A Structure Based Configuration Tool:
Drive Solution Designer - DSD

**Christoph Ranze[2], Thorsten Scholz[1], Thomas Wagner[1], Andreas Günter[3],
Otthein Herzog[1], Oliver Hollmann[2], Christoph Schlieder[1], Volker Arlt[4]**

University of Bremen, TZI[1],	encoway GmbH & Co KG[2]	University of Hamburg[3],	Lenze AG[4]
Universitaetsallee 21-23,	Universitätsalle 21-23,	Dept. of Computer Science and	Hameln
D-28359 Bremen	D-28359 Bremen	HiTec e.V.	Hans-Lenze-Strasse 1
{cs, herzog, scholz, twagner}	{hollmann, ranze}	Vogt-Koelln-Str. 30,	D-31763 Hameln
@tzi.de	@encoway.de	D-22527 Hamburg	arlt@lenze.de
Tel: +49 421 218 7090	Tel: +49 421 246 770	guenter@informatik.uni-hamburg.de	Tel: +49 5154 82 2534
Fax: +49 421 218 7196	Fax: +49 421 246 7710		Fax: +49 5154 82 1920

Abstract

In this paper, we describe the configuration tool *Drive Solution Designer* (DSD). The DSD is used by sales engineers of the company *Lenze AG (www.lenze.com)* for the configuration of complex drive systems in order to make on-site offers together with the customer. The aim of this process is to generate a consistent solution which fulfills the functional requirements of the user along with optimization criteria such as price and delivery time. The preparation of a technical offer requires fundamental knowledge of complex physical and in particular technical correlations of drive components, in depth knowledge of the product catalog as well as high empirical knowledge about the order of the parameterization of the components. In order to meet these requirements knowledge-based AI-techniques are required. In the DSD we use a domain independent incremental *structure-based configuration* approach with different knowledge representation mechanisms and a sophisticated declarative control. Currently DSD is used with great success by approx. 150 sales engineers of the company *Lenze* for the design layout task. The introduction of the DSD lead to a drastic time reduction for drive solution development and reduces incorrect solutions to nearly 0 percent.

Task Description

The growing complexity of drive configurations in industrial sales and distribution scenarios makes high demands on sales engineers. This is caused by the highly dynamical product evolution in combination with constant improvement of the product features and the increasing complexity of customer-stipulated solutions as well as tighter optimality criteria.

This applies especially to products which require profound expert knowledge. The degree of complexity mainly depends on the structure of the products and the closely

---

connected sales scenario. In [Hollmann et.al. 2001] we distinguish three different sales scenarios:

Click & Buy: This is the most simple scenario. Only non-customizable, complete products are offered to the customer from a product catalog (e.g. www.amazon.com). The customer is able to choose a product without taking expert configuration knowledge into account. Complexity- and consistency problems need not be handled and the use of AI-technologies is not necessary.

Customize & Buy: In this scenario, products can be customized with regard to the customers desires. This scenario is found at many car manufacturers like BMW (www.bmw.com) or at computer vendors like Dell (www.dell.com). The combination of valid configurations of components is still restricted and can be represented explicitly (e.g. within the product catalog). Thus complexity and consistency still play a minor role and can be handled by standard information technologies.

Configure & Buy: This is the most complex scenario and requires the use of AI-technologies. The vast amount of possible combinations of components and the multitude of complex domain-specific restrictions lead to a serious complexity problem as it is infeasible to list the set of possible solutions. Thus the generation of a valid solution requires explicit expert knowledge and therefore cannot be done by customers on their own. Even with detailed expert knowledge the creation of valid solutions remains a difficult task which in some cases leads to faulty configurations. An example of this scenario is the *Lenze*-domain handled by the DSD.

