91

SAND--98-8505C

PRE: A framework for enterprise integration.

CONF-980540--

R. A. Whiteside
Sandia National Laboratories
MS 9012, P.O. Box 969 Livermore, CA, 94551, USA
(510) 294-3565, FAX: (510) 294-1230, raw@ca.sandia.gov

E. J. Friedman-Hill
Sandia National Laboratories
MS 9214, P.O. Box 969 Livermore, CA, 94551, USA
(510) 294-2154 FAX: (510) 294-2234, ejfried@ca.sandia.gov

R. J. Detry
Sandia National Laboratories
MS 0807, PO Box 5800, Albuquerque, NM
(505) 844-7722, FAX: (505) 844-2067, rdetry@sandia.gov

19980507 090

RECEIVED
MAR 27 1998
OSTI

Abstract

Sandia National Laboratories' Product Realization Environment (PRE) is a lightweight, CORBA-based framework for the integration of a broad variety of applications. These applications are "wrapped" for use in the PRE framework as reusable components. For example, some of the PRE components currently available include: our product data management (PDM) system, our human resources database, several finite element analysis programs, and a variety of image and document format converters. PRE enables the development of end-user applications (as Java applets, for example) that use these components as building blocks. To aid such development, the PreLib library (available in both C++ and Java) permits both wrapping and using these components without knowledge of either CORBA or the security mechanisms used.

Keywords
CORBA, integration, framework

DTIC QUALITY INSPECTED 2

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

1 INTRODUCTION

Like many organizations, Sandia National Laboratories has a variety of distributed electronic resources. The most apparent or these are databases: financial, human resources, product, *etc*. However, there are other important electronic resources: engineering design advisors, finite element simulators, data format converters, manufacturing devices, *etc*.). Sandia's Product Realization Environment (PRE) is a lightweight, CORBA-based framework for the integration of this broad variety of electronic resources.

Figure 1 illustrates the general idea. Each of these applications) is "wrapped" to plug into an enterprise-wide wiring harness (illustrated by the grid). This wiring harness and associated "wrapper" let the application be used as an enterprise component. Using this wiring harness, a PRE client can, for example:

- Put data into the component.
- Get data out of the component.
- Request that some computation be performed.

Furthermore, these components might use services provided by each other. For example, structural analysis component might retrieve the model to be analyzed out of the PDM (Product Data Management) component. It might then put the analysis results back into the PDM for archival purposes.

An end-user interacts with the PRE system through some kind of user interface (a Java applet on a Web page for instance) which accomplishes its tasks by utilizing these wrapped applications as components. Such a user interface might be very general, giving access to all of the components in the system. More likely, however, its use is more special-purpose: utilizing a few components to do specific task via a simple interface. Note that the operation might itself be quite complicated, requiring co-ordination of several of these enterprise components. Using PRE, however, we can present a simple user interface to this task.

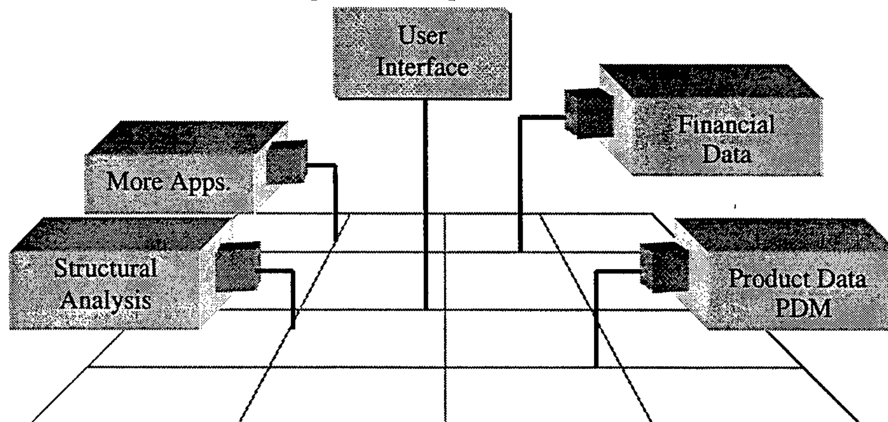

Figure 1. Application integration with PRE

The PRE framework is based upon the Common Object Request Broker Architecture (CORBA), which provides industry standards for writing plug-and-play software. These standards are promulgated by the Object Management Group (OMG), a consortium of hardware and software vendors. In implementing PRE, we have used the CORBA-compliant Orbix product from Iona Technologies.

