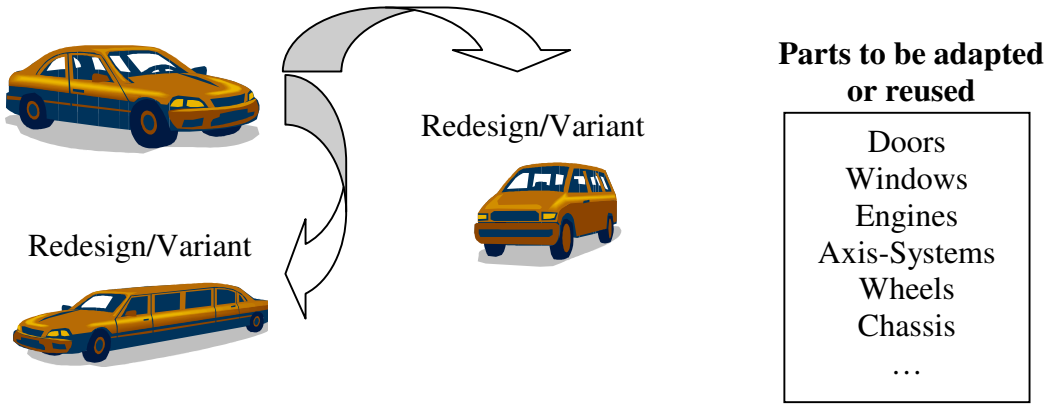

Figure 1. Example Scenario of parametric or variational product design

As example consider the situation shown in figure 1, where a redesign of a product is needed at a company "C" to create an advanced product. A product design can be captured through parametric modelling, if the product stays overall the same, with only minor adjustments, that can be derived from the original product design and be expressed in parametric or variational concepts, such as constraints. A redesign could involve the replacement of components of supplier "X" by components of another supplier "Y" because supplier "Y" provides a component of higher quality. For the redesign there is a need to access the product data of supplier "Y" to introduce and test the new component. Of high importance to supplier "Y" is the distinction between public and private knowledge, since supplier "Y" has an interest in keeping his intellectual property secret.
One solution which is widely adapted by the suppliers is to hand out unsharp information, through a model catalogue or more sharply through selected specifications. This means that the needed values have to be transferred by hand into the used product model at company "C". Another approach which abstracts over the details of the product model is to specify a component through a standalone configuration application which can be used if the interfaces

are known. Normally a high amount of useful information is hidden from the integrator. There are several approaches to support the design task by knowledge based systems [5]. For those systems it only the homogenous situation within one company is considered.

2. Related Work

Constraint programming and reasoning has a long history. The use of constraints is not fixed to the domain of CAD but stretches over a wide range of application fields where the solution of combinatorial problems, decision problems or reasoning on unsharp information is needed. Current research, after the seminal works of Yokoo, Ishida and Kuwabara [6] focuses strongly on the application of distributed constraints using agent networks.

Recently developed constraint languages include:

- The Object Constraint Language (OCL) [7] which is a part of UML
- Mozart [8], a development platform for intelligent, distributed applications based on the programming language Oz. It has an emphasis on distributed constraint solving.
- The ECLIPSE Constraint Logic Programming System [9], which contains several constraint solver libraries, a high-level modelling and control language, interfaces to third-party solvers, an integrated development environment and interfaces for embedding into host environments.

3. Constraint Model and Distributed Constraint Satisfaction (DCSP)

In this paragraph we will further discuss the example scenario and show why it can not be solved using regular constraint programming. Constraints are a classic method to describe component structures in CAD. We give a short introduction in constraint logic programming to show the applied methods and motivate our approach in comparison. For further reference consult [10].

Constraints in the domain of CAD can be divided into three groups:

- Absolute Constraints
- Geometric Constraints
- Algebraic or Dimensional Constraints

Absolute Constraints represent fixed values in a design system, such as coordinates or fixed expressions (constants). Geometric constraints express relations between geometric objects, such as “is parallel to” or “is perpendicular to”. Algebraic constraints describe functional dependencies between parameters of objects. The task of finding a valid geometric expression, while fulfilling all active constraints, is described through the constraint satisfaction problem (CSP). If the problem is distributed across several domains, we speak of a distributed CSP (DCSP).
For a constraint system to be solved by a computer, it is formalized to be used with a calculus. A calculus is a set of transformation rules. It uses a step called resolution to derive the next step by applying a transformation rule. Such a calculus can also be viewed as a state-transition system. The transition system is defined as a Triple $(Z, T, \mapsto)$, where S is a set of states, $T \subseteq Z$ is the set of terminal states and $\mapsto \in Z \times Z$ is the transition relation, for which holds the following:

$$S \not\mapsto S' \text{ for all terminal states } S \in T \text{ and all } S' \in Z.$$

We will now use a small transition system and a calculus to build a DCSP for the scenario. Each state in the transition system is equal to an assignment of values to the constraint system or a transformation of its rules. A transition from one state to another is reached by applying a resolution step (see section 3.1). Under the precondition that the conditions (or constraints) are fulfilled, a transition can be made to another state, starting from the one on which it was defined.

