Using Standard Tools to Build an Open, Client/Server Prototype

Bernard S. Hirsch

Hewlett-Packard Company
930 East Campbell Road
Richardson, Texas 75081 USA
(214) 699-4197
bernie@bsh3185.ssr.hp.com

Abstract. This paper will benefit software developers, MIS managers, and end users because it will help explain some of the practical benefits and implications of the Open Software Foundation (OSF) Distributed Computing Environment (DCE) in the context of a prototype application environment that was developed using the OSF DCE. The environment, a Financial Desktop, consists of a series of OSF Motif and MicrosoftWindows based clients which obtain information and resources transparently from a series of DCE based services, that reside on a range of heterogeneous computing hardware and software, including multivendor operating systems, networks, architectures, and databases. This paper will explain how DCE was used to implement this Financial Desktop in such a way that installed assets were leveraged, new technologies were integrated,and the focus of control for this environment has shifted away from a single hardware or software vendor. The use of the DCE remote procedure call (RPC) is discussed with respect to the role that it plays in this environment.

1 Introduction

The seemingly elusive goal, whereby the wealth of information and services in an enterprise is transparently accessible to end users on demand, is one about which much is written and discussed. Furthermore, there is an ever accelerating requirement to accomplish this in an open, distributed computing environment.

The first requirement that the computing environment be distributed is due to several ongoing trends in which an organization's business units, functions, data, users, and computing equipment are all now more and more distributed. The second requirement that the computing environment be open is resulting due to the need for these enterprises to control their own destiny. That is, they do not want to be reliant on a single hardware/software vendor when using information technology to help meet business goals. With the frantic pace at which new technologies, products, and methodologies are introduced nowadays, an enterprise would like to be in control to

incorporate these to increase its competitive advantage, while still leveraging its many existing computing investments.

This paper describes how a prototype of an open, distributed application environment for stock brokers was created in which financial information and services are transparently accessible across a range of heterogeneous computing hardware and software, including multivendor operating systems, networks, architectures, and databases. Further, it is shown how this environment, the "Financial Desktop" (or FDT), was created using standard, off-the-shelf tools and technologies including the Open Software Foundation (OSF) Motif graphical user interface and components of the OSF Distributed Computing Environment (DCE). Using this open approach it is shown how installed assets are leveraged, new technologies can be integrated, and how the focus of control for the environment has shifted away from a single hardware or software vendor.

First the FDT user interfaces and services are described in detail, and the tools and technologies that were selected to implement the prototype are presented. Next, implementation details and experiences are discussed, followed by a summary of the prototype development. Finally, conclusions are drawn on the prototype environment and related work is introduced.

2 Description of the Prototype Environment

2.1 User View: Graphical User Interfaces

The setting for the prototype is an application environment for a stock broker, that consists of a series of hardware and software components working together in unison, so that the stock broker can very efficiently and transparently get his or her job done, by having presented at their desktop all of the needed information and services. The software components that come into play in this environment are a collection of services (or servers) and user interfaces (or clients).

From the stock broker's perspective, four (4) user interfaces "drive" the entire application (see Figures 1, 2). That is, all interaction with the system is done by simply entering some information -- such as the customer account number -- and then pressing a button in the graphical user interface (GUI). All of the information that the stock broker needs to know about that customer, for example, and the status of his or her investments are then automatically and transparently presented to the stock broker in the same unified interface. The four GUIs (e.g., clients) are described below. [1]