Thus, the CORBA foundation on which PRE is implemented is a *component* technology. The CORBA standard specifies how to create and use a software component:

- How to specify its interface (CORBA IDL).
- How to discover and name the (possibly remote) component.
- How to make a request of the component (down to the order of the bits on the wire).

Using CORBA, a developer can create a very wide variety of these software components. The PRE *framework*, built atop the CORBA *component* technology, specifies which interfaces a developer *should* implement. That is, it specifies things like "If you are an application, then this is exactly the interface you must implement." A client programmer can then easily use your application component, since all applications provide the same interface. Many ideas in PRE borrow from the Discus project at Mitre (Mowbray, 1995). Further, it captures experience from our previous work in enterprise integration (Friedman-Hill, 1996), and in agile manufacturing (Whiteside, 1996).

2 PRE ARCHITECTURE

The major components of the PRE architecture are illustrated in Figure 2. This captures the same concepts illustrated in Figure 1, but exposes some of the structure in the communications grid of that figure. In the upper right of Figure 2 are the applications that are "wrapped" to run in PRE. These are used by the user interfaces to provide end-user functionality (illustrated in the upper left). These end-user interfaces employ the wrapped applications and the PRE Core services (the lower part of the figure) to deliver functionality.

The major pieces of the PRE architecture called out in that figure include the data transport, the integrated applications, user interfaces, the trading service, the conversion broker, and security.

2.1 Data objects

CORBA and the PRE data object provide the basic data transport mechanism in PRE. All applications integrated into PRE accept their inputs, and emit their outputs as PRE data objects. A data object is a container for structured information, and is essentially an associative array of named properties. Methods are provided for getting and setting values of properties. A property can be a single number or string, or it can be a multi-megabyte data file. Based on this simple structure, Data objects can be used to represent simple documents, C-like struct objects, tabular information, etc. A MIME string is used to denote the type of a

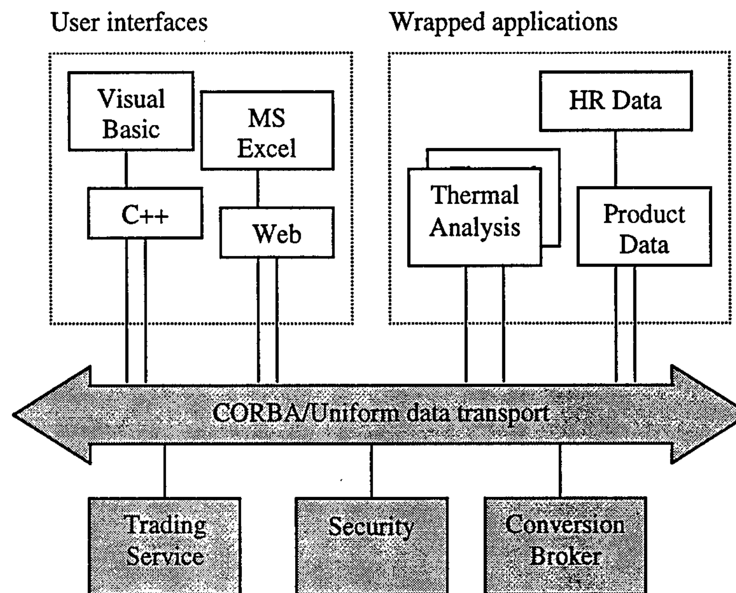

Figure 2. Major elements of the PRE architecture

PRE data object. Using MIME types to describe content has a number of advantages:

- MIME is an internet standard for describing types of things. Web browsers already use these types for figuring out how to render data, thus our data object integrates nicely with the World Wide Web.
- The browser's database for MIME types already provides a mapping of MIME types to applications that end users have already had to deal with once. Our desktop GUI can use this to decrease installation complexity for deployment (since we'll need the same kind of information).
- The MIME format permits electronic mail to include enhanced text, graphics, audio, and other types, in a standardized and interoperable manner. Future plans include the transmission of PRE data objects *via* multimedia mail.