This can be written as:

$$\frac{\begin{array}{l} Condition_1 \\ \ldots \\ Condition_n \end{array}}{S \mapsto S'}$$

It is possible to capture several aspects of the situation of the scenario described in section 1, through the use of the distributed constraint satisfaction problem (DCSP). The DCSP is a CSP with the following enhancements:

- Each location of the DCSP forms a local reasoning domain
- The reasoning domains are interconnected through interfaces
- All reasoning domains together form the DCSP

Depending on the distributed nature of a DCSP setup, there are several possibilities in the information flow to resolve constraint satisfaction. Several distributed algorithms exist to solve the DCSP [6]; the main difference between the solutions is, if the processing is sequential or if it has to be in parallel. Distributed constraint setups are shown in figure 2.
Both communication models in figure 2 are designed to have an information flow towards a "master product". The master product functions as described in the example scenario and represents a product in which to integrate other products from part delivering companies.

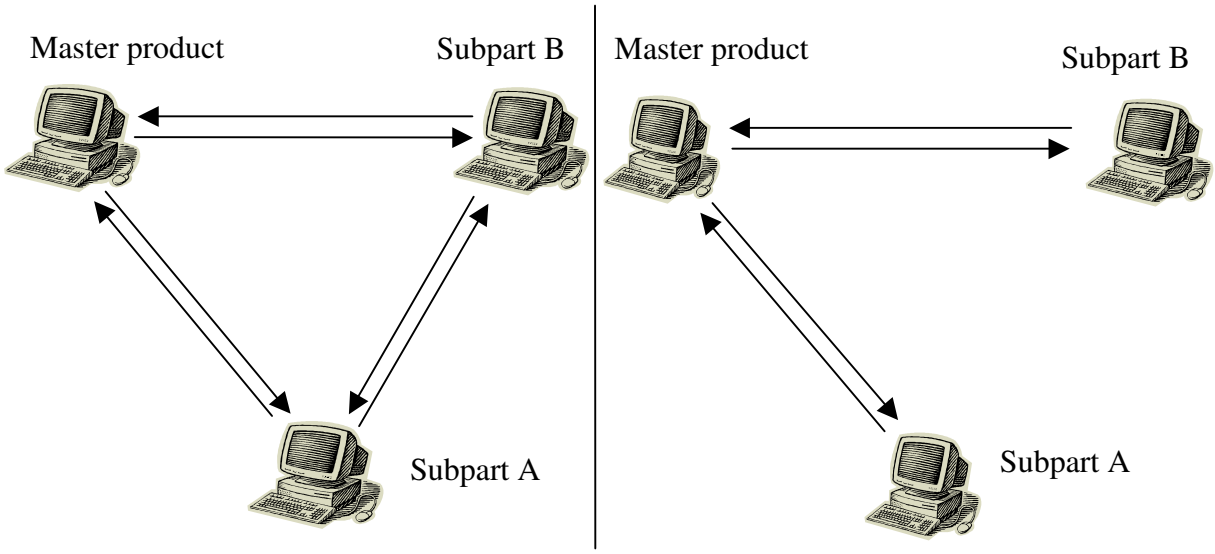

Figure 2. Information flow with inter-subpart constraints

On the left side in figure 2, the participating suppliers have equal communication rights. This setup however requires the involved parties' models to communicate directly. This can lead to problems when IP protection is needed. In order to communicate between models, the part

suppliers have to open the models for a competitor. This unwanted situation can be resolved partly, if the setup on the right side of figure 2 is used. Here the part suppliers only communicate through the master. The master synchronises communication, therefore the suppliers have to open the model only to the master company. There can be situations where this is also unwanted. The master also gets deep knowledge of the product, when constraints have to be established towards inner values of a subpart and when the information for the constraint resolution runs through the master. This leads to a need for methods of information hiding.
A constraint model serves to define possible relations between different values of geometric objects. The goal is to solve the constraint satisfaction problem, which is defined as one or all sets of variable values to a given constraint model.
The CSP consists of a finite set of variables X and domains D to which each variable is associated and a set of constraints C.

$$X = \{X_1,...,X_n\}, C = \{C_1,...,C_n\} \tag{1}$$

A constraint C_i consists of two parts, the subset S_i of variables on which it is defined and the associated relation rel_i:

$$S_i = \{X_{i1},...,X_{ij(i)}\} \tag{2}$$

$$S_i : rel_i \subseteq D_{i_1} \times \cdots \times D_{i_{j(i)}} \tag{3}$$

As example, a simple constraint system for Boolean algebra can be set up as follows:

C ::= true | false | $X =_B Y$ | $-X =_B Y$ | $X \wedge_B Y =_B Z$ | $X \vee_B Y =_B Z$ | $X \oplus_B Y =_B Z$ | $C \wedge C$

3.1 CLP-calculus for constraint solving

We will describe shortly a calculus for the processing and solving of a CSP, based on constraint handling rules (CHR) [11]. The advantage of CHR is to be able to cover new domains by providing user-defined constraints. The rules of similar systems are mostly fixed. The solver of the CHR is able to transform a constraint system by using the following rules:

- Simplification
- Normalization
- Propagation
- Entailment
- Solving
- Optimization
- Consistency check

A user defined constraint [11] for less-than-or-equal (=<) can be defined using syntactical equality (=) in three relations:

- Reflexivity: A=<B $\leftrightarrow$ X=Y | true
- Antisymmetry: A=<B $\wedge$ B=<A $\leftrightarrow$ A=B
- Transitivity: A=<B $\wedge$ B=C $\rightarrow$ A=<C

The operational semantics of CHR use is given by a transition system with initial and final states. The sequence of the solution steps is normally not deterministic and often relies on

search for value sets if there resolution steps can't be applied. Failed states have to be detected to determine the unsolvability or to start some sort of backtracking to come to a final state.