The Drive Configuration Task

The problem of designing drives (e.g. for printing- and sorting machines, automated saw, etc.) lies in finding a consistent combination of an engine, a gear and a drive controller while starting from a very limited set of basic data like motor power and torque. The configuration has to fulfill to a high degree the imprecise requirements of the

customer while being as cost-efficient as possible. The selection of the required components and their parameterization implies a high level of engineering knowledge and detailed knowledge about the product catalog. The goal was to develop a software configuration tool for the *Lenze AG (www.lenze.com)* to support sales engineers in this complex task.
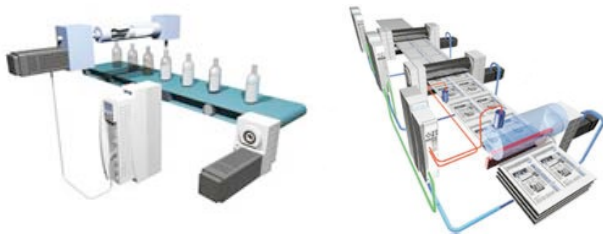

Figure 1: Sample applications: Installation (left) and printing machine (right)

The complexity of this special task is captured by the fact that there are thousands of different engines, gears, engine controller and additional components and their variants manufactured by *Lenze* (and even more off-company products). If all the combinations of all components were to be considered, this would lead to a solution space of 10^{21} of possibly consistent products. Additionally, the complexity is increased by the fact that central components like engines and gears may have hundreds of parameters and restrictions constraining possible solutions.

When parameterizing and specializing components, there is a highly complex constraint-net between the parameters within as well as the parameters of other components. These constraints reflect the complex physical correlations (both mechanical and kinetic) and result in a difficult consistency problem.

A special problem arises from the assistance aspect. In order to support sales engineers intuitively, the control mechanism of the system has to be able to represent the sequence of specifying the values for the parameters. This has a great impact on the quality of the configured solution. Additionally, the values have to be specified according to the individual requirements of each customer in order to get high quality solutions.

Because of this general set-up, it is very time-consuming to achieve a high-quality and consistent solution in direct cooperation with the customer.

In practice, the basically hand-made configuration of a drive system did not ensure that the resulting solution was complete and consistent. This resulted in a number of problems, when an error occurred, since the faulty solution was the foundation of the offer. The main problems can be summarized as follows:

1. The consistency of the generated solution, composed of a number of engines, gears, drive controllers and support components, has to be checked by engineers at the main manufacturing site of *Lenze*. This means that all solutions developed at the customer's location are subject to inconsistency and therefore possibly cannot be built.

2. The validation of a solution (i.e. a detailed sketch) is a very time-consuming task. In some cases, highly qualified engineers need days to perform this work.

3. New, innovative products are often not taken into consideration by sales engineers, since it is almost impossible for them to keep up with all new developments.

4. This also may apply to more cost-efficient solutions by choosing standardized components which can be a serious drawback on the profitableness and might result in longer delivery times.

5. The sales engineers are in most cases highly qualified specialists in a certain field, who excel in finding optimal solutions for their field. There is no easy way to make this knowledge available to other non-experts in the specific domain.

System Goals

The goal of the project was the development of a software tool to support knowledge-based configuration in order to assist sales engineers in preparing an offer. The main system goals can be described as follows:

- *Domain-independent configuration engine:* Although DSD was designed to solve a specific problem, the configuration engine itself should be domain-independent, so that it can be used in different projects. This ensures the future availability of software updates and support.

- *Assisting the configuration task:* The system is supposed to assist the user in the configuration task without taking over the whole work. It should propose solutions for the given problem, but leave the key decision to the sales engineers.

- *Increasing the speed to place an offer:* Placing an offer for a drive solution could take anything from 3 to 30 hours. Assisted by the system, this time should be reduced significantly.

- *Handling the amount of variants:* The system should make it possible to handle the high amount of possible combinations. It should be possible to browse through the space of currently valid components at any given time.

- *Ensuring the consistency of the offer/solution:* As a key requirement, the final configuration should be consistent according to the physical knowledge and the product catalog.

- *Controlling the configuration process:* The system should be able to guide the user through the configuration process in an intuitive way based on empirical knowledge of other experts.

- *Building and maintaining the knowledge base:* As the domain of drive systems is a rapidly changing domain, it should be easy to build and maintain the knowledge base.

Additionally, a number of other requirements like a modern software architecture for web-integration, consideration

of alternative solutions and building a case-base for successful solutions have also been made.

Application Description

The software tool "Drive Solution Designer" (DSD) has been developed by the TZI (Center for Computing Technologies) of the University of Bremen, the companies *encoway GmbH & Co KG,* and *Lenze* AG.