2.2 Wrapped applications.

All of the real functionality in PRE (the useful computation) is provided by the various applications wrapped and integrated into the framework. Application "wrappers" integrate these into the PRE framework, making these applications useable as programming components. User interface developers then use these components to accomplish tasks desired by the end-users.

All applications integrated into PRE support the same interface, consisting of six methods. Thus, the first tasks of an "app-wrapper" is to understand the functionality of the program being integrated into PRE, and to decide how this

functionality should be mapped onto these six PRE application methods. Here are the six methods in the PRE application interface:

`convert()`. If part of the functionality of the application can be viewed as a type conversion, then it should support the convert method. PRE applications as data converters have turned out to be a very extensively used aspect of PRE. A Web page, "Conversion Central" has seen substantial use at Sandia, mostly for conversions among various image formats (*i.e.*, TIFF to GIF, CAD format to VRML, many formats to PDF, ...).

Another use of the convert method has been in wrapping legacy applications. Many of our old engineering and design FORTRAN programs, for example, can be usefully viewed as conversions from an input deck into output results.

`query()`. The `query` method is implemented by applications with data that can be queried. Certainly wrapped databases fit into this category, but other applications might be `query`-able as well.

`get_data()`. Servers that can emit data on demand implement this operation. For example, a word processor application integrated into PRE might implement `get_data` to permit retrieval of the current document. Our PRE interface to Sandia's Product Data Management (PDM) system provides this method as a mechanism for retrieval of documents from the PDM.

`put_data()`. This is the inverse of `get_data`: if a client might want to "put" some in data into your application, implement the `put_data` method. For example, our PRE interface to the PDM implements this to enable clients to add new documents to the repository.

`execute()`. This operation is for scriptable aspects of an application. Thus, applications with a script language should implement `execute` to give client programs access to this functionality. Further, the signature of the `execute` method is very general, so that any kind of application functionality that does not fit cleanly into one of the other categories can be provided through this method.

`launch()`. If an application is being used, for example, to perform a data format conversion, then it is not particularly desirable to have the user interface (UI) associated with that application be displayed to the end user. Indeed, it would probably be distracting. In some cases, however, displaying this UI to the end user is the whole point of the operation. The `launch` method us used by a client program to explicitly state that the application in required to present its user interface to the end user.

2.3 PRE Trading service

The PRE Trading service provides a "yellow pages" registry of the component applications available in PRE. Although there is now an official OMG interface to this type of service, our design of the PRE trader predates this, and we do not conform to the standard. Future versions of PRE will address this incompatibility.

Applications integrated into the PRE environment register with the PRE Trader so that they can be discovered and used by others. These "others" might be client

programs that will discover and use the component automatically, or human developers who can use the component to accomplish some task.

The kinds of things registered at the PRE trader include general information about the application, as well as details of operations supported by the application (`convert`, `query`, *etc.*). The kinds of general information available for each application include:

- Name: A one-word description of the component. Suitable for display, for example, on a button.
- Summary: A one-line description of the component. Suitable for display, for example, in a status line.
- HelpURL: The location of the on-line help for the component.
- Email-contact: An address to which to send gripes (or kudos!)

The other information registered at the PRE Trader about each application consists of the list of methods implemented by the component, and the data types appropriate to each. For example, the Conversion Broker (described in the next section) uses this information to use conversion services dynamically added to the PRE environment.

2.4 Conversion broker

A client that wishes to use a PRE component to convert a data object from one type to another (from some CAD format into VRML, for example) should take the following steps:

- Query the Trading service to see if any PRE component has advertised such a conversion capability.
- If an appropriate component is available, call its `convert()` method to effect the conversion.

The PRE Conversion Broker simplifies this task by handling the trader query automatically. You can ask the Conversion Broker to perform any conversion. It doesn't actually implement any conversions itself; it merely queries the trader on your behalf, and uses the resulting PRE application to do the conversion. The PRE Conversion Broker provides a simple interface for converting data from one type into another.

