Classic and PROSE Stories: Enabling Technologies for Knowledge-based Systems

Jay I. Berman
Harry H. Moore IV
Jon R. Wright

The complexity and scope of telecommunications systems equipment brings special challenges for the order-realization process. In particular, one wants to maintain a simple-to-use interaction between the sales force and the customer, and yet supply downstream engineering and manufacturing processes with sufficient information for manufacturing, billing, and installation. In this paper, we describe a unique approach to developing expert configurators for AT&T Network Systems equipment, using knowledge-based technology that meets the needs of a diverse set of users.

Introduction

From the point of view of someone who develops and supports customized configuration applications, the equipment used in modern telecommunications networks is modular in design, applicable to a wide range and scale of problems, and very complex. Customized solutions are an essential part of doing business, partly because they are the key to providing the most economical networking solutions. Configurators, software applications used to configure sales orders one level up from a bill of materials, play an important role in the process of delivering the right product to customers. We believe that knowledge-based configurators, systems that encode configuration rules as data that can be extracted and manipulated by external programs, have something special to offer.

Perhaps the best known knowledge-based configurator is R1/XCON, developed in the late 1970's.[1] The R1/XCON configurator was groundbreaking in several ways. It not only demonstrated the feasibility of knowledge-based systems in the real world, but it helped establish production rule systems as the dominant method of knowledge representation used by developers of expert systems. Production systems have three major components:

- A local database, also called the *knowledge domain*,
- A set of production rules that operate on the database, and
- A control system that determines what rule is to be applied at any given time.

Programming for this paradigm is sometimes referred to as *rule-based* programming.[2]

By their nature, configurators contain a great deal of knowledge about products. Such knowledge is an important resource, much like the data in corporate databases. Because knowledge of a product is a valuable commodity needed by many different people in a company, expert configurators provide an excellent starting point from which to examine the practicality of reusing a knowledge base by other organizations. While the reuse of knowledge encoded in expert systems often has been discussed within the expert-system community, there are, as yet, few working examples.

Although proven effective for many kinds of expert system applications, in practice, the reuse of production rules has some drawbacks. For one thing, there is a tendency to mix *control* and *domain* knowledge in the same set of rules. Consequently, production rules seldom contain purely declarative knowledge. This makes domain knowledge difficult to reuse, because the control structure implicit in the rules must be adapted to the new application.

The PROSE (PRoduct OfferingS Expertise) configurator platform developed at AT&T Bell Laboratories is based on a knowledge-representation system, that is, a subsystem of an intelligent system that provides knowledge representation services.

Panel 1. Acronyms and Terms Used in This Paper

Assertions — See forward-chaining rules.

C-Classic — C language implementation of the research language Classic used by PROSE.

Classification — An automatic process (using subsumption) that adds new information to a C-Classic knowledge base.

Completion or propagation — The process of mechanically adding to a knowledge base the logical consequences of new facts and assertions.

Contradiction detection — The identification of information mismatches and the provision of alternate choices.

Dependency maintenance — A system of records that maintain information about the relationships among facts, rules, and assertions in a knowledge base.

DACS — Digital Acess and Cross-Connect System trunk termination equipment

Description logic — A family of languages derived from the KL-ONE knowledge representation language.

DS3-16 — 45Mbits/s digital signal transmission rate

FPQ — Firm price quote

Forward-chaining — C-Classic is said to be forward chaining because it derives the logical consequences of any new information it receives as input and saves them for later use.

Frames — A hierarchical representation of data.

Individuals — Instances of C-Classic concepts used to represent equipment configurations in PROSE.

Iterative deepening — A search algorithm used by the PROSE system to derive configurations that meet the customer's specifications.

Legacy — An embedded, pre-existing system used downstream from the order-processing flow to price quotes, orders, etc.

Logic rules document — Standardized specification format used by PROSE to develop a knowledge base.

OA&M — Operations, administration, and maintenance

Object-structuring primitives — A small set of primitive operations.

Roles — A way of describing structure and linking individuals together.

PROSE — PRoduct OfferingS Expertise

RFP — Request for proposal

Semantic nets — A graphical representation of data.

STS1 — 53 Mbits/s synchronous transfer signal

TCE — Telephone company engineered

PROSE uses a class of knowledge-representation systems called a *description logic.* This class offers distinct advantages to applications which require the reuse of a knowledge base. The description logic used by PROSE is called *C-Classic.*[3] It is a C-language implementation of the research language, CLASSIC.[4] As such, C-Classic has close ties both to research on knowledge representation and to database theory.[5]

C-Classic was selected by the PROSE project largely because the reuse of the knowledge encoded in expert configurators suggested a possible solution to an important business problem—inconsistencies in product knowledge within the order-realization process. Briefly, PROSE synchronizes product knowledge by making available a single base of information on a product, or products, that satisfies the needs of those who both order and engineer products.

In addressing the product synchronization issue, PROSE contributed to some important innovations in knowledge-based systems. In particular, PROSE represents an interesting synergy between research and development in which both benefited significantly from the interaction.[4] This reinforces the notion that technology transfer is bi-directional, and not simply a process of handing off technology to developers.

