System Design for Reusability – Task, Current State, and Future Research

Hauke Arndt[1], **Frank Feldkamp**[1], **Michael Heinrich**[1] and **Klaus Dieter Meyer-Gramann**[1]

Abstract: In this paper, an IT support for vendors of transportation systems and their subsystems in the early phases of the product design is presented: ***SDR, System Design for Reusability***. Water pumps for cooling combustion engines serve as a pilot application, an editor for the water pump designer was implemented. SDR supports a reusable and modular product design by different modeling features: systems with ports, a taxonomy with system types, interfaces with types, and modular constraints between parameters of systems. SDR stresses the principles of modular system design. This requires a thorough and consistent segmentation of the system at all levels. Moreover, a well-chosen set of interfaces between the modules is defined.

SDR serves as a design assistant. A block diagram editor allows to describe systems, ports and interfaces. Constraint propagation and constraint solving is seamlessly integrated in the modeling of the system structure. SDR supports a least commitment approach during the product design process in order to capture uncertainty in early design phases adequately and to support reusability. Future research will tackle specific issues of reasoning, user interactions, and documentation.

1 INTRODUCTION

The vendors of transportation systems and their subsystems (products like cars, engines, etc.) are faced with a dramatically increasing pressure to reduce their costs and to accelerate their product development and their innovation processes. In addition, the market demands customer-adapted solutions. These solutions often must include customer services like operating a technical system, and contracts assuring the reliability, availability, and life-cycle costs of the products. Vendors must master the entire system and gain the system leadership. They design the technical system and specify the interfaces; the suppliers have to design and manufacture the subsystems of the entire system. In the following, design is meant as the process of engineering technical systems.

Considering this situation, the vendors cannot focus on improvements in detail. The only way to stand competition is to optimize the entire system. These reduction efforts must cover products as well as production processes.

Possible measures to tackle the costs are:

- To **modularize** the product and its design and to **reuse** solutions for partial problems. By doing this, the variety of components necessary to cover all market segments decreases and the costs for production can be reduced, but enough flexibility to offer customer-oriented solutions remains.
- To treat in parallel technical aspects and cost during the design process (**design-to-cost**).

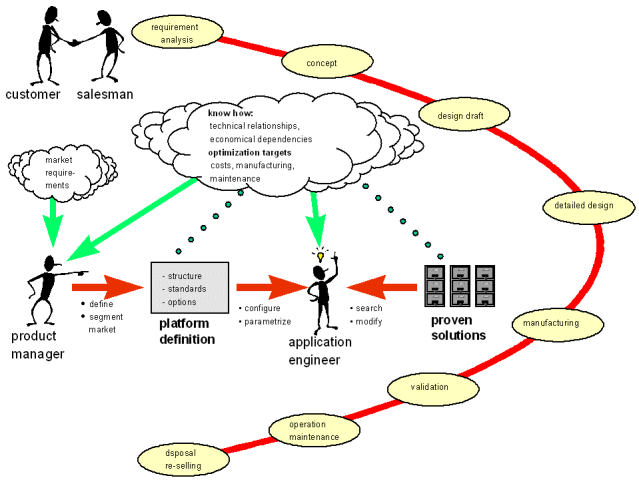

Figure 1. Defining and producing reusable systems

Different divisions of the DaimlerChrysler group have to organize the process chain as sketched in Figure 1. An efficient IT support is crucial for fulfilling this task. In the following we present a pilot application from a subsystem vendor of the DaimlerChrysler group.

2 A PILOT APPLICATION

Water pumps are an integral part of many combustion engines in automobiles. They effect the circulation of the cooling water through the engine block in order to cool the engine. The water pumps modeled during our pilot application are produced in the automotive plant of the DaimlerChrysler Group in Berlin-Marienfelde, Germany.

Designing water pumps is a typical routine task. For bidding, a draft design and a cost estimation for the water pump which is to be designed and manufactured is elaborated. To fullfill a customer's order, a detailed design of the water pump must be worked out. This design of a water pump is partly based on the reuse of existing parts / designs of water pumps. But typically for each new design of a water pump, some new parts have to be designed or existing parts have to be adapted, for instance

[1] DaimlerChrysler AG, Research and Technology FT3/EW, Alt-Moabit 96a, D-10559 Berlin, Germany, email: {Hauke.Arndt, Frank.Feldkamp, Michael.B.Heinrich, Klaus.D.Meyer-Gramann}@DaimlerChrysler.com

geometrically. In addition to these tasks, design management and retrieval are the main topics in this domain of designing water pumps.

