AI Magazine Volume 14 Number 3 (1993) (© AAAI)

A Knowledge-Based Configurator That Supports Sales, Engineering, and Manufacturing at AT&T Network Systems

Jon R. Wright, Elia S. Weixelbaum, Gregg T. Vesonder, Karen E. Brown, Stephen R. Palmer, Jay I. Berman, and Harry H. Moore

■ PROSE is a knowledge-based configurator platform for telecommunications products. Its outstanding feature is a product knowledge base written in C-CLASSIC, a frame-based knowledge representation system in the KL-ONE family of languages. It is one of the first successful products using a KL-ONE–style language. Unlike previous configurator applications, the PROSE knowledge base is in a purely declarative form that provides developers with the ability to add knowledge quickly and consistently. The PROSE architecture is general and is not tied to any specific telecommunications product. As such, it is being reused to develop configurators for several different products. Finally, PROSE not only generates configurations from just a few high-level parameters, but it can also verify configurations produced manually by customers, engineers, or salespeople. The same product knowledge, encoded in C-CLASSIC, supports both the generation and the verification of product configurations.

PROSE (product offerings expertise) is a knowledge-based engineering and ordering platform that supports sales and order processing at AT&T Network Systems (AT&T-NS). The cornerstone of the PROSE architecture is a product knowledge base written in C-CLASSIC, a knowledge representation system in the KL-ONE language family that was developed at AT&T Bell Laboratories (Borgida et al. 1989). Currently, PROSE is being used to provide configurations for sales proposals and to generate factory orders for manufacturing. Some examples of products that are currently being configured by PROSE are the cross-connect systems DACS IV-2000 and DACS II CEF as well as the remote cell sites for the AT&T Autoplex mobile telephone system. We expect PROSE to be deployed for highly optioned products across all AT&T-NS business units.

The PROSE platform is closely integrated with the corporate infrastructure for ordering products, and it has communication links to the mainframe systems that support order processing and manufacturing. PROSE can produce a detailed materials list and pricing for sales proposals, it can electronically place orders and initiate billing, it can send manufacturing specifications to the factory, and it can produce instructions for on-site installers. Most importantly, the PROSE architecture is general and is not tied to any specific product.

The motivation underlying the PROSE project was to solve what we initially called the data-synchronization problem. In a large company offering complex products, ordering information is typically distributed among a variety of sources, both formal and informal. The distributed, informal nature of this critical information makes it difficult to maintain in an up-to-date, valid, and consistent way.

The official repositories of product information at AT&T-NS are the engineering drawings. As technical documents, they cannot be read and understood by everyone. Consequently, the ordering information in the engineering drawings is reworked into paper ordering guides, informal spreadsheet programs used by account executives, and various personal computer (PC)–based configurator programs. The product information contained in these sources frequently becomes obsolete and out of synch with the engineering drawings.

Inaccurate orders, when combined with products that are so highly technical in nature, cause delays in order processing and manufacturing and can result in billing discrepancies. PROSE seeks to centralize this information, or product knowledge, in a single source, and put it in a form that can be made available to anyone who needs it. Having every team member working off the same page, so to speak, greatly reduces rework in the ordering process, improves quality, and reduces cost.

Our experience is that within the context of the PROSE application, consistency checking has somewhat the feel of programming in a strongly typed programming language, where inconsistent and incorrect uses of data types are caught by the compiler.

The earliest and best-known configurator application that used techniques pioneered in the AI community was developed at Digital Equipment Corporation in conjunction with Carnegie Mellon's John McDermott (McDermott 1982; McDermott and Bachant 1984; Barker and O'Connor 1989). The research version was called R1 and later become known as XCON in its production version. R1 used production rules to represent knowledge about configuring Digital's computer systems.

Although production rules had advantages over the conventional development approaches that had been tried at Digital prior to R1, some drawbacks surfaced after the deployment of R1 in 1981. The most serious was the effort needed to maintain an up to date, consistent, and valid collection of production rules. Digital estimated that 40 to 50 percent of the R1 product knowledge changes each year (Bachant 1988). By some estimates, there have been as many as 6000 R1 production rules. The rate of change, coupled with the sheer number of rules needed to adequately represent R1's product knowledge, made R1 software maintenance an expensive process. Subsequently, special techniques had to be developed to make software maintenance easier and more cost effective (Bachant 1988).

For a configurator application, product-ordering conventions serve as software requirements. Responding quickly to changes in requirements is especially important in this application domain because the inability to order new product features through a configurator dramatically affects utility. Development schedules for a configurator tend to be driven by the pace of change in the product, not by the developer's sense of what can be delivered when.