Among the innovations of the PROSE project are:

- One of the first successful applications that uses a description logic for knowledge representation,
- A purely declarative-knowledge base in which a limited form of knowledge reuse is demonstrated,

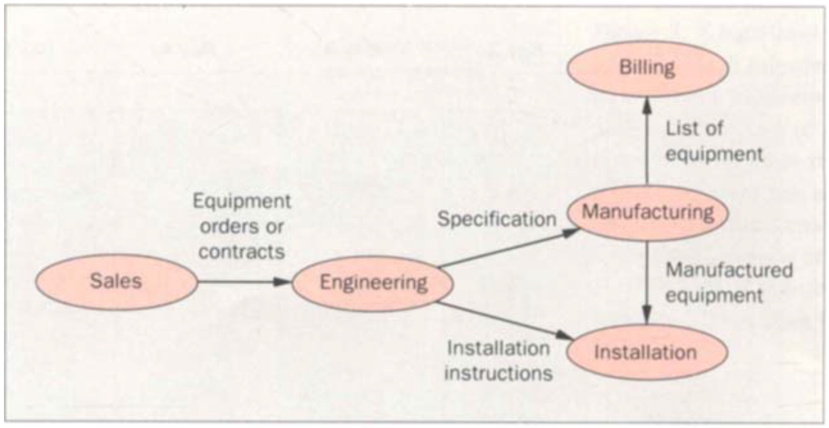


Figure 1. This illustration provides a high-level view of the stages involved in ordering and delivering products within an assemble-to-order or manufacture-to-order industry.

- A general architecture for configurators, not tied to any specific product,
- Close integration with the order-processing infrastructure at AT&T Network Systems, and
- A specification technique, similar to an application generator, that allows non-programmers to maintain product knowledge.

Product Syncronization

Figure 1 provides a high-level view of the stages involved in ordering and delivering products within an assemble-to-order or manufacture-to-order industry. In such industries, the expense and complexity of individual products require the producer and consumer to contract for the purchase of products before they are actually manufactured. In large part, the manufacturing of telecommunications equipment fits this description.

Sales of telecommunications equipment are often organized around large-scale contracts for which vendors bid after the customer announces a request for proposal (RFP). Typically, engineering begins after the bid is accepted by the customer. An engineering stage is required for several reasons:

- The manufacturing entity must be provided with an unambiguous specification of the product that was purchased by the customer, and
- Typically, the equipment is customized for a specific site and, therefore, installers must be provided with site-specific instructions.

Each stage shown in Figure 1 is associated with a slightly different view of the product. The view at the customer/sales level, by necessity, is simplified and feature oriented. In contrast, downstream processes, such as manufacturing and installation, require a low-level, detailed description that focuses on the physical design of the product. Engineering, in this context, is the process that translates the high-level view, shared by customers and the sales team, into the detailed, lower-level views required by the downstream stages.

Typically, each stage has its own supporting systems and business processes. For example, a sales team might rely on customized computer spreadsheets to return quick estimates of equipment quantities and pricing. While useful in the sales context, these tools lack the detailed structural information and technical accuracy required by manufacturing and installation.

On the other hand, computer systems used by the engineering community often require users to have a level of product knowledge expected only of experienced engineers. In addition, engineering tools require many inputs and do not often provide the quick turnaround needed for a sales team to explore price/functionality tradeoffs that are an integral part of the bid-and-proposal process.

The use of separate systems to support each stage has both positive and negative consequences. While all groups of users have tools that are closely tailored to their jobs, the task of updating and maintaining separate databases, as products change and new products are introduced, must be closely synchronized for all databases if they are to work together properly. In effect, product knowledge is distributed in various forms across the databases shown in Figure 1. The supporting systems associated with each stage also need to be updated together, adding to the complexity. The potential is

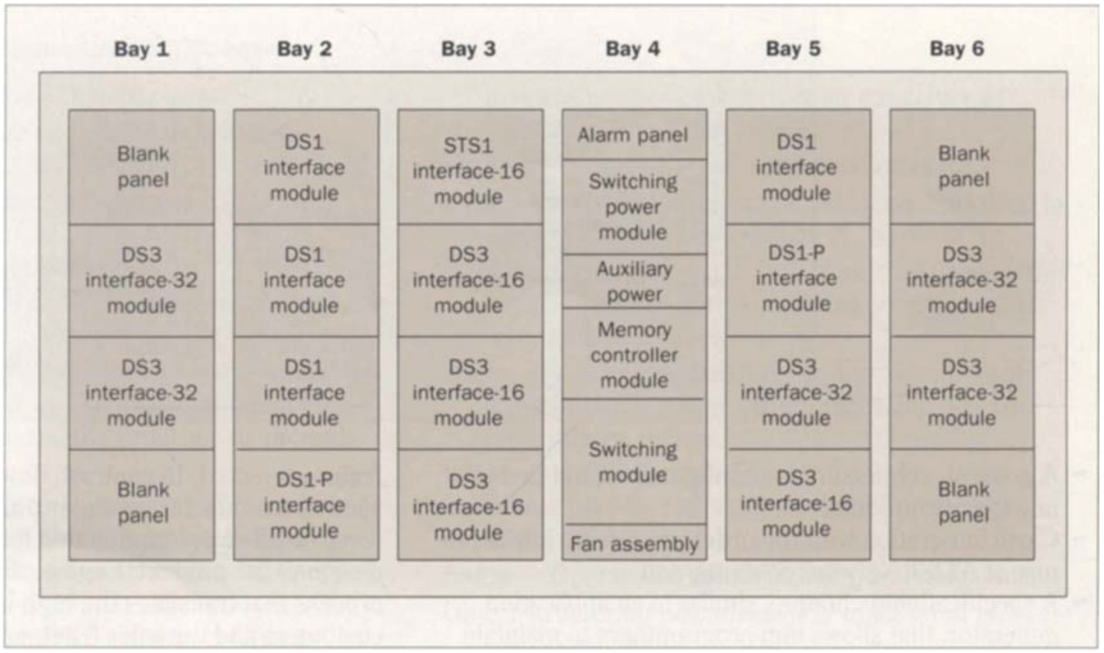


Figure 2. The hierarchical structure for a DACS IV-2000® cross-connect system is a very common characteristic of telecommunications equipment. The configurator problem is how to map, from this high-level view, a description of the internal structure that is required by manufacturing and installation.

enormous for the databases to get out of sync, requiring rework and causing delay as equipment orders are processed through each stage. This is often described as the *product synchronization* problem.