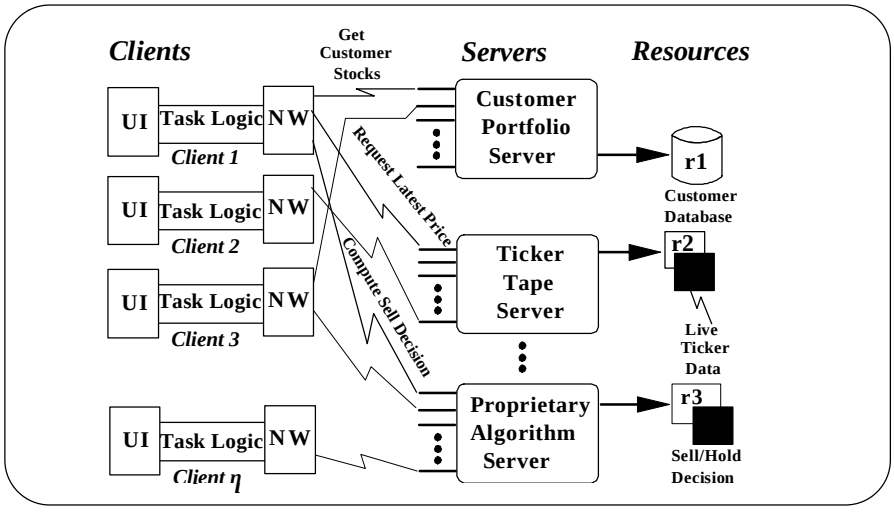

Fig. 1. FDT Client/Server Architecture

Marketminder

Using a "real-time" feed to the Dow Jones, Marketminder simultaneously presents the NYSE and NASDAQ ticker tape data (e.g., latest stock prices) and allows the stock broker to query the latest price for a particular stock. Specific information on each stock presented to the stock broker includes:

a) Stock Name.
b) Current, High, and Low stock prices for the current trading period.
c) Stock price change between the current and the previous trading periods.
d) Volume.

Financial Desktop (FDT)

FDT allows the stock broker to query a customer information database for individual customer and portfolio data. A heuristic function analyzes various data and suggests whether the time is right to sell particular customer stocks. Specific information presented to the stock broker includes:

a) Customer name, social security number, and address.
b) Stock symbol, number of shares, and purchase price for each customer stock.
c) Current trading price for each customer stock owned.
d) Sell/Hold recommendation for each stock owned.

New Customer

New Customer allows new customer data and customer stock portfolio data to be added to the customer information database.

Customer Report

Customer Report generates letters to all of the customers of the stock broker that own a particular stock. The letters advise them of pending recommendations and actions to sell particular stocks based on some activity. Additional functionality included in the "Customer Report" client allow the stock broker to either (a) look up a stock company name based on a Dow Jones stock symbol, or (b) look up Dow Jones stock symbol based on the company name.

2.2 Resource View: Backend Services

From a functionality perspective, a significant amount of processing is transparently occurring to be able to present all of this information to the stock broker within each of these four GUIs (see Figures 1, 2). A description of the various server data andoperations in this environment will help explain some of this processing. Five (5) servers provide access to all of the information and services within this prototype envirnoment. The five servers are described below. [2]

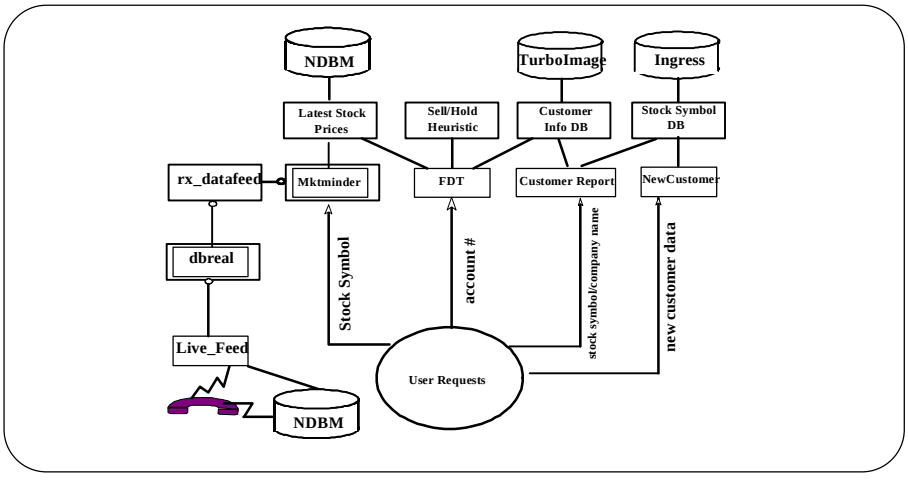

Fig. 2. FDT System Architecture

Customer Information Database service

The customer information database service is implemented as an MPE/iX TurboImage proprietary network database of customer information that manages the following data:

a) Customer account number, customer name, social security number, address, and number of stocks owned.
b) Stock symbol, number of shares, and purchase price for each customer stock owned.