It provides extensive assistance for a sales engineer in the *Configure & Buy* scenario and its usage resulted in a considerable increase of the quality of the products as well as in a significant reduction of the development period. It is a software assistant used by *Lenze* sales engineers at the customer's location in order to validate technical offers for drive systems with respect to their possible construction with the aid of *structure-based configuration* (Günter/ Cunis 1992, Günter 1995a.). Based on the domain-independent configuration engine EngCon, the DSD supports ETO (Engineering to Order) configuration.

Consequently, the DSD does not only calculate physical correlations with the help of various formulas, but it also computes a correct solution which can be built and delivered, containing a sensible combination of the Lenze products.

Configuration with EngCon

The central component of the DSD is, as already mentioned, the domain-independent structure-based configuration engine EngCon. EngCon is a standardized configuration platform based on long-term AI research (Hollmann et. al. 2000). The configuration engine allows the stepwise assembly of a drive system by constantly controlling the consistency of the partial solution. In contrary to the common configuration approaches which perform something similar to breadth-first search resulting in all possible solutions, structure-based configuration performs a depth-first search, resulting in only one given solution (global consistency can not be ensured for partial solutions without performing a complete look-ahead search). At the first glance, this may seem a drawback, however, this is not the case: The complexity of the domain of offer validation for technical drives is extremely high. With constraint-based configuration engines, all possible solutions would be made available to the salesman who in turn would have to choose one concrete product out of this large set.

By adding domain specific knowledge about the configuration task as well as providing sophisticated modeling techniques, the structure based configuration provides a single solution. There are three types of knowledge used in the configuration task: The product catalog, the relationships between the components from that catalog and a control structure on how to perform the configuration task.

If an offer for a drive solution is going to be configured with the DSD, the user starts with choosing a certain type of product. By specifying parameters and relations for this initial concept, it is decomposed into its parts, which in turn are decomposed again, until a solution is found. To make sure that only a consistent solution is produced the configuration engine EngCon applies various AI techniques described in the AI technology section. All the time, the user is guided by a configuration control, which helps to produce high quality solutions by applying domain expert knowledge about the order of the decomposition task.
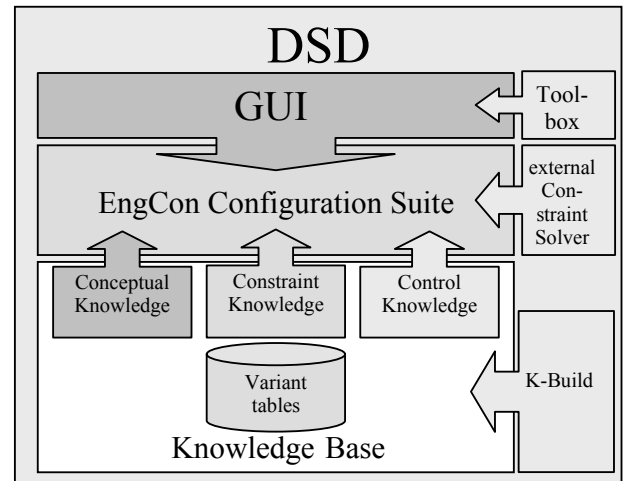

Figure 2: Architecture of the DSD

System Architecture

The DSD has been completely implemented in Java™ 1.3 technology, using the standard API as well as Java Swing. There are three major components in the DSD: The domain-independent configuration engine EngCon, the user interface, and the knowledge bases (see figure 2).

Within the central component EngCon, taxonomical inferences and constraint-propagation is performed in order to ensure a consistent solution resulting from the configuration process. EngCon utilizes interchangeable external constraint solver for propagating constraints over infinite domains and intervals. These may rank from C-libraries (CSP) to Prolog Systems (CLP) connected to EngCon via the Java Native Interface (JNI). Additionally to these constraints, EngCon uses variant tables which are stored in a database accessed through the ODBC/JDBC interface.

The second vital part of the DSD is the knowledge base, containing the product knowledge. The knowledge base contains three types of knowledge: The component knowledge, the constraint knowledge, and the control knowledge. For the maintenance of the knowledgebase, a domain-independent, Web-based service tool is available, the K-Build.