In addition, it seems to us that the term knowledge-acquisition bottleneck is especially meaningful for configurator applications. The R1 project, partly in response to the fact that the Digital product knowledge was too complex for one person to maintain, developed schemes for factoring rules into modules so that the maintainers could specialize within the product domain (Bachant 1988). Having access to people who understand the product is a key element in the success of a configurator project.

Thus, a configurator application such as PROSE has three critical problems to address: (1) the acquisition of product knowledge, (2) rapid and sometimes unexpected changes in product knowledge, and (3) the complexity of software enhancements and maintenance.

In part, PROSE responds to these problems by taking advantage of knowledge representation techniques originally introduced by KL-ONE (Brachman and Schmolze 1985). Although there has been active research on KL-ONE–style languages since 1975, and research prototypes have demonstrated feasibility in several cases, heretofore few successful production software applications have used a KL-ONE–style representation (O'Brien, Brice, and Hatfield 1989). However, we think other successes are likely to follow.

The use of C–CLASSIC, whose ancestry can be traced directly to KL-ONE, provides the PROSE platform with several key advantages. With some exceptions to be discussed later, product knowledge in PROSE is isolated to a single module—the product knowledge base. C-CLASSIC encourages a reasonable organization for the product knowledge and enforces internal consistency. Inconsistencies in the knowledge base are often flagged in the compilation stage and, at other times, are caught during testing. Both kinds of inconsistencies are identified by C-CLASSIC's internal integrity-checking mechanisms.

Our experience is that within the context of the PROSE application, consistency checking has somewhat the feel of programming in a strongly typed programming language, where inconsistent and incorrect uses of data types are caught by the compiler. C-CLASSIC's consistency checking has had a beneficial effect on both the maintainability of the PROSE product knowledge and the quality of the configurator's output.

Like that of its predecessors, the simplicity of C-CLASSIC's description language and the tractability of its inference algorithms are linked. C-CLASSIC provides only a few primitive operators with which knowledge can be described. These operators were chosen at least in part to avoid intractability in the underlying subsumption algorithm (Levesque and Brachman 1987). In particular, the description language lacks true disjunction and has no way to express negation. Nevertheless, we have not encountered major problems when we encoded the product knowledge for our AT&T Network Systems products.

To the contrary, we feel that C-CLASSIC has encouraged the encoding of product knowledge in a natural way. Subject-matter experts with a variety of engineering and business backgrounds, when provided with a small amount of assistance from someone who understands C-CLASSIC, have been able to easily relate to and understand the product knowledge encoded in C-CLASSIC.

In this context, standard software-engineering techniques such as code inspections take on a special meaning. Essentially, these sessions perform double duty as verification exercises. Typically, a product expert participates and often clarifies misunderstandings in the ordering knowledge for a product. In most cases, the C-CLASSIC expressions are close to the expert's intuitive understanding of the product, providing an uncommonly strong basis for communication between the developer and the product expert.

C-CLASSIC's contribution to the PROSE project is unmistakable. Maintenance and customization of a product configurator for specific user communities can be accomplished in a clean and straightforward way. Reuse of the descriptive product knowledge is one of PROSE's most interesting features, and it has some genuine benefits. In particular, the sticky problems associated with updating, synchronizing, and distributing product knowledge to the appropriate people are much easier to control in the PROSE environment.

The PROSE Application

The PROSE platform is geared toward configuring telephone switching and transmission equipment. By their nature, these products are complex and have many optional features. Although there is a trend toward scaling down the number of available options for individual products, customers like the ability to customize products to their specific needs. To provide a concrete example of the capabilities of the platform, we briefly describe the DACS IV-2000 cross-connect, which was the first product to be made available within the PROSE platform.

DACS IV-2000 is a digital cross-connect system that processes digitized signals at a DS1 or DS3 rate.[1] A complete lineup consists of nine 7-foot frames (called *bays* when they are equipped and working) connected by cabling. The positions in the lineup are significant and are numbered from left to right. Each bay contains as many as four shelves or modules of electronic gear. A 6-bay DACS IV-2000 configuration is shown in figure 1.

There are 13 types of DACS IV-2000 bay, 3 of which appear in figure 1. The modules within a bay can be equipped with different kinds of circuit packs depending on what capabilities are desired. In addition, compatible cabling and software must be ordered. Although we have not tried to produce an exact calculation, the number of possible configurations is large, perhaps exceeding 100,000 or more. The cost of a complete nine-bay lineup, including spare circuit packs, can easily extend into seven figures.

The time needed to process orders prior to manufacturing is called the *up-front order interval*. The rework and delay associated with the processing of invalid configurations during the order interval is a significant contributor to the cost of providing a new DACS IV-2000. Significant benefits are associated with reducing the length of the order interval, not the least of which is increased customer satisfaction. For DACS IV-2000 prior to PROSE, the time period for getting manually produced equipment specifications to the factory was generally 7 to 14 days. PROSE is capable of delivering valid orders to the factory at the push of a button.