Its server operations include:

i.) "Get customer data" retrieves all of the data in (a) above, given a customer ccount number.
ii.) "Get customer portfolio data" iteratively retrieves all of the stock information in (b) above for each customer stock held.
iii.) "Add new customer" adds the new customer and customer portfolio data [specified in (a) and (b) above] to the database.
iv.) Get customer owning stock" iteratively retrieves all of the customers that own a particular stock, given a specific stock symbol.

Dow Jones ticker tape service

The Dow Jones service provides a real-time feed to the latest NYSE and NASDAQ stock prices. Either the Telerate or Prodigy dialup services can be used as underlying services.

Latest Stock Price Database service

The latest stock price database service is implemented as a UNIX NDBM database that maintains the latest stock price for each of the stocks coming across the ticker tape.

Stock "sell/hold" heuristic analysis service

The "sell/hold" analysis service provides a recommendation to the stock broker as to whether to sell the customer stock or hold onto it, based on historic and current market conditions.

Stock Symbol Database service

The stock symbol database service provides a mapping from stock name to stock symbol, or from stock symbol to stock name, for each of the stocks in the NYSE and NASDAQ. This database service is implemented as a relational database using Ingres.

The following server operations are supported:

i) "Get stock symbol" retrieves the Dow Jones stock symbol, given the company name.
ii) "Get company name" retrieves the company name, given the Dow Jones stock symbol.

2.3 Hardware, Operating Systems, and underlying Network

The customer information database service is implemented using an existing TurboImage network database on a HP 3000 Series 900 business computer running the MPE/iX operating system.

The Dow Jones ticker tape service, latest stock price database service, and the "sell/hold" analysis service are all implemented on the following hardware and operating systems:

a) HP 9000 Series 700 workstations and Series 800 servers running the *HP/UX* operating system.
b) IBM RS/6000 Model 320 workstation running the *AIX* operating system.
c) DEC DecStation 3100 workstation running the *OSF/1* operating system.
d) DEC VaxStation 3100 running the *VMS* operating system.
e) DEC DecStation 5000/200 workstation running the *Ultrix* operating system.
f) Siemens-Nixdorf workstation running the *SINIX* operating system.

g) Groupe Bull workstation running the *BOS* operating system.
h) Stratus fault-tolerant minicomputer running its variant of the *UNIX System V Release 4* operating system.

The stock symbol database service is implemented using the Ingres relational database on the HP 9000/730 workstation running the HP/UX operating system.

The clients are implemented on the following platforms:

a) HP 9000 Series 700 workstations running the *HP/UX* operating system.
b) Intel 80386 PC's running Microsoft *DOS 5.0* and *Windows 3.0.*

The clients and servers are networked together using both Ethernet and IEEE 802.5 token ring, using TCP/IP and UDP/IP protocols.

4 Selected Tools and Technologies

The technologies and tools that were selected to be used to implement the Financial Desktop application environment are briefly discussed.

4.1 User Interfaces

Motif 1.1 using UIMX interface builder

It was decided that OSF Motif 1.1 was the primary GUI technology to be used to implement the client user interfaces. The familiar and intuitive appearance and behavior of an interface that complies with the Motif style guide empowers end users to be productive immediately. Further, UIMX (also called Interface Architect) from Visual Edge was selected as the interface builder for the Motif GUI, due to its ease of use, quick prototyping capability, and ability to generate pure Motif C code, so that portability to future client platforms can occur very easily. In addition, since very efficient C source code is generated, the prototype interface code can also be deployed quite appropriately in a production environment. [2]

Microsoft Windows 3.0 SDK

The Microsoft Windows 3.0 SDK was used as a secondary, supplemental GUI technology to develop MS Windows interfaces for two of the FDT clients that are to run natively on the PC platforms. Although MS Windows is not standard or open, it is the most popular GUI in use today on PCs, and thus it is strategic to many users.

