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.

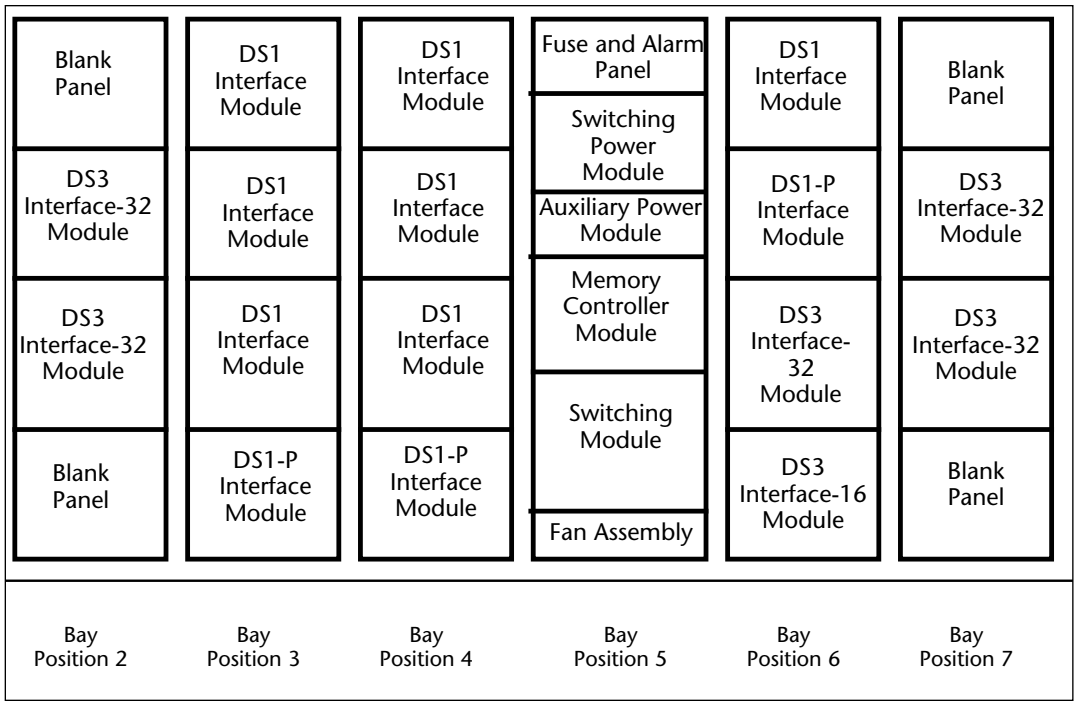

Figure 1. A 6-Bay dacs iv-2000 Configuration.

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 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.

<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.

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

(**define-concept** <concept-name> <concept>)
(**define-primitive** <concept-name> <concept>)
(**define-disjoint-primitive** <concept-name> <concept> <partition-index>)

define-concept defines an equivalence between <concept-name> and <concept>. **define-primitive** and **define-disjoint-primitive** define <concept> as a necessary (but not sufficient) condition for <concept-name>. <partition-index> is used for forcing disjointness among concepts defined with the same defining concept.

(**define-individual** <classic-individual> [<concept>])

Host individuals (numbers and strings) are implicitly defined by the system. **define-individual** defines a CLASSIC individual as an instance of <concept>, which defaults to **classic-thing**, the ancestor of all CLASSIC concepts.

(**define-role** <role>)
(**define-attribute** <role>)

define-attribute implicitly forces all individuals that reference <role> to have exactly one filler.

(**define-rule** <rule> <named-concept> <rule-concept)

Figure 3. C-CLASSIC Object-Defining Functions.

without losing previously asserted facts or inferences.

At any given time, the collection of objects that has been described to C-CLASSIC is connected in a network called a *classification graph*. When an object is defined or modified, C-CLASSIC searches the classification graph to find a location for inserting or relocating the object. Changes in the graph might cause objects already in the graph to be reclassified, or they might cause one or more rules to be fired.

During classification of a new object, a contradiction between this object and the existing classification graph might be discovered, or a propagation might result in a contradiction. To help the user recover from such error conditions, C-CLASSIC has standardized error-handling and error-reporting capabilities. Error recovery is facilitated by storing dependency information in the classification graph. By including information that indicates exactly how an individual attained each portion of its structure, error recovery can be done in a timely fashion. Using the dependency information, C-CLASSIC also permits users to retract previously asserted facts.

