1996 NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM
JOHN F. KENNEDY SPACE CENTER
UNIVERSITY OF CENTRAL FLORIDA

DEVELOPMENT OF A COTS-BASED DISTRIBUTED COMPUTING ENVIRONMENT BLUEPRINT APPLICATION AT KSC

Dr. Isaac Ghansah, Professor
Department of Computer Science
California State University Sacramento
Sacramento, California

KSC Colleague - Bryan Boatright
Communications/Networks

Contract Number NASA-NGT10-52605

August 9, 1996

ACKNOWLEDGMENTS

I would like to thank my NASA colleague, Bryan Boatright for the opportunity he gave me to work on an interesting problem this summer. He was available when I needed clarification for what was required as well as getting in touch with appropriate personnel. He also gave me a private tour of network facilities at KSC as well as provide an environment which made me productive.

Mark Page and Henry Yu, both employees of INET were available to answer questions, professional or otherwise. I appreciate their patience with me.

Mr. Greg Buckingham (NASA program director), Dr. Roger Johnson (UCF Program Director), and Ms. Kari Stiles (Administrative Secretary) did an amazing job of organizing and coordinating many social and professional activities that made it an enjoyable summer.

John Schnitzius (NASA Advanced Network Systems) freely gave me ride to and from work. John gave my family and I so much help in many other ways that I do not have enough words to say thanks to him and his wife Loretta. Anna Maria Ruby of NASA and Carlos of INET were always ready to talk about all kinds of things. Thank you.

Finally, I want to thank my wife Becky and my children (Fred, Joy, and Kofi) for being patient with me while I was busy spending many hours on this project.

ABSTRACT

This paper describes a blueprint that can be used for developing a distributed computing environment (DCE) for NASA in general, and KSC in particular. A comprehensive, open, secure, integrated, and multi-vendor DCE such as OSF DCE has been suggested. Design issues, as well as recommendations for each component has been given. Where necessary, modifications were suggested to fit the needs of KSC. This was done in the area of security and directory services. Readers requiring a more comprehensive coverage are encouraged to refer to the eight-chapter document prepared for this work.

1.0 INTRODUCTION

There is currently a need to develop a distributed computing infrastructure at NASA Kennedy Space Center (KSC) to support current and future activities involving both national and international partners and customers. The deployment of such an infrastructure represents a foundation for a secure distributed computing base. Future services will be integrated into the environment efficiently and effectively. This infrastructure also provides the framework for developing distributed applications in a coherent manner. The objective of this research is to develop a commercial off-the-shelf (COTS) based open distributed computing environment (DCE) for application at KSC. The main benefit of an open system is availability of low-cost application software.

1.1 Current Distributed Computing Environment at KSC

In the case of network protocols, KSC currently has a myriad suite including AppleTalk, SNA, IPX(Novell), DECnet (Phase IV), and TCP/IP. Current applications include e-mail (SMTP MIME, Microsoft, ccmail, POP, X.400 and Quickmail), real time packetized telemetry (via PC GOAL), and data warehousing.

1.2 Future Distributed Computing Needs at KSC

From the management and policy direction point of view NASA is moving in two main directions. First, there is the performance based contracting shift which will allow NASA to divorce itself from the day-to-day oversight of contractors. Second, future operations of NASA will involve the international space station. All the major industrialized nations are involved with building parts of the international space station although the United States to a large extent and Russia to a lesser extent will share most of the burden.

Technically, the future of distributed computing within NASA is as follows. From the network protocol point of view, NASA is moving towards TCP/IP suite of protocols. From the point of view of e-mail/messaging systems, NASA is embracing two key technologies, including the Microsoft Exchange Mail and standard POP mail clients. As far as user interface is concerned, NASA is looking to using World Wide Web technology extensively. In the future, general applications will involve client-server computing/databases. In the light of this a standard naming convention is desired such that one could look at the database and retrieve any information automatically. Other future applications involve transaction processing for on-line procurement such as handling purchase orders, travel vouchers, time cards, etc. on line.