Central to the PROSE application, no matter what aspect is being discussed, is the *materials list,* which is a description of the materials needed to assemble and install a configuration. It is used for producing a bill of materials for the shop floor, billing and shipping to customers, generating instructions to installers, and communicating with customers about the product. In essence, the materials list serves as a manufacturing specification, telling the factory what to assemble.

For a nine-bay lineup, there would be separate materials lists for each of the bays plus separate lists for DACS IV-2000 software and cabling. A completed order also includes installation instructions (where to locate and how to wire each bay). PROSE generates all the information that is associated with manufacturing and installing the equipment it configures.

PROSE has three interfaces that support different aspects of sales, engineering, and man-

… the sticky problems associated with updating, synchronizing, and distributing product knowledge to the appropriate people are much easier to control in the PROSE environment.

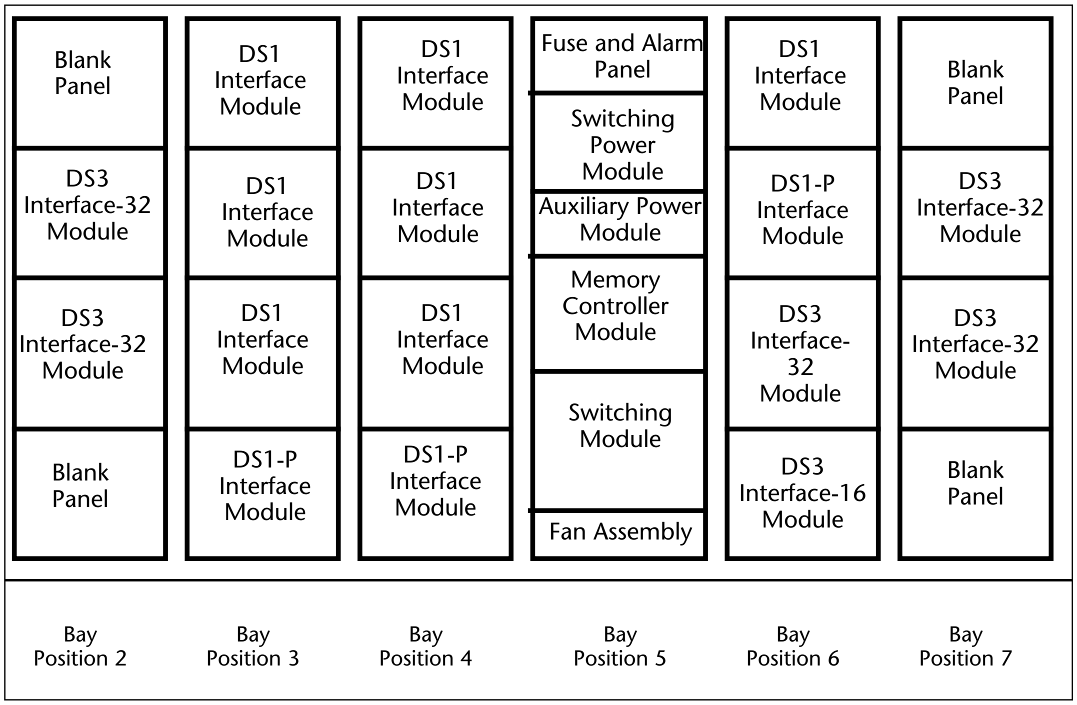

Figure 1. A 6-Bay dacs iv-2000 Configuration.

ufacturing. Distinct user communities are served by the three interfaces, but all three draw on the same product knowledge base. Having a single-product knowledge base allows PROSE to avoid problems associated with synchronizing knowledge and data or resolving conflicts in several software applications.

The FPQ (firm price quote), or pricing, interface: Because accurate price quotes are not possible without knowing all the equipment needed by an application, sales teams have technical consultants with the responsibility of producing price quotes with itemized lists of equipment and prices. From a few high-level parameters, FPQ can produce a price quote for a complete nine-bay DACS IV-2000 lineup, including compatible software releases and cabling, in a few minutes. FPQ output is such that it could be turned into a valid order and sent directly to the factory.[2] Frequently, technical consultants use FPQ to explore what-if scenarios to help the customer find the right configuration.

The SPEC (specification), or engineering, interface: SPEC is intended for AT&T engineers who might be working either on internal AT&T applications or as consultants to outside customers. SPEC requires more input from the user than the FPQ interface, but it is also more flexible. Engineers have the choice of keying in capacity parameters and feature choices, or they can specify the quantity and the type of circuit pack for each bay. Like FPQ, SPEC output is in the form of an order that can be sent directly to the factory's ordering and billing systems.