In addition to the C-CLASSIC interpreter, the system includes a library of C-CLASSIC functions that can be called from C. With this library, a customized interface was developed for PROSE. The system provides error data through the C interface, enabling PROSE to have its own customized error-handling procedures.

PROSE Knowledge Base Organization

Engineering documents that describe each product, including comprehensive information on the product's acceptable configurations, are the official source of product information at AT&T. For example, the DACS IV-2000 knowledge base represents a section of the engineering drawings called Table A, which describes all the pieces of equipment that can be ordered for a product, the time at which each item can be ordered, and the procedure for determining the desired quantity of each item. Product experts often write concise summaries of the information in Table A using their own notation. The summaries contain descriptions of simple constraints, called *compatibility rules* by our experts, and we have adopted this terminology.

Compatibility rules are generally derived from the physical structure of the product. However, other factors are sometimes involved, and in general, it is not possible to derive all compatibility rules by simply knowing the structure of the product. For example, some compatibility rules represent artificial constraints imposed by marketing, others represent an attempt to make the product easier to order, and still others represent constraints required for cost-effective manufacturing.

Figure 4 shows several compatibility rules and their C-CLASSIC representations. The two rules are associated with a type of DACS IV shelf called a DS1 IF shelf. The description that corresponds to a DS1 IF shelf is named DS1_IF in the C-CLASSIC knowledge base.

For the purposes of ordering a product, a DACS IV-2000 shelf has 6 attributes: signal capacities for ds1 and ds3 signals (ds1_lines and ds3_lines); the quantities of ds1 and ds3 circuit packs needed to satisfy a given signal capacity (ds1_packs and ds3_packs); the quantity of so-called common circuit packs (a general term used for a bundle of power and interfacing circuit packs); and pmgr-type (performance-monitoring generator and receiver) circuit packs, which are used to monitor the ds1 and ds3 signals. The value restrictions (for example, "(range 0 224)") on DS1_IF in figure 4 represent the legal ranges of these attributes for a DSF IF shelf.

Because the pmgr packs and the ds1 packs are inserted in the same slots on a DSF IF (seven slots available in all), the rule DS1_IF_max_ds1_packs limits the legal range of ds1_packs with the simple formula ds1_packs <= 7 - pmgr using a computed rule.

Calc is a general routine that accepts an algebraic expression and returns a C-CLASSIC concept. In the case where pmgr is filled with the integer 2, Calc returns the concept expression (all ds1_packs (upper-limit 5)).

The expression (test-c fills? pmgr) is a filter or guard that is used to determine when an individual has enough information for computed-concept Calc to be applied.

The set-descriptor function immediately below DS1_IF_max_ds1_packs simply attaches a string to the rule that elaborates what the rule means. We anticipate that description strings will eventually be used to enhance error-handling capabilities or the ability of PROSE to explain what it is doing in the context of the application (the Common Lisp version of CLASSIC already has a rudimentary way to explain what it is doing). For now, the explanations are simple English elaborations of what the rules mean.

The rule DS1_IF_eq_ds1_packs describes how to obtain a value for ds1_packs when the ds1 capacity of a DS1 IF shelf is known.

For a new configuration, information about the user's selection of features is passed to the knowledge base by filling roles or adding value or number restrictions to individuals. As information is added, these individuals are classified under the DACS IV-2000 concepts, triggering computed rules and causing the appropriate integrity checks to be performed. The rules sometimes cause propagations and chaining such that additional integrity checks are performed, and additional rules are applied.

It is possible for a configuration to be overconstrained. From the user's point of view, a configuration becomes overconstrained when an incompatible set of features is selected. Perhaps the user wants to have both feature *x* and feature *y,* but feature *x* and *y,* when combined, exceed some capacity limitation of the equipment. In such cases, PROSE provides customized error messages based on the C-CLASSIC error-handling features. These messages describe the problem adequately, but PROSE lets the user decide what feature (or constraint) to change or withdraw.

The compatibility rule idea is rather deeply imbedded in the existing process and the thinking of the product experts. For example, paper documents describe the compatibility rules for each product. These documents are used to support manual validation procedures for incoming orders at the factory.