As pointed out earlier, the future operations of NASA involve the International Space Station. For that reason there will be need for communication with international partners, contractors, and Space itself (space station and other space components). The international partners also use different protocols. The foregoing calls for open standards, ones supported on multiple vendor platforms. In addition, the current view is to move

away from the mainframe environment (a central point of failure) towards distributed client/server environment (a decentralized system with no single point of failure).

Therefore, a comprehensive, integrated, open, secure, and multivendor distributed computing environment (DCE) is needed that is transparent to applications. The only DCE we know of that satisfies these requirements is OSF (Open Software Foundation) DCE. OSF is a consortium of all the major computing firms including IBM, HP, DEC, etc. Visibly absent from OSF are Microsoft, AT&T and Sun Microsystems. Although Microsoft makes DCE-like products, it does not currently have a comprehensive integrated DCE. AT&T and Sun have a different DCE called ONC (Open Network Computing). Major components of ONC DCE are the popular Sun RPC and NFS. Both Sun RPC and NFS are published standards. Unfortunately, ONC DCE is not comprehensive.

The rest of this report is organized according to the major components of a generic DCE. Section 2 discusses Remote Procedure Call (RPC). Section 3 elaborates on Distributed Time Service (DTS). In section 4, Distributed Security Service is extensively discussed using the SAAINT acronym. Section 5 discussed Distributed Directory Service (DDS), section 6 discusses Distributed File Service (DFS) , and section 7 covers Distributed Transaction Service (DTRS). Finally, section 8 concludes this paper. In each section we will look at each of these components from the point of view of why OSF DCE satisfies its requirements at NASA KSC. In addition, where necessary, we will state modifications that should be made to satisfy KSC requirements.

2.0 REMOTE PROCEDURE CALLS (RPC)

RPC is the foundation of any complex distributed computing environment. All communication that takes place in a DCE uses the RPC. RPC is based on the client/server model. Its objective is to simplify distributed programming so that the distributed programmer is productive.

2.1 RPC-based Design Issues

The first step in designing an RPC-based distributed application is to identify all the operations and specify them formally using an interface description language (IDL). Subsequently, an IDL compiler can be used to produce the necessary client stub code and server stub code automatically. A client program (written by the programmer) and the client stubs are linked into one object code. The server stub and server program (written by programmer) also are linked into one object code. Note however, that the client code runs on the client and the server code runs on the server. The interface specification is the key to the RPC. To check a design of a distributed application one has to look at the interface description. If the interface description is clear, concise, and complete, the analyst does not have to look too much into the other parts of the code (i.e. stubs, client program, server program etc.) to figure out what is going on.

2.2 Recommendation

OSF DCE RPC is based on the HP/Apollo RPC. Its interface definition language is clear, concise, and complete. For example, each interface is uniquely identified globally with a UUID (universal unique identifier). In addition although at-most-once is the default semantics, the interface specification allows the designer to specify other appropriate semantics (e.g. maybe, idempotent). Finally, if an operation (which is essentially a procedure) has parameters they are qualified with as input ([in],), output ([out]), or both as the case might be. On the other hand Sun RPC is at-least-once by default but has no way of specifying alternate semantics at the interface.

3 DISTRIBUTED TIME SERVICE (DTS)

A time service is needed for coordinating activities in a DCE. The time service keeps a single, global, monotonically increasing virtual clock which is used to synchronize machine (both client and server) clocks across the network. There are many situations in a DCE where synchronized physical clocks are needed. They include although not exhaustively: security, distributed file service, replicated systems, and concurrency control.

3.1 DTS Design Issues

Time service is obtained by arranging for some time servers to be connected to some authoritative time sources such as Universal Coordinated Time (UTC). Accurate UTC sources exist around the world. Machines acting as clients obtain synchronized time from these time servers. The challenge is to filter the variable network delays experienced by the time packets, and to make sure that time always goes forward and never backwards (this can be caused by inevitable clock drifts). The time service must be distributed so that there is no single point of failure. This is done by using multiple time servers. If the time service is to scale to a large number of machines, a hierarchical (multi-level) scheme is necessary for organizing the time servers. The most accurate servers are connected to the most accurate UTC receivers. These servers will be the highest level servers. Lower level servers receive their time from these higher level servers. Fault-tolerance is achieved by obtaining time information from at least three servers at the same time and rejecting ones which are totally out of line. The best estimate is selected from the midpoint of the accepted time intervals.

