Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12*, 373–382. Printed in the USA.

SyDeR—System design for reusability

FRANK FELDKAMP, MICHAEL HEINRICH, and KLAUS DIETER MEYER-GRAMANN
Daimler-Benz AG, Research and Technology, IT for Engineering Lab., Alt-Moabit 96a, D-10559 Berlin, Germany

(Received October 31, 1997; Accepted February 5, 1998)

Abstract

A fixed set of components with a fixed set of properties is often regarded as a defining characteristic of configuration design problems. In configuration design for larger and more complex systems (e.g., street cars), however, this often is not really true. The reason is that sometimes the component library offers no component that meets the actual requirements, so that a new component has to be added to the library. In general, configuration design for larger products must be interactive: interactive in the sense that the user controls the configuration process and in the sense that the phases of knowledge acquisition and configuration are partly mixed. This paper describes a method and a corresponding software tool (SyDeR, System Design for Reusability) that support the interactive configuration design of complex products, especially in the tendering phase. It combines three different technologies:

- structural modeling of technical systems;
- a library for technical solutions, which is based upon taxonomies and allows the reuse of the technical solutions;
- constraint techniques to propagate design decisions and check designs for consistency to support interactive configuration design.

This paper gives an overview of the functionality of the SyDeR tool and describes the main ideas behind the modeling language, which is especially oriented towards the requirements of system design problems. It also explains how we integrated structural modeling, taxonomies, and advanced constraint reasoning techniques into real-world applications.

Keywords: Configuration Design; System Design; Knowledge-based Systems; Case-based Reasoning; Constraints

1. REQUIREMENTS FOR A CONFIGURATION DESIGN TOOL

SyDeR is intended to support engineers with design tasks in bidding and order processing. These tasks share the following typical characteristics: Contrary to the widespread definition of configuration design problems (Coyne et al., 1990), we do not assume a fixed set of components. In many application areas (e.g., street cars), technical progress and advanced customer desires make it necessary to extend the set of components available with each project the engineer works on. As a consequence, a configuration design tool should include a graphical editor that allows entering new components—and also the configuration knowledge that comes with them—while working on a design project. This in turn requires that the modeling language is not too sophisticated, so that the engineers can handle it.

Components are not the only thing that can be reused between different projects. Often, parts of the system structure are used again and again; e.g., the overall structure of a car engine water pump is quite the same for every engine (see below, application case study). The configuration design modeling language should allow specifying structures which are suited to be reused. Moreover, the structure information should be stored in a way similar to the way the components are stored in the tool, and the tool must allow specifying new structures. Besides that, it should make structure knowledge explicit by displaying it to the user, e.g., in a graph-like form, and not leave it implicit.

Besides that, the tool must support interactive configuration design. A completely automatized, batch-like approach works only for small configuration design problems. On

Reprint requests to: Frank Feldkamp, Daimler-Benz AG, Research and Technology, IT for Engineering Lab., Alt-Moabit 96a, D-10559 Berlin, Germany; E-mail: feldkamp, heinrich, meyer@dbag.bln.daimlerbenz.com.

large-scale problems, users want to have more influence on and control over the configuration design process, for several reasons:

- The user wants to see how the design tool created the solution, replay the solution process, and take a closer look at certain steps (explanations for the configuration design).
- The user wants to control the design process by retracting the design decision made by the tool. If he or she does so, the configuration design tool can be expected to automatically retract all decisions by the tool that depend on the one which the user retracted (and only the dependent ones!).
- The tool should never retract user decisions.
- In case of a deadlock in the configuration process, the user should get some hints on which decision to retract in order to resolve the conflict.
- In general, it is often desirable to have the possibility to explore the dependencies between different configuration steps.

Another important aspect concerning the configuration design process is that large configuration design problems are usually tackled by the strategy of structural decomposition. The engineer breaks the system which has to be configured down into subsystems, which are again subdivided, etc. This way, he or she reduces the complexity of the configuration problem so as not to deal with the details hidden in the subsystems.The modeling language as well as the configuration design tool should take this strategy into account.

There is still another strategy to handle the complexity and uncertainty that is inherent in larger design problems: a least-commitment approach to the definition of parameter values. Usually, in an early configuration phase, it is not desirable to commit to precise values for certain parameters because it is still not clear what the optimal value is. But normally an idea is obtained in which range the value should be, so an interval range can be specified, by which the parameter value is covered. The configuration design tool should accept intervals as parameter values and also be able to handle intervals.

2. INTRODUCTION TO SyDeR

To meet these requirements, the SyDeR tool combines three different approaches:

1. A design library for configuration design cases, components, structures, etc., organized in a taxonomy.
2. Structural modeling of technical systems.
3. Constraint techniques to propagate design decisions and check designs for consistency to support interactive configuration design.

SyDeR allows the search for complete and partial solutions in a design library and their modification to meet given requirements. While modifying the design, the engineer can use the SyDeR tool to perform calculations and check the design for consistency. The design library also serves as a knowledge base containing design descriptions as well as reasons for design decisions (justifications for propagated parameter values and free-text design rationales) made in earlier design projects and background information about the design domain, e.g., definitions for design parameters or explanations concerning best practices. By making this information available, SyDeR can play the role of a corporate memory for engineering knowledge.

The rest of this paper is organized as follows: It first explains the SyDeR perspective on configuration design, then deals with the three basic technologies employed in SyDeR (structure modeling, taxonomy-based design library, and constraint techniques) and finally describes how SyDeR and its underlying design method benefits system design.