Consequently, product experts are most comfortable thinking in terms of compatibility rules. Although more satisfying representations might exist, we are convinced that representing compatibility rules directly in C-CLASSIC is currently the only realistic choice. C-CLASSIC makes it possible to represent these rules in a rather straightforward way so that any time a rule must be changed, only a localized piece of code is affected. We find the expressiveness of the C-CLASSIC description language, when enhanced with a few hand-coded test functions, to be completely adequate for our purposes.

```
(define-primitive DS1_IF
 (and
   shelf
   (all ds1_lines (range 0 224))
   (all ds1_packs (range 0 7))
   (all common_packs (range 0 1))
   (all pmgr (range 0 7))
   (all ds3_lines (range 0 0))
   (all ds3_packs (range 0 0))
   )
)


(define-rule DS1_IF_max_ds1_packs
 (and
     DS1_IF
     (test-c fills? pmgr)
  )
  (computed-concept Calc  (ds1_packs <= 7 - pmgr))
 )

(set-descriptor DS1IF_max_ds1_packs
 "There are seven slots in a DS1IF shelf for both pmgrs and DS1 packs")

(define-rule DS1IF_eq_ds1_packs
 (and
   DS1_IF
   (test-c fills? ds1_lines common_packs)
  )
 (computed-concept  Calc (ds1_packs = (ds1_lines / 28) - common_packs))
)

(set-descriptor DS1IF_eq_ds1_packs
 "Formula for computing ds1_packs from ds1_capacity and common_packs")
```

Figure 4. DACS IV-2000 Knowledge for a DSF IF Shelf.

Each product knowledge base in the PROSE platform is not simply an undifferentiated collection of compatibility rules. Rather, there is a standard way in which such rules are organized within a product knowledge base for all products. We refer to the C-CLASSIC structure that describes such an organization as the *order template*.

The order template describes the items and the logic for assembling a valid order. It is the organizing principle lying behind each knowledge base. Although it differs in certain details from product to product, the general outline of the order template is the same for all products.

C-CLASSIC provides an important benefit that we have not yet discussed. Because all objects

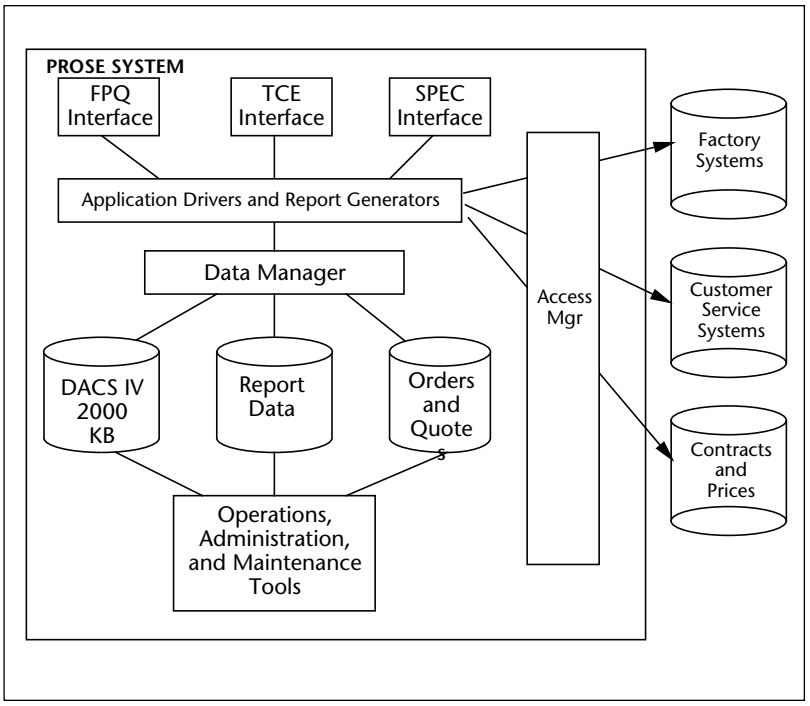

Figure 5. PROSE Software Architecture.

are classified in the C-CLASSIC system, the description of each object must, in a sense, be consistent with all other object descriptions. Inconsistent descriptions are detected at compilation time, that is, when the knowledge base is loaded into C-CLASSIC. In the context of the PROSE platform, the inconsistencies represent either incorrect knowledge or incorrectly encoded knowledge, and they must be investigated and corrected. The detection of inconsistencies by C-CLASSIC has been an important debugging tool within the PROSE platform.