4.2 Application Interoperability

DCE Remote Procedure Call (RPC)

It was decided that the OSF DCE RPC would be the enabling technology to be used to achieve application interoperability between the four FDT clients and the five FDT servers. The primary decision criteria here was the openness of DCE, as an enormously popular consortia-sponsored interoperability standard, and as such, the expectation that it will be increasingly available on almost every computing platform. For the prototype, early versions of DCE were used for x386 Windows-based PCs, Series 700 HP/UX workstations, and MPE/iX minicomputers, to achieve interoperability across the various FDT clients and servers.

Other decision criteria also affected the selection of the DCE RPC. First, the transparency, and scalability afforded the application developer and end users is much greater for applications architected with DCE RPC than with some of the less robust, pure messaging technologies. The familiar and intuitive local procedure call semantics empower software developers to be productive developing distributed applications with minimal training. Second, DCE RPC greatly simplifies the development of the clients and servers by automatically generating client and server stub (or "glue") C code. Most distributed computing complexities are taken care of automatically by the DCE RPC tools, generated stub code, and the DCE RPC library. Finally, DCE RPC can just as easily be used to implement a production worthy implementation as it can to achieve the prototype described in this paper, since this third generation technology was designed with these goals in mind. [3, 4]

5 Implementation Details and Experiences

5.1 Background

The FDT prototype was originally developed in early 1991 at HP Dallas using Network Computing System (NCS) 1.5.1 RPC and directory services on HP Domain/OS, HP/UX, and MPE/XL operating systems. NCS 1.5.1 supports a non-threaded distributed computing environment with a non-hierarchical name space with no additional security. [5] This technology was used because DCE, even in snapshot form, was not yet available from OSF. In addition, the UIMX Motif prototyper tool was used to generate the client user interfaces. Since one of the purposes of the prototype was to demonstrate heterogeneous distributed application interoperability and data access, some rudimentary visual feedback was added such that the screens of the various RPC servers "light up" green whenever an RPC request is serviced.

DCE Implementation

In July of 1991, OSF decided to use this prototype environment as the basis for its DCE demonstration to the support the September 17, 1991 worldwide availability announcement of DCE. As such, the FDT client and server code was reengineered o use DCE RPC and DCE threads, so that any DCE licensee can utilize the prototype environment for demonstration purposes, by building it with their DCE implementation. At the time of the implementation, the latest version of the DCE code was Snapshot 5, and the CDS security services were not yet mature enough to be used. The FDT code was ported to OSF/1 on the DECstation 3000 workstation and the HP PA-RISC 720 workstation, as the two primary development environments.

5.2 Utilization of DCE Core Components

RPC and Threads

The DCE remote procedure call was heavily utilized throughout the FDT environment to accomplish both distributed data access and distributed computation. Idempotent call semantics were used throughout, since read only access was he only requirement for the initial implementation. The new DCE RPC context handle feature was also utilized in the data access interfaces so that servers can maintain client state and clean up if necessary. The endpoint mapper daemon (rpcd) was not utilized since well-defined endpoints were used in the Interface Definition Language (IDL) files. This is poor implementation practice, in general, but was used here due to the infancy of the DCE software. Finally, the broadcast attribute was specified for the *send_tick()* procedure of the Dow Jones Ticker Tape Service, so that multiple MarketMinder clients/servers are instantly informed of new ticker tape activity.

Threads were not used explicitly in any client implementation except for a workaround that was required for some initializations that were not occurring in early DCE code. Threads were used implicitly by every FDT application server so that multiple clients are serviced simultaneously. It was for this same reason that context handles were also required. The manner with which multi-threaded servers are created in DCE is very straightforward and thread complexities are transparent to the developer.

Cell Directory Service Emulation

Part of the source code port from NCS to DCE included porting NCS location broker (*lb_$*) calls to the DCE NSI calls to register services into and lookup services from the Cell Directory Service (CDS). The original goal was to utilize the new automatic binding feature in DCE, whereby with no explicit

programming on the part of the application developer, clients will automatically and seamlessly connect with their appropriate servers, and if those servers should fail, the clients will then automatically reconnect with other equivalent servers.

