The Design and Implementation of PartNet

Robert Mecklenburg
Don Brown
Michael J. Clark
Dave Crandall
Sean Dalby
Andrew Hwang
Ben Spigle

UUCS-95-024

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112 USA

October 10, 1995

Abstract

PartNet is a federated database for providing interactive online access to mechanical parts catalogs. The data contained in the vendor's product database is exported to the federated database using a network-based distributed database protocol. A Single coherent view of these vendor databases is provided by a Query Server which clients access to pose queries and receive answers. The client interface programs are simple and can be executed on current desktop computers. The system is scaleable to thousands of vendors and tens of thousands of customers. We feel this approach provides better service at less cost than traditional paper or CD ROM catalogs.

1. Introduction

PartNet is a project to provide direct, interactive online access to mechanical parts catalogs. This access relies on the Internet network [1] to provide an efficient communications medium for transferring parts information from vendors to customers. This approach has many advantages over both traditional paper catalogs and CD ROM-based methods [2]. Both paper and CD ROM provide a more traditional "batch oriented'' style of access to parts data. Normal manufacturing and production delays mean that customers cannot rely on this information to be entirely up to date or complete (because of space limitations). Due to the discrete nature of catalogs and ROM disks it is not possible to search all catalogs simultaneously (without the number of disk drives equal to the number of catalogs). It may also not be possible to acquire all catalogs (even from a single catalog distributor) due to shipping or publishing constraints (e.g., a new vendor has been added to the catalog suite, but the customer cannot get the catalog until the next product release). The PartNet system overcomes these problems by providing immediate access to all vendors simultaneously. All information a vendor is willing to distribute is available including images and animation. When new vendors join the PartNet catalog or when an existing vendor adds new products, their information is immediately available to customers through the distributed PartNet software system.

The design of PartNet focuses on a small number of important issues. First, the architecture must be scalable to thousands of vendors and tens of thousands of customers. It should be possible to start with an initial installation of a single vendor and a few customers and grow on demand. As the subscription rate increases, the system should be dynamically configurable to handle the increased load. Second, the system should be tolerant of network failure and processing delays. Since the system relies on databases maintained by vendors at the vendor's site, the catalog information will be widely separated both geographically and logically (e.g., network "distance''). If answering a query requires all vendor systems to be operational and timely then eventually no query could ever be processed. Finally, the software comprising the system must be "portable'' in the most general sense of the word. Vendor databases are likely to run on the full spectrum of computer hardware, use a wide variety of database management system (DBMS) software, and encompass many different data formats. All of this diversity must be managed and translated into a unified format suitable for an online parts catalog.

One of the underlying assumptions of PartNet is that many vendors already have or will want to store their product information in online databases. Success in the PartNet system is then a matter of exporting this product information in a controlled fashion to the customers who request it. Although this assumption may not reflect current business practice we feel it is an obvious and economically sound choice. Vendors must already maintain inventory and manufacturing databases, many also have design databases. These can be unified by a comprehensive product database which includes traditional catalog data, availability and delivery information, images, as well as non-traditional data such as animation, CAD models, and vendor tutorials. Information stored in online systems is easier to access, maintain and deliver to end users and will reduce the cost of delivery and dissemination.

Finally, to further improve the accessibility of parts information we support the Hypertext Transfer Protocol (HTTP) used by the World Wide Web initiative [3]. This entails generating browsable hypertext views of vendor catalog data and providing this as Hypertext Markup Language (HTML) files. In this capacity centralized Query Servers will act as directories in the Web.

We begin the design discussion by reviewing the computational environment our system requires. Following that, we discuss the architecture of the PartNet system as a decomposition into cooperative services and their responsibilities. We then review auxiliary tools for accessing and maintaining PartNet. This is followed by summary of the implementation choices we have made for the initial implementation. Finally, we discuss the status of the project, results and suggest directions for the future of the system.

1.1 The Environment

PartNet is intended to perform in a dynamic and heterogeneous computing environment. The Internet is a loose collection of thousands of computers from every computer vendor running almost every operating system. The common thread these systems share is their physical connection to the telecommunications network and their support for the TCP/IP networking protocols [1]. Vendors are distributed geographically as well has having the normal variety of computer hardware and software. Customer's computer systems are even more diverse since they may not even support multitasking.