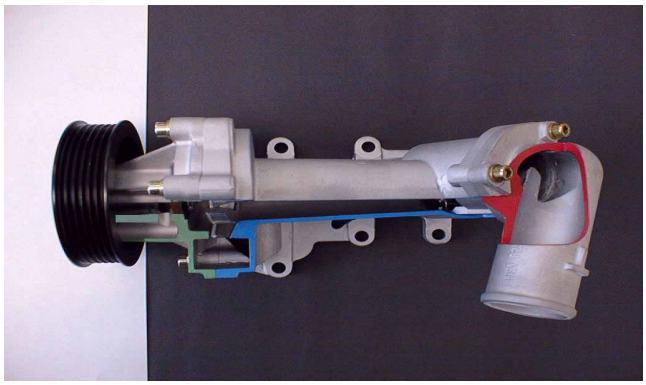

Figure 2. A typical water pump (cross-section view)

2.1 Hydraulic Aspects

The characteristic figures of the water pump depend on the flow speed, differential pressure, and pump rotation speed and have to be determined. The shape of the impeller, the number and shape of its blades, the interaction of the impeller with the spiral are the set-screws to determine the hydraulic behavior. The relationships between flow speed, differential pressure, rotational speed, and the shapes of impeller and spiral are very complicated and are described by estimating formulae in this phase. From a general overall problem-solving viewpoint, the main aspect is the integration of this part of problem solving into the other design activities (like the design of the shaft and the bearing, the driving system).

2.2 Aspects of the Mechanical Drive

The mechanical forces of driving and driven components must fit. For instance, if a belt-pulley system is chosen for driving the pump, then the rotational and axial forces acting on the shaft and the bearing are to be considered for their design. But if an electrical system is chosen, then the specifications of the electrical engine is decisive.

2.3 Geometric Aspects

Examples of geometric aspects are the diameter values of the shaft and the bearing depending on the chosen material; the shape of the impeller and the spiral influencing the size of the pump housing; the relationships between pulley diameter and housing size determining the choice of the connection between the pulley and the shaft. Another important geometric aspect is how the water pump fits into the predefined packaging space without unwanted interferences with other systems around the engine of the automobile.

3 MODELING PRODUCTS WITH SDR

SDR is the abbreviation for "System Design for Reusability" and it is a software tool developed by DaimlerChrysler's Division Research and Technology. Modeling a product with SDR breaks down into three tasks:

- modeling the properties of a product and its sub-systems,
- modeling the product structure and
- modeling the design logic behind the product.

Modeling the properties is usually achieved by defining attributes for systems describing properties (parameters) like size, weight, cost or material. The product structure can be presented by means of a product tree (part-of-tree). Rules or constraints can describe the dependencies and the reasoning behind the product, thus defining its design logic.

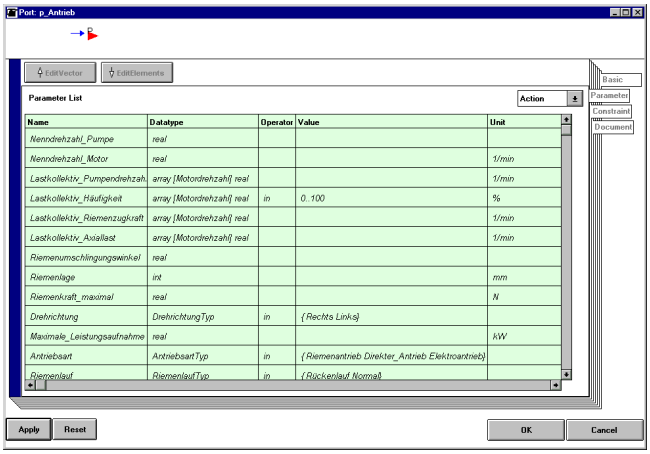

Figure 3. Parameters describing the required hydraulic and mechanical properties

The product structure can be analyzed from two perspectives: vertical and horizontal. The **vertical product structure** is usually modeled by means of a part-of-relationship. Each system is linked to the sub-systems it consists of via such a part-of-relationship. These systems again are linked to their constituent sub-systems, etc. The transitive closure over the part-of-relationship – beginning with the product root node – yields a product tree.