The Conversion Broker provides another bit of value-added as well. Suppose that one PRE application has advertised its ability to convert from type A into type B, and another PRE application advertises the conversion from type B into type C. In this case, a query to the Trader for a conversion from type A into type C will yield no hits: no application has advertised such a service. However, the conversion is, in fact, possible by converting first into the intermediate type B. The Conversion Broker is able to carry out such sequences automatically if no direct conversion is available. Merely call the `convert()` method on the Conversion Broker, and the required sequence of conversions will be carried out.

2.5 Security

The PRE security architecture supports a number of different security models, selected by plug-able library modules. Our basic approach has been to secure the PRE framework, not to construct a security mechanism that can be used by CORBA-based systems in general. The approach can be summarized as follows:

- Define and implement methods that can be used by the initiator and acceptor to negotiate a mutually acceptable security model and establish a security context if necessary.
- Pass a security token, the contents of which will depend upon the security model to each method in the framework.
- Perform the necessary security operations (authentication, authorization, delegation, data protection).
- Handle as much of the security as possible within PreLib development libraries.
- Allow the application owner to control access to his application and to perform authorization external to the framework if so desired.

The PRE security system provides for the use of a number of different security models. How the security system functions is dependent upon the security model that is agreed upon by the initiator (client) and the acceptor (server). The system provides a mechanism for the client and server to negotiate a compatible and agreeable security model, although in most cases the environment will dictate the security model.

The client can request that a particular security model be used. This allows the client to request the use of a model other than the default model of the server. This will most often be used to allow the client to request a higher level of security than that usually used by the server. The server, however, has the final choice about which security model is to be used.

The security models that are supported in the latest release of PRE are shown below:

Model Name	*Description*
SEC_MODEL_NONE	essentially turns off the security system
SEC_MODEL_NAME	uses unauthenticated username and hostname, but exercises most of the functionality of the security system, including logging
SEC_MODEL_DCE	uses the DCE implementation of GSSAPI to perform authentication, authorization, and delegation

The three security models reflect the phases in the development of the PRE security system. The first model, SEC_MODEL_NONE, was created to allow

applications to exist without security. It enabled the development and initial implementation of the security methods without requiring any code changes on behalf of application developers. This model can still be used to turn off security.

The second model, SEC_MODEL_NAME, provides an intermediate step between no security and full security. When using this model, a context is established between the client and the server, although the context is not secure. The security token contains information about the context, from which the client's username and hostname can be determined. The client's information is not authenticated and cannot be relied upon to enforce serious security requirements. However, this model can be useful in a stand-alone environment or when security really isn't an issue. Furthermore, the extensive logging performed when using this model can provide some insight into the operation of the system.

Based upon DCE, SEC_MODEL_DCE offers a full range of security services including authentication, authorization, and delegation. Encryption is not currently enabled, although that is a capability of the system. On many networks, using a wire-encryption mechanism such as SSL could better perform encryption. If it turns out that encryption inside the framework is necessary, it can be enabled.

Java servers and applets

To support the use of Java servers and applications with DCE, the necessary hooks into the security system have been implemented using the Java Native Interface. This allows Java servers and applications to utilize the security services and inter-operate fully with C++ clients and servers.

To support the use of applets, a secure proxy has been developed that works in conjunction with DFS Web Secure from IBM to allow applets to access a DCE server. When a user requests access to the applet, DFS Web Secure authenticates the user and stores the user's DCE credentials securely on the server. Information about the credentials is passed to the applet via a parameter. The credential information is then added to the security token and passed to the secure proxy. The proxy can then use the user's credentials to establish a secure context with a DCE-based server and securely invoke its methods.

3 IMPLEMENTATION ISSUES

PRE consists of five facets or layers:

- The PRE CORBA interface definitions in the OMG Interface Definition Language (IDL).
- Official implementations of some of these services.
- The Framework Developer's Kit (FDK) Libraries (PreLib)
- User-developed server applications integrated into PRE
- User-developed interactive client applications.