The core PartNet software requires a large address space, multiprocessing, high performance mass storage, and fast network access. We have targeted our prototype system to run on the average Unix workstation with 32Mb of RAM. We assume that both the interface software at the vendor's site and the Query Servers will run on such machines, although we have no bias against other operating systems with similar capabilities (e.g., VMS, Windows/NT).

The customer interface, however, is assumed to run on much simpler hardware and software (e.g., PC with Windows). In fact, the customer's computer may not even have direct access to the Internet. This is accommodated by providing a Query Server on a gateway machine which is connected both to the Internet and to the customer's internal network.

Finally, we assume that the Internet network provides modest bandwidth capabilities consistent with current performance. Peak transmission of data using the optimized File Transfer Protocol (FTP) can achieve data rates of 110 kilobytes per second with average transfer rates between 20 and 40 kilobytes per second. This suggests that network bandwidth is of concern and the architecture should address this issue.

2. PartNet Architecture

PartNet is designed as a federated database management system [4] specialized for read-only access. Each vendor site presents a database of parts information available for access by customers. These databases are managed by the vendor using various (possibly proprietary) database management systems. Our goal is to allow controlled export of this vendor maintained information to the customer on demand. Furthermore, it is important that the mechanism for delivering data be as distributed as possible to allow scaling up to many vendors. The process structure of PartNet is depicted in **Error! Reference source not found.**.

Customers typically interact with the system through a graphical interface which connects to a centralized Query Server. This Query Server receives queries about parts which are then routed to vendors who supply those parts. Each vendor site provides one or more Vendor Database Interface (VDI) processes which execute the query and return the answer to the Query Server. The Query Server in turn forwards the data to the original requesting customer. A single Query Server will handle several hundred simultaneous customers. As the load increases Query Servers are replicated.

This process architecture addresses several basic issues inherent to distributed databases. First, the diversity of vendor database software is managed by a single coherent interface exported by the Vendor Database Interface (VDI). Each VDI is responsible for mapping from vendor specific formats into canonical PartNet format. This translation includes DBMS query language, part attribute and value conversion. The Query Server provides a centralized process for routing queries to vendors, managing global information, and caching vendor data to reduce latency and improve throughput. The existence of the Query Server process also dramatically reduces the NxM connectivity problem inherent with allowing customers to talk directly with vendors. Finally, the customer-side user interface software is kept simple to allow execution on low-performance, low-capability hardware.

