An Industrial-Strength Description Logic-Based Configurator Platform

Deborah L. McGuinness and Jon R. Wright, AT&T Labs—Research

USING DESCRIPTION LOGICS AS A FOUNDATION, THE AUTHORS DEVELOPED COMMERCIAL CONFIGURATORS FOR LARGE, COMPLEX TELECOMMUNICATIONS PRODUCTS.

MODERN TELECOMMUNICATIONS equipment is highly modular and can scale to a wide range of applications. Usually, the equipment's cost and complexity requires that it be manufactured-to-order, or at least assembled-to-order. In this context, orders double as specifications, describing *what* should be manufactured as well as *how* the product should be installed. Producing a correct and complete order for such equipment can be challenging when requirements are incomplete, inconsistent, or when the final product is large and complicated. A good order is technically correct and meets customer requirements for network capacity and growth without over-engineering. Incomplete configurations can lead to cost overruns if the missing elements are discovered during manufacturing. If they are not, faulty products can result. Either way, the customers are unhappy.

We have tackled the configuration problem for a number of large telecommunications products sold by AT&T and Lucent Technologies. Our Prosc configurators are based on CLASSIC,[1] a description logic-based knowledge representation system developed at AT&T Bell Laboratories. CLASSIC is freely available for academic purposes, and commercially available for other purposes, at *http://www.research.att.com/sw/tools/classic*. AT&T has distributed CLASSIC to more than 100 universities; it is also in use for internal projects at AT&T, Lucent, and NCR. Implementations are available in LISP, C++, and C.

CLASSIC is part of a description logic family that was developed with the goal of balancing usability, expressivity, and complexity. We have found it to be well suited to our configurator needs. Because it attempts to provide predictable performance in all cases, CLASSIC is less expressive than many description logic systems, but it has been widely used in both industrial applications and academic systems.

Some of our configurators have been in use since 1990. They have processed more than $4.5 billion in orders and have documented many benefits, including reduced order processing time, reduced staffing, and product-knowledge consistency checking.

Background

We began developing a configurator based on description logic in 1988 when we were approached by a group of system engineers involved in the re-engineering of an important business process, which at AT&T is sometimes called the Engineer, Furnish, and Install process. The EF&I process begins with customer-sales interaction and ends when a product is successfully manufactured and installed. The engineering group had identified product configuration as key to EF&I business processes.

Among the most important goals of this re-engineering project were:

- decreasing the time between new equipment sales and installation, and
- reducing the rework caused by mistakes and inaccuracies in incoming orders.

1094-7167/98/$10.00 © 1998 IEEE

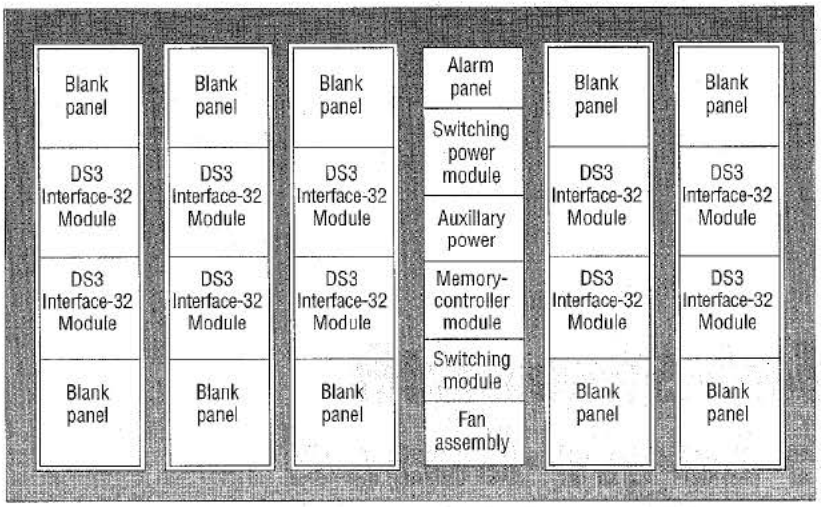

Figure 1. The hierarchical structure of a DACSIV-2000 cross-connect system.