The TCE (telephone company engineered), or customer-service, interface: Customers sometimes configure products on their own without going through either FPQ or SPEC. In such cases, the customer submits what is essentially a proposed materials list. Because invalid configurations cannot be assembled, it is essential to know if the list represents a valid configuration. The TCE interface allows a customer service clerk to key in the materials one item at a time. PROSE validates the configuration and formats it so that it can be entered in the appropriate order-processing systems.

In addition to serving a diverse community of users, PROSE must deal with products that constantly change in response to the marketplace. Although we have observed variations, the rate of change for certain products approaches that reported by R1-XCON (40 to 50 percent a year).

For knowledge engineers, however, the real problem is that the scheduling and timing of changes is not within their control. The inability to produce valid orders for new products and enhancements to existing products is problematic for any manufacturing entity. To really be useful, configurators must change in lockstep with new product offerings. Although the solutions to these problems are partly methodological (for example, early notification of changes from the design community), the use of C-CLASSIC has played an important role in our ability to respond rapidly with quality results.

C-CLASSIC

C-CLASSIC (Weixelbaum 1991) is a frame-based knowledge representation system derived from the KL-ONE family of languages (Brachman and Schmolze 1985; Brachman, Fikes, and Levesque 1983; Patel-Schneider 1984; Woods and Schmolze 1993). It is a direct descendant of CLASSIC (Borgida et al. 1989), which was written in Common Lisp and had the benefit of years of research on semantic nets and frame systems.[3] Because of the declarative nature of the information encoded in a C-CLASSIC knowledge base, it and other similar languages are sometimes referred to as *description logics.*

C-CLASSIC inherits its two most salient features from CLASSIC: a simple description language and tractable inference algorithms (Borgida et al. 1989). C-CLASSIC is an interpreted language written in C and portable to any UNIX system. C-CLASSIC provides three basic types of objects: (1) *concepts* (or frames), which are assertions or descriptions about the state of the world; (2) *individuals,* which are particular instantiations of concepts; and (3) *roles,* which provide a way to relate individuals.

C-CLASSIC provides a simple rule-firing mechanism. A rule consists of a left-hand side and a right-hand side. The left-hand side is a concept, and the right-hand side can either be a concept or a function that returns a concept when called on an individual. Whenever an individual is classified under a concept, all rules that have this concept as the left-hand side fire on the individual, adding the right-hand side concept or the result of the function call onto the individual's descriptor.

Concepts are built up through composition of components that primarily include previously defined concepts and various types of role restrictions. In addition, a controlled escape mechanism to the C language is provided through test functions and computed rules. Test functions are used to test if an individual satisfies criteria that are otherwise inexpressible in C-CLASSIC. Computed rules are

```
<concept> ::=               <concept-name> |
                            (at-least <integer> <role>) |
                            (at-most <integer> <role>) |
                            (between <integer> <integer> <role>) |
                            (exactly <integer> <role>) |
                            (all <role> <concept>) |
                            (fills <role> [<individual> ...]) |
                            (one-of [ <individual> ...]) |
                            (range <number> <number>) |
                            (lower-limit <number>) |
                            (upper-limit <number>) |
                            (test-c <function> [ <c-classic-object> ...]) |
                            (test-h <function> [ <c-classic-object> ...]) |
                            (and [ <concept> ...])

<rule-concept> ::=          <concept> |
                            (computed-concept <function> [<c-classic object> . . .]) |
                            (computed-fillers <function> <role> [<c-classic object> ...])

<individual> ::=            <host-individual> | <classic-individual>

<host-individual> ::=       <integer> | <float> | <string>

<classic-individual> ::=    <symbol>

<concept-name> ::=          <symbol>

<role> ::=                  <symbol>
```

Figure 2. C-CLASSIC *Description-Language Syntax.*

used to compute the right-hand side of rules that are otherwise inexpressible in C-CLASSIC. Figure 2 shows C-CLASSIC's description-language syntax, and figure 3 shows how to define C-CLASSIC objects.

C-CLASSIC provides the following types of inference: (1) automatic classification of new concepts and individuals into an existing knowledge base; (2) completion or propagation of logical consequences, including but not limited to inheritance; (3) contradiction detection; (4) simple forward-chaining rules (or triggers); and (5) dependency maintenance (for retraction and error recovery).

All these inference mechanisms are used in PROSE. Classification and inheritance are used to organize the knowledge base into understandable pieces. In addition, an important side-effect of C-CLASSIC's ability to classify and propagate logical consequences is that internal consistency is maintained within the knowledge base. Sometimes a user can request a combination of features that does not represent a legal configuration. Contradiction detection is used to detect such errors. Next, as we discuss in the subsequent section, rules are needed to represent the product knowledge adequately. Finally, users might sometimes change their minds in the middle of a PROSE session. Dependency maintenance gives them the opportunity to retract an action

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.