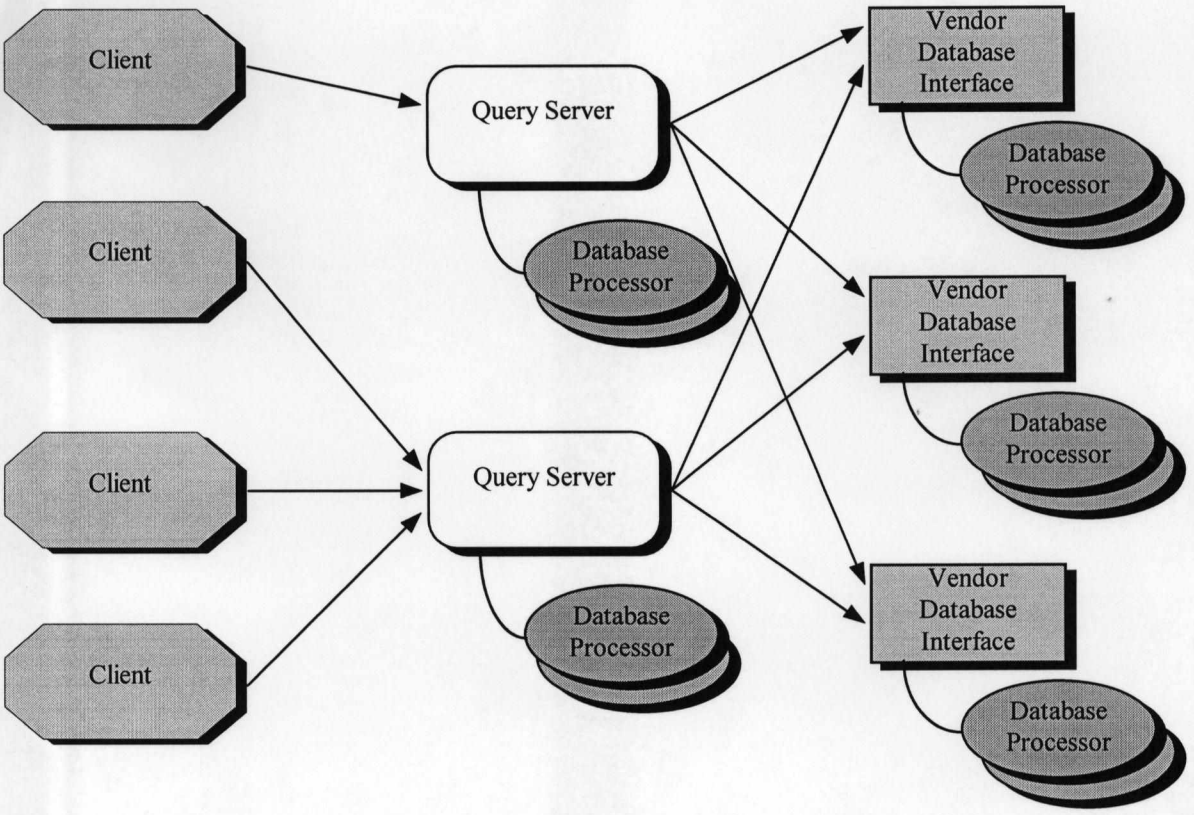

Figure 1: The PartNet process architecture.

Communication between processes occurs via a message-based command and response protocol. This allows a simple, portable implementation which is as efficient as Remote Procedure Call systems for average messages, but without the added implementation complexity.

2.1 The Query Server

The Query Server is the "glue'' which binds PartNet together. It provides a centralized service which can be accessed through either a well-known network address or by name from an Internet name server. Since it is centralized it forms a locus for routing information, global data management, and performance monitoring. We assume greater computational power at a Query Server host which can be used to reduce network traffic and latency through caching which may not be possible at the customer site (due to fewer computing resources).

The major responsibilities of the Query Server are:

- manage a set of customers and vendors,
- route messages from customers and vendors,
- control access to global data (e.g., table of contents and part lists),
- cache data,
- log transactions.

As the central router for messages the Query Server must ensure that each customer is serviced fairly and that no customer process is "orphaned'' or "mislead''. In particular, answers to customer queries are delivered to the customer incrementally as each vendor supplies their portion of the answer. It is important that the customer not mistake a partial answer for a complete one.

For each query submitted by a customer, the Query Server determines which vendor is capable of supplying an answer and forwards the query to that set of capable vendors. This dramatically reduces network traffic when compared with forwarding every query to every vendor. To properly determine the capable vendors, the Query Server must know all parts supplied by each vendor and it must update this information as it changes.

Other information the Query Server manages is global data such as the "table of contents" and parts list. The table of contents simply records the names of all parts available in the PartNet system in a hierarchical format suitable for browsing. Both the table of contents and parts list are global in that they unify all the information supplied by all vendors. This requires that each vendor report the parts they supply and a taxonomy of how these parts relate to the rest of PartNet.

To improve throughput and reduce latency, the Query Server caches the answers to queries. When a query is received from a customer, the cache is first scanned for other queries referencing the part requested in the current query. If any are found, the cached query is analyzed to determine if its answer describes a superset of the current query's answer. If this is true the current query can be answered directly from the cache without the overhead of forwarding the query to vendors. This feature will become more useful as the size of PartNet grows. We anticipate users presenting naive queries which return many parts, then refining the query to reduce the size of the answer. Only actual use of PartNet can determine if this pattern of usage, and consequently the utility of caching, is realized.