Guiding the sales engineer through the configuration process is the GUI, visualizing the technical details (see figure 3). On top of it, the so called sketch is visualizing the already chosen components, providing a rough overview about the configuration process as well as component details. The sales engineer is able to enter the data required

for building a drive solution in various sheets and graphs on the bottom part of the GUI.
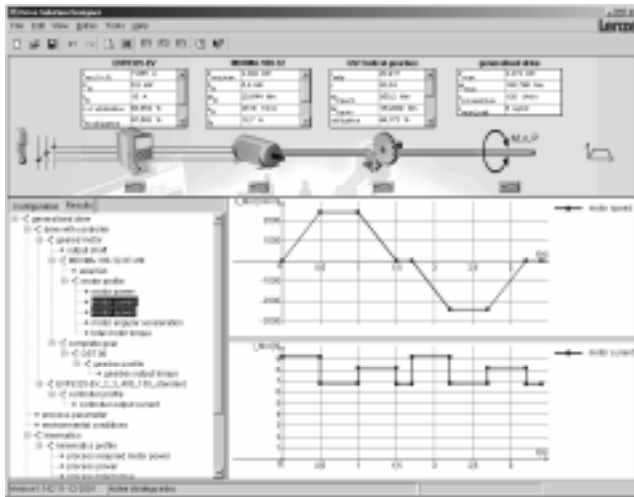

Figure 3: The DSD User Interface

Additionally to these components, a set of tools for calculating the correct parameters for components is available to the user in order to facilitate the process of configuration.

Uses of AI Technology

In other domains e.g. the telecommunications domain, classical configuration systems have been successfully deployed (e.g. (Focacci, et.al. 1997), (Chow and Perett, 1997), (Bach 2000)). Unfortunately, these constraint-based configurators proved inappropriate for the task described here. The main reasons are:

- **Knowledge acquisition** in constraint-based configuration is difficult for non constraint experts.
- **Search space size**: In the DSD, search space is far too complex to compute each and every solution from a given number of starting parameters. Nearly every concept has a number of real intervals whose values will be restricted during a step-by-step configuration process.
- **Control**: The succession of the configuration steps has a deep impact on the quality of the solution. In a constraint system it is hardly if not almost impossible to model the knowledge about the order of the configuration. The knowledge about which order leads to a high-quality solution is an important resource of an experienced sales engineer. It is necessary to describe this knowledge and make it available for other, less experienced sales engineers.
- **Assisting the configuration task:** The customer neither knows, how many components the configured solution requires nor its exact parameter values. He only knows what the solution is supposed to do. Consequently, a

system assisting the salesman in the incremental process of parameterization is needed.

For these reasons, a strict constraint-based solution did not suffice and a different solution had to be chosen.

Structure-Based Configuration

An interesting alternative to the strict constraint approach are the methods for heuristic, structure-based configuration developed in the projects TEX-K and PROKON[1] (e.g. Günter1995b) (see also: Hollmann et.al. 2000). The key requirement for the use of this approach is the component structure of the domain. Especially in technical domains like the *Lenze-drive-system* domain the component structure is commonly found.

This approach is based on three different knowledge representations:

1. Knowledge about the objects of the domain are represented in an ontology according to their taxonomic- and partonomic relations and the relevant attributes.
2. Dependencies between objects and their relations and attributes are represented with constraints by a meta-constraint modeling language.
3. Knowledge about the control process i.e. order of configuration steps, calculation methods and the priority among them as well as the use of conflict resolution methods is declaratively described by strategies.

The configuration process is started by the user (sales engineer) choosing a task object (e.g. a printing machine) from a list of predefined objects. Based on this task-object EngCon generates a first simple partial solution by instantiating the concept. Afterwards the control component calculates all configuration steps that have to be performed in order to determine terminal values for all relations and attributes and adds them to a configuration agenda. Three kinds of configuration steps are supported: a decomposition-, a specialization- and parameterization step. A decomposition configuration step restricts the values of has-part relations, which may result in the instantiation of new objects. Given a has-part relation, *has-part: engine [0 2]*, which the user or the system is able to restrict to *[1 1]* then a new instance of the concept *engine* is created and appended to the current partial solution. Similar to the decomposition step the specialization step restricts the values of the ISA- relation and the parameterization step restricts the attribute values. New or restricted values may result in new configuration agenda entries.

A key feature of the structure-based approach is that not only the value of already instantiated constraints are restricted incrementally but also new constraints may be added to the already given constraint net. Therefore the control component checks after each configuration step if new constraints have to be added, due to new components (after decomposition), or new or more restricted attributes

---

[1] These projects have been supported by the *German Ministry for Education and Science*.