With this in mind, we developed a prototype configurator for the FT Series G fiber-optic transmission system. The FT Series G configurator was never released, but it was effective enough that a software development team was assigned to the project in early 1990. The project, now called Prose (*prod*uct *o*fferings *e*xpertise), delivered its first configuration to AT&T's Merrimac Valley Works in Andover, Massachusetts, later that year. We assisted our software development collaborators in eventually developing and deploying 18 Prose configurators, and they have played an important supporting role in re-engineering the EF&I process.

Early lessons. Over the years, we learned many lessons. Two early discoveries were particularly important to our success. First, realizing that product knowledge was scattered across different organizations, we decided to merge the product knowledge into a knowledge base that was both consistent and commonly accessible.

Second, given the dynamic nature of complex telecommunications products, we quickly realized that product knowledge maintenance would be an ongoing concern. Also, the key products from a business standpoint—new products and top sellers—change the most. During the 10 years we worked on configurators, we observed change rates as high as 40 to 50% per year for some products (as measured by turnover and changes in configuration rules and constraints). Consequently, knowledge acquisition and maintenance were high on our list of problems to solve, and drove much of the work we did on technical problems.

Many people involved in the EF&I process need to understand the product knowledge underlying a specific order—the whys and why nots of a particular product configuration. This need overrode many other issues for us. Given this, one member of our team developed some novel approaches to explanation and truth maintenance in description logics.[2,3]

The knowledge problem. In real-world applications, success seldom comes from focusing solely on technical issues. In fact, to solve knowledge acquisition and maintenance problems for the Prose configurators, we had to consider the context in which knowledge originated and was distributed throughout the EF&I business process. This context imposed specific requirements on our configuration platform.

We selected description logics as a basis for our work because the field has several important features for addressing configuration problems, including:

- object-oriented modeling;
- rule representation, organization, and triggering;
- active inference and knowledge completion;
- explanation, product training, and help-desk support;
- ability to handle incrementally evolving specifications;
- extensible schemas;
- reasoning mechanisms that handle incomplete or ambiguous information;
- inconsistency detection, error handling, and retraction; and
- modularity.

These needs are common in many complicated deductive tasks, but we believe there were particularly critical in our configurator applications. In particular, the fact that knowledge was distributed and used by different people in different geographical locations produced some special challenges.

We are not alone in using description logics for configuration. Ford Motor Company, for example, has an automotive configurator[4] based on another description logic. To our knowledge, we have developed the first commercial description logic-based configurator, as well as the longest lived and arguably most significant description logic-based configurator environment. The sidebar, "Description logics" offer further information on the method itself. Following, we offer a simple example and use it as a context to explore each of the description logic features above.

DACS IV-2000

For efficiency, telecommunications networks bundle individual phone calls into composite signals known as DS1 and DS3 signals. These are standards in the US domestic market. Just as individual phone calls are switched by switching machines, higher level signals are also switched, in a manner of speaking: software in a cross-connect system controls the signals and can reroute them in case of network congestion, outage, and so on.

As an example here, we chose our DACS IV-2000 cross-connect system. Although simplified for our discussion, the example covers a representation and reasoning structure that is isomorphic to our deployed configurators.

Figure 1 shows the basic physical structure of the DACS IV-2000. Essentially, it is composed of equipment bays, each of which has shelves for electronic assemblies, which in turn have slots for circuit-packs (not shown in Figure 1). The equipment is modular and thus engineers can add new functionality and capacity to existing systems simply by adding circuit packs.

A DACS IV-2000 has a single switch bay and up to eight interface bays. The interface bays connect to the outside world (incoming and outgoing signals) and the switch bay "switches" the signals under software control.

A more detailed description of the DACS IV-2000 configuration is available.[5] We also have built a configurator demo for stereo equipment[6] and have recently ported it to the Web. For most people, this is a more acces-

sible domain than telecommunications, yet is reasoning isomorphic to that of the deployed telecommunications configurators. The demo is available at *http://taylor.cs.vassar.edu/stereo-demo/*.

Description logics and the DACS system

We now have more than eight years' experience with deployed configurators based on description logics. Most of the telecommunications equipment we encounter has several levels of hierarchical nesting: circuit packs, assemblies, and bays or frameworks for the assemblies; individual frameworks are themselves composed into higher level entities, fiber-optic links, wireless cell sites, and so on. The hierarchical structure inherent in most telecommunications equipment lends itself naturally to object-oriented methodology and, in particular, to declarative techniques. Thus, in producing a configuration, we are not so much interested in how a piece of equipment works, as we are in its structure: the components and their interrelationships.