However, since the CDS portions of DCE were not ready at that time, a C library was created which emulated CDS (actually, RPC NSI) functionality and behavior. Conditional compilation allowed use of either the native DCE CDS or the emulation library, while providing transparency to the application. In addition to emulating the automatic binding and rebinding on server failure, the library randomly picks a server to which the client will connect, if there are multiple equivalent servers available. This will, in effect, amortize the various RPC calls -- and thus the RPC load -- across the various RPC servers, creating a simple form of RPC load-balancing.

The additional step of emulating CDS required the static definition of server locations (e.g., hostnames) for each interface definition in a configuration file. This is sufficient for a small, static prototype environment, but this methodology will not scale up to a larger, more dynamic, and more realistic environment.

What worked well here, in addition to the transparency to the application source code, is that clients compiled with the emulation library will interoperate with both servers compiled with the emulation library as well as with servers compiled with the true RPC NSI interface -- in effect, creating two namespaces. The converse is not true, however.

5.3 Demonstrating DCE Interoperability

Demonstrating distributed application interoperability and data access is not an intuitive concept. If application components are interoperating correctly, user requests are processed and results are presented. However, it is not obvious where events and actions are happening. As such, a not insignificant amount of effort went into providing visual feedback as part of the FDT environment, such that one is able to discern where distributed computing is occurring.

Specifically, the lighting up of screens was expanded to display specific vendor logos on both client and server machines. To accomplish this, an additional RPC parameter was added to each function signature in the FDT IDL files to denote either from which machine an RPC is being called or where it is being serviced:

```
[idempotent] void db_stocks_open([in]       handle_t      h,
                                 [in,out]   long          *vendor_id,
                                 [out]      file_handle_t *fh);
```

The background of the server machine's screen, for example, will display a bitmap of the client's vendor logo (i.e., the HP or IBM logo) when the RPC is serviced. And the client GUI will display a bitmap of the server's vendor logo when the RPC completes. In this way, it is easy to determine between which machines an RPC is being rocessed. The user can then press a toggle button which alternates between visual feedback mode and "live" mode, in which RPCs can complete at full speed with no visual feedback.

5.4 Replicated Application Servers

FDT application servers were replicated to demonstrate the concepts of utilizing DCE for high availability, load balancing, and multiple custom server implementations in a distributed computing environment. High availability is demonstrated by unplugging the network connection from an FDT server during a series of RPC calls. Then, when the ensuing RPC is requested on that now unavailable server, the client delays slightly and then rebinds to another equivalent server, with its new server vendor logo displayed in the client GUI. In the FDT clients, the default RPC timeout was changed from about thirty-two (32) seconds to either four (4) or eight (8) seconds. Depending on which application server to which a client connects, a different server implementation is executed. For example, on a customer database lookup, either a Unix flat file is sequentially searched or an HP MPE TurboImage network database is searched.

In the prototype environment, the general replication problem is scaled down and some assumptions are made so that only application servers with no update interface can be replicated. If no assumptions are made -- which in a real application environment cannot be done -- either a replicated database or a distributed transaction processing (TP) monitor would be required for an application server managing a database. Since the former solution requires a monolithic, single database vendor implementation, the latter TP monitor solution would have been preferred, and the Transarc Encina technology would have been selected since it is already integrated with OSF DCE. In this scenario, an update is no longer a non-idempotent (i.e., at most once) operation, but instead is transactional (i.e., exactly once), in which each replicated database is updated with the scope of that single transaction. In effect, the Encina TP monitor solution is used to keep heterogeneous databases in synchronization by utilizing a two-phase commit across the various databases. [6]

5.5 Developers' Skills and Roles

Three developers were intially involved in the building of the FDT prototype. Each had different backgrounds and skillsets that were effectively utilized in creating different portions of the FDT clients and servers. It was found that a high degree of concurrent development was achieved in large part due to the inherent requirement to first define the client/server interfaces in the DCE IDL files. This is

somewhat akin to defining an object in object-oriented analysis and design methodologies. DCE encourages the view of resources and their services as objects and methods, and the requirement that this definition be done first allows the DCE client and server developers to proceed independently once an IDL file is agreed upon.

The developer with legacy database management skills proceeded to create the customer database schema, load the database, and develop the interface routines (e.g., methods) that comply with the service's IDL file. In effect, this developer was encapsulating the legacy database with a layer of openness that any client developer could then access once given the IDL file -- and could access it without knowing its underlying implementation or even being knowledgeable about that implementation. A customer database object has now been created.