Beside the vertical structure, the **horizontal product structure** plays an important role in the design of a product, although many product structure models (e.g. those in EDM systems) ignore it. It defines how systems are connected and which relationships exist between the systems that belong together building one assembled system. The horizontal structure is modeled with the help of a connected-with-relationship, leading to graphs which describe the part topology.

In order to distinguish between different connections from one system to another, SDR offers the modeling construct of attaching ports to systems. If for example a certain system *A* has a mechanical connection to another system and an electrical connection to a third system, you attach two ports, mechanics and electrics, to the system *A*.

These are also the basic ideas behind the SDR product model. This product model describes systems by listing their properties and the constraints that hold on these properties. It also uses ports to define the potential connectivity of systems. But, as we will see in the following, it adds a few more features that enhance reusability.

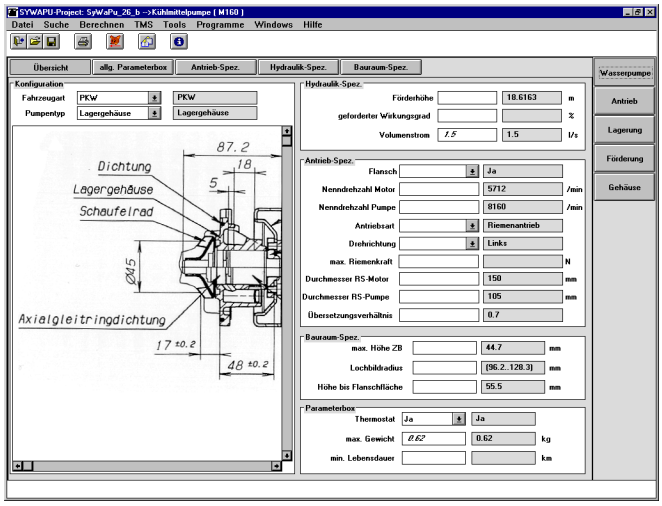

Figure 4. The application of the water pump design in SDR

3.1 The First Step to Reusability: Introducing Types

Abstraction is the key to reusability. In order to get rid of the context of a certain system description, the engineer would prefer to reuse a more generalized and more abstract version of the system to be designed that is not burdened with the specific details of a particular design context. This generic system definition can be provided by a system type where the actual systems in the different design contexts are instances of this type.

The system type models have those properties and rules which hold for all instances of this type. Types allow to reuse system definitions in other design contexts by creating instances of this type in the new design context.

In order to enable reuse of knowledge between types, an inheritance hierarchy of the types (taxonomy) is implemented. Properties as well as rules/constraints are inherited.

Besides increasing efficiency, type hierarchies also support a least commitment approach to design. The engineer is not forced to commit early to detailed design decisions he can not reasonably justify. Instead, in early stages of design, he / she commits to more abstract classes (e.g. to the type "motor") than in later stages (e.g. to an AC-motor with 360 V).

The inheritance hierarchy also helps to get an overview of the available system types, which is important for the reuse of systems. Usually, the reuse process is semi-automatic. The engineer selects a certain abstract system type (e.g. AC motor) and specifies a few desired properties. SDR now searches for types which meet these requirements and propose them to the engineer. An inheritance hierarchy helps the engineer significantly to define a starting point for the search.

So far, we have treated product structure and system types quite independently. To closely integrate them, it is necessary to introduce a third kind of entity (besides types and instances) that represents a system used to build another system. Being used in the structural definition of another system *type* (which is in turn supposed to be reused) is a new role that can not be captured by means of types and instances which only deal with the reuse of the system itself.

In SDR, this task leads to *system refinements* (describing types), *system applications* (corresponding to the usage of a system inside another system) and *system instances*. A simple example: The SDR model of a streetcar with three vehicles, i.e. three wagons each of them having two bogies, has 3 x 2 = 6 *system instances* of bogies. The *system refinement* of the vehicle has two *system applications* of the type bogie.