Object-oriented modeling. Configuration applications usually have some sort of structured domain. Our DACS IV-2000 application has a knowledge base that includes a concept taxonomy and instance descriptions. Figure 2 shows a piece of the concept taxonomy. At the top is a general concept, DACS-IV-THING, with subconcepts DACS-IV-BAY, DACS-IV-SHELF, and DS3-EQUIPMENT. At the bottom are DS3-EQUIPMENT, DS3-32-SHELF, and COMBO-BAY. Subconcepts are related to parent concepts by an "isa" relation (denoted by dark arrows), which means, for example, that anything that is a DS3-32-SHELF is also DACS-IV-SHELF.

Concepts are structured descriptions and can have many encoded restrictions. Lower level concepts inherit restrictions from higher level concepts, but contain additional restrictions used to distinguish them from their parents. Hence, lower level concepts are more specific and higher level concepts are more general. The inheritance hierarchy encodes restrictions at the most abstract level possible. Like most description logics, CLASSIC supports strict inheritance, and thus restrictions represent true statements about a concept and all its subconcepts. The most general concept in Figure 2, DACS-IV-THING, contains information that is common to all of DACS-IV-THING's subconcepts. Such information might include knowledge about price—that a price exists, is non-negative, has a certain upper bound, and so on. More specific information that distinguishes component classes like DS3-32-SHELF from DS3-EQUIPMENT is associated with more specialized concepts, such as specific price ranges.

The COMBO-BAY concept illustrates some of this structure. It has roles for each of the four shelves in the bay, and for additional equipment such as cabling. Each role has:

- a *number* restriction, which restricts the number of "fillers" than the role can contain; and
- a *value* restriction, which limits the fillers to a certain "type" and is involved in propagation (more on this later).

Instances, often called individuals, are similar to objects in object-oriented languages. In Figure 2, they appear as circles and their labels have numbers attached. Each individual has a unique identity. Individuals can inherit from concepts but not other individuals. We distinguish instance inheritance

Description logics

Description logics are a subfield of knowledge representation, started by people attempting to formalize what it meant to represent and reason with descriptions. Previously, the preferred modeling notation had been a graphical notation that used structured objects connected by labeled arcs, such as those used in semantic networks.[1] Such a notation easily represents inheritance. However, many applications need reasoning that goes beyond simple inheritance. This motivated serious investigation into the formal theory behind the graphical conceptual model.[2] What researchers discovered was that the graphs were open to many possible interpretations. Thus, a precise semantics for the representation language was needed.

Beginning with the KL-ONE[3] system, description logics—also called terminological logics, structured inheritance networks, and KL-ONE-like systems—required a precise syntax and semantics for the representation language. Thus, they provide a compelling natural modeling language and a powerful logical deductive engine that has attracted both theoreticians and the applied community.

The theoretical community has extensively studied description logics and produced a store of knowledge about their computational complexity and inferential power at many levels of expressivity. Applications developers like description logic because it works for a range of representation and reasoning tasks and is predictable in terms of completeness and termination.

Description logics automatically "classify" new object descriptions with respect to previously defined descriptions. Thus, a knowledge engineer does not have to draw all the links in a graph, but instead can have a system automatically deduce implied links. For example, if one description (*stereo*) required at least two fillers for a specified role (*speaker*) and a new description (*2AmericanSpeakerSystem*) was entered that contained two American-made speakers as fillers for the speaker role, then *2AmericanSpeakerSystem* would be classified as a description at least as specific as the *stereo* description. The *stereo* description would then "subsume" the more specific *2AmericanSpeakerSystem* because every instance of the subsumed concept must be an instance of the subsuming concept.

Description logics typically look for information that is implicit in the stated information. For example, if a system is told that *A* and *B* fill a role, then there is an implicit statement that there are at least two fillers for the role. Description logic based systems will typically distinguish between "told" and "derived" information and only allow modifications to the told information.

Description logics are distinguished from many reasoning systems by their use of *open-world reasoning* instead of the closed-world reasoning common in databases. Description logics do not assume that information that is not known is false. For example, in the previous situation, a description logic system will infer that the role has at least two fillers but it will not assume that the role has at most two fillers just because only two are known at the moment.