Knowledge Representation for Expert Configurators

A concrete example may help explain some of the main issues in knowledge representation for configurators. Figure 2 shows a lineup of a piece of transmission equipment for a DACS IV-2000® cross-connect system. It plays an important role in automating the cross-connection of high-speed optical and electrical trunks for both switched-voice and leased-line voice and data transmission in the toll network. For our purposes, the most important point to notice about DACS IV-2000 is that it is hierarchically structured, which is very characteristic of telecommunications equipment. A working DACS IV-2000 lineup consists of from two to nine bays, each of which contains shelves or assemblies of electronic gear called modules. Information processing is accomplished by circuit packs that fit into slots in the modules. The bays are connected to each other and to the network by cabling.

Engineers build configurations by specifying, in large part, the internal structure of telecommunications equipment—the location of shelves and circuit packs, interconnections of the cabling, and so on. This data is simply too detailed for the customer/sales interaction, and too labor intensive to be cost effective for bid-and-proposal work. Consequently, configurators written for a sales team contain a great deal of information beyond the physical structure of a product, such as information about features and ordering conventions, and constraints on feature combinations.

Customers and sales people tend to think of a DACS IV-2000 system in terms of how it functions within a working telecommunications network. In that context, a DACS IV-2000 is a network element or node with a certain signal capacity and, perhaps, a small number of high-level features. The configurator problem is one of how to map, from this high-level view, into a description of the internal structure that is required by manufacturing and installation.

Figure 3 shows a high-level view of a database in which knowledge about ordering products is kept separate from knowledge about the physical structure of products. Examples of the types of knowledge that can be encoded in the order model are the acceptable values and ranges of feature choices, constraints on the legal combinations of features, and ordering conventions or standards that are used to structure the output from the configurator.

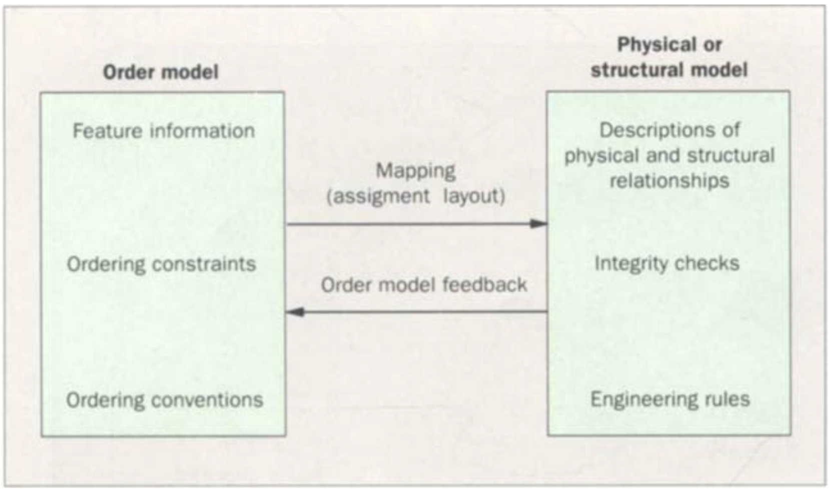


Figure 3. A high-level view is shown of a database in which knowledge about ordering products is kept separate from knowledge about the physical structure of products. Examples of the types of knowledge that can be encoded in the order model are the acceptable values and ranges of feature choices, constraints on the legal combinations of features, and ordering conventions or standards that are used to structure the output from the configurator.

A description of the physical structure of the product is the most important part of the physical model. Not every detail of the product need be represented, however, only what is required to properly feed the downstream processes. Physical dimensions, for example, may be largely irrelevant, but knowledge about slot locations for shelves and circuit packs may be essential.

Integrity checks and engineering rules are also basic parts of the physical model. Integrity checks are used to prevent incompatible equipment types from being included in the same order. *Engineering rules* are statements of how to compute certain quantities of equipment in the language of a product expert. They are implemented as simple forward-chaining rules in C-Classic. Such rules, when invoked, trigger other rules, which in turn can invoke other rules, etc. When an item is listed for a system, the forward-chaining software triggers a list of additional equipment that must be added to complete the order for that item. For example, if a certain capacity is exceeded on a shelf, engineering rules may be activated to add power to that shelf. Engineering rules are a labor-saving device for people with a deep knowledge of a product, because they are saved from having to specify every last detail of the order. Sales people or customers, however, who may have less product knowledge, depend heavily on engineering rules to produce technically correct configurations.

Mapping between the high-level order model and the detailed physical model is a crucial aspect of a telecommunications configurator. In the context of telecommunications systems, the generic term describing this mapping is *assignment*. In the abstract, assignment is a special case of a constraint-satisfaction problem in which demand generated from the order model must be satisfied, given the constraints imposed by the physical model. Optimization can, and often does, play a role here and, although it may or may not be heuristic in nature, is often required to find an economical solution.