3.2 Recommendation

A DTS which satisfies all of these requirements is the OSF DCE DTS. The way the time service is organized is follows. Assuming that KSC has one DCE cell, at least three DTS servers will be required. Each NASA center will similarly have as least three DTS servers. At least one of the DTS servers will be designated as global DTS server. It will synchronize itself to other NASA centers' global DTS servers and to other KSC local servers.If there is the need to synchronize to machines in other countries, a similar

arrangement can be made for a higher level (say country) DTS servers to synchronize to other country DTS servers, and so on. A flexibility of the OSF DTS also allows coexistence with other time services such as the Internet NTP (Network Time Protocol).

4.0 DISTRIBUTED SECURITY SERVICE

Security should be one of the primary goals of a distributed system for an organization such as NASA KSC which is involved in a mission-critical enterprise. We believe a good, comprehensive, distributed security service should have the characteristics of a SAAINT (will not allow anything to go wrong). We use SAAINT as an acronym for Secrecy (which means the same as Privacy or Confidentiality in security circles), Authentication, Authorization (Access Control), Integrity, Non-repudiation, and Trust. Let us look at the technical aspects of the components of a distributed security service and show how they can be applied to a DCE for NASA in general, and KSC in particular.

4.1 Distributed Security Design Issues

There are two major ways of ensuring secrecy: secret key and public key. In the secret key case the same key, $K_{A\text{-}B}$ used to encrypt and decypt. $K_{A\text{-}B}$ is a secret known only by A and B. In the public key case, A has two keys: $KPub_A$ and $KPriv_A$. $KPub_A$ is a public key used to encrypt and whereas $KPriv_A$ (a private key that A keeps secret) is used to decrypt. Examples of secret key system are DES, IDEA, RC2, RC4, and RC5. Examples of public key systems include Diffie-Hellman key exchange, RSA, and DSS.

Authentication is the most difficult problem to solve in a DCE because of the openness of the network. In such an open network, someone (an intruder) can impersonate, replay , or modify a message. The best known authentication protocol is Kerberos. In Kerberos, a principal has to prove itself to another principal by obtaining a ticket from a key distribution server (KDS). The KDS has secret key it shares with each principal based on the principal's password. Kerberos allows mutual authentication and also allows A and B to perform secret communication after the authentication, using a random secret session key generated by the KDS.

Authorization is implemented by using an access control list (ACL). It is useful to think of ACL as a matrix. An element of the matrix, ACL_{ij} , is the set of operations that an authenticated principal (person, application program), i, can perform on an object (file, printer), j. Because keeping a long list for all principals might not scale well in a large organization such as NASA it useful to use a hierarchical organization based on groups. A possible organization for human principals is as follows. Every employee at KSC belongs to the group called KSC-Empl. KSC-Empl group in turn belongs to another group called NASA-Empl. With this scheme the management of the ACL is distributed because someone at each site keeps track of employees only at that site. However, the authorization decision will take a longer time. Note that the hierarchy can be extended to cover the entire world, such as International space agency.

Integrity protection is useful in documents that have been digitally signed. It is also useful in situations where we have an unsigned message such as e-mail which we want the computer to automatically detect if it has been changed by someone. Integrity is typically implemented using a **message digest (MD).** A message digest is a one-way function which takes a variable length message, *m*, and produces a fixed length bit string *MD(m)*, usually 128 bits.*)*. Suppose Ann wants to send a message, *m*, to Bryan. Ann computes $MD(m, K_{A-B})$ and sends it along with the message, *m*, to Bryan. $MD(m, K_{A-B})$ is called **message integrity check (MIC)** or **message authentication code (MAC).** Another way of producing the MIC is to compute $K_{A-B}\{MD(m)\}$. Examples of message digests include MD2, MD4, MD5, and SHA. SHA seems to be the most secure because it produces a 160-bit hash.