References

1. J. Sowa, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann, San Francisco, 1991.
2. R.J. Brachman, "What ISA Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," *Computer*, Vol. 16, No. 10, 1983, pp. 30-36.
3. R.J. Brachman and J.G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, Vol. 9, No. 2, Apr.-June 1985, pp. 171-216.

from concept inheritance in our diagrams by using different arrows. Roles define relations between individuals. When one individual is related to another we include an appropriately labeled line in our diagrams—in Figure 2, the Bay#9 and Shelf#19 individuals are related by the thirdshelf relation.

One reason that we organize the knowledge base in this fashion is that it simplifies summarization. Because CLASSIC recognizes the class membership of all its instances, we can retrieve and count all the instances of DS1-ASSEMBLY for summary reports and the like after configuration is completed.

We created the DACS IV-2000 knowledge base working with a domain expert. In the beginning, the database of instance information, such as particular circuit packs and their specifications, was hand-compiled. Later in the project, we developed a domain-specific translator for our product experts to use. The translator was based on a notation that product experts and technical consultants used to express configuration rules and constraints. The notation they were using was informal and imprecise, but after we sharpened it, it became the main vehicle for knowledge acquisition.

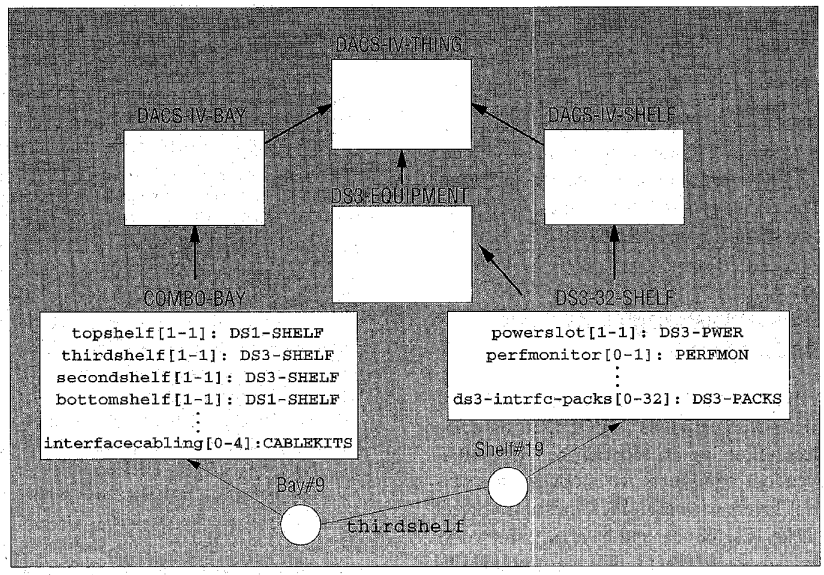

Figure 2. Concept hierarchy for a small portion of the DACS IV-2000 knowledge base.

Rule representation, organization, and triggering. Typically, a knowledge base contains class rules and definitions. It's common to see constraints that apply across different structural components of a configuration. In our domain, for example, software has to match hardware. Thus, certain hardware selected in a DACS IV-2000 configuration, such as circuit packs for processing DS3 signals, might require a specific software release (sometimes called a "generic").

Figure 3 shows a simple diagram sequence that illustrates how we use rules to implement such constraints. The top-level node in Figure 3a (BAY-LINEUP) represents a concept definition with two roles: software and interface-bay. In addition, the interfacebay role has a value restriction—uppershelf[0-1]. In other words, any instance playing the role of interface bay in the DACS-IV lineup cannot have more than one uppershelf. BAY-LINEUP has one subconcept, BAY-LINEUP-G2, which has an associated rule (indicated by the white arrow). Individuals have a unique identity and are numbered (Lineup#12 and Bay#27).