3. CONFIGURATION DESIGN ACCORDING TO SyDeR

The ideal configuration design process consists of two steps: enter the design specification and obtain the design fulfilling the specification. As mentioned earlier, this is possible for small-scale configuration problems (Heinrich & Jüngst, 1991), but larger problems require more interactivity and more choices for the user.

3.1. Interactive Configuration Design

In a more interactive approach, configuration is an interplay between specifying and selecting component types, guided by a structural decomposition strategy which introduces decomposition steps in between. The engineer starts the configuration process by selecting a component type on the top level of the system (the kind of system configuration desired). After this selection step, the parameter values are specified for the top level component. The tool applies configuration design knowledge to derive consequences from the specification, i.e., new parameter values. Now, the engineer performs a decomposition step to descend one level in the structure hierarchy.

Here, the cycle starts again. For each of the subcomponents, the engineer selects a type, specifies the component, lets the tool parametrize the component, and further decomposes the system to be configured.

In the selection step, the tool can—on the engineer's wish—search through its library and come up with some predefined solutions that meet the specified requirements, or the engineer can select a type on his or her own.

This configuration design loop assumes that

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

- there never arises a conflict between specified values,
- that we only need the specification parameters to parametrize the components, and
- that the library of components has every component in store we might ever need.

These premises cannot be taken for granted, especially if you are not dealing with larger systems like street cars, subway trains, or trucks. We will examine in the next paragraphs how a design cycle without these premises might look like.

3.2. Realistic configuration design for larger systems

A realistic configuration design process supported by SyDeR is given in Figure 1. The engineer starts with the design specification as explained above. The constraints in the model allow the derivation of further system parameters from the specification and detect inconsistencies, so that the engineer can deal with them.

In the next step, the design library is used to search for the optimal system and interface types. The type assignment brings new information and design knowledge into play. New parameter value domains are inherited as well as additional constraints. The new value domains are intersected with those previously defined, and thus narrow the value domains. The additional constraints are employed to derive new values and perform additional checks on consistency. Again, the constraints should be so complete that the subsystem applications are completely parametrized.

If performed in this way, configuration design is still very efficient, even under the new premises. The engineer can concentrate on specifying the system and resolving inconsistencies. Moreover, design quality is enhanced by the use of predefined and proven solutions. The precondition for such an efficient design process is, of course, a "good" library, based on a clean and thorough modularization and comprehensively modeled constraints.

4. TAXONOMIES, CASES, AND DESIGN KNOWLEDGE IN SyDeR

There are three taxonomies in SyDeR: organizing system types, interface types, and parameter types (see structure modeling for details on systems and interfaces). As we will explain below, we think it is crucial for efficient knowledge base maintenance to model types for interfaces between

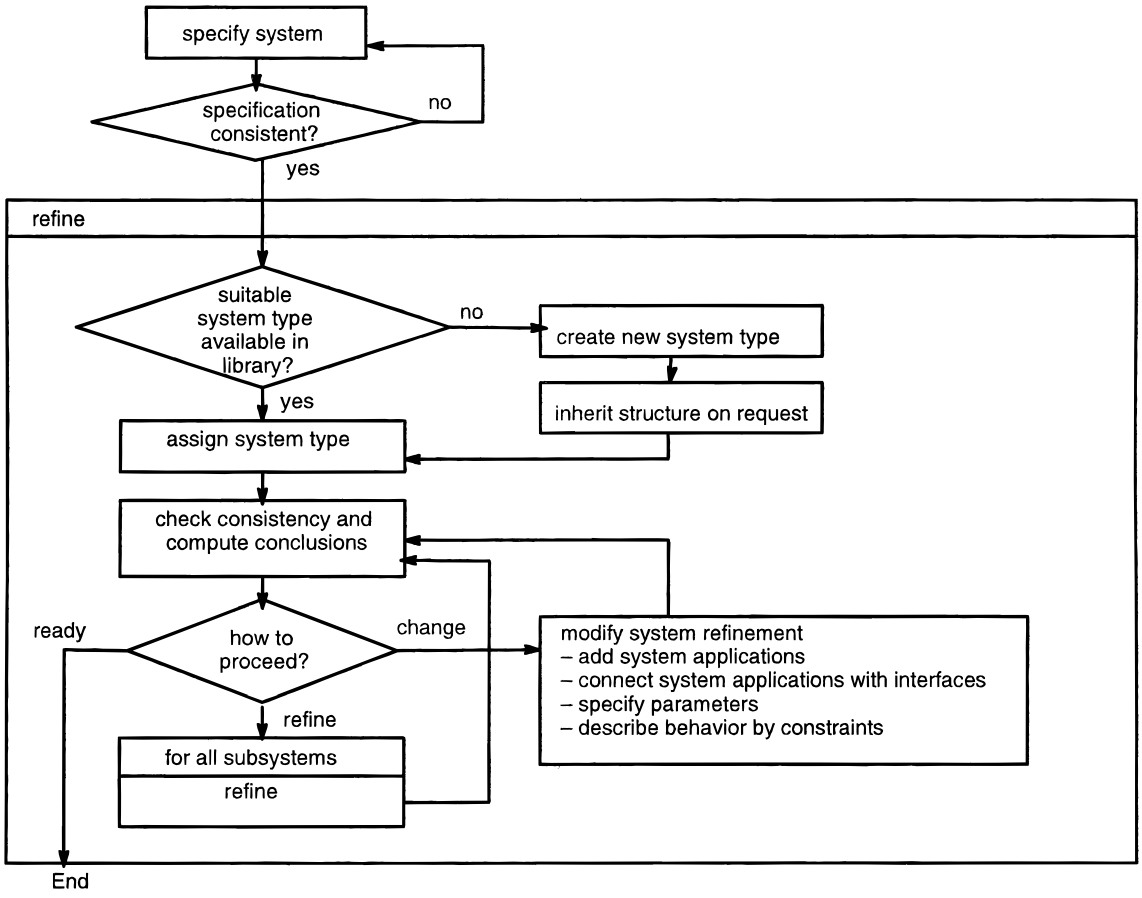

Fig. 1. The system design cycle.