Using a cache leads to the problem of cache consistency and invalidation. In short, a problem occurs when a vendor updates part information while the Query Server has copied information about that part. In this case, the customer may be given out of date information about a part. This problem is solved by requiring the VDI to inform the query server whenever parts information is changed. To reduce the burden on the VDI and reduce network traffic, the Query Server associates a lifetime with each answer. When the lifetime has expired the answer is removed from the cache. Removing parts information from the cache in a timely manner also prevents the cache from growing without bound. The lifetime should be long enough to allow reasonable performance gains while short enough to minimize the load on the network and VDI.

Finally, the Query Server is responsible for monitoring the performance of the PartNet system as a whole. This includes ensuring that vendors and customers are not "orphaned'', recording timing statistics on network latency and bandwidth, recording quantity of information delivered by each vendor to each customer (e.g., for billing purposes), and recording general usage patterns. Due to faults in networks and software it may be possible for customers or vendors to become unreachable. This should be noted and should not cause other components (e.g., Query Server or customer interface) to fail. Also, by recording network performance statistics, the Query Server can improve the user interface by anticipating delays.

2.2 The Vendor Database Interface

PartNet does not impose a particular DBMS or database management paradigm on participating vendors. This is important since vendors may have invested enormous time and expense in building existing databases. Furthermore, the vendor may even have a proprietary database management system tailored to their specific data. Any attempt to replace this database or impose some standardized format will result in vendors who are unable or unwilling to participate in PartNet.

To avoid excluding vendors by requiring a standard database and query language, PartNet provides an interface process which responds to the PartNet communications protocol and implements database queries through native calls to the vendor database. This interface process manages:

- network communication,
- table of contents and names,
- concurrency, and
- caching.

We discuss each of these responsibilities in turn.

The first responsibility of the VDI is to manage network communication. Even if a vendor database directly accepted the PartNet command language, additional software would be required to identify the available Query Servers and manage network connections. When a VDI is initiated it identifies a Query Server through either a well known port and address or using the Internet name service. After establishing a connection to this Query Server it requests a complete list of active Query Servers with which it should register. By registering with a Query Server the vendor signals its readiness to receive and process queries. VDIs are able to accept connections from new Query Servers as they are added to the network and manage communication from Database Processors (see Figure 1) which are spawned to perform actual database queries.

Once a vendor has registered with a Query Server, it transmits a table of contents describing part categories the vendor supplies and their relationship in the taxonomy of known mechanical parts. The Query Server receives this table of contents and matches its entries with those in the canonical table of contents. If the Query Server is presented with an unknown part category, the part category may be tentatively accepted. This allows new parts categories to be added by vendors to the PartNet system pending review.

The VDI presents the table of contents and detailed part category descriptions of a part categories and characteristics using unique part and attribute ids. This allows provides a mechanism for mapping from a vendor's naming scheme into the PartNet canonical scheme. For instance, vendors may represent a part number in their database as "PN'', "part_number'', "part_no'', etc. Since the names of mechanical parts and their attributes are not standardized the PartNet system adopts a single canonical name for each category and attribute and assigns a unique identifier (an integer) to each. This identitifier is used by all PartNet software to refer to a part or attribute, while the canonical name is presented to the customer by the user interface. Only the VDI needs to know the translation of part ids specific names.

Since a VDI will be connected to several Query Servers which are in turn connected to many customers, a vendor database may be asked to answer several different queries in a short space of time. Ideally we would like to answer all queries immediately with response time related only to the delay imposed by the vendor's own DBMS. Unfortunately, there may be an arbitrary number of simultaneous queries limited only by the total number of customers. Also, many databases and operating systems are limited in the amount of concurrency a single program can achieve. For instance, a single program executing a database query might be required to wait until that query is processed by the DBMS and the answer returned before being allowed to initiate another query. This is overcome in the PartNet design by creating several database processors (see Figure 1) which execute queries synchronously, but in parallel with each other (i.e., process-level parallelism). These processors are discussed in more detail in Sections 2.3 and 4.2. The VDI process serializes all commands, but since each command can be handled very quickly (i.e., by forwarding the command to a Query Server or a database processor) no command is forced to wait an undue amount of time for processing.