The developer with a background in GUI design was given the liberty to create a GUI that the user would be comfortable with, and as such dealt primarily with ergonomic and style-related issues. The third developer, in effect, acted as an integrator between the user interface object and the customer database object. This developers' role was to translate user interface messages (e.g., callbacks) into customer database messages (e.g., business transactions) and to create the necessary application logic to have the DCE client perform as desired.

These three roles are prevalent throughout the FDT clients and servers. When an object and its method already exists, it can simply be reused. This is the case, for example, in the Latest Stock Price Database with the *request_price()* method used by both the Marketminder client and again later by the FDT client. This reusability of server operations is one of the primary benefits achieved through thoughtful and generalized IDL design.

6 Summary of Prototype Development

Before the summary of the steps involved in developing the FDT prototype application environment is presented, it should be emphasized again that the selected tools and technologies promote parallel development of the application. The benefit of this approach is not only that the prototype will be produced more rapidly since it is produced in parallel, but also that the learning curve is minimized since current skillsets can be partitioned into these three pieces. That is, user interface experts can concentrate exclusively on building GUIs, and specific technology experts can deal solely with providing access to an underlying technology/service by developing the server and server operations. An expert on the TurboImage database can provide access to TurboImage data by implementing the previously agreed upon server operations, and a client developer can then access TurboImage data without having to know anything about the TurboImage database, by simply invoking the appropriate server operation.

The primary steps, then, that were followed to create the FDT prototype are outlined below:

1. ***Client/Server Interface Definition:*** For each service, the client and server software developers need to agree upon the client/server interface definition and its supported server operations. These server operations include the operation name, input and output parameters required by the operation, and possibly some additional, optional attributes. This information is specified in a DCE IDL file. These server interfaces and server operations were discussed above in "2 Description of the Prototype Environment". One of the server operations for the customer information database service is specified as:

```
[idempotent] void get_customer_data([in]     handle_t     h,
                                    [in]     char         acctNum[4],
                                    [out]    customer_t   *custData,
                                    [out]    long         *numStocks,
                                    [out]    long         *status);
```

2. ***Interface Compilation:*** For each interface, the DCE IDL compiler is executed taking the DCE IDL file as input and producing as output the respective client and server side stub code.

3. At this point, the client and server software developers can start their parallel developments, since their interface has now been formally defined.

   3a. ***Server Implementation:*** The server software developer will develop the implementation for the server operations agreed to in Step 1, above. In the implementation of each server operation, the server developer needs to manipulate the underlying resource as specified by the operation. This entails using TurboImage system calls, for example, in the customer information database server, to query or update this database. In the stock symbol service, the server developer needs to issue imbedded SQL statements to retrieve the requested stock symbol or company name.

   3b. ***Client Implementation:*** Development of the client application can also be done in parallel here, by first specifying the formal interfaces between the GUI and the application logic. What needs to be agreed to and specified up front are (1) the callback function names and parameters, so that the appropriate function can be called when the user presses a button, and (2) the names and types of the GUI objects/widgets so that the client software developer can read from and write back to the appropriate user interface elements.

3b-i. ***GUI Development:*** The user interface developer will create the appearance and behavior of the GUI and will link the user requests for action (e.g., button presses) with client application "callback" functions. The information presented in the user interfaces was discussed above in ″2 Description of the Prototype Environment". UIMX was used here to create the Motif GUIs, and the Microsoft SDK was used to create the MS Windows GUIs.

3b-ii. ***Client Application Logic:*** The client software developer will develop the necessary "callback" functions and client application logic, and will call the server interface operations as needed. Also, the client software developer needs to read from and write to the appropriate graphical user interface elements. Some pseudo-code for the FDT client shows a sample of the flow of processing that occurs when the stock broker enters a customer account number and presses the "***OK***" button:

```
void ok_callback();
{
...
/*
 *  First read the customer account number from the GUI.
 */
acctNum = XmTextGetString(textWidget);
...
/*
 *  Next invoke the customer data "query" operation.
 *  Note:  This is an RPC call.
 */
get_customer_data(bindingHandle, acctNum, &custData,
                  &numStocks, &status);
...
/*
 *  Then, put the customer data just obtained to the screen.
 */
XtSetValues(firstNameLabel, data1, count1);
XtSetValues(lastNameLabel, data2, count2)
XtSetValues(ssnLabel, data3, count3);
XtSetValues(addrLabel, data4, count4);
...
}
```