Each layer refines the concepts developed by the layer below it. This structure has made it possible for the functionality offered by PRE to evolve, sometimes

dramatically, without invalidating end-user code. For example, the CORBA IDL definitions have changed as PRE has evolved, but because application developer's code is targeted toward PRE's libraries and not directly to the CORBA layer, this code did required no modification. Furthermore, the layered structure has made user code relatively insensitive to changes to (and bugs in!) third-party software at lower layers. In the following sections, we will examine the contributions of these layers to PRE, stating at the lowest level.

3.1 PRE CORBA interfaces

PRE's CORBA interfaces are quite simple - they comprise a total of eight interfaces, fourteen global types, and sixteen exception types. The interfaces are designed so that they can be easily used as building blocks for more complex structures. Full details of these interfaces are available in html format in the PRE distribution and at www-collab.ca.sandia.gov/pre/fdk/idl. The html was automatically generated by idldoc (Friedman-Hill, 1997).

The PRE Interfaces

Common: Ancestor to all other interfaces, Common contains operations for basic security, retrieving version and meta-information, and initializing and destroying objects.

Data: As described above, PRE data objects provide the basic data transport between components in PRE, and use the MIME standard to describe content. The MIME standard also allows for utilization of local formats that are not internet standards. Such local type or sub-type names must begin with "x-". This will certainly be useful in holding input data for various applications (for example, `application/x-antipasto`, for input to the antipasto program). PRE also define several new main types. These include `x-struct` for C-like structures and `x-table` for tabular data.

Observer: A simple object, similar to Data but capable of holding only one property. The Observer interface is used for asynchronous notification: the single property is used as a representation of the status of some operation.

App: The fundamental actor in PRE; all services will export this interface as a means of offering their functionality. This interface adds to **Common** the set of standard App methods (`convert`, `query`, *etc.*). Each of these operations also exists in an asynchronous form that accepts an Observer as a third argument. The App interface serves as ancestor to all the remaining interfaces.

Trader: Serves as a database of meta-information about a PRE installation. The PRE Trader contains descriptions of the capabilities and locations of installed Apps, and details about the various x-struct and x-table MIME types that have been defined.

AppFactory: Creates App objects. Because AppFactory inherits from App, it can implement `query()` to respond to status requests, for example.

DataFactory: Creates Data objects. Again, can implement App methods: for example, `query()` to locate existing Data, `put_data()` to move Data objects

between factories, convert() to alter existing Data objects. The DataFactory uses the Trader to learn how to construct Data objects of various MIME types.

ObserverFactory: Creates Observers. This interface allows processes to initiate asynchronous processes and quit, because such a process can obtain an Observer object from an ObserverFactory; this Observer object can then monitor the status of the asynchronous process.

3.2 Official implementations

Several of the PRE interfaces are represented solely by privileged implementations that, together with PreLib, form the PRE runtime system. These include official implementations of the DataFactory, the Trader, the File Manager, and the ConversionBroker. Additionally, there is a standard ObserverFactory service, which some of the other core services use in their implementations. In this section we discuss some implementation issues and experience related to these services.

Our implementations of these interfaces are all in C++ and are available on Sun/Solaris, SGI/IRIX, HP/HPUX , Intel/Windows 95, and Intel/Windows NT. These are installed on the Unix boxes using our own installation system called "AppPkg". On the Windows platforms we provide self-installing executables build with the InstallShield product.

The Data Factory.

The standard Data Factory offers Data object persistence (the presence of which is implied in the IDL) and several optimized data transfer modes (which are not specified in IDL.) Furthermore, it itself is implemented as a library, so that end-user applications can contain an embedded Data Factory for efficiency reasons.

The standard Data Factory optionally offers simple object persistence. The implementation is completely hidden from users of the service, and indeed, persistence has been implemented in several different ways. Our initial implementation was in terms of a simple relational database, and used the popular shareware product Mini SQL (see http://www.hughes.com.au). Production usage demonstrated that installing a Data Factory that depended on the presence of an external RDBMS was problematic. Thus, the current implementation uses flat file data storage, offering increased performance and simplified installation.