PROSE Architecture

Three PROSE installations are currently up and working. All are based on the SUN 490 platform. The first, which processes orders for transmission products, is located at AT&T's Merrimack Valley Works Data Center in Massachusetts. The second, used for Autoplex, is at AT&T Bell Laboratories in Whippany, New Jersey, and the third, a microelectronics system, is at AT&T's Dallas Works Computer Center. Each PROSE installation allows access to more than one product configurator.

PROSE users access the PROSE computer through AT&T's DATAKIT corporatewide area network. The PROSE computers are also on the corporate XNA network so that PROSE can access AT&T's mainframe computers.

A high-level view of the PROSE software architecture is shown in figure 5. The top of the picture shows PROSE's three user communities. The three user interfaces contain menus and forms, pick-and-choose options, and pop-up windows that make the application user friendly.

Feature selections and choices are passed to the PROSE knowledge base through an application driver and a data manager. Report data such as installer's notes, equipment codes, and a few other items are stored in flat files. Orders and quotes are also saved in flat files so that users can do some of the work on an order, interrupt it, and then return later to complete the same order.

Intelligent programs tend to be factored into control, operations, and data at a high level. The application drivers, the data managers, and the knowledge base can be thought of as the control, operations, and data for PROSE. The following example illustrates this arrangement.

In the case of DACS IV-2000, the application driver for FPQ (the pricing module) is basically a simple search program over the space of DACS IV-2000 configurations. An FPQ user enters the desired DS1 and DS3 signal capacity that defines the goal state for the search program. For efficiency, the minimum and maximum capacities of each bay type are precompiled into the knowledge base and not calculated dynamically.

Users are mostly interested in solutions with the minimum number of bays because the number of bays is the major determinant of cost. The FPQ application driver searches first for the one-bay solutions. If no one-bay solution is found, it searches for two-bay solutions, and so on. If solutions are found at any level, the program continues to search the level exhaustively and returns all solutions to the user. Programs at the data manager level support the search by supplying successor nodes and data for testing. Although the search algorithm is customized for the application, it appears to be related to iterative deepening (Korf 1985).

AT&T's corporate systems are used to either price the quote or send the order to the appropriate AT&T factory. From the PROSE side, these functions are performed by software called the access management interface and access manager. These modules provide an application-to-application protocol between PROSE and the mainframe applications.

PROSE also contains a suite of operations, administration, and maintenance tools.

These tools assist in system administration, including adding and removing users, doing backups, and installing new software. Also included here are tools to update the knowledge base and other data files.

Although PROSE uses AI technology, the majority of PROSE's software is written using conventional techniques. In fact, for the DACS IV-2000 configurator, only 15 percent of all PROSE code for the DACS IV-2000 makes up the knowledge base. Thus, although the knowledge base is a significant part of PROSE, procedural programming is still necessary to produce a useful, production-quality product. This reinforces our previous experience with AI applications (Wright, Zielinski, and Horton 1988; Ackroff et al. 1990).

Most of the non-CLASSIC code is reusable and does not need to be rewritten for new product configurators. Currently, to produce a new configurator, the major pieces that have to be rewritten are the user interface and the knowledge base. Although these modules will always be product specific, and new ones must be provided for each configurator, we are moving toward the development of standard methodologies that will greatly improve our productivity for product-specific modules.

PROSE Deployment

The PROSE platform was designed and developed by a team of seven system engineers and developers. The first release of PROSE, release 1.0, had a production interval from conception to delivery of 8 months.

Use

Table 1 shows the PROSE 1.0 schedule from initial contact with the customer to the released product and includes knowledge base and application design, development, system test, and documentation.

Table 1 does not cover the development of C-CLASSIC. C-CLASSIC was available before PROSE 1.0 development began. Table 1 should not be construed as representing the effort needed to develop and deploy a new product configurator under the PROSE platform. Development schedules for new products today are a fraction of what they were for the first product configurator in 1990.

Currently, PROSE is used in all AT&T-NS sales regions. PROSE users include regional engineers, technical consultants, account executives, members of the design community, and product management. We expect that new configurators for additional products will be developed under the PROSE platform in the coming year.