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

systems in a separate taxonomy as a backbone for the knowledge base. It can also be helpful to keep parameter types in a taxonomy, although this is not as important as the interface type taxonomy.

The taxonomies are implemented as single inheritance hierarchies. This allows for flexible modeling of knowledge about component types and interface types. Knowledge modeled with the help of SyDeR is basically expressed in terms of parameters and constraints on the one hand and structure description on the other hand. SyDeR uses interfaces and ports to describe the product structure (see below, structure modeling).

Parameters and constraints that are valid for a whole group of objects are defined at that level in the taxonomy which corresponds to this group of objects. Therefore, redundancy in the model is avoided because facts common to a group of objects are modeled once for the group as a generic supertype and not for each object independently. Structure knowledge about a component can be inherited on demand by the user. We did not implement automatic inheritance because structure tends to vary stronger than parameters and constraints between types and subtypes. It would be annoying to automatically inherit a structure that does not fit.

This flexible modeling is not only essential for economy in modeling, but also to adequately support the design process. Usually, it becomes only gradually clear during the design process which kind of part is used to implement a certain function. For example, the designer starts with defining that a certain part is a pump and later on decides that a certain subtype of pump is adequate for this project and even later selects a specific kind of pump. If knowledge about pumps were only modeled at the leaf types of the taxonomy, there would be no chance for the design tool to support the engineer during the earlier stages of design. It could just start to draw inferences about the design when the specific pump is chosen. This can be avoided by modeling generic types as supertypes of real types, including parameters and constraints, as far as is known on that level of abstraction.

The taxonomies also serve as libraries from which the engineer selects complete or partial solutions for the current design problem. It is important to note that the library offers cases for each level in the system structure, so that the taxonomy reflects the structure model.

When the engineer has to find a solution for a system or interface, the taxonomy tree can be scanned to look for a system or interface type that can be employed in the design, or the SyDeR tool can be instructed to search the taxonomy for a certain needed part. The search is specified by selecting a start node in the taxonomy (defining the kind of part being sought, e.g., a water pump) and a set of restrictions on system parameters typically formulated as intervals, e.g., specified performance parameters. SyDeR then presents a set of proposals from the library. The application engineer takes a closer look at some of them by examining their attributes and reading comments attached to the proposals by the engineers who designed the case in the first place.

If one of the proposals seems to be a good candidate for fulfilling the requirements, it can be checked out in the current design by evaluating the consequences using the constraint machine (see below) which propagates parameter values (and thus performs design calculations) and checks parameter values for consistency with design rules and other parameter values.

The SyDeR knowledge base is not restricted to knowledge which can be processed by the constraint machine. It also provides the engineer with background information about the application domain (e.g., parameter definitions or details concerning design standards or recommended best practices for design). The engineer can also extend this knowledge by explaining the reasons for the design decisions (in plain text). This is not only helpful to understand later on the design but also to document design rationales for other engineers.

5. STRUCTURE MODELING IN SyDeR

The SyDeR structure model allows the description of the vertical as well as the horizontal structure of a product to be configured. The vertical structure describes how the product is broken down into subsystems/subcomponents on several levels, resulting in a product tree. The horizontal structure describes how the components inside another component are connected to each other and what their logical relationships are.

A major contribution of the SyDeR tool is how it models the horizontal structure. The basic idea is that formally defined interfaces connect components. This approach has two advantages:

- A strict modular approach to configuration design is supported.
- Interfaces serve as a stable backbone to the configuration design knowledge base.

Modularity is the very key idea behind configuration design. The product that has to be configured must be broken down into modules with clearly defined properties and interfaces, so that there are no relationships between components other than those described by interfaces. A weak modularization causes unnecessary costs in production, stockkeeping, product support, and product redesign. So, the configurability of a product or product family is a good indicator for design quality in terms of modularization and for the profitability of the product family. This gives an idea how the formal techniques embodied in SyDeR not only help to configure a product but can also help to improve product design.

In our opinion, interface typing is an important point in structure modeling for another reason. Interface definitions are much more stable than system definitions. An example: the definition of the RS232 interface has been unchanged for decades in which the systems using this interface (e.g.,

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

computers) have been completely redesigned for several times. So, it makes sense to use the interface definitions as a backbone for the design knowledge base which undergoes only slight changes over time. If the interfaces are defined carefully, system definitions can be exchanged without affecting other system definitions in the knowledge base, resulting in less knowledge maintenance expense. This idea is already used by the "resource-based configuring paradigm" (Heinrich & Jüngst, 1991).

Before we discuss the SyDeR structure model in detail, we introduce some SyDeR terminology. Components, modules, systems, and similar entities are subsumed under the term *system*. The connections between systems are called *interfaces*. Interfaces are plugged into systems *via ports* attached to systems. Systems are classified as *system types* which are organized in an inheritance hierarchy, called *system taxonomy*. Correspondingly, *interface types* are organized in an *interface taxonomy*.