Description logics are said to have active inference because they deduce the logical consequences of given information. Once a description logic system recognizes that an object is an instance of the rule's left-hand side, the rule is triggered and asserts a new fact on the object. For example, suppose we add the information that the upper shelf of Bay#27 is filled with a DS3-SHELF (see Figure 3b). The filler could be any of type of DS3-SHELF. When this happens, the knowledge base recognizes that Lineup#12 is an instance of BAY-LINEUP-G2 (Figure 3c), because the preconditions for it being a BAY-LINEUP-G2 have been met. Because a rule is associated with this new concept (indicated by white arrow), the knowledge base also asserts that Lineup#12 is a BAY-LINEUP-G2 (Figure 3d). Consequently, Lineup#12 acquires several new parent concepts in this process. These new concepts may provide more information about Lineup#12, and the cycle of recognition and assertion can be repeated in a chaining fashion until all possible information is derived.[7]

In the telecommunications domain, rules fall into two classes: hard and fast rules (such as auxiliary power supplies are required when capacity exceeds a given amount) and rules of thumb (such as one performance monitoring circuit pack per bay is usually—but not always—sufficient). Because CLASSIC does not allow default reasoning and exceptions, all products configured by the knowledge base must abide by the hard and fast rules; hence, the rules of thumb are universally enforced just like other rules. CLASSIC does not support explicit defaults. If a default representation were available, we would have implemented our rules of thumb as defaults.

In some cases, the Prose platform implemented a form of defaults via its user interface. This method is compatible with *input completion*, one of the simplest proposals for handling defaults in description logics. If the user does not specify certain input values, the system "completes" the input with preselected default values. In another application, we implemented the default information as a "guided system". Users could then chose to follow the rules of thumb by asking for a guided system or they could over ride this information by deleting the guidance request.

Active inference and knowledge completion. After the interface guides the user through a few simple questions, the underlying description logic provides active inference. For example, users answer questions about the desired system capacity, as well as their preferences on a few high-level features, such as performance monitoring. With these inputs, the configuration application performs a search using the CLASSIC knowledge base to determine a range of solutions that satisfy these inputs. It also determines if follow-up questions need to be asked. The configurator then generates a complete (abstract) product description.

For example, imagine that the user chooses a capacity in terms of DS1 and DS3 signals, but does not specify a particular level of quality. CLASSIC deduces the minimum number of circuit packs of each type that will meet these requirements, then adds shelves, bays, and cabling required to feed that equipment. A description logic-based system might also present several abstract solutions. When the user selects one, the system might then ask for further input—such as service quality information—before completing the configuration.