Non-repudiation is used in the context of **digital signatures**. This is essential for unclassified memos, some e-mail, and electronic commerce (in the Internet for example).The digital signature scheme should be such that if A delivers a message to B, 1) B can verify, the identity of A; 2) A cannot later repudiate the contents of the message; 3) it is not possible for B to concoct the message A sent; and 4) the signature cannot be used for another document. Secret key signatures do not provide strong non-repudiation because the KDS (who knows everybody's secret) is involved in both signing and verification. Public key non-repudiation is strong. All real-world digital signature standards use the public key scheme. RSA is versatile, and has been around for a relatively long time. It is by far the defacto world standard. DSS, developed by NIST, is newer but not popular. In particular, RSA can be used to provide both secrecy and signatures, whereas DSS is designed only for signatures.

For the security service to be trusted, **key management** is essential. As usual let us look at how this is handled in both secret key and public key systems. In the case of secret key systems we have seen that the KDS keeps a shared secret for each principal. In a large system there could be multiple KDS's each responsible for users in its own **domain** (or **realm** as used in Kerberos). For example, in the case of NASA there will be a domain KDS serving the domain for each center. There will be one for KSC. Then for cross-domain authentication, every domain has to establish trust relationships with every other domain. In reality this means that each KDS has to be registered as a principal in the KDS of every domain it trusts. In a large system (such as international/intercorporate/interagency internetwork) this chain of trust relationships is not scaleable. A more scaleable arrangement is to allow a principal in domain A, to communicate with a principal in domain C, through domain B, without being registered in domain C. What we mean by registered is that they share a secret key. In this case there are two questions to answer. First, how does a principal in A find the domain path to principal in C? Second, how do we decide whether the domain path was acceptable? The first question is answered by making sure that the path followed is embedded in the naming convention. Since X.500 or DNS both use a hierarchical naming scheme this problem is solved. For example in the case of NASA the domain relationships will be in the from of the tree shown in figure 1. In figure1, KSC is a master domain at Kennedy Space Center.