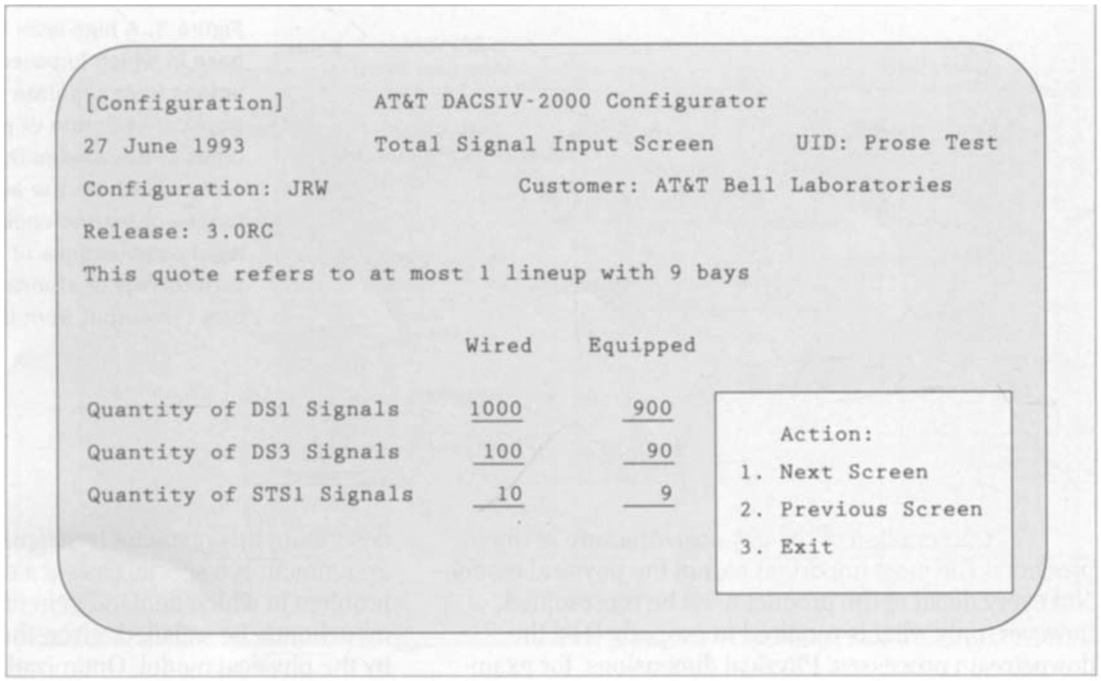
Feedback from the physical model to the order model is another important aspect to consider. One form of feedback takes place when equipment is added after engineering rules are triggered. Such equipment is not explicitly requested by the user, and yet must be accounted for in the final order.

The PROSE Configurator Platform

Fifteen configurators have been successfully developed using the PROSE configuration platform. Most of these employ a knowledge-representation scheme similar to that shown in Figure 3. One interesting aspect of the PROSE project is that, for each configurator, one C-Classic knowledge base drives different user interfaces for sales, engineering, and customer service users.

Figure 4 shows the main screen for the sales/customer interface, called the firm price quote (FPQ) interface for DACS IV-2000. To obtain a meaningful DACS IV-2000 configuration, including pricing, the sales team need only specify a release identifier and six numbers representing the desired signal capacity of the system. The terms *Wired* and *Equipped* are jargon that refer to a distinction between planned and working

Figure 4. The main screen is shown for the sales/customer interface, called the firm price quote (FPQ) interface for DACS® IV-2000. To obtain a meaningful DACS IV-2000 configuration, including pricing, the sales team need only specify a release identifier and six numbers representing the desired signal capacity of the system.



```
[Configuration]          AT&T DACSIV-2000 Configurator
27 June 1993             Total Signal Input Screen      UID: Prose Test
Configuration: JRW                 Customer: AT&T Bell Laboratories
Release: 3.0RC
This quote refers to at most 1 lineup with 9 bays

                              Wired    Equipped

Quantity of DS1 Signals       1000       900        Action:
Quantity of DS3 Signals        100        90     1. Next Screen
Quantity of STS1 Signals        10         9     2. Previous Screen
                                                 3. Exit
```

capacity. In other words, the DACS IV-2000 configurator takes into account the need to grow from initial working capacity to the customer's planned capacity over time.

The seven inputs can be thought of as the smallest set of features required to describe a DACS IV-2000 cross connect. The FPQ application uses these inputs to define a *goal state* for a search routine, that is, a description of the desired features of the configuration. The search routine starts with the minimum DACS IV-2000 lineup, then adds equipment according to knowledge encoded in the structural model, testing for a solution as equipment is added. The search technique used is related to *iterative deepening*.[6] For DACS IV-2000, more than one configuration may satisfy the goal state at any level of the search, and all solutions at that level are returned to the user for evaluation.