(after specialization or parameterization). In this approach, constraint propagation is not handled as a calculation function but rather as a consistency-ensuring process which is invoked automatically after each of the current partial solutions.

A final solution is reached once all attributes and relations have been restricted to terminal values.

The central components, ontology, configuration control, and constraint module will be described in more detail in the following sections.

The Ontology Component

For the modeling of the conceptual knowledge (i.e. the product catalog), the proprietary knowledge representation language EngConKR has been developed. Instead of using already existing languages (e.g. OIL and Ontolingua for details see: Fensel, Horrocks, Harmele and Decker, 2000, Farquhar, Fikes and Rice 1996), a separate approach has been chosen for two reasons: First, while the functional requirements for EngCon could be described in a clear way, the requirements for the expressional power of the representation language were elusive without having modeled part of the application domain (i.e. EngConKR has been extended with discontinuous intervals, like *[[0.0 20.15], [30.42 78.85]]*). Secondly, even though existing ontology-languages like Oil support e.g. real numbers, given reasoners like FACT are not capable to perform the adequate inferences.

The domain knowledge is modeled by ISA-specialization- and has-parts relations. The specialization relation has a stringent semantics, i.e. attributes from superior concepts are inherited and maybe restricted by more special attributes. Multiple inheritance is not allowed. In addition to the standard attribute types like sets, intervals and atomic values discontinuous intervals are allowed also.

Taxonomic inferences: The fundamental assumption for the inference mechanism of EngCon is the (strong) closed world assumption (CWA). A situation, where there is no way to specialize an instance of a partial solution to a leaf concept, will be interpreted as a conflicted solution. Even though this assumption is insufficient for many domains, it is well fitted for technical domains. Products in EngCon are modeled as a leaf concept. If an instance may not be specialized to a leaf concept, there is no product for the current task. Thus it seems plausible to presume CWA for this domain.

Based on the CWA, EngCon has two fundamental taxonomical inference modules: a dynamic and a static one. Optimally, the static inference module may be applied prior to the configuration in order to propagate terminal values from the leaf concepts alongside to the specialization hierarchy. With the help of this mechanism, a wide range of conflicts resulting from improper input may possibly be prevented.

The dynamic taxonomic inference module supports primarily four types of inferences:

1. automatic specialization to n-th level
2. automatic specialization to a leaf concept
3. automatic specialization along the has-part relation
4. automatic pre-decomposition

The first inference is equal to the common classification inference of description logic (DL). If the value of an attribute of an instance allows for a classification to a more special concept, an automatic specialization is triggered. The automatic specialization to a leaf concept is implicitly performed by the first inference. It takes a conflict avoidance role by ensuring that a specialization to a leaf concept is possible, if the static taxonomic inference module is not used. The fourth inference, the automatic decomposition, is better regarded as a heuristic rule instead of as a logic inference. It is only applicable in a true top-down configuration. If it is specified in a has-part relation of an instance of the concept *engine* that the engine has at least one cooler (that is i.e. *[1 inf]*), as a result of this inference an instance of a cooler will be automatically added to the solution.

The Control Component

For the simulation of an expert's procedure, EngCon employs an agenda-based control guided by declarative control knowledge (Günter A, 1992). For this purpose, a knowledge engineer defines strategies on how the configuration process of an application is divided into phases, and additionally defines the order of the tasks and their calculation method. By this, it is possible to model different kinds of procedures for different groups of users.

Schematically, the control flow is as follows:

1. *Definition of the task*
2. *Creation of an initial partial solution*
3. *Determining the current strategy*
4. *Creation of an agenda:* The configuration focus defined in the strategy determines which entries are going to be included into the current agenda. These entries specify which attribute of which component of the current solution will be modified next. Here, strategy-specific agenda selection criteria define the order of the entries in the agenda. The agenda contains all tasks to be processed in a strategy and may be accessed at any time by the user.
5. *Selection of agenda entry:* The next step is choosing the entry with the highest priority.
6. *Selection of calculation method*: With the help of the order of the calculation methods defined in the strategy, the method to be employed will be chosen. Valid calculation methods are: Defaults, dynamic defaults, calculation function, user input. If the highest priority calculation method may not be applied, i.e. a default is not applicable the next calculation method with a lower priority is used. If there is no automatic method left to apply, the user will be asked for input. The interfaces within the EngCon configuration engine are open for the

integration of new calculation methods, i.e. simulation without much effort.