The final responsibility of the VDI is to aid in Query Server cache management. To improve throughput and reduce latency the Query Server caches answers to customer queries. The details of this caching are discussed in Section 2.1. It is essential that a customer is never given out of date information because the

cache is inconsistent with the vendor's actual data. This is known as the problem of cache consistency and consequently, cache invalidation. To aid in maintaining a consistent cache, the VDI must monitor answers to queries it receives as long as the data is held in a Query Server cache. If this data ever changes, the VDI must notify the appropriate Query Server that the original data is now invalid.

2.3 The Database Processor

The Database Processor is a slave process of the VDI and Query Server which performs actual database queries using the native DBMS interface. The purpose of this separate process is to overcome the singly threaded nature of many operating systems and DBMS interfaces. While a single Database Processor may perform one query at a time waiting for DBMS to process the query and return an answer, a collection of Database Processors can handle multiple queries in parallel. These processes are managed by a single scheduler which maintains a queue of pending queries and a suite of available server processes. Queries are scheduled on idle processors and query answers are delivered using the standard PartNet protocol. It should be emphasized that this does not require a multiprocessor to execute. It is merely a mechanism to achieve process-level parallelism in a singly threaded DBMS or operating system.

Although this does not achieve the ideal goal of the fastest query processing possible (which would require a cpu per query), it does provide a reasonable mechanism to maximize throughput and tune query processing. A simple algorithm would allocate a fixed number of Database Processors as determined by past query loads. A more sophisticated algorithm could dynamically adapt to query loads by spawning additional Database Processors as query arrival rate and system load dictate.

2.4 User Interface

Although user interface specifics are a matter of implementation choices, certain features of the problem so profoundly affect the user interface that it is reasonable to provide architectural support for them. One such feature is the vast amount of parts information to which PartNet will provide access. Navigating through the unified parts catalog which PartNet provides without imposing some overall structure on the data may be difficult. To address this problem, we have devised a hierarchical taxonomy for identifying categories of parts which is presented to the user as a table of contents.

The table of contents is developed by consensus as each vendor determines the sub-categories which differentiate one type of part from another. The development of this taxonomy will be moderated by PartNet to ensure that no conflicts or inconsistencies occur. The table of contents will form the initial interface to PartNet which users will navigate.

Due to the distributed nature of PartNet the user interface should have several added features. Since a query is executed in a distributed fashion, users are able to view the partial results of queries. The interface should clearly indicate that an answer is not complete and what percentage of vendors have responded. It is also possible for a customer to execute several queries simultaneously, so a display of pending queries and their status is useful.

PartNet is designed to deliver multimedia information as well as traditional text. This implies a windowing interface for full functionality. Nevertheless, it is reasonable to provide text-only or even "batch mode" interfaces for users with fewer computational resources. An interactive text-only interface could provide support for simultaneous queries and viewing partial results, while a batch interface can deliver queries and answers through electronic mail.

2.5 Query Servers as Directories

Query Servers will add another critical link to the information distribution infrastructure by serving as information navigators. Navigation is a significant problem with the World-Wide Web (WWW). The Web has an arbitrary, cyclic graph structure in which it is easy to become "lost". Also, there is no systematic mechanism for information providers to make their products known to users of the Web. Query Servers provide a means for users to perform value-based searches of the Web thereby help them quickly find the information they need.

Query Servers will perform the function of service advertisers. When a new Vendor Server comes "online", it will contact its assigned set of Query Servers and inform them of the products it provides. Interfaces to the Web such as Mosaic and Netscape will provide a link to a Query Server where designers can browse available products. The Query Server also holds the table of contents and parts lists which serve as a "yellow pages" of part types for clients to browse. The Query Servers could also serve as directories for non-catalog services on the net.

As a special service to vendors participating in PartNet a Query Server can be asked to perform characteristic search on a specific vendor database or set of databases. By creating a hypertext link from a vendor's "home page" to a PartNet Query Server and requesting single vendor search PartNet provides a vendor with a search engine especially for their own data. At the same time their data is available through the PartNet federation.