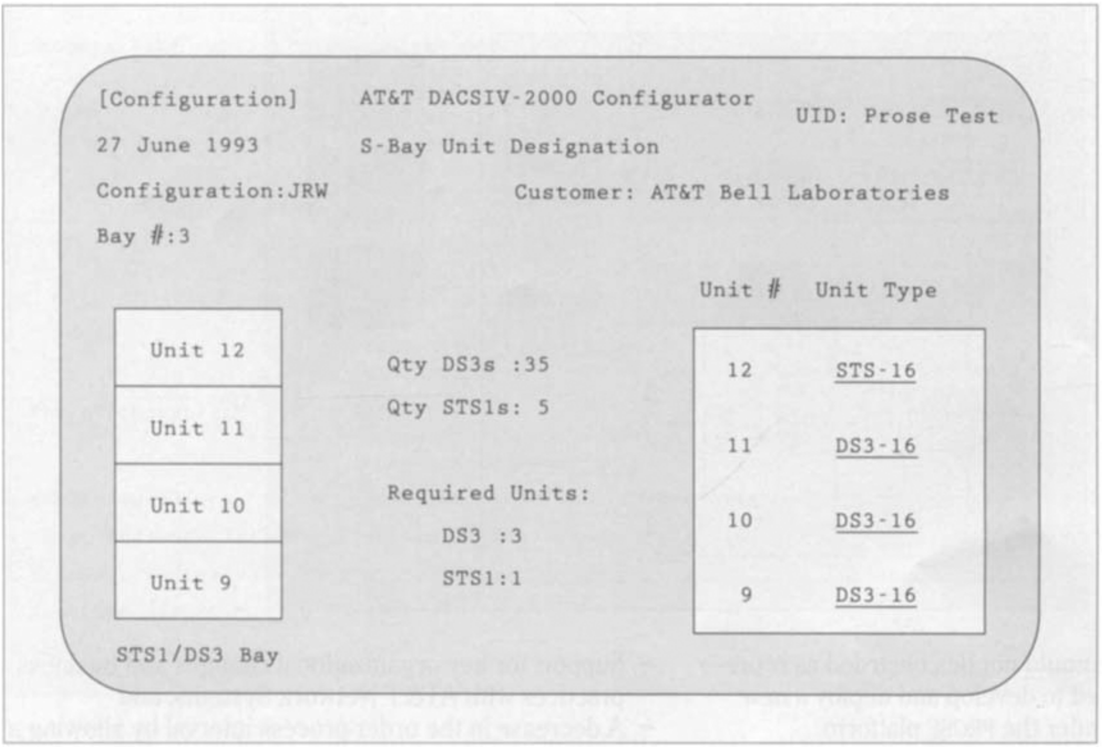
In contrast, the PROSE interface for engineers, called SPEC for *engineering specification*, naturally requires more inputs and more knowledge about the product than the sales/customer interface. But it also has more flexibility. The required inputs cover several screens. An example of one such screen is shown in Figure 5.

Without going into the details, the SPEC input screen shown in Figure 5 provides a way for engineers to describe certain aspects of a DACS IV-2000 item called an S-bay. For this particular S-bay, there are four shelves (Units 9-12), three for types DS3-16 (16 terminations of 45Mbits/s lines) and one for STS1-16 (16 terminations of 53 Mbits/s lines) respectively. Other input screens would be provided to describe the DS3-16 shelves and the STS1-16 shelf. Entering this data triggers both integrity checks and engineering rules such that verification of the order is one of the services provided by the knowledge base.

Equivalent DACS IV-2000 configurations can be derived from both the SPEC and the FPQ applications. For the SPEC application, however, users must provide the mapping from feature to physical model themselves. The FPQ search routine provides this mapping for users. In general, the SPEC interface operates on the physical model directly, in contrast to the FPQ interface, where users operate directly on the order model and information is passed to the physical model via the mapping function.

Software Architecture

There are three PROSE installations deployed and operating, one at the Merrimack Valley Works Data Center for the Transmission Systems Business Unit products in Massachusetts, one in St. Louis, Missouri, for the Network Wireless Systems Business Unit products, and one



```
[Configuration]     AT&T DACSIV-2000 Configurator
                                                  UID: Prose Test
27 June 1993        S-Bay Unit Designation

Configuration:JRW            Customer: AT&T Bell Laboratories

Bay #:3

                                            Unit #   Unit Type

  Unit 12        Qty DS3s :35                 12       STS-16
                 Qty STS1s: 5
  Unit 11                                     11       DS3-16

  Unit 10        Required Units:              10       DS3-16
                     DS3 :3
  Unit 9             STS1:1                    9       DS3-16

  STS1/DS3 Bay
```

Figure 5. The SPEC input screen provides a way for engineers to describe certain aspects of a DACS® IV-2000 item called an S-bay. For this particular S-bay, there are four shelves (Units 9-12), three for types DS3-16 (16 terminations of 45Mbits/s lines) and one for STS1-16 (16 terminations of 53 Mbits/s lines) respectively. Other input screens would be provided to describe the DS3-16 shelves and the STS1-16 shelf.

at the Dallas Works Computer Center for the Microelectronics Energy Systems Business Unit products. All are based on the Sun 690 hardware platform. Multiple configurators run on each of these installations.

Figure 6 shows the PROSE software architecture. The top of Figure 6 shows PROSE's three user interfaces for FPQ, telephone company engineered (TCE), and SPEC. The three user interfaces contain menus and forms, pick-and-choose options, and pop-up windows that make the application user friendly. On some occasions, AT&T customers, such as regional telephone companies, may configure their own orders and send them directly to an AT&T customer service organization. Such orders are called telephone company engineered (TCE) orders, and they can be verified using the PROSE TCE interface.

Feature selections and choices are passed to the PROSE knowledge base through an application driver and a data-manager level. These are library calls to C-Classic, surrounded by defensive programming and error-handling code. Report data, such as installer's notes, equipment codes, and a few other items, are stored in flat files so that users can do some of the work on an order, interrupt it, and return later to complete the same order.

Embedded, pre-existing systems, also called *legacy* systems, are used downstream from the order-processing flow to price quotes and orders, enter an order in an order tracking system at a factory, and so on. On the PROSE side, these functions are performed by the *access manager interface* and the *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 (OA&M) tools. These tools assist in system administration, including adding and removing users, doing backups, and installing new software. Also included are tools used to update the knowledge base and other data files.

PROSE Deployment

The PROSE platform was designed and developed by a team of seven system engineers and developers. The PROSE Release 1.0 schedule extended from January 1990, the initial contact with the customer, to the released product in August 1990. The schedule included knowledge base and application design, development, system test, and documentation.