January 1, 1990	Initial contact made with customers.
April 1, 1990	DACS IV-2000 is selected as first product offering. PROSE knowledge base development begins.
June 1, 1990	PROSE 1.0 platform development begins.
August 30, 1990	PROSE 1.0 is available. First DACS IV-2000 order is processed.

Table 1. PROSE Development and Release Schedule.

Benefits from Use

Assuming the continued deployment and use of PROSE in the field, the following benefits are expected in AT&T's order-processing environment: (1) a reduction in operating costs because of the elimination of errors on orders detected by AT&T clerks and the order rework that is carried out to clear these errors; (2) a reduction in operating costs with the consolidation of databases and positions; (3) a decrease in the interval for updating product design changes in the order process infrastructure by eliminating manual interpretation and transcription of drawing information; (4) support for key organizational changes and business practices within AT&T-NS; (5) a decrease in the order process interval by allowing a user to configure, edit, and send the order to the AT&T factory interactively. Existing order process intervals range between 5 percent and 20 percent of what they were before PROSE was introduced.

Future Plans

We have plans for platform enhancements and growth in four areas: (1) knowledge base development and maintenance tools, (2) generic application programs, (3) reusability of the product knowledge, and (4) support of new products.

Knowledge Base Development and Maintenance Tools We would like a product expert to update the knowledge base. The product expert is usually not a programmer and has little or no experience using C-CLASSIC. To make it easy for the product expert to update the knowledge base, we would like to supply tools to update the product information easily. We are currently exploring ways of integrating PROSE with the existing design capture tools being used within AT&T-NS.

In addition, we feel that it is possible to develop an application-specific methodology for developing a new product knowledge base. The methodology would exploit the capabilities of C-CLASSIC to identify redundancies, contradictions, and incompleteness in the product knowledge.

Interestingly enough, we think there are common organizing principles present in all the existing product knowledge bases. These commonalities can serve as the guidelines for a standardized knowledge engineering methodology that could be replicated and improved for the development of all new PROSE knowledge bases.

We are currently at work developing this methodology. The first step in our strategy is to describe all the kinds of knowledge that have to be collected to develop a new product knowledge base as well as a standard format for recording this knowledge. Step two will be to write a translator that can turn the product knowledge into a C-CLASSIC knowledge base. A working prototype of such a translator currently exists and is adequate for handling all the products currently available through PROSE.

From our point of view… we feel that the use of C-CLASSIC and the architecture it encourages has contributed something important to the PROSE project

Generic Application Programs Currently, a new user interface, as well as certain other application programs (such as error-handling routines), have to be written to reflect the structure of a new product. Because all the information about the product is or could be located in the knowledge base, it would save time if PROSE relied on general application programs that queried for data encoded in the knowledge base. Thus, to develop a new configurator, one would collect all the product-specific information, put it in the standard format, and run the translator.

Because the benefits are so significant, we are beginning to think about the architecture of a generic collection of application programs that could be used with any product knowledge base. For example, the user interface could query the knowledge base, following accepted conventions, to determine what information to present to the user. Because the correspondence between the knowledge base and the menu structure in PROSE's user interface is fairly straightforward, we feel that this area is worth pursuing.

Reusability of the Product Knowledge Earlier we discussed one form of reuse—that which occurs when several application programs access the same product knowledge. However, it seems to us that we can take advantage of the modularity provided by C-CLASSIC's object-centered nature to build higher-level configurators. For example, the equipment currently available through PROSE is used by engineers to piece together elements of a working telecommunications network.

SLC series 5 carrier equipment and DDM-2000 shelves, which are two kinds of transmission equipment available through the PROSE platform, can be combined to make something called an access node. There is no reason why a higher-level template couldn't be put together that describes how to assemble an access node. In theory, one could continue adding more structure above the existing product knowledge to provide the customer with higher and higher–level solutions without changing or replicating the product knowledge at the lower levels.

Support of New Products PROSE currently handles configuring and order processing for the AT&T-NS DACS IV-2000, the DACS II CEF cross-connect, the SLC series 5 carrier system, the E2 power system, the DDM-2000 multiplexer, and Autoplex remote cell sites. Although we expect to spend some of our time on knowledge base maintenance and user interface enhancement, new product configurators are planned for the PROSE platform in the coming year.