3. Names, Queries, and Tolerance

A significant problem in federated databases is resolving conflicts in the shared data model. These conflicts manifest themselves through inconsistent data in response to queries. Three problems discussed here are maintaining a consistent table of contents and parts list, translating queries, and support for diverse units systems and tolerances.

3.1 Semantic Consistency of Global Information

The table of contents and parts list have been discussed briefly in previous sections; here we discuss some difficult semantic interpretation problems these tables create. Parts categories in PartNet are identified by unique id. A part category has associated with it a collection of attributes, types and units for these attributes and annotations describing them. Unfortunately, even with this level of detail describing a part category, there is the possibility that a purely software driven solution may incorrectly identify one part category as another. As a simple example, suppose one vendor registers a gear with the Query Server whose attributes consist of a diameter and a number of teeth. For this set of attributes it is possible that some other vendor has already registered a different type of gear with the same name and the same attributes. Adding more attributes only reduces the probability of this collision, the probability will never be zero. The same problem occurs when locating part categories in the table of contents.

The goal of PartNet is not to provide a universal parts identification system, rather it is to provide a unified parts catalog. The difference lies in the degree to which we require a completely unambiguous parts description language. Our approach to this problem is to involve vendors in the parts identification and differentiation process. When vendors add part categories to the PartNet system they must ensure that their category and attribute names unambiguously identify their product. This does not seem unreasonable since the vendor is already involved in entering product information into their own computer systems.

3.2 Query Translating

When merging part attributes from many vendors, it is likely that some vendor will supply an attribute not supplied by any other vendor. If the user poses a query containing this attribute how should the system respond? There are three possible choices:

- reject the query,
- send the query only to vendors supplying all attributes,
- send the query to all vendors supplying the part category and omit the attribute for vendors which don't supply it.

Given that we anticipate many vendors supplying the same part category, the first option leads to an unusable system. The second and third choices can be restated as "for those vendors not supplying the attribute, assume the attribute is never matched'' and "for those vendors not supplying the attribute, assume the attribute is always matched''. Since there are queries for which each of these choices are reasonable, the choice must be selectable by the user (possibly with a default).

3.3 Units and Tolerances

When handling attributes with units, it is important to be able to convert freely between units systems. Furthermore, a conversion from one system to another will typically yield values which no longer compare "properly'' due to floating point rounding errors. We address this problem by implementing a combined units and tolerancing facility which allows users to indicate the units applicable to a particular query, whether conversion to other units systems should be performed, and the tolerance on those conversions if performed. This facility is configured with reasonable defaults which allow most users to ignore these details if absolute control over the query is not required.

4. PartNet Core Implementation

Here we describe the basic approach used to develop the PartNet software. In general, we will concentrate on the software framework and avoid the exact details of implementation. We will discuss specific algorithms only where the solution is either novel or of particular interest. PartNet is written in C++ as a tool kit for managing parts information in a distributed environment. Since it is a tool kit we leverage off of existing code to a surprising degree. For instance, less than 10% of the PartNet source code is specific to any one component program. The PartNet framework focuses on a small collection of objects which manage the fundamental aspects of networking, database management, command and control, and describing parts and their attributes. We will discuss the interesting features of each of the objects in turn.

4.1 Networking

Network communication is mediated by a single network object in each PartNet process. This object maintains a list of all network connections, is capable of establishing new connections and controls the extent to which network failures are visible to the PartNet process. The main event loop executed by the various PartNet servers is mediated by the network object which identifies network partners requiring service and the order in which they are serviced.

One of the problems in scaling PartNet is the limitation on the number of simultaneous open network connections a single process can maintain. In the prototype, network connections are established between every VDI and every Query Server. This limits the total scale of the system to about 100 vendors and 40,000 customers. This limitation can be overcome in several ways. One is to partition the PartNet network by product. A user looking for gears, transmissions, or motors would connect to one Query Server while a user needing fasteners or connectors would reference a different Query Server. This routing can be

handled transparently to the user by PartNet itself. Another technique for handling greater network connectivity is to provide network access through network server processes which multiplex I/O to the main Query Server. These techniques can all be incorporated within the network object with little or no impact on the remaining PartNet software.