The standard PRE Data Factory also offers a mechanism to transfer a set of properties in a single network transaction. Recall that data object may contain many properties, and the PRE IDL a remote procedure call for the retrieval of each. In the case of array or tabular data where hundreds of individual properties must be retrieved, this leads to severe inefficiencies of the Data Factory library. it is used only by the PreLib layer, and are invisible to the applications programmer. It should be noted that this optimized transfer mode is not part of the public interface

The File Manager.

Another form of optimized data transfer is the handling of single Data properties that are very large. Some of our engineering applications, for example, have multi-

megabyte inputs and outputs. The PRE File Manager is an implementation of the App interface that manages a flat file space of these large files. When a Data property is set to contain a large file, the property in the Data Factory actually will contain only a pointer to a file owned by a File Manager service. The decision to store a file in the File Manager is made by PreLib, and is invisible to the end user. The File Manager can either use CORBA to transfer the data, or it can use a separate authenticated TCP connection (Pyarali, 1996); while the former is more convenient when firewalls are in use, the latter is more efficient.

A further optimization is possible if the programs exchanging data happen to be located on the same machine. In this case the File Manager can allow end-user applications to get direct access to the files in the File Manager's store. Again, the decisions about when this is feasible are made by PreLib, so that two communicating applications do not need to know when they are collocated; the applications can be written in a location-independent way, and get the benefit of optimization when it is possible.

The Trader.

The PRE Trader is a simple interface to an underlying RDBMS. Again, details of the implementation are hidden, and several alternatives have been tried. An initial implementation, using Mini SQL, resulted in some of the same problems observed for the Data Factory. Currently our Trader is implemented using an in-house embedded RDBMS library, offering simplified installation and maintenance.

PRE Apps and MIME types are described in a small language named 'TREG' (Trader REGistration). A TREG description of an App or type is presented to the Trader via the `do_register()` operation. This information is then available for querying via the Trader's `query()` method.

The Observer Factory.

The standard PRE Observer Factory service is a simple implementation of the CORBA ObserverFactory interface that adds Observer persistence. Persistence is implemented via the standard Data Factory library. The observers implemented by this factory merely store the status information given to them; end-user applications can retrieve it at a later time.

3.3 The PRE FDK

The PreLib libraries insulate the applications programmer from the details of PRE's CORBA-based implementation. There are two parts to PreLib: the client libraries and the server libraries. Client libraries are used to communicate with other PRE entities; the server libraries greatly simplify the act of writing a PRE server application.

PreLib is available in a number of languages. The full library is available in both Java and C++, while the client-side library is also available in Perl and Visual Basic.

Client libraries.
In PRE, every operation must be authenticated via the security system. This important function can be difficult to accomplish correctly and consistently. Furthermore, even if done right, these security details can often obfuscate the actual purpose of a method call. Therefore, PRE's security functionality is handled for the programmer by PreLib.

There is a PreLib class (in C++ and in Java) for each of PRE's CORBA interfaces. For example, the PreData class represents Data objects, and offers methods to set and get properties. The PreData class also handles, for example, the dispatching of large Data properties to the File Manager service, or the use of multiple-property transfer modes. Similarly, there are PreApp, PreTrader, and PreDataFactory classes for communicating with the corresponding PRE entities.

Additionally, PreLib offers a number of convenience classes. For example, a PreConverter class allows the programmer to easily use the Conversion Broker application. A PreTable class provides an interface to tabular data which is implemented entirely in terms of the underlying Data interface. Classes named PreLocalData and PreLocalTable provide the programmer with a simple interface to the multiple-property transfer modes of the Data Factory.

Each method call to a remote object made via client-side PreLib hides a great deal of complexity from the programmer, including security, manipulation of CORBA data types, use of CORBA functionality to locate remote services, *etc.* As a result, our users' code has remained insulated from out underlying implementation of PRE.