Discussion

In this final section, we focus on whether we have selected a good software architecture for the PROSE configurator. From our point of view as software developers, we feel that the use of C-CLASSIC and the architecture it encourages has contributed something important to the PROSE project.

Change and modification is an integral part of the process of developing a configurator, and telecommunications products change continuously in response to the marketplace. Further, although having well-designed products helps, change is inevitable for reasons having nothing to do with product design. As the business needs of our customers change, they desire new capabilities in the products they buy, and AT&T responds accordingly. The key point is that the pace and nature of the changes are out of the hands of the software developer. Products change in response to marketplace forces, and the configurators must be ready when the new products are ready.

Planning for sometimes arbitrary but necessary change and using an architecture that is responsive are essential if PROSE is to be a successful product. It should be no surprise to

the AI community that factoring the product into control, operations, and data modules at a high level has proved to be a key element in the ability of PROSE to deal with change.

A few examples might help us make our point. Not long after PROSE 1.0 was introduced, some rather sweeping changes in the DACS IV-2000 product, collectively called DACS-IV Generic 2, were introduced. These changes were implemented in the PROSE knowledge base without affecting PROSE at either the data manager or the application manager level. Further, simply by modifying the knowledge base, these changes were made available to the FPQ, SPEC, and TCE applications simultaneously.

Similarly, as PROSE was introduced to new customers, we found that some customers were only interested in seeing certain classes of configurations in the output of the DACS IV-2000 FPQ. These changes were implemented through localized changes in the FPQ search algorithm, essentially by allowing the customer additional control over the definition of a goal state through the user interface.

C-CLASSIC itself plays a key role in the management of change. Although this system is moderately sized, without some form of mechanical assistance, the knowledge engineer has a difficult job maintaining consistency while frequently updating the descriptions with new product knowledge.

We find that the consistency checking provided by C-CLASSIC feels somewhat like programming in a strongly typed language: Many errors are detected in the compilation stage. The bottom line is that the knowledge engineer can feel confident about attacking changes in the product structure, making it feasible to keep pace with new product knowledge.

Acknowledgments

No AI application like PROSE becomes successful without contributions from many diverse sources. The number of people who have contributed at different times during the project makes it impossible to list everyone by name. Briefly though, the research department at AT&T Bell Laboratories headed by Ron Brachman helped us in ways too many to describe. Clearly, the developers and system engineers in Harry Moore's and Jay Berman's organizations and their respective managements were essential. They turned a good idea into reality, which is probably the toughest job of all. Many forward-looking people at AT&T-NS provided support at the right times. In particular, we would like to thank John Ehasz and Dennis Dibert, who believed in what we were doing and were willing to try something new.

Notes

1. DS1 (1.5 megabytes [MB] a second or the equivalent of 24 circuits) and DS3 (45 MB a second or the equivalent of 672 circuits) are standard digital transmission rates in the United States.

2. In some cases, a small amount of additional information, such as the desired location of certain circuit packs, might be required.

3. A prototype of the PROSE system was developed using Common Lisp CLASSIC.

References

Ackroff, J.; Surko, P.; Vesonder, G.; and Wright, J. 1990. SARTS AutoTest-2. In *Practical Experience in Building Expert Systems*, ed. M. Bramer, 209–226. New York: John Wiley & Sons.

Bachant, J. 1988. RIME: Preliminary Work toward a Knowledge-Acquisition Tool. In *Automating Knowledge Acquisition for Expert Systems,* ed. S. Marcus, 201–224. Norwell, Mass.: Kluwer Academic Publishers.

Barker, V., and O'Connor, D. 1989. Expert Systems for Configuration at Digital: XCON and Beyond. *Communications of the ACM* 32(3): 298–318.

Borgida, A.; Brachman, R.; McGuinness, D.; and Alperin-Resnick, L. 1989. CLASSIC: A Structural Data Model for Objects. In Proceedings of the 1989 ACM SIGMOD International Conference of Data, 59–67. New York: Association of Computing Machinery.

Brachman, R., and Schmolze, J. 1985. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2): 171–216.

Brachman, R.; Fikes, R.; and Levesque, H. 1983. KRYPTON: A Functional Approach to Knowledge Representation. *IEEE Computer* (Special Issue on Knowledge Representation) 16(10): 67–73.

Korf, R. 1985. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27:97–109.