Ports are characterized by the interface type that may be plugged into the port and identified by a name. Ports do not form a taxonomy of their own. Port descriptions are part of the system pattern and apply for system applications as well as for system refinements, so they are modeled in the system taxonomy.

The information about port-interface compatibility is just one example for a broad range of syntactic consistency checks that SyDeR can perform on design models. Syntactic checks save a lot of time because simple mistakes can be identified immediately. Moreover, they raise the design quality. Further checks examine the number and direction of interfaces connected to a port. Of course, SyDeR also performs semantical checks on the design description (see section on constraint techniques).

6. HOW TAXONOMIES AND STRUCTURE MODELS MIX

Systems belonging to a system type appear in two roles in the configuration design knowledge base. The first role is that of a part in the structure definition of another system (e.g., the traction/brake system is part of the system light rail vehicle), and the second role is as a system that is described in terms of its inner structure (e.g., the traction/brake system is broken down into its subsystems). These two roles correspond to two different views on the system type. One is from the outside (when the system is part of another system), the other is from inside (when the system is broken down into its subsystems).

To model these two roles of a system type, two new terms are introduced in the SyDeR modeling language: *system applications* and *system refinements*. System applications appear in the structure definition of another system, and system refinements define the inner structure of the system itself. How are the system type, the system application and the system refinement related to each other? The system type defines parameters and constraints that are also valid for both the system type and the system refinement. This shared set of knowledge is called the *system pattern*. The system application can extend this pattern by adding parameters or constraints that are valid only in the context of a certain system refinement, but not in other contexts. Anyway, this should be a last resort in modeling a product. Extending the system pattern makes sense when the engineer is not really sure which kind of new system type he or she will define and temporarily wants to keep a modified system application as a prototype for the new system type.

The system refinement describes the internal system structure by listing the subsystems and by showing how the subsystems are connected to each other (see Figure 2). Here we also see that system applications appear in the refinements of other systems.

Ports attached to a system application are mapped to corresponding so-called *external ports* in the system refinement (see Figure 3).

Interfaces are allowed to have an internal structure, too. For example, a bus interface can be broken down into several wires, which are again treated as interfaces. External ports handle the interface refinement. Viewed from outside the system refinement or interface refinement, the external port is connected to one interface. Inside the refinement the port may be split up into several ports, each corresponding to one of the subinterfaces. (Think of a connector of a electronic system which is internally connected to different wires.)

SyDeR also takes into account that usually there are many different views on a configuration design, often based on different disciplines like mechanical design, functional design, electrical design, or cost estimation. The engineer can define different views on the system and attach subsystems, interfaces, parameters, and constraints to one or several of these views. It is possible to define a separate system refinement for each view the corresponding system type is attached to, so that the different structures that belong to the different views can be modeled separately. If the engineer concentrates on one view, SyDeR only presents data attached to this particular view.

7. CONSTRAINT TECHNIQUES IN SyDeR

Constraints in SyDeR describe technical, financial or other relationships between system parameters, port parameters, and interfaces parameters. Relationships can be laws of physics, rules to estimate cost, or design standards that apply. What makes constraints useful for interactive configuration design purposes is that they can be used in either direction depending on the current demand to compute unknown parameter values (more precisely, value domains), and to check whether the relationship is satisfied if all parameter values are given. As a consequence, the engineer is not forced to make a distinction between input parameters and output parameters, which would define a more or less fixed configuration design process.

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

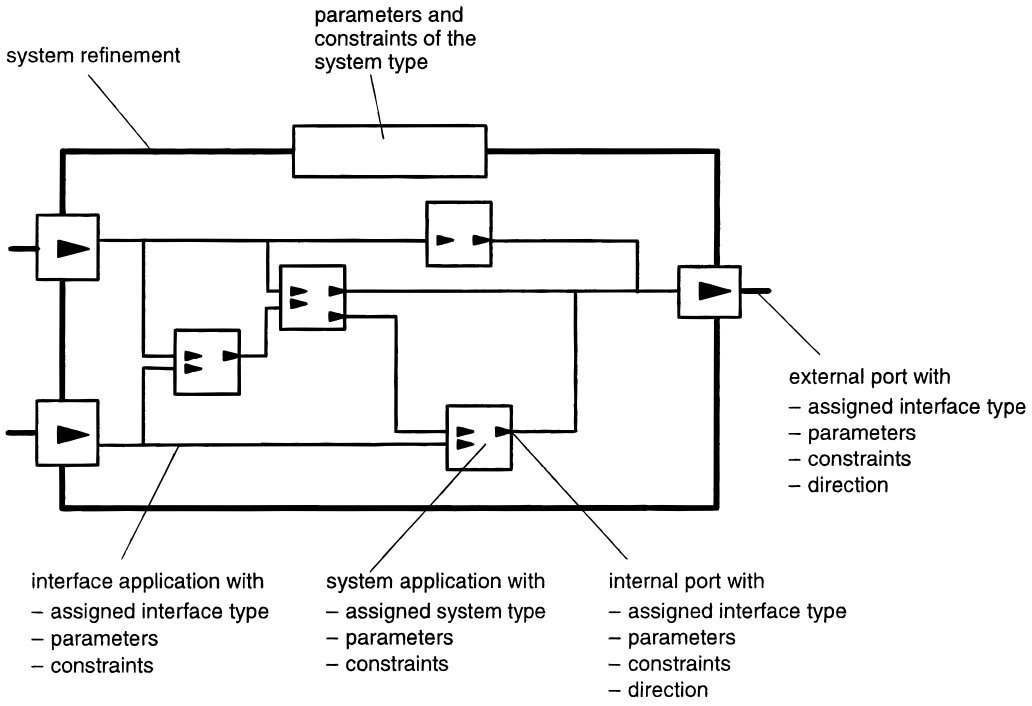

Fig. 2. System refinement: internal structure.