A CHR-calculus works on states $\langle F, E, D\rangle \nu$, where F are the constraints remaining to be solved, and E and D are constraints that have been accumulated and solved so far. The initial state has the form $\langle F, true, true\rangle \nu$, the failed state $\langle F, E, false\rangle \nu$ and the final state $\langle true, E, D\rangle \nu$. The following rules are applied to process a constraint system [11]:

Solve:
$$\langle C \wedge F, E, D\rangle \nu \mapsto \langle F, E, D'\rangle \quad \text{if } C \text{ is built-in and } CT \models (C \wedge D) \leftrightarrow D' \tag{4}$$

Introduce:
$$\langle H \wedge F, E, D\rangle \nu \mapsto \langle F, H \wedge E, D\rangle \nu \quad \text{if } H \text{ is a CHR constr.} \tag{5}$$

Simplify:
$$\langle F, H' \wedge E, D\rangle \nu \mapsto \langle B \wedge F, E, H = H' \wedge D\rangle \nu \quad \text{if } (H \leftrightarrow G \mid B) \text{ in } P \text{ and } CT \models D \rightarrow \exists \bar{x}(H = H' \wedge G) \tag{6}$$

Propagate:
$$\langle F, H' \wedge E, D\rangle \nu \mapsto \langle B \wedge F, H' \wedge E, H = H' \wedge D\rangle \nu \quad \text{if } (H \rightarrow G \mid B) \text{ in } P \text{ and } CT \models D \rightarrow \exists \bar{x}(H = H' \wedge G) \tag{7}$$

The discussed constraint model of a CSP however is sufficient to fully describe the scenario of section 1, if all parties agree upon full information sharing. We will describe shortly the process of solving a DCSP.
A computation of a DCSP is a sequence of $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$ of transitions. It is successful or finished when it reaches a terminal state, which is written as $S \rightarrow {}^*T$. Similarly, a DSCP consists of a global sequence, which is build by local sequence sets of the participating part problems. This can be written as $SG_0 \rightarrow SG_1 \rightarrow SG_2 \rightarrow \ldots$, where SG_n is a final set of local sequences $SL_0 \rightarrow SL_1 \rightarrow \ldots$ that has reached a terminal state in itself or is in a configuration state, waiting for input from a dependency on another local configuration problem. This is shown in figure 3.

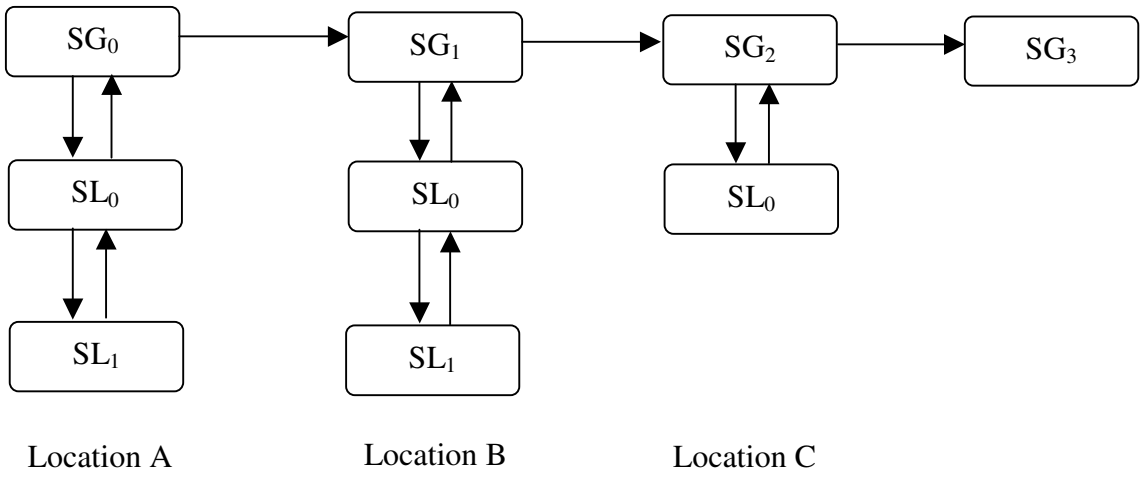

Figure 3. Distributed configuration states without dependencies

After solving the local constraints in each location A, B and C, the global state SG_3 has to be reached to solve the constraint system. This state can only be reached if no intermediate location came into an unsolvable state. The solution in such a case would be to negotiate a new set of input values for a specific location, so the global model would also be updated. In a concurrent design as described in the scenario, the need of the master product is the driving force. There is no possibility to automatically negotiate new values for the master model by a search algorithm or similar. A value change has to be brought to the attention of the designer as it is mostly to be decided "by hand".
The reasoning in a distributed setup has risks of disclosing internal information of the participating models. The consequence of this problem is shown in figure 4, where a constraint exists between parts of locations B and C. Therefore an agreement has to be reached between states SL_0 at location C and SL_1 at location B.