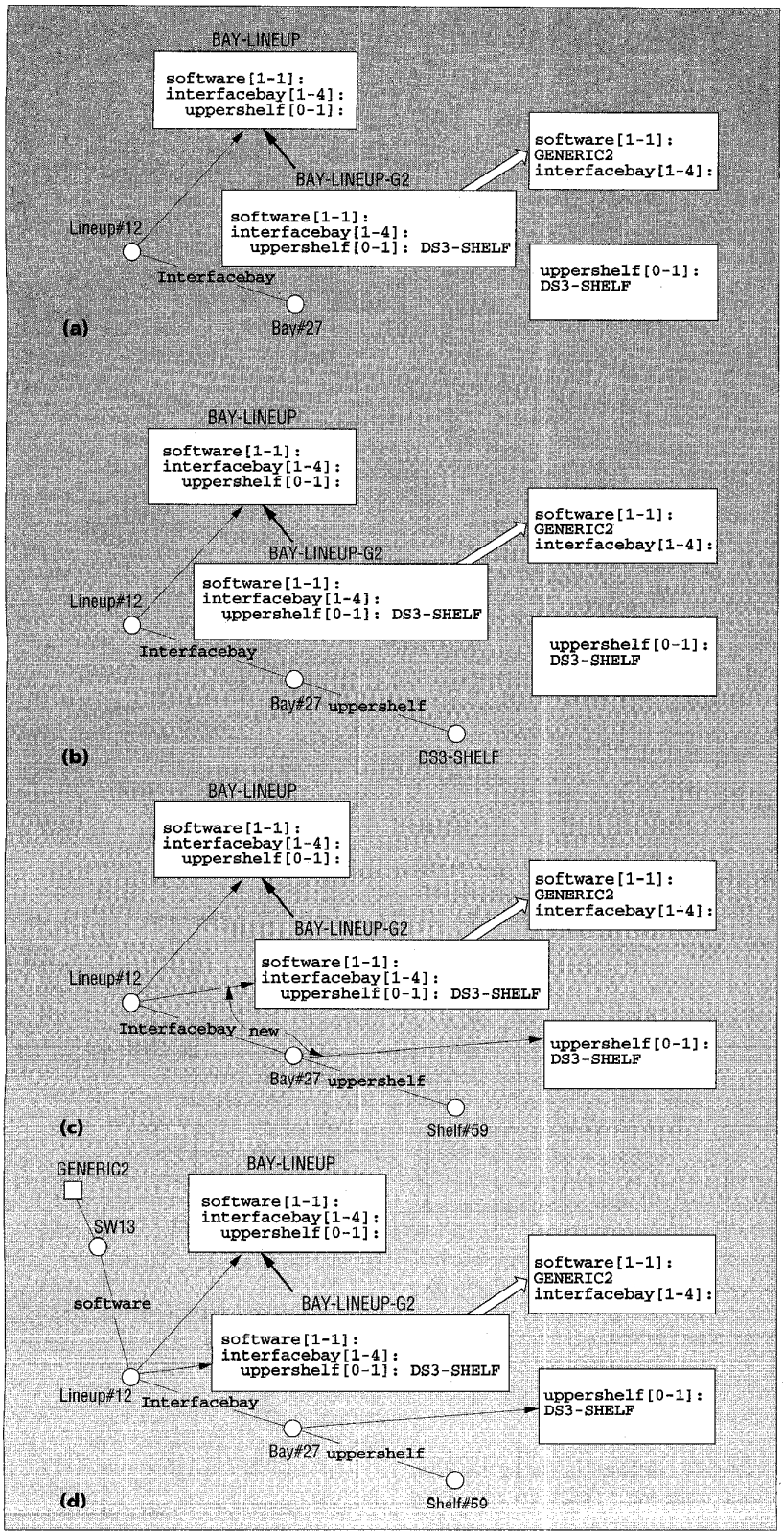

Figure 3. The rule triggering process: (a) individuals Lineup#12 and Bay#27 are created to hold the configuration state; (b) the upper shelf of Bay#27 is asserted as a DS3-SHELF; (c) the new information about Bay#27's upper shelf causes Bay#27 to be reclassified as a BAY-LINEUP-G2; and (d) a rule fires that derives the software requirement for Lineup#12.

CLASSIC calculates the logical implications (or *deductive closure*) of the information users provide. The user can view the completed information on any component or the entire system by clicking an icon or other interface element.

One way in which description logics achieve knowledge completion is to propagate information from one instance to another via value restrictions. Figure 4 illustrates how this works. In this example, Shelf#53 and Shelf #61 are both instances of DS3-SHELF (Figure 4a). Bay#27 is a DS3-BAY that has inherited value restrictions for the upper-shelf and lower-shelf roles such that any filler of the upper-shelf role must be a DS3-32-SHELF (that is, it has a capacity of 32 DS3 signals) and any filler of the lower-shelf role must be a DS3-16-SHELF. When Shelf#53 is added to the upper-shelf role, the value restriction associated with that role is propagated to that individual: it becomes a DS3-32-SHELF (Figure 4b). If anything about Shelf#53 conflicts with this new information, an error condition is raised, and Shelf#53 is prevented from filling the upper-shelf role.

Explanation, product training, and help-desk support. CLASSIC can justify all of its beliefs.[2] Not only can users view any piece of information; they can also have any deduction explained.

In our first example, if the user asks why GENERIC2 software became part of the configuration, she learns that adding a DS3-SHELF to the lineup caused a rule to fire—which, in effect, required GENERIC2 software. Similarly, the propagation in shown in Figure 4 can be supplied if a user wonders why Shelf#53 is a DS3-32-SHELF. The explanation facility can answer other questions such as why one object does or does not subsume another object, why a rule was asserted, or why an error occurred.

Inferences can also have associated templates that present explanations in terms that are acceptable to the user. For example, in some of our systems, we present answers in English by associating natural language templates with explanation options in pop-up menus. We could also use forms-based templates compatible with a help system that presents information in a familiar form.

Explanations and inspection functions let product engineers and help-desk personnel view product information in one source, which could also be made publicly available. The ability to justify, explain, and generally

DOCKET ALARM

Explore Litigation Insights

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

Real-Time Litigation Alerts

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

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

Advanced Docket Research

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

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

Analytics At Your Fingertips

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

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

API

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

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

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

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.