Relationships in design mainly fall into two categories: algebraic relationships (e.g., laws of physics) and relationships over finite domains (e.g., different materials). As a consequence, both algebraic constraints and constraints over finite domains have to be handled by a constraint-based design support system.

The SyDeR inference engine consists of two main modules: a constraint machine (based on the DeltaBlue constraint machine) and a JTMS (justification-based truth maintenance system) (Doyle, 1979). The inference engine delivers the following tasks in configuration design:

- propagate specified or defined parameter values,
- check parameter values for consistency,
- handle parameter value intervals,
- handle sets/finite domains as parameter values, and
- provide interactivity in the configuration design process.

The former two tasks are what constraint systems are usually used for. They compute new parameter values from known parameter values exploiting the constraints which model the relationships between parameters. They also allow the design states to be checked for consistency by performing consistency checks on the constraint network. These two constraint applications are common, basic automatization steps in configuration design.

Less common is that a constraint machine handles intervals to permit a least commitment approach to configuration design.

During the design process, the engineer gradually narrows the possible value domain for the design parameters. A few parameters may have crisp values from the beginning but most are specified in a certain range or only restricted by coarse default ranges. When more information is added to the design description, the propagation of restrictions across the constraint network causes the value domains to become narrower and narrower. This corresponds to our design philosophy which regards design as a process that step by step narrows the design space until the space of acceptable design solutions is reached.

Constraint systems that only handle exact values cannot support this aspect of system design. Therefore, we decided to implement a constraint system that reasons over value domains (with monotone functions) instead of exact values. So, design decisions concerning parameter values in SyDeR are always decisions over value domains rather than over crisp values and are represented as constraints that restrict the value domain for a parameter. Another important extension is that SyDeR can handle constraints defined over parameters which are vectors, often used to model parameters that depend on the mode of operation. Note that a system design must be valid for each possible mode of operation.

SyDeR does not parametrize the whole system after pressing a button (although it could do so), but it segments the system and computes only those parameters the user wants it to compute. The strategy implemented in SyDeR offers the user a propagation control which uses the structure of the hierarchical model itself.

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

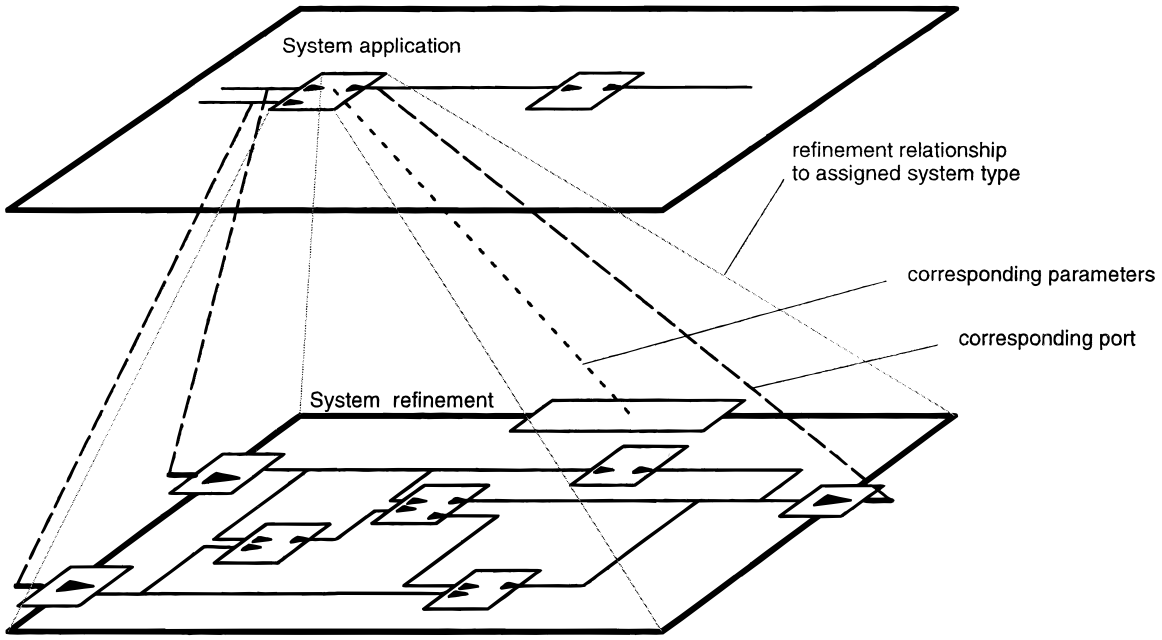

Fig. 3. System refinements and system applications.

Essential for interactive configuration design is the JTMS coupled to the constraint machine. It keeps track of the propagations executed by the constraint machine and thus provides a service basic to several important aspects of interactivity.

- explain parameter values derived by the configuration design tool by showing which parameter values had which effect on other parameter values *via* which constraints;
- allow the user to retract decisions and automatically retract dependent value propagations;
- in case of an inconsistency, compute the minimal set of possible culprits (imposed constraints) for the inconsistency; and
- give the engineer the chance to continue his or her work even if the design state is inconsistent.