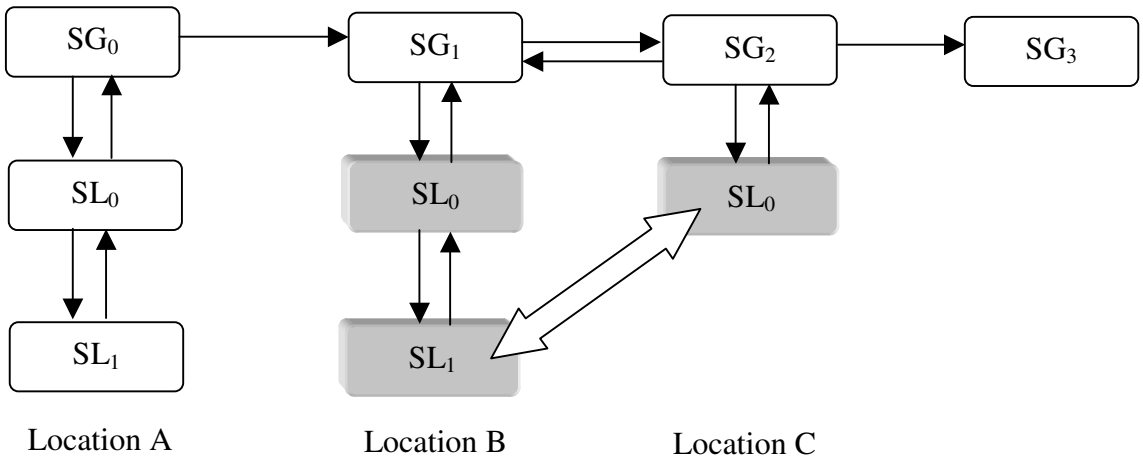

Figure 4. Distributed configuration states with dependencies at SG_2-SL_0, SG_1-SL_1

The steps get to know information from each other by the constraint. If this is unwanted, there is no possibility to complete global configuration, since the individual parts can't complete the configuration.

3.2 Information Propagation

To examine the difficulties of unwanted information propagation in distributed configuration setups, we analyze the possibilities of information access in several situations. It is assumed that the parties have control over the access rights of the participating variables of the product models. A mechanism to achieve this is described in section 3. In the view of a "master product" it is essential to make a distinction between driving and driven parameters. A driving parameter has a constant assigned at the beginning of a propagation process.
The value of a driven parameter is defined as a dependency or constraint on other driving or driven parameters. One driving parameter has to be in the dependency line, so that the depending values can be computed. In figure 5 there are two situations, where A is dependant on local and remote variables. The upper situation is solvable, since A has access to B. In the lower situation there is the constraint set to $B \wedge C$, which can never be solved, since access to C is forbidden. If A needs access to C, it has to request access rights from the owner of C. After A knows C, the access rights of C have to remain attached to whatever happens with the knowledge provided by C, so that it can't be propagated to another party. This is especially difficult in chained constraint setups. There has to be a negotiation method that prevents such setups between restricted information when the problem is formalized.

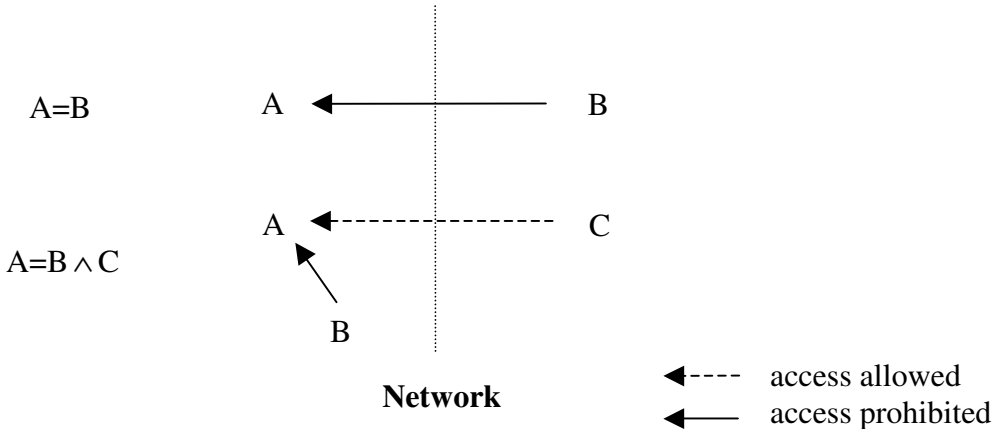

Figure 5. Direct Information Access

Figure 6 shows the problem of indirect information access. The value of C (upper situation) and B (lower situation) becomes known to A, although direct access to the respective parameters is not possible or allowed.