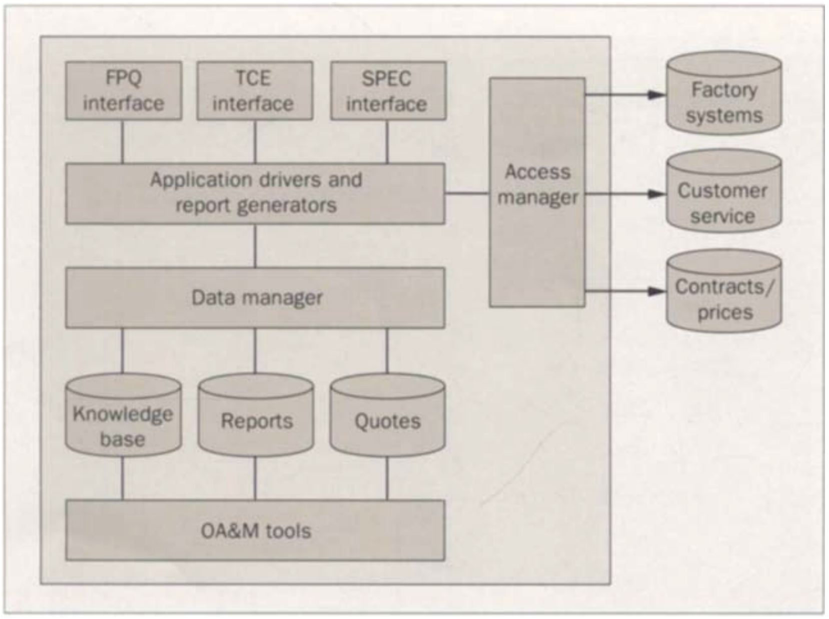


Figure 6. An overview of the PROSE software architecture.

This example should not be construed as representing the effort needed to develop and deploy a new product configurator under the PROSE platform. Configurators for complex products have been developed in a little as four months. However, the entire team strives for continuous improvement in product quality and delivery intervals, and metrics on both have been steadily improving since the first PROSE release. Currently, PROSE is being used in all AT&T Network Systems sales regions. PROSE users include regional engineers, technical consultants, account executives, members of the design community, and product management.

As noted, there are 15 PROSE configurators supporting three AT&T Network Systems business units, and new configurators for additional products are being developed under the PROSE platform in 1994.

Benefits. The following benefits have been achieved in AT&T's order-processing environment:

- A reduction in operating costs, due to the elimination of errors on orders detected by AT&T personnel, and order rework that is carried out to clear these errors,
- A reduction in operating costs through the consolidation of databases and positions,
- A decrease in the interval for updating product-design changes into the order-process infrastructure, by eliminating manual interpretation and transcription of drawing information,
- Support for key organizational changes and business practices with AT&T Network Systems, and
- A decrease in the order-process interval by allowing a user to configure, edit, and send the order to the AT&T factory interactively. This interval compares favorably with the previously existing process that took anywhere from two days to two weeks.

C-Classic Knowledge Representation

Now we will look at the development of PROSE. As noted, the knowledge-representation system selected by the PROSE project, C-Classic, is a member of a family of languages known as *description logics.* These languages are derived from KL-ONE (for knowledge language), a knowledge-representation system that was developed in the late 1970's.[7] Such languages have a close link to modern database theory,[8] and they are closely related to knowledge representation methods called frames,[9] and to *semantic nets.*[10]

Underlying all description logics is the notion of organizing a small set of primitive operations, sometimes called *object structuring primitives*, into a *description language* (see Brachman[11] for the original thinking on this idea). The choice of primitive operations for the description language is crucial, because they are known to have a profound impact on the complexity of the underlying inference algorithms.[12] Consequently, the designers of

```
<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>[<cl-classic-object>...])
<individual>::=            <host-individual>|<classic-individual>
<host-individual>::=       <integer>|<float>|<string>
<classic-individual>::=    <symbol>
<concept-name>::=          <symbol>
<role>::=                  <symbol>
```

Figure 7. An example of the C-Classic description language is shown here. The internal structure of concepts and individuals is described using value restrictions (the *all* primitive) and number restrictions (*at-most* and *at-least* operators).

description logics play close attention to the choice of the description-language primitives. Description logics all share the characteristic of being based on a coherent, well-defined logical theory.

C-Classic has an exceptionally simple description language. The fundamental building blocks in C-Classic are *concepts, individuals,* and *roles.* A *concept* or *concept description* is an expression composed from the primitive operations as defined by the description language. A C-Classic knowledge base consists of both concepts and instances of those concepts called *individuals.* They are related to each other in an underlying data structure sometimes called a *concept hierarchy.* The most general concepts are located high in the hierarchy, and more specific concepts are lower in the hierarchy. Whether or not a given concept is more specific or more general than another concept can be determined by examining their concept descriptions. C-Classic has an algothrim, often called the *subsumption* algorithm, that mechanically determines whether one concept is more general or specific than another.

Concepts inherit descriptions from other concepts located higher in the hierarchy. C-Classic is said to have *multiple inheritance* because a single concept can inherit from more than one parent. *Roles* provide a way of describing structure and linking individuals together.

The following example illustrates how concepts, individuals, and roles are used to describe and represent products in a PROSE knowledge base. DACS IV-2000 has an assemblage of electronic equipment called a DS3-16 shelf. The structure of the DS3-16 shelf is represented in C-Classic by a concept. In effect, the concept describes the rules of composition for the shelf—the fact that certain slots in the shelf accept only certain kinds of circuit packs, that cabling has to be of a certain type, and so on. However, many DS3-16 shelves are often included in real DACS IV-2000 configurations. These shelves are represented in C-Classic by individuals. However, they all follow the same rules of composition as defined by the DS3-16 concept description.