The last two points deserve further explanation. Inconsistencies are conflicts in terms of the constraint network and can be easily detected by the constraint machine, when the consistent parameter value domains turn into empty intervals or sets. However, the question on how to resolve the conflict is much more difficult to answer. A lot of different constraints may have played a role in the process that has led to the inconsistency. Fortunately, the JTMS can compute the minimal set of user decisions (represented as constraints) that lies at the core of the inconsistency. It can be guaranteed that no other constraints than those in the minimal conflict set have to be revised in order to resolve the conflict.

This offers the opportunity to resolve conflicts in a flexible way. The engineer is not forced to retract the last decision he or she made, but can retract one of the decisions which caused the inconsistency and have the consequences of this retraction computed by the constraint machine and the JTMS.

Sometimes, it is not possible to resolve the inconsistency immediately. Maybe the inconsistency touches the work of another colleague, so that you may want to discuss the topic with him or her first. It may also be that the fastest track through the "configuration design landscape" from the specification to the final solution is through "forbidden territory," and it would mean a serious waste of time to find a "legal" (read completely consistent) path. In this case, the user retracts the constraint(s) that restrain him or her and carries on with his or her work. SyDeR keeps the retracted constraints in the model, but ignores them in the update of the constraint network. Later on, the user can enable the constraints again and have them check the design.

The implementation is based on the DeltaBlue algorithm (Freeman-Benson et al., 1990). It was extended to handle value domains and a justification-based truth maintenance system (JTMS) (Doyle, 1979) was included.

8. AN APPLICATION CASE STUDY: CAR ENGINE WATER PUMPS

We verified the SyDeR way of engineering with different examples stemming from the Daimler-Benz group: street cars, total rail systems, and car engine water pumps. In this paper, we explain the ideas behind SyDeR using one of these applications—the car engine water pump.

We will discuss the earliest phase of pump design, where the engineers sketch a rough pump configuration and esti-

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

mate development and production costs for the pump. While a water pump is not as complex as a light rail vehicle or a truck, it is still complex enough to pose an interesting configuration design problem and is better suited for a short paper.

A water pump delivers cooling water through a car engine. It consists of several main components: impeller/spiral, pulley, axis/bearing, and housing. The central part of the pump is the impeller which—together with the spiral housing around it—creates the pressure needed to pump the water through the engine. The impeller is driven by an axis which is in turn driven by the engine *via* the pulley. The axis is held by the bearing.

The SyDeR model for a water pump has three layers: The top level describes the pump as a whole, the second layer models four function blocks (each centered around one of the main components described above), and the third layer describes the components belonging to these function blocks.

The pump configuration process starts with the definition of the pump type (e.g., truck pump, car pump, single pump, compound pump, etc.). This is the first selection step in the SyDeR configuration design cycle. The next step (specification!) is to specify the performance parameters for the water pump: water pressure, rotations per minute, volume per second, Additionally, the engineer can enter the space the water pump has to fit into.

Now, the engineer starts to configure the first function block of the pump, the impeller block (a decomposition step), and lets SyDeR search the taxonomy for an impeller that satisfies the specified performance criteria. SyDeR proposes several alternatives, from which the engineer picks one as a start (selection step). The choice of impeller brings new parameters into play, like the size of the impeller, etc. As the impeller performance strongly depends on the geometrical form of the spiral housing around it, selecting an impeller always means selecting a spiral housing too. The engineer can now define further parameters for the impeller, have SyDeR compute other parameters, and check whether the overall configuration is still consistent.

The engineer can now configure other parts of the pump. Taken together with some data about the engine, the impeller data facilitates parametrizing the axis and specifying the axis bearing in the bearing function block. To find a suiting bearing, the engineer again lets SyDeR look for a bearing that meets the specification and picks one of the proposed parts.

Let us have a closer look on how the bearing is specified because it demonstrates some of the ideas embodied in SyDeR. The specification parameters for the bearing (i.e., the forces in axial and radial direction it has to bear) are derived from forces at the pulley and from the dimensions of the axis *via* some leverage equations. This calculation is not modeled with one single constraint, attached to the bearing and referencing the pulley and the axis. This would make it impossible to reuse the bearing without the other components around. Instead, the input parameter values needed for the calculation are "moved" *via* "transportation constraints" at the interfaces between the three systems and the force calculation is done where it belongs according to the laws of physics: at the axis, which serves as a leverage between the pulley and the bearing. After the resulting forces at the "bearing end" of the axis are computed, these values are passed over from the axis to the bearing *via* similar "transportation constraints." This way, all three systems and the interfaces in between can be reused in other environments.

During the configuration process, geometrical data about the selected parts is matched with the spatial specification for the space allowed for the water pump. In parallel, SyDeR estimates development and production cost based on cost data about components that also lies in the component library.

In a short time, the engineer has a reliable conceptual configuration of the pump, together with cost estimates he or she can trust. Using the flexible SyDeR approach reaps several benefits in this application.

New components can be easily added to the library without affecting other components. The engineer can easily define, e.g., a new type of drive (e.g., a toothed wheel drive instead of a pulley drive) and plug it to the pump because of the neat and clean interface definitions. This would be much more difficult if a configuration design tool is used which models dependencies between components directly (without interfaces). Probably, it would have been necessary to call in a knowledge engineer who is familiar with the knowledge base.