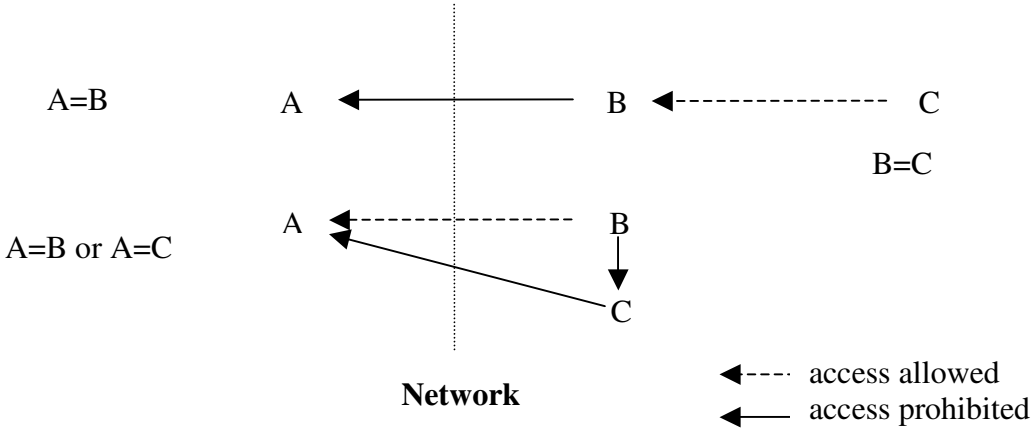

Figure 6. Indirect Information Access

This is not a flaw of access methods but by the logic system used to solve the DSCP. It was not designed to work in an area where restriction of information is necessary. It is inevitable that information gets propagated in the reasoning process. If it is annotated to reflect the users' rights, it is possible to circumvent the users' intention, if the reasoners don't work in a secured environment where access is protocolled or the information is encrypted.

4. Preserving IP in Constraint Reasoning

As discussed above, the existing DCSP approaches need to have access to all involved information to compute a valid solution, which is then solved sequentially or in parallel. This in turn forces companies to fully open their models if they need to build an integrated view of a combined product.

We will now present a solution that enables reasoning over a complex, distributed product model, while providing the possibility to preserve intellectual property. In opposite to the DSCP, the reasoning over distributed product models requires only the ability to export and import decided parameter values. Instead of opening arbitrary access on the participating models, we introduce a concept called "reasoning mediator" (RM) which serves as the interface definition. The RMs can be specifically configured to exclude information from the

reasoning process by reasoning over publicly announced knowledge only. There is no access to or views of knowledge that is not to be distributed outside a reasoning domain. Such knowledge will be regarded as uncertain knowledge. As a central meeting point for expression and solving of constraints, we suggest a blackboard-architecture where all publicly available information is stored and dependencies are negotiated.

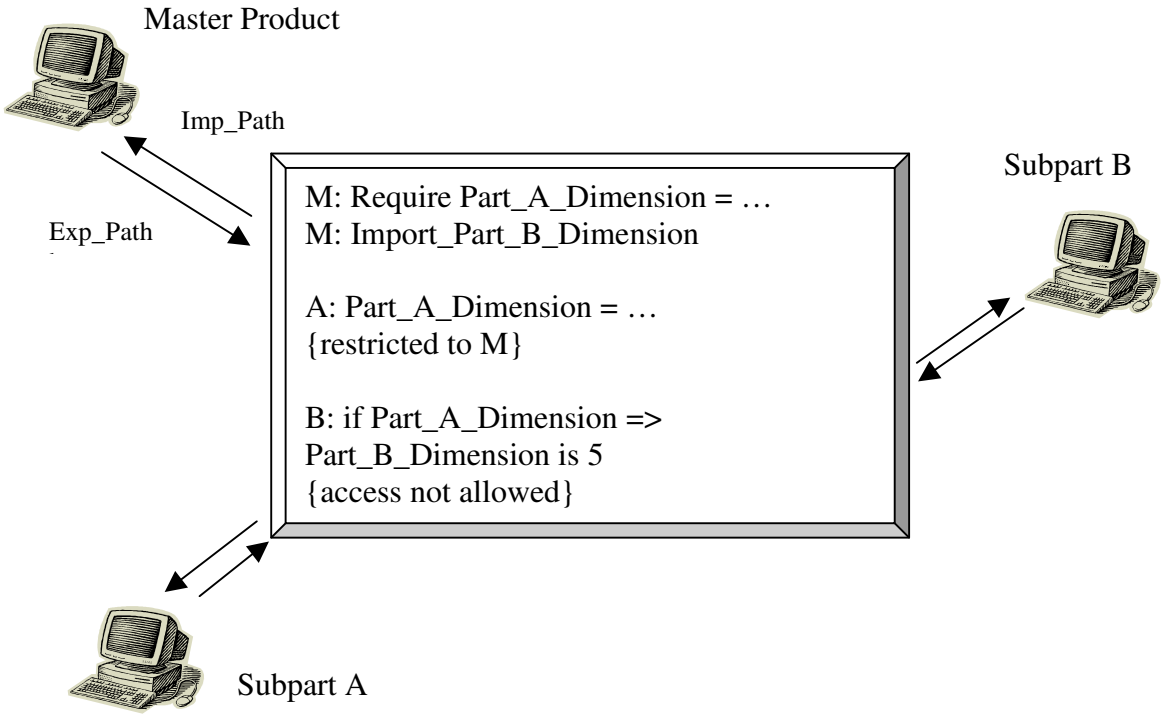

Figure 7. Information Access through a blackboard-architecture

The information on accessible interfaces, reasoning rights and the respective values are broadcast through the RMs onto a blackboard as shown in figure 7. This has the advantage that there is no need for the part suppliers to communicate and exchange information directly. Also it serves as a common knowledge repository for all participating parties. Only the published information is known, there is no need for interaction deep inside the involved product models. The blackboard serves as a global interface definition.
In order to preserve intellectual property, there is a need to mark which information is not allowed to be shared through a reasoning process. This can be done by specifically announce interface variables that can be used to access a model. Another difficulty lies in direct and transitive dependencies where one can determine the value of a local variable by questioning an external value. To reflect driving and driven parameters, we need two types of variables for the input and output direction. This would protect the internal model structure from deep access needs.
To protect the data from being exposed to all parties, there are two possible methods:

- Protect exchanged data with asymmetric encryption. This means that involved parties can communicate directly and but only the intended receiver can read the data adressed to him.
- The transmitted data has to be kept in a secure environment. This can be realised by installing a communications-blackbox at each part supplier, through which he makes contact to the global model. The blackbox guards the security policies of data which it receives and sends.

Also the constraints represent valuable design knowledge, since they represent knowledge about geometric solutions. Consider the simple constraint C_1: a=2*b. If both the variables "a" and "b" are accessible, one can determine the constraint function directly or an approximation of it by setting a value for "b" and observing the result in "a". As consequence it might be also necessary to restrict the amount of access times on certain variables, thus giving the access providing company control of the process. The difficulty herein lies in the reasoning itself that can have access to restricted information by making a suitable request.
To mark variable content that needs to stay private, we make an enhancement to the variable definition. A variable now consists of two parts, its value and a set of access rights that tells if it is available for local or external reasoning or not at all:

$$X = \{(X_1, RM_1),...,(X_n, RM_n)\}, RM_i = \{D_{RMi}, R_{RMi}, F_{RMi}, C_{RMi}\} \quad (8)$$

The need of information hiding and IP protection is reflected by the access rights of the RMs. Each RM_i is defined by a set of D_{RMi}: Direction: Input/Output, R_{RMi}: Values Range, F_{RMi}: Access flags – user, times and C_{RMi}: Boolean Configuration flag.
The direction specifies the possible information flow, it can be allowed read access, write access or both. Access can be restricted to selected members of the blackboard. To avoid abuse of the provided information, there can be restriction of access times to reduce or keep track of information access to avoid reading out all possible configurations. To further hide configuration possibilities, the value range can be limited. The blackboard system has to be secured from being abused by a party. All transactions have to be monitored. This is specifically important for the reasoning mediator definition. The transaction logs have to be made available to the parties which have agreed on an interface mediator definition.
The RMs function as enhanced variable descriptions and form a logic system on the blackboard, RMs can be constrained against each other. The reasoning takes place in two areas, on the blackboard and in the sub models. The interfacing between the sub models and the blackboard is accomplished through the enhanced functions of the RMs. Each model has a set of RMs which can be divided in sets of RM_{input} and RM_{output}, based on their configuration of D_{RMi}.
If all members of RM_{input} are defined, then the model will come to a final state if it can be configured based on the input or a failed state if there is no such possibility. This decision can be made because the model is in the local view not constrained against other models. If the model comes into a final state, then all RM_{output} variables are defined with a fitting value.
After solving all constraints between the models, all RM_{output} and all RM_{input} of the blackboard are defined. The master model can read the RM_{output} and can come to a conclusion. If a model failed to configure, based on the given RM_{input}, the blackboard will remain in a (maybe temporary) failed state. It can be detected by the master model that there is no further solution by reading the states of the configuration flags.
Compared to the direct access approach, a blackboard-architecture has the several advantages. The integration of uneven knowledge sources is transparently managed by the control system of the blackboard. Each knowledge source is independent which eases development and maintenance. The blackboard architecture allows participating applications to adapt to changing requirements more flexibly because it is application independent, and is easily applied to new problem domains.

The basic knowledge on the blackboard is of three types:

- Driving parameters
- Result (or driven) parameters
- Constraint definitions

A difficult task is to secure the protection of certain information. Once the information is further processed, it can be possible to infer what the unprocessed information was. Initial ideas to achieve this have be discussed in this section, a few additional problems will be addressed in section 5. Another aspect is that the access has to the knowledge has to be made platform independent so that companies can attach different systems to the blackboard (see also section 5). This way, different reasoning systems can come to a global solution.

5. Extending Information model through UML

Another problem when several models of different companies come together to form a uniform model is the multitude of used CAD-systems, formats and logical reasoning languages.

As a tool to represent and model the information flow of driving and driven parameters we use the Unified Modelling Language (UML) [7], as it is a graphical modelling language that is widely known and suits the semantic requirements of the interconnection modelling. It can be used to model structural relationships of part assemblies within a specific domain [12]. UML provides a rich toolbox for describing concepts involved, such as class, state and activity diagrams. It is accompanied by the Object Constraint Language (OCL), which allows the declaration of restrictions on objects. The UML/OCL can be used to define relations between objects, which go further in the semantic expressiveness than conventional restrictions; it can be used to build a semantic network, which represents the product knowledge [5].