4.2 Database Management

This object provides a "virtual database manager" interface to vendor specific database software. The object provides functionality sufficient to manipulate the database without relying on the details of the DBMS software. The database object is used for three purposes: database control, query language support, and cache management. The database control features are straightforward. They allow the controlling PartNet process to open and close the database and determine data characteristics such as record layout and data types. Query language support is much more subtle.

The PartNet system employs its own query language syntax which is a simplified form of SQL [5]. As users compose queries in the graphical interface the software derives an SQL-like textual representation which is passed as a command to the Query Server which, in turn, forwards the query to vendors able to respond. The database object supports a query action which transforms the PartNet query parse tree into a valid DBMS-specific query and executes it. This translation process will be different for different vendor databases but will provide the same interface to the PartNet software.

Likewise, the answer to a query will be a collection of part descriptions consisting of <attribute,value> pairs describing the part. These answers are collected into a list of row objects with each item corresponding to a row or record in the vendor database. Thus, the database object forms an interface to the DBMS control functions, the query language, and the data model. Actual DBMS specific functions are performed by classes derived from the database object.

4.3 Command and Control

PartNet communicates between its various processes by passing command objects. These objects are derived from a common root class which provides the framework for their behavior. Basic command characteristics include the network partner who issued the command, the time the command was issued, and a unique transaction identifier. All commands respond to an `execute` function which performs the action appropriate for the issuing process.

Command objects can be communicated between processes in a variety of ways. The prototype implements a textual translation of objects into a simple command language grammar. This is convenient for prototyping since the Query Server and VDI can be manipulated interactively through the telnet application. The textual transmission protocol can be replaced with a more compact form by changing two functions. Note that if the textual command is less than the size of a network packet there will be little or no performance gains from sending binary data.

Another reason for maintaining a simple textual interface is to allow PartNet to be an open system. The current interface is suitable for use by a wide variety of applications due to its simple LL(1) grammar.

4.4 The Table of Contents and Parts List

The table of contents and parts list objects form the more visible components to the user. A table of contents is a singly rooted, acyclic directed graph (DAG) which describes the universe of mechanical parts by decomposing part categories into differentiating features. The parts list is much more subtle. A vendor part is described by a part category (or *PartCat*) object in PartNet. The PartCat represents the "platonic ideal" for a part in that it associates with a part identifier the union of all attributes (which are also given

identifiers) provided by vendors. These attributes are given non-conflicting names and annotations along with implementation data such as data type and units (if any). The VDI is responsible for mapping from these PartNet identifiers back to the vendor specific nomenclature.

Note that the set of attributes describing a single part category is related to the set of vendors participating in PartNet at any instant. If a vendor goes off-line (for instance, to perform backups) and that vendor is the only one supplying a particular attribute for the part category, the attribute should be disabled from the set of queryable part attributes although it is not removed from the PartCat itself.

This same variability applies to the table of contentts. As vendors join and leave the PartNet system the table of contents is immediately updated to reflect the currently queryable parts.

4.5 Transaction Management

Finally, we have a transaction management object which records commands and their responses. The basic task of the transaction manager is to record which network partner sent a particular command. Since a Query Server manages many "conversations'' concurrently, when it receives a response command it must route that response to the process which initiated the query. This can be done by identifying each command with a unique identifier and recording the query identifier in the response. If a response command is received with an "originator'' command identifier the original command is found and the sending process identified. The response command is then forwarded to the original process. If a simple command is received, one that does not require a response or forwarding, it is executed, logged, and discarded. Commands representing an answer with part information are handled by the cache manager.

When part information is received by the Query Server, the data is immediately forwarded to the appropriate customer. Furthermore, the data is then logged in the transaction table and written to a local cache in the form of a local DBMS accessed through the database object. When further queries about that particular part are received by the Query Server, the transaction table is scanned for queries whose answers can be proven to form a superset of the current query. If any answers are found the Query Server then answers the customer query directly from its cache. Since answers are received independently from each vendor and cached for a fixed amount of time, some answers will have been purged from the cache while other answers are still resident. In this case, the transaction table will contain information about which vendors responded to a particular query and whether their data is still resident. The cache manager can then answer a query directly from the cache for that data which has not been purged and can resend the query to those vendors for which the data has been purged.