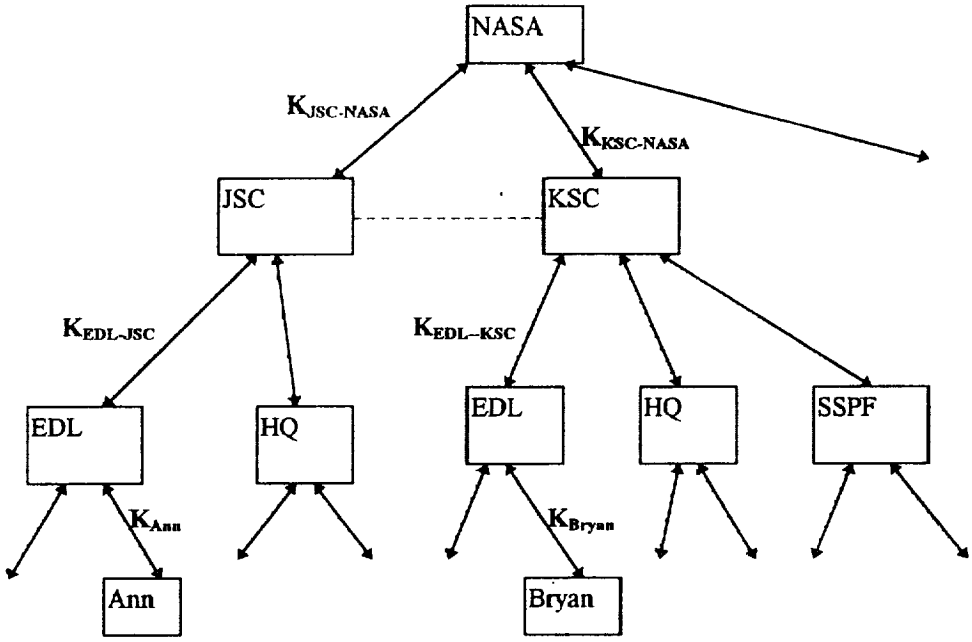

Figure 1 Hierarchical Trust Relationships in Multi-Domain Secret Key Systems

Suppose Bryan (or workstation on his behalf) wants to perform secure communication with Ann, Bryan will have Ann's X.500 name which is of the form /c=usa/o=nasa/ou=jsc/ou=edl/cn=Ann. Because the name follows the path in the hierarchy, Bryan can ask for a ticket to talk to Ann from EDL, who will ask for a ticket from KSC, and so on, until finally, Bryan can receive a ticket to talk to Ann. This is exactly how Kerberos V5 works. Kerberos V5 is used in OSF DCE. The answer to the second question (how do we know if the path transited is acceptable?) is left to the application by Kerberos V5. There is a TRANSIT field which lists all the domains transited in order to obtain the ticket. Ann will look at this list to determine if it should authenticate Bryan. Note that we do not have to follow the default hierarchy. We can use shortcuts. For example, in figure 1, if KSC's KDS trusts JSC's KDS (have registered shared keys) then we do not have to go through NASA.

Key management in public key systems is necessitated by the fact that we want to make sure that somebody's public key is authentic, otherwise, we could have impersonation. Authentic public key is ensured by using a KDS called **Certification Authority (CA).** The CA keeps the public keys of all principals. Anyone needing someone's public key asks the CA who returns a signed public key. A scalable CA is organized similar to the hierarchy in figure 1. Thus each KDS in a domain is a CA. Instead of shared keys a

domain CA issues certificates for its children and it obtains certificates from its parent. This hierarchy is sometimes called **certification hierarchy.**

4.2 Recommendation

The author recommends using Pretty Good Privacy (PGP) for secure e-mail and OSF DCE Security Service for everything else. The reasons are as follows. PGP is in the public domain, has received the test of time, and is more efficient than PEM. For example, PGP does compression. PEM does not. The only advantage PEM seemingly has over PGP is its specification of how an authentic public key is discovered. When an X.500-like certification hierarchy becomes available this argument will be moot. OSF DCE uses Kerberos V5 security system which has a better trust (key management) system than anything else available in its class such as Microsoft's Windows NT security.

5.0 DISTRIBUTED DIRECTORY SERVICE (DDS)

A distributed directory service (DDS) is a general purpose service which provides information to clients about services available in a distributed system. The distributed directory service is a fundamental building block of any distributed system because it is essential for network transparency and in many cases provides convenience for running applications automatically. For example a DDS can store a mapping of host names to IP addresses of servers so that clients can find them. Similarly, it can provide information about location, type, status of printers, e-mail address, phone number, and addresses of employees, to name only a few.

5.1 DDS Design Issues

Because a DDS deals with a large number of objects, the naming mechanism will affect its scalability. As noted earlier in this document, a hierarchical arrangement is the best way to design any scaleable system. Both DNS and X.500 provide hierarchical name space. Their naming formats reflect the hierarchical arrangement. Details can be found in chapter 5. X.500 is more flexible. On the other hand DNS has the larger installed base. Because a DDS is queried frequently for information (almost every operation requires access to it), its performance (availability and response time) is enhanced with replication and caching. Because the information in a DDS does not change often primary/secondary update protocol is used.

5.2 Recommendation

We strongly recommend X.500 for NASA in general, and KSC in particular. Because many current and almost all future applications will involve client-server computing a *standard naming convention* is desired such that one could look up and *retrieve any information automatically*. For example a print queue can be placed in a directory service such as X.500 so that different printers can be automatically selected for printing. In general, the future calls for a X.500-based directory service database that is a central

repository of all information. Note that the database will contain information about resources and not the resources themselves. Such a directory service will provide the following services and more for KSC.

- It will reduce network traffic as opposed to some current protocols that use broadcasting to find information about who has the service.
- It can answer a query such as "Where is the closest fax machine, printer, etc.?"
- It's database could be updated as needed. For e.g. printer status updated once a day.
- The X.500 directory service can be used as the basis for certificate management and distributed security service. This infrastructure is needed for verifying signatures, doing private/secret communication (e.g. e-mail) via encryption, etc. in the public key environment.
- The X.500 DDS can also be used as the basis for hierarchical trust relationships in secret key system such as Kerberos V5 used in OSF DCE. The current Microsoft Windows NT domain trust architecture being promulgated for NASA [ ] has the drawback of not being scaleable.