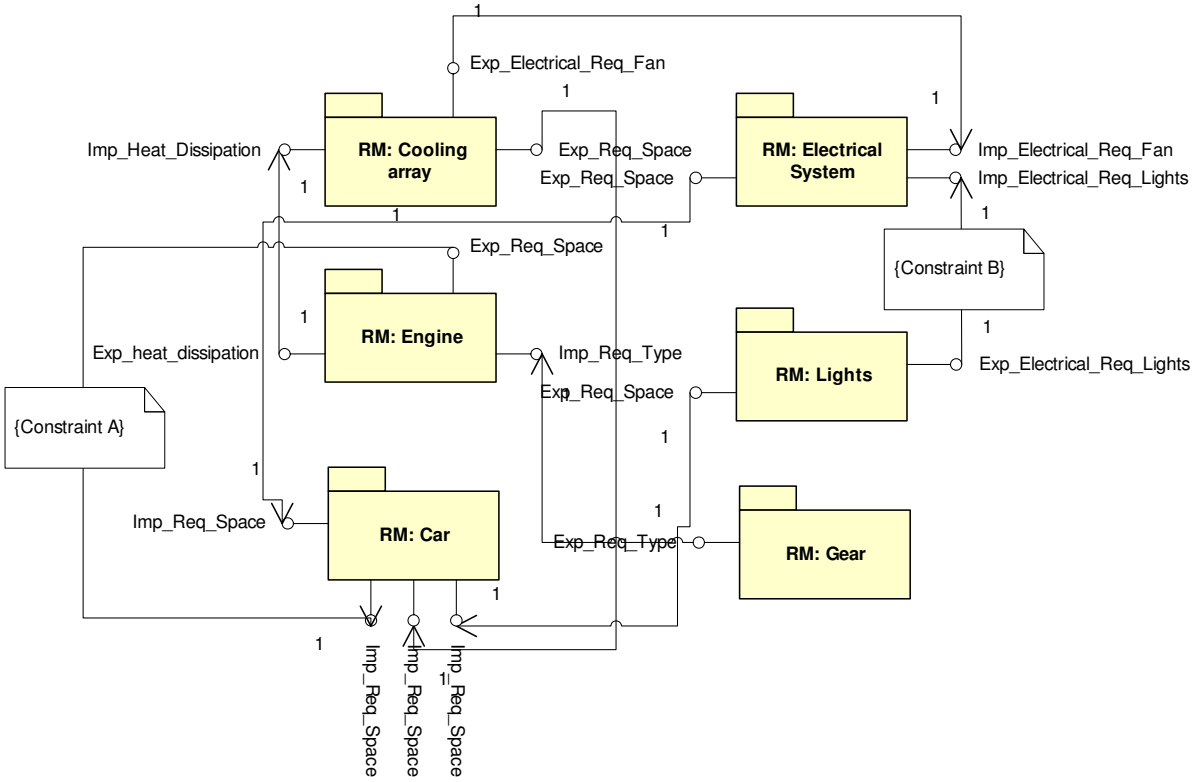

Figure 8. Blackboard global view using UML

In order to capture a wider range of possible relationships between objects and models, the power of expression has to be enriched. A promising way to achieve this is to use the object-oriented paradigm in for the exchange of product development data. We will use the UML as a layer system do describe semantic dependencies as “is subpart”, “is used explicitly”, “is kind of” and “is related” and model the involved information flow between the models.
The highest level of information flow is shown in figure 8, where the exchange of data between parts is modelled. Export and import paths are given as interfaces on packages. The packages represent subparts of the model. They can be described further using class diagrams and constraints. The constraints will have the extended semantics as described in section 4. In the packages the semantics known from the object orientation can be applied, such as inheritance and distribution of features.
Common problems within the supply chain management during early design are the incompleteness of product specifications and uncertainty of which subparts to be included in the final product. The problem of incompleteness of supply is solved through a negotiation by UML. The parties sit together and exchange in effect a delta-table, where the basic specification and possible derivations of in- and output variables are fixed. A configuration party can rely on this table. If there is need to enhance certain components beyond the limits, the parties have to sit together to re-establish a new common domain for possible input and output values. The driving force is the part integrator who constructs the main product. He knows where incompleteness can arise in his configuration and have a placeholder put in the place that takes any input and outputs statically the minimal parameters that a final component should deliver.
The problem of uncertainty which components to use, which also arises in early product development can be solved by using several product models in parallel, where each model uses another component. This way all possibilities can be played through. To minimize the number of effective models, they can also be unified into one model, where different subcomponents can be selected using an additional input parameter. This means that the main product manufacturer can cycle through different components in his model and test and validate several possible setups.

We view attributes as driving parameters and methods as driven parameters. The methods are defined as taking variables as input. If the source of the variable comes from another object, it is marked as external. Constraints can be setup to choose the right component, as a Diesel or Petrol Engine. An example assembly is shown in the UML diagram in figure 9.