7. *Execution of calculation method:* The result of the modification method is acquired.

8. *Calculation of consequences:* The current partial solution is retrofitted with the results of the calculation method and the consequences are calculated: *(a) Taxonomic inferences, (b) Constraints:* resulting from i.e. new components perhaps new constraints have to be added to the constraint net.

9. *Updating the agenda:* First, redundant entries resulting from constraints or taxonomic inferences are removed from the agenda and it is checked whether new entries have to be added to the agenda. Finally, it is evaluated whether a change of strategy has to be performed, and the configuration cycle starts again with 5.

The Constraint Component

Constraints are used for the description of the complex dependencies between the parameters of the objects of the ontology/the product catalog. Three different concepts have been realized:

1. Complex functional and predicate constraints are used to represent equations and functions. They are propagated in an interchangeable external constraint solver[1].

2. Extensional constraints (tuple of values) explicitly list the set of possible variants and are managed within an external database.

3. Additionally, Java-Constraints provide the flexibility for the user to integrate functions and calculations into the configuration, which are impossible to realize with functional or extensional constraints (i.e. summing up attribute values of several concepts).

The constraints are represented in a declarative manner in a meta modeling language and may be modified and extended without having to recompile parts of the whole system[2].

Future Work

Although we applied a wide range of AI-techniques, some extensions seem to be reasonable. The practical use of the system by the sales engineers showed, that once the user runs into a conflict, an detailed explanation of the underlying reasons is required. There are two possible sources for conflicts: taxonomic inferences and the constraint satisfaction. While taxonomic conflicts may be handled with a kind of dependency network, constraint conflicts are much more tricky because of the dependency between several constraints in the net. How complex constraint conflicts can be presented to the user, however, remains an open question. A more abstract question is how explanations can be generated (not only in the case of conflicts) which sup-

port users with the appropriate background knowledge in order to understand the underlying reasoning processes. Plan recognition methods are in investigation in order to provide the configuration engine with useful information about the intention and the background knowledge of the user to guide the explanation generation (see (Carberry 2001) for an overview).

Although user specific defaults may be defined, it seems to be reasonable that a user model should be learned automatically. Also, even though it is already possible to re-use a partial solution for various applications (variants), it is not possible to reconfigure complete solutions in the sense of CBR.

Application Use and Payoff

The DSD was introduced in May 2001, when it was shipped to the sales engineers of the *Lenze AG*. Since then, it has been used by about 150 sales engineers, who have been trained to use the system for technical offer preparation.

Using structure-based configuration for this construction and validation process has led to a number of benefits, which result in a high competitive advantage.

Time Reduction for Offer Placement

The time for placing an offer has drastically been reduced. Before the introduction of the DSD, the sales engineers had to get to the customer, take down the requirements for the solution and make a first sketch of the drive. This process took anything from 3 hours to 30 hours working time with an average of 4 hours.

With the help of the DSD, the time it takes for a sales engineer to come to a consistent, high quality solution has been significantly reduced by a factor of 5-10. Now, in easy cases, a consistent solution without the need of asking other experts can be found in approx. 15 minutes if a lot of standard components and values (defaults) can be used. The more complex tasks now take up to 150 minutes, if a lot of things need to be adapted. In average, 30 minutes are now required.

Reduction of Errors

Prior to the introduction of the DSD, too many of the validated offers had to be revised in the course of the actual preparation. These errors caused high costs to both *Lenze* and their clients, since the sales engineer had to check back with the customers again and the delivery of the drive was delayed.

By configuring the solution with the DSD, these errors were reduced to almost 0% of the offered drive systems since the knowledge of various domain experts now assist the sales engineers in executing the offer.

Increase of Quality and Use of new Components

Due to the fact that configuring a drive solution with the DSD uses the experience of many domain experts, the overall quality of the solution was significantly improved

---

[1] The currently used constraint solver is subject to privacy.
[2] This requirement led to the disqualification of i.e. the ILOG-solver.

by providing less expensive solutions which have a better performance. Even though the domain of drive systems is a very complex one, with thousands of different components, some of these are still standardized. Also, newly developed components were rarely included into a solution, since the sales engineers could not always keep up with the new product inventions.