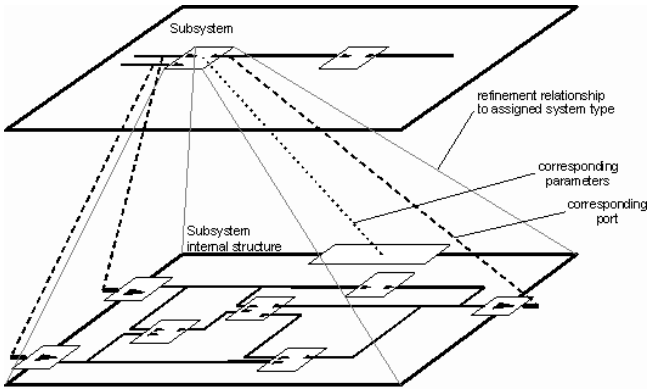

Figure 5. System application and system refinement

3.2 The Second Step to Reusability: Introducing Interfaces

Reusing a system basically means to move it from one context to another. This can only work properly if the system's relationships to its environment (i.e., its dependencies with other systems) are clearly defined and made explicit. A convenient way to model these relationships of a system to the outside world (that means with other systems) is to define interfaces to other systems.

SDR's modeling language for interfaces is as rich as the one used for systems. One can attach properties and constraints to interfaces, interfaces are typed, and properties are inherited along an interface taxonomy. Types and inheritance yield the same benefit for interfaces as they do for systems. Interfaces offer a number of advantages:

- Interfaces explicitly describe which relationships exist between a system and its environment (neighboring system modules) and make clear under which terms this system can be reused.
- Interfaces can model connecting parts between systems, relationships and dependencies between systems (e.g., distances), or even background knowledge about laws of physics etc. Therefore, interfaces are fundamental for modularity.
- Interfaces tend to have a longer life-span than systems, and thus provide a stable backbone for the design knowledge base.
- Interfaces are important for modularity, as they describe the dependencies between systems. Interfaces also help to model the technical reasons for system relationships.

In our application studies for the SDR tool, we have used interfaces for a variety of purposes:

- to model the data flow from the initial specification down to the component parameters;
- to model the flow of force along mechanical components;
- to model the relationships between subway tracks and the subway lines running on these tracks;
- to model the transportation line for bags in an airport luggage system;

- to model the electric connections and control systems in a street-car;
- to model the geometric interrelations between parts of the water pump.

Figure 6 shows the system refinement of a water pump with four system applications (boxes), three external ports for describing the required properties, and interfaces between these applications and ports (lines).

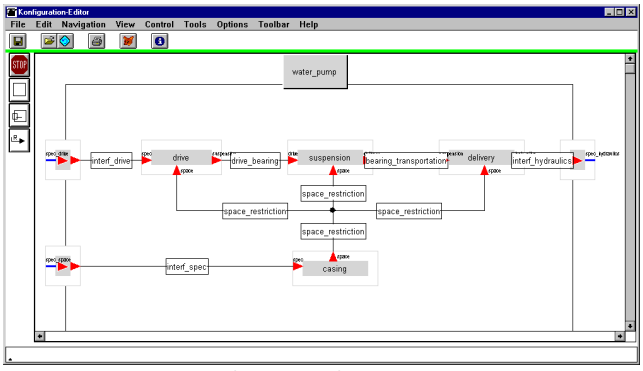

Figure 6. System refinement of a water pump

3.3 The Third Step to Reusability: Modular Constraints

The last step in achieving as much modularity as possible is to modularize not only the product, but also the design logic behind it. This is a crucial factor for the maintenance of the knowledge base behind a configuration design tool, which often suffers from a lack of modularity in the knowledge base and therefore cannot be maintained cost-efficiently. Often, local changes in a knowledge base do not remain local, but lead to changes in other parts of the knowledge base, which in their turn cause other changes etc. In order to keep local changes local, the knowledge base has to be modular, too.

SDR constraints attached to systems fall into one of three classes, depending on how much the parameters/variables in the constraint are spread:

- constraints over parameters at one single system,
- constraints over parameters at subsystems of one system,
- constraints over parameters which are distributed over several systems.

Figure 7 shows a modular constraint between a system which is attached to the interface between the casing and the other subsystems of the water pump (driving, suspension, delivery). Each subsystem must fit into the casing and therefore must have a smaller overall-length than the casing. Notice that the constraint does not need to be changed if another subsystem is added.

Constraints of the first kind are local and raise no problem in terms of reusability, as they have no connection to other systems. Constraints over parameters at subsystems occur frequently, e.g. to compute the total weight of a system by summing up the weights of its subsystems. If we restrict ourselves to constraints connecting different subsystems of the same system refinement, these constraints do not limit reusability. Therefore, the subsystems of a certain system type are part of its definition and thus available any time the system is used.