Another use of the transaction table is to identify "dead'' or "orphaned'' network partners. Since every command sent to a customer or vendor is recorded in the table until a response is received all outstanding commands are in the table. A time stamp is associated with each so that a scan of the transaction table can quickly determine exactly how long a customer or vendor has been waiting for a response. Appropriate action (probably determined by the user interface) can then be taken.

Finally, as commands are purged from the transaction table, they are logged to allow off-line analysis of customer usage patterns, network load, demographic information, and charge back information.

4.6 The User Interface

We have discussed a wide variety of user interfaces which we can support. Among these are graphical window-based applications, interactive, but text-based applications, batch oriented electronic mail interfaces and hypertext-based applications. Such a diverse set of user interfaces is possible because of the simple text command format and the process layering architecture around which PartNet is built.

The graphical interface for PartNet, the interactive text interface, and the e-mail interface are all relatively simple. They access the Query Server using the same command language/object set used by the backend processors. For the e-mail interface we assume that users formulate queries with our simplified SQL query language. In this case, the receiver strips the mail headers and forwards the query to the Query Server as usual where it is reconstituted into a query object. The interactive interfaces should include support for executing multiple queries simultaneously and managing partial queries.

The other interface discussed is that of the World Wide Web. Here we have developed a Common Gateway Interface (CGI) process which handles WWW requests. When the CGI process is invoked it maps the received uniform resource locator (URL) into PartNet SQL and sends it to the Query Server. The CGI interface then collects the responses, formats them into an HTML document, and returns the document to the requester. The Web software can then access vendor catalog data directly through hypertext links. Maintaining a dual representation is avoided by formatting responses on-the-fly. The forms interface to the Web is defined by the CGI process.

5. Current Status

The current status of PartNet includes a beta-test version of the Query Server, Vendor Database Interface, and Database Processor. The system runs on a distributed network of 6 Sun workstations and totals 90,000 lines of C++. As mentioned in Section 4 most of this code is reused in each of the PartNet processes. The prototype is currently simulating five vendors using two different DBMSs to manage the vendor data. It is also able to manage non-textual data such as images and CAD data files. As part of the prototype we have simulated several vendors with databases currently holding over 2,000 spur gears. A custom Motif user interface and a World Wide Web interface are both available.

6. Conclusions and Future Work

The PartNet system implements a federated database for accessing mechanical parts catalogs from a large set of vendors in real time using the Internet as the communications medium. This system has distinct advantages over traditional paper catalogs or CD ROM systems. PartNet queries are performed by each vendor in parallel providing superior response times. Answers are complete and accurate at the time of query since all updates to vendor data is performed by the vendor at the time the data changes. Images, sound, animation, and CAD models can be delivered as easily as text. The system is scalable to thousands of vendors and tens of thousands of users.

The problems of naming conflicts in federated databases are solved pragmatically by relying on communication between vendors and coordinated by PartNet. Units conversion is performed where necessary and is configurable by users. Several user interfaces are available including custom Motif and World Wide Web interfaces.

Future work includes improving the handling of large numbers of vendors and users, better management of query processing delays, and support for e lectronic commerce.

References

[1] Comer, D. E. (1991). *Internetworking with TCP/IP*, volume 1. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition.

[2] Williams, M. A. (1993). Highlights of the online/CD-ROM database industry: Opportunities through technology. In *Proceedings of the 14th National Online Meeting*, pages 1–4.

[3] Berners-Lee, T., Cailliau, R., Groff, J., and Pollerman, B. (1992). World-Wide Web: The information universe. *Electronic Networking: Research Applications and Policy*, 2(1):52–58.

[4] Ullman, J. D. (1988). *Principles of Database and Knowledgebase Sysetms*, volume 1. Computer Science Press, Rockville, Maryland.

[5] Chamberlin, D. D. et al. (1976). SEQUEL 2: A unified approach to data definition, manipulation, and control. IBM J. Research and Development, 20(6):560–575.