With the introduction of the DSD, the configuration engine now proposes these standardized components, if they are fitting into a solution and thus improving its quality and delivery time. In the same instance, the DSD takes care, that the additional components combined with the standard one are fitting to it as well as newly developed components are taken into consideration for the solution.

Reduction of Costs

Prior to the introduction of the DSD, many components chosen were individual solutions. This led to higher production and maintenance costs for the individual solutions.

With the help of the DSD, more standard components are considered for drive solutions. This helps reducing the production costs – a standard component is easier to produce in less time – as well as reducing the maintenance costs by allowing the usage of standard spare parts. Thus, a considerable competitive advantage is gained, since lower costs result in more satisfied customers.

Application Development and Deployment

The development of the EngCon configuration engine was initiated in 1998. Starting with experiences from the already mentioned research projects (TEX-K, PROKON), a project team was initially formed by *Lenze* and the TZI of the University of Bremen. At the TZI of the University of Bremen, the development staff consisted of a project manager, two research staff, a software engineer and several students.

The TZI was responsible for the development of the domain independent configuration engine EngCon as well as providing support for the knowledge engineering process. The project team at *Lenze* was responsible both for the development of the knowledge base and the DSD application. This team consisted of a project manager, a software engineer and two domain experts.

Having a modern and web-enabled application in mind, JAVA was chosen as the implementation language. During the first 18 month the core functionality of the configuration engine was designed and implemented at the TZI. With a delay of three months, In parallel, the team of domain experts at *Lenze* defined the expressional requirements and developed a prototypical knowledge base for the application DSD. In course of this, the knowledge representation had to be adapted several times.

At the beginning the EngCon configuration engine was designed with the *unified modeling language* (UML) using the Rational Rose. Since several problems arose with the round-trip engineering using Rational Rose to update and develop the software model, the usage of the tool for this purpose was given up. Instead it was only used for the review processes. In order to implement the UML-model a beta version of the Java development environment *Visual J* was chosen. But due to several update delays and compatibility problems, a change to the *Inprise JBuilder* was made.

A first milestone was reached in 1999 - a first implementation of the EngCon engine and a prototypical version of the application DSD. As a result of market surveys, the stakeholders decided to set up a separate company to continue the development of the configuration engine EngCon towards a product. As a result *encoway GmbH & Co KG* was founded early in 2000 as a spin-off of the TZI of the Bremen University. After two years, *encoway* now has a staff of 30.

Since then, the development and implementation tasks for the software (EngCon, DSD) where shifted from the TZI to *encoway* to ensure product quality (maintenance, services, documentation) for the systems. During this period the project team at the TZI enhanced the underlying methodologies e.g. for conflict handling and explanation generation and prepared them for an integration into EngCon.

After three months of field tests and several adjustments, the DSD was finally shipped in May, 2001. Since then, it has been in use by about 150 sales engineers for the technical offer preparation.

The overall investments for the project EngCon and the application DSD are summed up to approx. 30 man years.

Maintenance

The domain of technical drive systems is changing very rapidly. New developments in engineering result in specialized or completely new components, which either add to the already existing ones or completely replace them. In order to make sure, that these new drives, gearings and other related components are made available to the customers, the knowledge base has to be adapted to these changes. Therefore, maintenance of the knowledgebase is a key requirement for the success of the application.

The KBuild software tool for the configuration engine EngCon, which domain-independently allows to maintaining the knowledgebase has been developed for this purpose. KBuild is not only a tool for maintaining the knowledgebase, but is used as well for the initial knowledge acquisition and building of the knowledgebase itself.

In KBuild, validation of the newly inserted concepts is performed with the help of the same taxonomical inferences used within the EngCon configuration engine as described in chapter 'Uses of AI Technology'. Constraint knowledge is modeled with the same tool. With the help of the KBuild knowledge acquisition tool, the maintenance of the knowledge base can be performed by any domain expert.