The third kind of constraints is more tricky and the one that usually hinders reuse. One cannot assume that several systems are always used and reused together just because there is a constraint that touches all of them. Moreover, it is not quite clear which system the constraint should be attached to. The SDR solution is to attach it to none of the systems, but to an interface that connects the systems. This has the advantage that it does not arbitrarily place chunks of knowledge at one system where others are also affected. It also stresses that interface modeling is very helpful in reducing the maintenance cost for a knowledge-based configuration design tool.

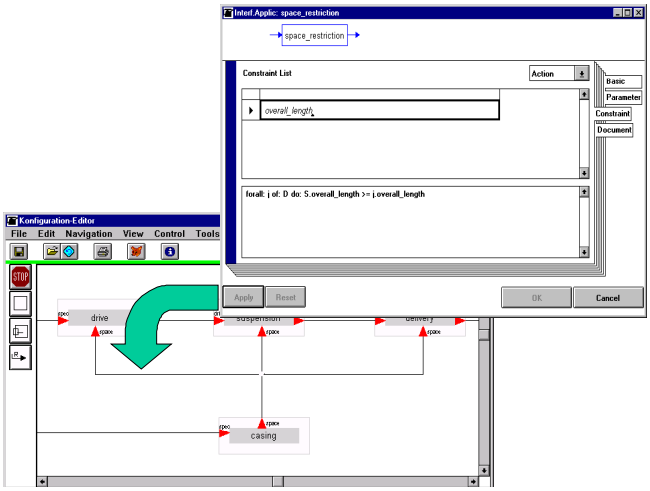

Figure 7. A constraint between systems attached to an interface

4 SDR AS A DESIGN ASSISTANT

SDR is an assisting tool which relieves the designer from routine work steps and focuses on the early design stages (see Figure 1). By getting rid of boring and time-consuming tasks he / she can concentrate on problems which demands the engineer's creativity and experience. As a formalized description of all significant relationships between subsystems belongs to a SDR model, SDR is able to find automatically all consequences of the engineer's decisions and can keep track of the design process. A SDR user can explicitly describe a solution for a specific customer as well as a generic solution which offers the freedom of being adapted to different requirements.

The most simple way to use SDR is to regard it as a comfortable editor for system design tasks that captures design information and allows to easily navigate through a large system design space. Design information is structured according to the product structure. The model structure underlying SDR can model the vertical product structure (the classical product tree) as well as the horizontal structure between subsystems. The user (designer) mainly works with an editor where he / she can plot the internal structure of a system by drawing a graph where nodes represent subsystems and links between the nodes represent interfaces between subsystems. He / she can pick a node to refine it, which means that a new editor window pop up where the user can enter the internal structure of this system. Parallel to this editor, SDR shows the resulting product tree in a second window. The user can also define parameters for each system and interface and attach comments to systems, interfaces and parameters.

SDR also performs computations and checks on the current system design, based on constraints attached to system and interface types defined by the user or reused from the taxonomy. Selecting a type forces the system to load the knowledge expressed in the constraints into the current design. This way, SDR does not only support the reuse of systems and components, but also serves as a corporate memory for engineering knowledge.
Integrating constraint techniques into SDR brings two challenges. The first is to integrate them seamlessly with the structure modeling and taxonomy techniques without losing the reusability of systems and interfaces. The second is to make the constraint machine flexible enough so that SDR can be an interactive assistant instead of a tool that forces the engineer to solve system design problems in a predefined way and hinder the engineer's creativity.

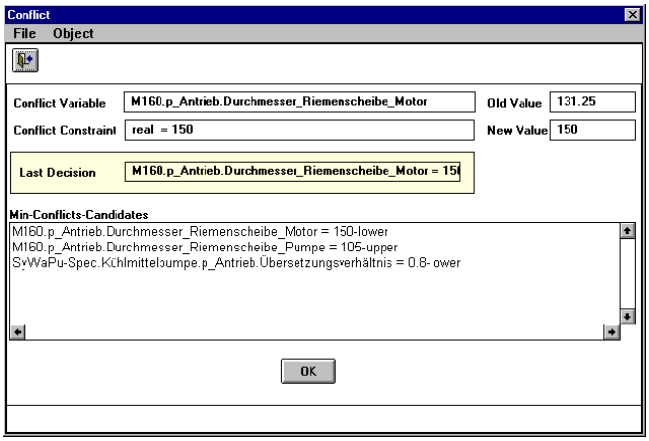

Figure 8. SDR showing an inconsistency using a conflict sheet

SDR supports a least commitment approach which is essential in system design for two reasons. First, it allows the engineer to express the uncertainty in both design specification and design decisions by assigning intervals or sets instead of crisp values to parameters. Second, it improves reusability. A motor designed to run at a range of 200 V to 230 V should be modeled with this interval instead of the crisp value of 220 V the motor runs within a certain design. The least commitment approach is supported by a system taxonomy and by reasoning on intervals, e.g.