The configuration design tool for water pumps can also easily be extended to cover a broader range of applications, e.g., oil pumps or complete cooling systems. Much of the configuration design knowledge that is common to both oil and water pumps can be reused in the oil pump knowledge base. It is also no problem to extend the scope of configuration from a subsystem (water pump) to the whole system (cooling system) because the complete knowledge base can be reused. Thanks to the clean modularization *via* clearly defined interface types, flexible migration paths for configuration design tools become possible that make it much easier to introduce knowledge-based configuration design tools into the business world and adapt them to changing business needs.

Figure 4 shows a snapshot of the water pump in the SyDeR modeling environment.

9. RELATED WORK

A lot of research has been done in each of the technologies mentioned above. Our work does not 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 working together. As far as we know, there is not much research done with a similar intent, with the exception of de Vries et al., 1997.

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

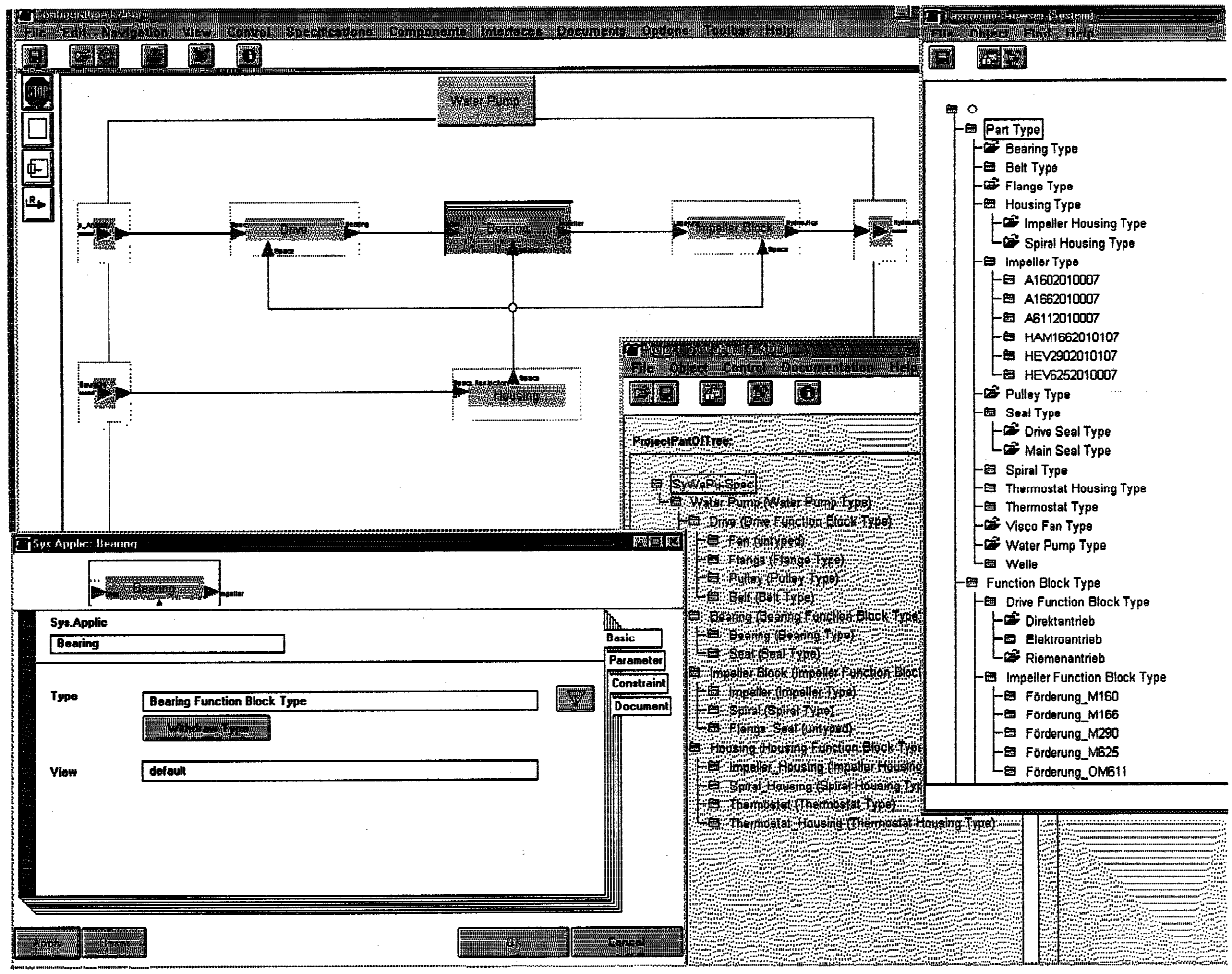

Fig. 4. Snapshot from the SyDeR modeling environment.

Taxonomies are well known from object-oriented programming languages like C^{++} or Smalltalk (Goldberg & Robson, 1989) and from "description logic" (Baader et al., 1990; Brachman & Levesque, 1985).

Typically, block diagrams are used for describing the internal structure of a system. Most simulation systems offer hierarchical structure modeling, often based on DEVS (Zeigler, 1990). In the domain of software engineering, dedicated object-oriented engineering methods are presented in Booch, 1994.

Constraint techniques are still an expanding field of research. An overview for finite domain constraints techniques is given in van Hentenryck, 1989, and a description for arithmetic constraints is presented in Jaffar and Lassez, 1987. A constraint logic programming language over intervals is defined in Benhamou et al., 1994. Jaffar and Maher (1994) gives an overview over constraint logic programming.

de Vries et al. (1997) describes a first approach of combining these basic techniques, used for simulation models to determine a consistent set of initial state variables. There are some differences between this work and ours, however. We understood from de Vries et al., (1997) 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; they do not use an equation solver but have to enter all functions belonging to a constraint explicitly; there is no mention of handling sets and value intervals. They also do not tackle problems of modularity.