Among other things, roles are used to link the various components of a configuration such that important structural aspects are preserved. A single bay in a DACS IV-2000 configuration can have up to four shelves of electionic gear. Roles are used to link the bay individual with four shelf individuals that together represent the physical structure needed for that particular configuration. Associated with these roles are special concepts called value restrictions (the *all* primitive in Figure 7). They help ensure that all configurations produced by the PROSE system are valid.

The C-Classic language performs the following kinds of inferences:

- *Classification,* the automatic insertion of descriptions in a concept hierarchy;
- Completion, or *propagation,*of logical consequences, such as the *all* referenced above, which completes the definition of a switching bay; that is, completes the list of required equipment;
- *Contradiction detection,* which identifies mismatches of equipment types, and might provide alternate choices;
- Simple *forward-chaining rules* (or triggers); and
- *Dependency maintenance,* or "truth maintenance" discussed below.

All languages derived from KL-ONE support an interesting operation known as *classification.* New concept and individual descriptions are classified in the hierarchy automatically by analyzing their internal structure and comparing that to concepts already encoded in the hierarchy.

Partly as a side effect of classification, there is a sense in which all C-Classic knowledge bases must be internally consistent. Because the underlying algorithms cannot properly classify descriptions that conflict with pre-existing knowledge, contradictory descriptions submitted for classification are detected and rejected by the system.

Classification in C-Classic can be used to drive the application of forward-chaining rules, called *assertions,* to individuals. The expert configurators developed by the PROSE team use this mechanism to build configurations from user inputs.

C-Classic maintains a system of dependency records that collectively form the basis for a *truth maintenance system* (TMS). In a word, it keeps you honest. It is possible for someone to select a feature and later change that selection. The TMS allows users to *retract* such *told* information from a C-Classic knowledge base, thereby unwinding any of the logical consequences depending on that information. The TMS is also an essential part of C-Classic's error-handling mechanisms.

Owing to a long tradition of research on the complexity of the inference algorithms in description logics, C-Classic inference is known to be tractable for most practical problems.[13] This is an important characteristic for any knowledge-representation system used in real applications.

C-Classic has its origins in research on knowledge representation conducted in the Artificial Intelligence Principles Department at Murray Hill, N.J. Interestingly enough, our experience in technology transfer on the PROSE project has demonstrated that there is something important to be gained both by research and by the application team. Brachman[14] has discussed the process from the research perspective. He cites three categories of changes that are the direct result of the technology-transfer process:

- Language improvements,
- Interface improvements, and
- System features.

He concludes that the C-Classic/PROSE experience shows that "implementation and use ... are vital complements to work on knowledge-representation theory."

Reusing Product Knowledge

Knowledge-based techniques are not the only method of implementing configurators. The most common method is the use of a collection of "if-then-else," or *nested case,* statements in a conventional programming language, like C or Pascal. We argue that such a method is no less an encoding of product knowledge than the techniques we have described here. The key difference in C and Pascal is that, once compiled into object code, knowledge about the product that was embodied in the source code becomes unrecognizable. This is an important resource that is, in a sense, lost.

Product knowledge represented in C-Classic is in a symbolic, declarative form. As such, it can be retrieved and manipulated. A purely declarative-knowledge base has some interesting characteristics. For example, although it may contain rules that perform actions, there is no flow of control information, and, therefore, the order in which inputs are received is irrelevant. External programs can construct a set of inputs in the most natural way, send them to a C-Classic knowledge base, and expect identical results.

Most importantly, however, the declarative nature of knowledge encoded in C-Classic lays the necessary groundwork for reuse applications. Krueger[15] has identified eight levels of software reuse. At the highest level are reusable software architectures (see Gelman[16] for a discussion of this topic, and Beck[17] for a concrete example of a reusable architecture). Hsueh et al.[18] also are concerned with reuse issues, although they focus instead on the definition and support of processes for predictable and reproducible results in software development.

Software reuse is foremost in our minds as well. We think there are common organizing principles present in the 15 PROSE configurators that can be extended. Simply recognizing and understanding a common problem structure provides substantial leverage in software development. However, we think it is possible to go further. In our case, such commonalities serve as the guidelines for a standardized knowledge-engineering methodology that can be replicated and improved upon for the development of all new PROSE configurators. We are currently using and further defining this methodology.

The first step we have taken in this direction has been to collect all the kinds of rules, etc., that are needed to develop a new product knowledge base, and develop a standard format for recording this knowledge. We call this specification format the ***logic rules document*** (LRD), and it is described in a document called the "PROSE Knowledge Engineering Guidelines."[19] The format is intuitive enough so that someone with product knowledge, but no experience with C-Classic, can read and understand an LRD.

In addition, we have developed a translator that produces a working C-Classic knowledge base from a specification document. Today, PROSE configurators are developed by producing an LRD and integrating the resulting C-Classic knowledge base with the other modules developed within the PROSE platform.

In a sense, the knowledge-engineering methodology, the specification format, and the PROSE translator institutionalize the common problem structure that we have uncovered in the configurator domain. The PROSE translator qualifies as a type of application generator, one of the eight levels of software reuse identified by Krueger, although it presently covers a limited domain.

Since all the information about a product is, or could be, located in a knowledge base, the reuse paradigm could be extended if PROSE relied entirely on generalized application programs that queried the knowledge base for data that distinguished one configurator from the other. Thus, to develop a new configurator, one would collect all the product-specific information, put it in the LRD format, and run the translator. For example, the user interface could query the knowledge base, following accepted conventions, to determine what information to present to the user.