Levesque, H., and Brachman, R. 1987. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence* (3)2: 78–93.

McDermott, J. 1982. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19(1): 39–88.

McDermott, J., and Bachant, J. 1984. R1 Revisited: Four Years in the Trenches. *AI Magazine* 5(3): 21–32.

O'Brien, J.; Brice, H.; and Hatfield, S. 1989. The Ford Motor Company Direct Labor Management System. In *Innovative Applications in Artificial Intelligence,* eds. H. Schorr and A. Rappaport, 81–87. Menlo Park, Calif.: AAAI Press.

Patel-Schneider, P. 1984. Small Can Be Beautiful in Knowledge Representation, AI Technical Report Number 37, Schlumberger Palo Alto Research, Palo Alto, California.

Weixelbaum, E. 1991. C-CLASSIC Reference Manual,

Release 1.0, Technical Memorandum 59620-910731-017M, AT&T Bell Laboratories, Murray Hill, New Jersey.

Woods, W., and Schmolze, J. 1993. The KL-ONE Family: Computer and Mathematics with Applications. *Artificial Intelligence* (Special Issue on Semantic Networks). Forthcoming.

Wright, J.; Zielinski, J.; Horton, E. 1988. Expert Systems Development: The ACE System. In *Expert Systems Applications to Telecommunications*, ed. J. Liebowitz, 45–72. New York: John Wiley & Sons.

Jon Wright is a distinguished member of the technical staff in the Software Technology Center at AT&T Bell Laboratories, Murray Hill, New Jersey. PROSE is one of several AI applications in the telecommunications domain that he has helped successfully bring into the world. He received a Ph.D. in cognitive psychology from Rice University in 1978.

Karen Brown is currently a member of the technical staff in the Software Technology Center at AT&T Bell Laboratories, Murray Hill, New Jersey. In addition to her work with PROSE , she has been both a system engineer and a software developer on several telecommunications and business operations projects at AT&T. She originated the concept of an AI-based configurator platform that led to the PROSE project. She received an M.S.E. in computer and information science from the University of Pennsylvania in 1983.

Stephen Palmer is a member of the technical staff in the PROSE Development Group at AT&T Bell Laboratories, Holmdel, New Jersey. He is the lead developer on PROSE and has worked on both hardware and software projects at Bell Labs. He holds an M.S. in electrical engineering from Monmouth College.

Harry Moore is a technical manager in the QUEST Partnership at AT&T Bell Laboratories, Holmdel, New Jersey. He has managed the development of many successful information systems, including PROSE. He received an M.S. in electrical engineering from Columbia University in 1979.

Elia Weixelbaum is a member of the technical staff in the Software Technology Center at AT&T Bell Laboratories, Murray Hill, New Jersey. He is responsible for the development of the C-CLASSIC knowledge representation language as well as several telecommunications expert systems. Weixelbaum, who joined Bell Labs in 1983, has a B.S. in mathematics from Brooklyn College and an M.S. and a Ph.D. in computer science in the area of formal languages from the Courant Institute at New York University.

Gregg Vesonder is technical manager of the Object-Oriented and Artificial Intelligence Technology Group at AT&T Bell Laboratories, responsible for transferring object-oriented and AI technology to the AT&T business units. Vesonder received a B.A. in psychology from the University of Notre Dame and an M.S. and a Ph.D. in cognitive psychology from the University of Pittsburgh. He was named a Bell Labs fellow for his work on AI.

Jay Berman is a technical manager in the AT&T Bell Laboratories QUEST organization. He is responsible for working with the AT&T business units to reengineer the product- and service-provisioning processes. Berman is also the product manager for the PROSE product offering, which includes defining the future architectural direction of the product to meet the needs of AT&T and its customers. He received his B.S. in electrical engineering in 1978 from The Cooper Union and his M.S.E. in electrical engineering and computer science in 1979 from Princeton University. Berman is also a member of the Institute of Electrical and Electronics Engineers.

Important Deadlines:

AAAI–94 Paper Submissions: January 24, 1994

IAAI–6 Paper Submissions: January 17, 1994

AAAI and IAAI Tutorial Proposals: 1 November 1993

AAAI–94 Videotape Presentation Submissions: January 31, 1994

AAAI-94 Workshop Proposals: October 15, 1993

1994 Spring Symposium Series Submissions: October 15, 1993