Conclusion

As has been shown, for more simple scenarios standard methods from computer science sufficed to fulfill the task

of customizing products. The configuration of drive systems, however, is a highly complex engineering task which requires profound knowledge about the whole range of products as well as an in-depth understanding of underlying physical dependencies. Therefore, the problem domain of drive construction required sophisticated AI-methods in order to become feasible. The Drive Solution Designer (DSD) which has been introduced in this paper, uses a structure-based configuration engine, EngCon to provide extensive assistance to sales engineers of the *Lenze* company for their daily work.

With the help of the DSD, their actual working time for an offer could be drastically reduced from an average of 4 hours to an average of 30 minutes.. This went along with a significant reduction of faulty offers, since the consistency of an offered solution is now ensured. By maintaining the knowledge base to keep up with the latest developments in the drive engineering domain, these innovations are now easier brought to the attention of the sales engineer who in employs them to turn build drive solutions. Regarding the benefits the DSD had for the daily work of a sales engineer as well as the fulfillment of the other requirements for the development of the software, it has to be considered as a very successful application.

The major component of the DSD, the structure based configuration engine EngCon, has been developed as a domain independent tool. This allows its wide-spread use by exchanging the knowledge bases and the GUIs with domain specific ones while keeping the configuration engine unchanged. Currently, there are applications in technical domains (industrial robots, pumps and sensors) as well as service domains (financial planning and insurances) under development. For these applications, prototypes have already build .

Acknowledgements

References

Arlt,; Günter; Hollmann,; Hotz,; Wagner. 1999. Engineering&Configuration - A Knowledge-based Software Tool for Complex Configuration Tasks. Workshop on Configuration at AAAI-99, http://wwwold.ifi.uni-klu.ac.at/~alf/aaai99/

Artale; Franconi,; Guarino; Pazzi. 1996. Part-Whole Relations in Object-Centered Systems. An Overview. In Data and Knowledge Engineering, North Holland, Elsevier 20:337-384.

Bach. 2000. IPAS: An Integrated Application Toolsuite for the Configuration of Diesel Engines. ECAII'00 Workshop Configuration, http://www.cs.hut.fi/~pdmg/ECAI2000WS/Proceedings.pdf

Carberry. 2001. Techniques for Plan Recognition. User Modeling and User-Adapted Interaction, Volume 11, Number 1-2, pp. 31-48, 2001.

Chow; Perett. 1997. Airport Counter Allocation using Constraint Logic Programming, in: Proc. Of Practical Application of Constraint Technology (PACT98), London, UK

Cunis; Günter; Strecker. 1991. The PLAKON Book - An Expert System for Planning and Configuration Tasks in Technical Domains. Springer.

Cunis; Günter; Syska; Peters; Bode. 1999. PLAKON - An Approach to Domain-independent Construction, 866-874. Tennesse USA: ACM-Press.

Farquhar; Fikes; Rice. 1996. The Ontolingua Server: A Tool for Collaborative Ontology Construction. Technical Report, Stanford KSL 96-26

Fensel; Horrocks; Harmelen von; Decker. 2000. OIL in a Nutshell. In: Knowledge Aquisation, Modelling and Management, 1-16, citeseer.nj.nec.com/fensel00oil.html

Focacci; Lamma; Mello; Milano. 1997. Constraint Logic Programming for the Crew Rostering Problem, in Proc. Of Practical Application of Constraint Technology (PACT97), London, UK

Günter. 1995b. KONWERK - a modular configuration tool, 1-18. Richter (ed.). Kaiserslautern: INFIX Press.

Günter, A.; Cunis, R. 1992. Flexible Control in Expert Systems for Construction Tasks. International Journal Applied Intelligence, Kluwer Academic Press 2.

Günter. 1992. Flexible Control in Expert Systems for Planning and Configuration in Technical Domains, PhD-Thesis. In: PHD-Thesis in Artificial Intelligence, Volume 3, INFIX press

Hollmann; Wagner; Günter. 2000. EngCon - A Flexible Domain-Independent Configuration Engine. Workshop Configuration at ECAI-2000, http://www.cs.hut.fi/~pdmg/ECAI2000WS/Proceedings.pdf

Hollmann; Günter; Ranze; Wagner. 2001. Knowledge-based Configuration - About Complex Highly-Variant Products in Internet-based Scenarios. KI 01/01 (German AI Journal), Arendtap.

McGuiness; Wright. 1998. Conceptual Modelling for Configuration: A Description Logic-based Approach. AIEDAM 12(4): 333-344.