Finally, it occurs to us that we can take advantage of the modularity provided by our architecture to build higher-level configurators. The equipment currently available through PROSE, for example, is used by engineers to piece together elements of a working telecommunications network. SLC-2000® Carrier equipment and DDM-2000® shelves, which are two kinds of transmission equipment available through the PROSE platform, can be combined to form what are called access nodes in the loop plant for customer line and trunk access to the network. 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 higher- and higher-level solutions to the customer. We envision the ability to link our existing configurators together in a larger system that configures at the network level, and yet supports engineering for individual network nodes.

Acknowledgements

The authors wish to express their appreciation to the following individuals for their generous support: Ron Brachman, Elia Weixelbaum, Karen Brown, Gregg T. Vesonder, Stephen Palmer, and the systems engineers and developers of our organizations who worked with us on this effort.

(Manuscript approved October 1993)

References

1. McDermott, John. 1982, "R1: A rule-based configurer of computer systems," *Artificial Intelligence*, Vol. 19, No. 1, pp. 39–88.
2. Vesonder, G. T. "Rule-Based Programming in the UNIX System," *AT&T Technical Journal*, Vol. 67, No.1, 1988, pp. 69–80.
3. Weixelbaum, Elia S. 1991. "C-Classic Reference Manual Release 1.0," AT&T Bell Laboratories.
4. Brachman, R. J., A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick. "The CLASSIC Knowledge

Representation System, or, KL-ONE: The Next Generation," *Proceedings of the International Conference on Fifth Generation Systems*, Tokyo, June, 1992.
5. Borgida, A., R. J. Brachman, D. L. McGuiness, and L. A. Resnick, "CLASSIC: A structural model data model for objects," *Proceedings of the 1989 Association for Computing Machinery (ACM) SIGMOD International Conference on Management of Data*, pp. 59–67, June 1989.
6. Korf, R. E. 1985, "Depth First Iterative Deepening: An Optimal Admissible Tree Search," *Artifical Intelligence*, Vol. 27, pp. 97–109.
7. Brachman, R. J., and J. G. Schmolze. "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, Vol. 9, No. 2, pp. 171–216, April-June, 1985.
8. Borgida, A., R. J. Brachmann, D. L. McGuiness, and L. A. Resnick. "CLASSIC: A structural model data model for objects," *Proceedings of the 1989 Association for Computing Machinery (ACM) SIGMOD International Conference on Management of Data*, pp. 59–67, June 1989.
9. Minsky, Marvin, "A Framework for Representing Knowledge," J. Haugeland (ed.) *Mind Design*, Cambridge, MA: The MIT Press, 1981.
10. Sowa, John F. (ed.). *Principles of Semantic Networks; Explorations in the Representation of Knowledge*, New York, Morgan Kaufmann, 1991.
11. Brachman, R. J. "A Structural Paradigm for Representing Knowledge." Ph.D. thesis, Harvard University, Cambridge, MA, 1977. Revised version published as BBN Report No. 3605, Bolt Beranek and Newman, Inc., Cambridge, MA, July 1978.
12. Levesque, H. J., and R. J. Brachman, "Expressiveness and tractability in knowledge representation and reasoning," *Computational Intelligence*, 3(2), Spring, 1987, pp. 78–93.
13. Patel-Schneider, P. F., "Intractability and CLASSIC," AT&T Bell Laboratories Memorandum, February, 1993.
14. Brachman, R.J., "Reducing Classic to Practice: Knowledge Representation Meets Reality," in B. Nebel, C. Rich, and W. Swartout (eds.) *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, 1992.
15. Krueger, C. W. 1992. "Software Reuse," *Association for Computing Machinery (ACM) Survey*, Vol. 24, No. 2, pp. 131–183.
16. Gelman, S. J., F. M. Lax, and J. F. Maranzano, "Competing in Large-Scale Software Development," *AT&T Technical Journal*, Vol. 71, No. 6, 1992, pp. 2–11.
17. Beck, R. P., S. R. Desai, D. R. Ryan, R. W. Tower, D. Q. Vroom, and L. M. Wood, "Architectures for Large-Scale Reuse," *AT&T Technical Journal*, Vol. 71, No. 6, 1992, pp. 34–45.
18. Hsueh, T. R., Houghton, T. F., Maranzano, J. F., and Pasternack, G. P. "Software Production: From Art/Craft to Engineering," *AT&T Technical Journal*, 1994, this volume.
19. Foster, C. H., J. H. Ostar, "PROSE Knowledge Engineering Guidelines, Translator Version 0.3", AT&T Bell Laboratories, 1993.

__Jay I. Berman__ is a technical manager in the Product/Service Provisioning Knowledge Domain of the AT&T Bell Laboratories QUEST Partnership. Mr. Berman joined AT&T in 1978 and is responsible for designing innovative solutions to enable the AT&T business units to provide fast provisioning processes for its products and services. He has a B.S.E.E. degree from The Cooper Union, New York, N.Y. and a M.S. degree in electrical engineering/computer science from Princeton University, Princeton, N.J.

__Harry H. Moore IV__ is the technical manager of the PROSE Development Group of the QUEST Partnership in AT&T Bell Laboratories. The group has developed and deployed 15 PROSE Configurators for the Transmission, Network Wireless, and Energy Systems Business Units. Mr. Moore joined AT&T in 1978. He has a B.S.E.E. degree from the University of Delaware, Newark, and an M.S.E.E. degree from Columbia University, New York City, N.Y.

__Jon R. Wright__ is a distinguished member of technical staff in the Software Technology Center at AT&T Bell Laboratories in Murray Hill, New Jersey. He is a designer, developer, knowledge engineer, and a consultant for a variety of knowledge-based systems at AT&T Bell Laboratories. Mr. Wright has a B.S. degree in psychology from the University of Tulsa, Oklahoma, and a Ph. D. in experimental psychology from Rice University, Houston, Texas.