5 CONSTRAINT SOLVING

The SDR inference engine enables SDR to be an interactive design assistant. It consists of two parts: a constraint machine that is based on the DeltaBlue algorithm [9] and was extended to handle value domains, and a JTMS (justification-based truth maintenance system, [5]). The inference engine performs the following tasks:

- handle numerical as well as symbolic parameter values,
- propagate value ranges (intervals for numerical parameters) and value sets (for symbolic parameters) to support a least-commitment approach to the definition of parameter values,
- check parameter value ranges and sets for consistency with constraints (an empty interval or set means an inconsistency),
- help the user to resolve inconsistencies by computing the minimal conflict set of value assignment to parameters that caused the inconsistency,
- continue the design process even when the design state is inconsistent.

Figure 8 shows how SDR presents a conflict. The designer has specified that the new water pump has a transmission rate of 0.8 (ratio of the diameters of the pump's and the motor's belt pulley). He / she has selected the water pump M160, an existing type. The diameter equals 150 (motor's belt pulley) and 105 (pump's belt pulley), resp. These three statements are inconsistent.

6 RELATED WORK

The idea of types is not a new one, of course. Their integration with a recursive product structure has been solved in a similar way by [4], [16]. Object-oriented product models and constraints can, for example, also be found in [18], [19]. Extensive interface modeling and the strict modular use of constraints are ideas that we have not seen published anywhere else. Our approach was influenced by the idea of resource-based configuring [12]: A technical system is automatically configured by assembling a set of given components, and the interdependencies between these components are described by means of resources. Configuration is mostly based on the closed world assumption. Working with SDR is not only about using a library of parts / objects but about designing new parts. So SDR is not just a configuration tool [11] but a design assistant as it supports the early phases of design.

Taxonomies are well known from object-oriented programming languages like C^{++} or Smalltalk [10] and from 'description logic' [3], [9].

Typically block diagrams are used for describing the internal structure of a system. Most simulation systems offer hierarchical structure modeling, often based on DEVS [19]. Dedicated software object-oriented engineering methods are presented in [2].

Constraint techniques are still an expanding field of research. [13] gives an overview for finite domain constraint techniques, [14] presents a description for arithmetic constraints. [1] defines a constraint logic programming language over intervals. [15] gives an overview over constraint logic programming.

[6] describes a first approach on combining these basic techniques, used in simulation models to determine a consistent set of initial state variables. There are some differences between this work and ours, however. We understood from [6] that their approach does not use typed interfaces organized in a taxonomy, although they model connections between components. Other differences concern the inference engine: their work is based on the SkyBlue algorithm, the non-incremental version of DeltaBlue [9]; they do not use an equation solver but have to enter all functions belonging to a constraint explicitly; there is no means of handling sets and value intervals. They also do not tackle problems of modularity.

Our work does not primarily aim at improving these basic technologies. Instead, we combine these technologies to create applications that can tackle complex, business-relevant application problems. Our field of research is the interaction between basic technologies in order to make them work together. We regard the improvement and integration of all these ideas and their implementation in the tool SDR as the major contribution of SDR to the scientific community.

7 FUTURE RESEARCH

Our future research will tackle the following issues.

7.1 Reasoning

- SDR will perform reasoning on sets. Example: each part has a set of different legal permissions. The permission of the entire product is calculated as the intersection of all part permissions.
- We think about connecting SDR to a commercial constraint solver or an equation solver. We are searching for suitable tools.
- SDR should reason on qualities of system parameter. An example: a device consumes electrical energy with a standard voltage of 230V which is required to be within a range of ±10V and a frequency of 50 Hz ± 1Hz.
- SDR should be able to select automatically some system types which best fit to a given system application or system instance. Either the user selects a type among some pre-selected types or the tool automatically selects a type. Possible techniques: similarity assessment as known from case-based reasoning or graph grammars.