7 Conclusions

A prototype client/server financial application for stock brokers was developed using standard, off-the-shelf technologies and tools -- OSF Motif and OSF DCE. The technologies and tools that were used to develop the application were selected primarily due to their openness, standards compliance, and their capability to also be deployed in a production environment. By prioriting openness as the highest decision criteria, it is shown how installed assets can be leveraged, new technologies can be integrated, and how the focus of control for the environment can be shifted away from the vendor and back to the customer, where it belongs. By using standard compliant APIs that are portable to many different systems (PCs, MPE/iX, and Unix), by using standard compliant protocols that are interoperable with other vendors' tools, technologies, and implementations, by using tools and technologies that are widely available by tens and hundreds of vendors, and by using tools and technologies that scale well from a prototype application to a fully deployed application, an enterprise can start to regain control of its computing destiny.

Specifically, it is seen that by using these open technologies, the investment in installed assets, such as the TurboImage database, can be protected, and in fact enhanced, by opening up access to the data to the entire enterprise. In this fashion, where the enterprise's business units can use this technology to methodically open up the access to their business data and services to anyone who needs to use them, business processes can start to be better optimized and reshaped.

It is also seen how new technologies, such as the Ingres relational database (and in the future, object-oriented databases and audio and video), can then also be easily added to the environment for the additional benefits of those latest technologies.

Finally, the client/server development process has demonstrated to lend itself very nicely to rapid and effective systems development, in part by allowing the capability for parallel software development, in part because the development tools (the DCE IDL compiler and the UIMX interface builder) automatically generate much of the source code, and in part because if the services are developed in a generalized manner, they can be reused many times over by many new clients.

8 Related Work

The FDT prototype is currently being used as an educational tool to help demonstrate some of the features and benefits of OSF DCE. An HP customer education course entitled "Hands-On With Open, Client/Server Technologies" (HOW) has been developed to give information technology professionals a hands-on exposure on what is involved in developing a client/server solution using true, standards-based tools and technologies, such as OSF Motif and OSF DCE. The

class is four (4) days long -- half lecture and half lab -- and utilizes the FDT prototype for the lab exercises. Two new clients and one new server have been added to the FDT environment in the development of this class to allow students to be added as customers, to allow customer reports to be generated, and to allow a two-way mapping between stock symbol and company name. [7]

Several other enhancements to the FDT prototype are currently planned or are already completed. The first enhancement is introduced in "5.4 Replicated Servers" in which the Transarc Encina TP monitor will be used to keep the replicated FDT application servers in synch with each other by the XA-compliant two-phase commit protocol. If it is determined that this significantly degrades interactive performance, then the Encina RQS technology will also be used to batch transactions into persistent queues for later processing. [6]

Another enhancement request is for better use of the DCE core components including rpcd, cdsd, and secd (for authentication, integrity, and privacy). Better illustration of the distributed computing is also planned, in particular with the use of multimedia features such as audio on the clients and servers. Additional functionality such as the ability to track and analyze specific stocks or customer portfolios over a period of time is planned with the purpose of better illustrating the benefits of reusability of DCE server interfaces.

References

1. Hirsch, B.: Building an Open, Client/Server Application, Interact, Volume 12, Issue 10, 100-115 (October 1992)
2. HP Interface Architect 2.0 Developer's Guide, Hewlett-Packard Company (1993)
3. OSF DCE Application Development Guide, Volumes I and II, Open Software Foundation (1993)
4. OSF DCE Application Development Reference, Open Software Foundation (1993)
5. Lyons, T.: Network Computing System Tutorial, Prentice-Hall (1991)
6. Spector, A.Z.: Preparing for Distributed Computing and Open OLTP, Transarc Corporation (1992)
7. Hands-On with Open Client/Server Technologies, HP Computer/Instrument Systems Training Course (1992)