Server libraries.
The PRE server-side libraries provide the same kind of simplification for server programmers as the client-side libraries do. The details of the security system, and of CORBA itself, are entirely hidden. Writing an application is reduced to the task of inheriting from a simple PreLib class (in C++ or in Java) and implementing only the methods that are of interest. This class is then manipulated by the real CORBA stub classes, provided by PreLib. This arrangement allows for not only the security to be handled by PreLib, but also exception propagation, argument conversion, etc. In addition, by default, the asynchronous variant of the core App methods are handled by all servers with no effort on the application programmer's part: notification is handled by PreLib.

3.4 Server applications

As developers use PreLib to users integrate more applications into the framework, the functionality available in PRE grows. The current set of applications integrated into PRE is rather diverse. Illustrating the range of applications type, these apps include:

- Databases: Access and Mini SQL.
- Thermal, structural, and optical analysis packages
- Our Product Data Management system

Also available is a range of data format conversion applications. Applications that convert among image formats have proven to be very popular PRE components.

PRE servers can be written in either C++ or Java, and generally arise in the following way. A programmer wishes to meet some functional requirements of an end user. He finds several components already available in PRE that are useful in providing this functionality, but some pieces are missing. He wraps a new application for use in PRE, because integrating it with existing PRE components is the fastest way to get the job done. The result is not only that the initial problem is solved, but also the newly wrapped application is available for future developers.

3.5 End-user functionality

The final piece of PRE is a client program that provides some kind useful capability to an end user. There is a larger set of implementation languages available to client developers than to application wrappers. Though C++ and Java are certainly available, client authors can also use perl (widely used by cgi-bin programmers) and Visual Basic: it's particularly fun to watch a PRE application run from an Excel spreadsheet. However, because of the ubiquity of Web browsers, most of the current PRE clients are implemented as either Java applets or as Web pages backed up by cgi-bin programs.

4 CONCLUDING REMARKS

PRE is deployed, production software. In addition to its use here at Sandia National Laboratories, PRE is installed and in use with several other projects. As part of a Cooperative Research and Development Agreement (CRADA) with Goodyear Tire, Inc, PRE is being used to help produce an integrated tread wear design environment. At Georgia Tech, PRE is in use as a testbed for the National Electronics Manufacturing Initiative (NEMI) plug and play framework definition project. In another CRADA (EUVL: Extreme Ultra Violet Lithography), PRE is being used to help develop a design environment for next-generation semiconductor manufacturing.

5 REFERENCES

Friedman-Hill, E.J. and O'Connor, J.W. (1996) The XCELL Integrated Product Realization Environment: a Toolbox for the Agile Construction of Custom Parametric Design Software, Proceedings of the Fifth National Agility Conference, Boston, MA,

Friedman-Hill, E.J. and Whiteside, R.A (1997) idldoc: Automatic documentation for CORBA IDL, Dr. Dobb's Journal, June 1997, 46.

Mowbray, T.J. and Zahavi, R (1995) The Essential CORBA, John Wiley & Sons, New York.

Pancerella, C.M and Whiteside, R.A. (1996) Using CORBA to Integrate Manufacturing Cells to a Virtual Enterprise, *SPIE Proceedings of Integrated Manufacturing -- Plug and Play Software for Agile Manufacturing*, November 1996, Boston, MA.

Pyarali, I., Harrison, T. and Schmidt, D.C (1996) Design and Performance of an Object-Oriented Framework for High Speed Medical Imaging, Computing Systems, **9**, 331-375.

Whiteside, R.A., Pancerella, C.M. and Klevgard, P.M. (1997) A CORBA-Based Manufacturing Environment, Proceedings of Hawaii International Conference on Systems Sciences, January 1997, Maui, HI.

6 BIOGRAPHY

Dr. Whiteside is a Principal Member of Technical Staff in the Distributed Systems Research group at Sandia National Laboratories.

Dr. Friedman-Hill is a Principal Member of Technical Staff in the Distributed Systems Research group at Sandia National Laboratories.

Rich Detry is a Senior Member of Technical Staff in the Scientific Computing Systems Department at Sandia National Laboratories in Albuquerque, NM.

Report Number (14) SAND--98-8505C
CONF-980540--

Publ. Date (11) 199803
Sponsor Code (18) DOE/DP, XF
UC Category (19) UC-705, DOE/ER

DOE