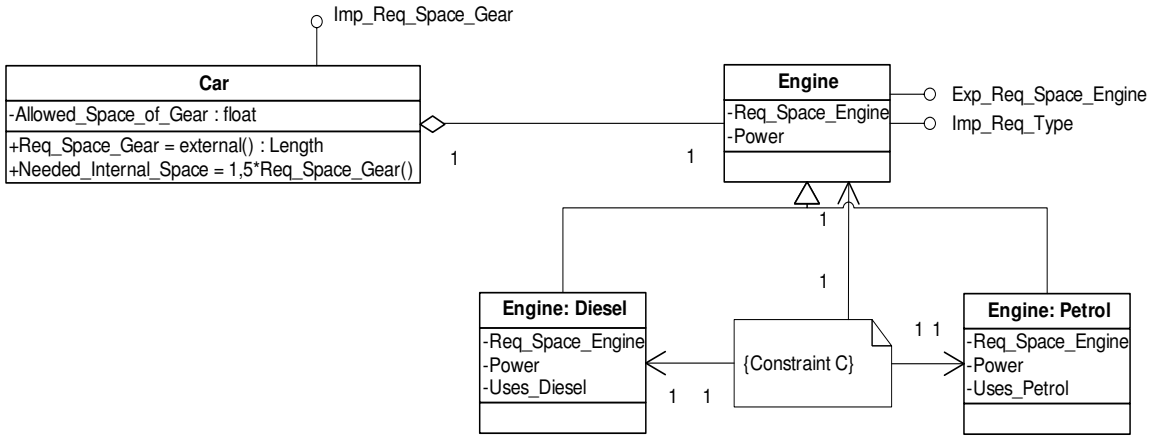

Figure 9. Blackboard setup using UML

The engine is a component that needs to be included in the car. The car has certain requirements for the engine which it announces. The section for engine configuration will take this action and search for a configurator suited to generate the specification for an engine type. If the requested engine would be a diesel engine, the supplier would try to configure a diesel engine based on the input of the car manufacturer. The resulting specification is then sent back to the manufacturer or another involved party from which the configuration request came. The information that is being sent out is annotated as required in section 4.
The UML layer makes it easier for the involved companies to decide between driving and driven parameters and to oversee the model dependencies. From the OCL comes the possibility to impose constraints on operations. The object oriented view is suited to model the IP aspects of the assembly setup, because it models the communication flow and shows how and where the information that leaves the domain of a supplier is used. This way participating companies can easily overlook the dependencies between parts and negotiate the publication of information.

The use of the object oriented paradigm delivers several advantages in the modelling of product assemblies:

Attribute encapsulation: Access to objects is only able from the outside; the inner structure is hidden from the user.
Unity of data, methods and relations: Features of the objects can be used transparently and provide for a natural treatment of problem expression.
Communication: The solution of a given problem is derived through the communication of the participating objects by message passing.
Classes: Classes are blueprints for objects; they express generalization and specialisation, which can be used to generate new and modified objects and reuse already defined structures in a new way, preserving the intent of the original creator.

6. Conclusion

We have discussed the problems of IP protection arising with complex product models on case scenarios. We have shown how they are normally solved using constraints in a distributed environment. In distributed product models, the distribution of intellectual property has to be limited to secure company knowledge. To limit the interaction between participating companies we have proposed a blackboard-architecture which serves both as interface definition and place for information exchange. To support IP restriction we have introduced interface mediators that specify attributes for information handling and can be configured to exclude certain property values. The establishment of automatic information flow between product models of different supplying companies makes it possible for a “master manufacturer” to react faster to customer needs. The proposed architecture can also directly be used in an intelligent CAD system where it serves to make design decisions automatically based on the communication between the connected product models.

References

[1] U. Rembold, B.O. Nnaji, A. Storr: Computer Integrated Manufacturing and Engineering, Addison-Wesley, 1994

[2] J. Shah, M. Mäntylä: Parametric and Feature Based CAD/CAM: Concepts, Techniques and Applications, John Wiley & Sons, 1995

[3] M. Mešina, D. Roller: Configuration management using standard tools, in: Perspectives from Europe and Asia on Engineering Design and Manufacture, X.-T. Yan, Ch.-Y. Jiang, N. P. Juster, eds., Kluwer Academic Publishers, 2004, pp. 645-658

[4] D. Roller, I. Kreuz: Selecting and parameterising components using knowledge based configuration and a heuristic that learns and forgets, in: Computer-Aided Design, Elsevier, Vol 35/12, 2003, pp. 1085-1098

[5] D. Roller, O. Eck, S. Dalakakis: Knowledge based support of Rapid Product Development, in: Journal of Engineering Design, Taylor & Francis Ltd, Vol. 15, No. 4, 2004, pp. 367 – 388

[6] M. Yokoo, et.al.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Transactions on Knowledge and DATA Engineering 10(5), 1998

[7] UML Specification, Version 2.0, Object Management Group (OMG) http://www.uml.org/

[8] The Mozart Programming System http://www.mozart-oz.org/

[9] The ECLiPSe Constraint Logic Programming System, Imperial College, London http://www.icparc.ic.ac.uk/eclipse/

[10] T. Frühwirth, S. Abdennadher: Essentials of Constraint Programming, Springer, New York, 2003

[11] T. Frühwirth: Theory and practice of constraint handling rules. The Journal of Logic Programming, 37, P. Stuckey, K. Marriot,eds., 1998, pp. 95-138

[12] R. Hochgeladen, P. Vyas: Entwurf komplexer Zusammenbauten mit UML, CAD-CAM Report 3/2004, Dressler Verlag, 2004

Sascha Opletal
University of Stuttgart
Institute of Computer-aided Product Development Systems
Universitätsstr. 38
Germany
Phone: +49-711-7816-335
Fax: +49-711-7816-320
E-mail: sascha.opletal@informatik.uni-stuttgart.de