7.2 User Interaction

- A user should enter the input parameter via Intranet.
- How to present the user the results of reasoning efficiently? How to filter the important data and constraints and to omit the other ones?
- The reasons of an inconsistency should be presented as shortly as possible. The final cause of a conflict should be discovered.
- The ability of SDR to check the constraint network on completeness and consistency will be improved. An example is a check whether some constraints contradict each other, i. e. they can ,in principle, not be fulfilled. An example: x = y AND y = z AND z = x + 1 can never be fulfilled.

7.3 Additional Features

- A documentation should be generated automatically. We have implemented a prototypical solution. A good way seems to connect SDR to a commercial report generating tool.

7.4 Conclusion

We do not have complete and sufficient solutions but we want to encourage a discussion about the topics.

ACKNOWLEDGEMENTS

We would like to thank Gerhard Trauzettel, Detlev Wohlgemuth and Axel Schermann from the water pump division, Ute John, Rüdiger Klein, and the other members of the FT3/EW team for their useful comments on this paper.

REFERENCES

[1] Benhamou, F.; McAllester, D.; Van Hentenryck, P.: *CLP (Intervals) Revisited*. SLP 124-138; 1994

[2] Booch, G.: *Object-oriented Analysis and Design. With Applications*, Benjamin/Cummings Publishing, 2nd Edition, 1994

[3] Brachman, R.J.; Levesque, H.J.: *Readings in Knowledge Representation*, Morgan Kaufman, 1985

[4] Callahan, S.: *Relating Functional Schematics to Hierarchical Mechanical Assemblies*, Proc. Fourth Symposium on Solid Modeling and Applications 1997, pp.229-239, ACM, New York, 1997

[5] Doyle, J.: *Truth Maintenance Systems*, Artificial Intelligence, 12(3):231-272; 1979

[6] Feldkamp, F., Heinrich, M., Meyer-Gramann, K. D.: *SyDeR - System Design for Reusability, Engineering Design and Automation (EDA)*, Maui, Hawaii, USA, 9th-12th August, 1998

[7] Feldkamp, F., Heinrich, M., Meyer-Gramann, K. D.: *SyDeR - System Design for Reusability*, AI-EDAM Special Issue on Configuration and Design, September 1998

[8] Feldkamp, F.; Heinrich, M.; Meyer-Gramann, K. D.: *Product Models and Reusability*. AAAI'99 Workshop on Configuration, July 19, 1999, Orlando, Florida, 1999

[9] Freeman-Benson, B.; Maloney, J.; Borning, A.: *An Incremental Constraint Solver*. CACM 33(1): 54-63; 1990.

[10] Goldberg, A.; Robson, D.: *Smalltalk-80. The Language*. Addison-Wesley, Reading, MA, 1989

[11] Günter, A.; Kühn, C.: *Knowledge-Based Configuration. Survey and Future Directions*. In Puppe, F.: *Knowledge-Based Systems*, 47-66, Springer 1999

[12] Heinrich, M., Jüngst, E.W.: *A Resource-Based Paradigm for the Configuring of Technical Systems from Modular Systems*, Proc. Seventh IEEE Conference on AI Applications (CAIA '91), 257-264, 1991

[13] van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*, MIT Press, Cambr., MA, 1989

[14] Jaffar, J.; Lassez, J.-L.: *Constraint Logic Programming*, Proc. 14th ACM Symp. on Principles of Programming Languages, Munich, 1987

[15] Jaffar, J.; Maher, M.: *Constraint Logic Programming: A Survey*, Journal of Logic Programming, Special 10th Anniversary Issue, Vols 19/20, May/July 1994

[16] Rappoport, A.: *A scheme for single instance representation in hierarchical assembly graphs*, Falcidieno, B.; Kunii, T.L. (eds.), IFIP Conference on Geometric Modeling in Computer Graphics, Genova, Italy, June 1993, Springer 1993

[17] Reiter, R.: *A theory of diagnosis from first principles, Artificial Intelligence*, 32(1), pp. 57-95, 1987

[18] de Vries, T.; Weustlink, P.; Cremer, J.: *Improving Dynamic System Model Building Through Constraints*; in: Computer Aided Conceptual Design '97, Proc. 1997 Lancaster Intern. Workshop on Engineering Design CACD '97; Lancaster University Engineering Design Centre 1997

[19] Zeigler, B.P.: *Object-Oriented Simulation with Hierarchical, Modular Models - Intelligent Agents and Endomorphic Systems*, Academic Press, Boston; 1990