10. SUMMARY

We described the three techniques used in SyDeR (taxonomies and cases, structure modeling, and constraints), explained how these three techniques work together in the system design process, and gave an example for a SyDeR application. In this paper, the benefits of the SyDeR method and the interactions between the three techniques are summarized.

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press

One contribution of the SyDeR approach is that it models both the vertical part of the structure and the horizontal connection with the structure, the latter using typed interfaces. This approach has two important benefits:

1. The knowledge base and the modeled product are clearly modularized, as components are described without reference to other components.
2. Interface definitions are much more stable over time than system definitions, which makes them a reliable backbone of the knowledge base.

Also important is that SyDeR makes the product structure explicit and thus gives the user a more intuitive approach to configuration design. The interaction between taxonomy and structure has led to a system type definition that might seem a bit complicated at a first glance (introducing an inside view and an outside view) but which is really helpful in modeling systems for configuration design purposes.

Constraints make the knowledge stored in the taxonomy much more powerful. Associating constraints to objects (e.g., system types, interface types) for which they are valid makes the knowledge base well structured and easy to maintain. A special feature of the constraint techniques used in SyDeR is that it supports the stepwise narrowing of value domains which is a typical feature of the system design process. Moreover, the combination of the constraint machine and a JTMS allows to design a system in an interactive way.

10.1. Modular design

SyDeR stresses the principles of modular system design. This requires a thorough and consistent segmentation of the system on all levels. Moreover, a well-chosen set of interfaces between the modules has to be defined. It is clear that because of the strong interdependency of the two tasks of modularization and interface definition, it is wise to execute them together and at once.

The modularization is a key process that decides over much of the profitability of a product family. It is also the precondition for an efficient product configuration process. If the modularization was carried through carefully, there is an exhaustive range of subsystems (modules) available on each system level, so that the engineer who is configuring a specific product does not need to design a completely new (sub)system but can pick a suitable module from the library and in a short time adapt and parametrize it to meet customer demands. SyDeR supports both the process of product range definition and product configuration.

REFERENCES

Baader, F. et al., (1990). Concept logics. In *Computational Logic*, (Loyd, J.W., Ed.), Springer, Berlin.

Benhamou, F., McAllester, D., & Van Hentenryck, P. (1994). *CLP(Intervals) Revisited*. SLP 124-138.

Booch, G. (1994). *Object-Oriented Analysis and Design. With Applications*. Benjamin/Cummings Publishing, 2nd Edition.

Brachman, R.J., & Levesque, H.J. (1985). *Readings in Knowledge Representation*. Morgan Kaufman.

Coyne, R., Rosenman, M.A., Radford, A.D., Balachandran, M., & Gero, J.S. (1990). *Knowledge-Based Design Systems*. Addison-Wesley.

Doyle, J. (1979). Truth maintenance systems. *Artificial Intelligence 12(3)*, 231–272.

Freeman-Benson, B., Maloney, J., & Borning, A. (1990). An incremental constraint solver. *CACM 33(1)*, 54–63.

Goldberg, A., & Robson, D. (1989). *Smalltalk-80. The Language*. Addison-Wesley, Reading, Massachusetts.

Heinrich, M., & Jüngst, E.W. (1991). A resource-based paradigm for the configuring of technical systems from modular systems. *Proc. Seventh IEEE Conf. on AI Applications (CAIA'91)*, 257–264.

van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Massachusetts.

Jaffar, J., & Lassez, J.-L. (1987). Constraint logic programming. *Proc. Fourteenth ACM Symposium on Principles of Programming Languages*, Munich, Germany.

Jaffar, J., & Maher, M. (1994). Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, May/July (Special 10th Anniversary Issue).

de Vries, T., Weustlink, P., & Cremer, J. (1997). Improving dynamic system model building through constraints. In *Computer Aided Conceptual Design '97, Proc. 1997 Lancaster International Workshop on Engineering Design CACD '97*. Lancaster University Engineering Design Centre.

Zeigler, B.P. (1990). *Object-Oriented Simulation with Hierarchical, Modular Models—Intelligent Agents and Endomorphic Systems*. Academic Press, Boston, Massachusetts.

Frank Feldkamp works at Daimler-Benz Research & Technology in the area of knowledge-based system design, intelligent CAD and knowledge management, with applications in automative and aerospace industry. He received a Dipl. in computer science from the University of Dortmund and is a member of the VDI (Verein Deutscher Ingenieure, German Engineers Association). E-mail: feldkamp@dbag.bln.daimlerbenz.com

Michael Heinrich is a senior scientist at Daimler-Benz Research & Technology. His research focuses on knowledge-based methods for design and configuration tasks, especially model-based approaches and their application, such as trucks, railway rolling stock, and aircraft design. He received a Dipl. in physics from the Technical University of Berlin, and he is a member of the Gesellschaft fuer Informatik (the German society for computer science). E-mail: heinrich@dbag.bln.daimlerbenz.com

Klans Dieter Meyer-Gramann works with Daimler-Benz Research & Technology in the areas of knowledge-based engineering, engineering knowledge management, and object-oriented modelling. He studied Mathematics and Computer Science at the Universities of Goettingen and Braunschweig and received a PhD from the Technical University of Braunschweig. E-mail: meyer@dbag.bln.daimlerbenz.com

https://doi.org/10.1017/S0890060498124095 Published online by Cambridge University Press