Currently, no technology exists that provides X.500-based DDS that is a central repository of most information. We do not know the exact reason for this. However, we suspect that it is because of its complexity. All is not lost because it is possible to custom-design such a system to fit the needs of an organization such as KSC.

We recommend using OSF DCE's DDS as a starting point for KSC. There are a number of reasons for this. First, its naming mechanism closely resembles X.500 although not exactly the same. Later, we will suggest recommendations to fit KSC needs. Second, it can be integrated with DCE security service. For example, a CDS server will not allow a user to access the database unless that user has been authenticated. Similarly, its authorization mechnism can be used to ensure that only the appropriate operation is performed. Third, because it is X.500-like the key management within secret key or public key case can be integrated relatively easily into its hierarchy.

How is DCE DDS organized? Basically, the system is organized into *cells*. Thus the service is called cell directory service (CDS). The client interface to CDS is called *clerk* and the directory database is called a *clearinghouse*. The clearing house is replicated and the clerk caches a successful information lookup to improve performance. Strictly speaking, CDS is not a central repository of all information in OSF DCE. OSF DCE actually has about four directory services. The first one is CDS which itself is basically a *name service*. It provides a mapping of host or RPC interface names to IP addresses. The second directory service is the *endpoint map* used for RPCs. The endpoint map is a mapping of RPC interface names and port numbers. When an RPC call is made, the client stub uses the interface name to fetch IP address from CDS. Then, the IP address plus interface name is used to fetch the port number from the endpoint map. The endpoint map is accessed from a well-known port number. The third directory service is the *file system's directory service*. This service is used to create, delete, etc. file directories as well as name, remove, etc., of files. Note that one of the major functions of a file directory service is to provide a mapping of file names to file IDs. The designers of DCE made the decision

to place the file directory service at the same location as the file service. To show that files are handled differently, every file name in CDS has **fs** (file system) as its prefix. The fourth directory service in OSF DCE is the *security directory service*. This directory which is next to the security server is called Registry database. It provides a mapping of principal names to principal attributes and privileges. Another directory which is closely related to it, *access control list* (used for authorization) is placed at the application server and accessed there. Once again, to show that principals are handled differently, the designers of DCE decided to prefix principal names with **sec** (security).

In order to design a directory service that is a central repository of all information such as in OSF DCE just described, we can pose the question. Is it possible to integrate these four (or five) directory services into a single one? To answer the question, let us look at how we could organize such a system. Basically we want a directory service that provides a mapping of names (file, RPC interface, persons, e-mail address, etc.) into IP adresses, port number, privilege attributes, etc. The key to doing this is to place in the DDS, for each unique name entry, not only the corresponding IP address but also all other relevant information such as port#, fileID, privilege attribute, ACL, etc. For example, in the case of RPC interface names, both IP address and port number will be in the DDS database. This way, there will be no need for an endpoint map at the server. If security is important for this interface, the privilege attributes will be entries as well. We could make similar suggestions for files or any other objects. We have answered the question.

However, we see a two disadvantages to such an integrated directory service. The first is security. It may not be a good idea to place security information in a system which is accessed by everyone unless we address all security issues associated with it and be sure it will work. The second issue is performance. We could potentially have more data crossing the network. To guard against that, it is better to place the data where it is needed. For example, in the case of RPC, having port information in the directory service means that we will transport it twice across the network, once during its retrieval from the DDS server database and then during the time the application server is accessed. On the other hand this scheme requires two accesses. So the tradeoff is between network transit time and database access time. With high speed network such as fiber the access time could be the more dominant. A research study has to be done here.

6 DISTRIBUTED FILE SERVICE (DFS)

The distributed file service (DFS) should be a major component in any useful distributed system. In particular, all application programs will be developed using the distributed file system. In addition, a distributed file service simplifies the way remote files are handled.

6.1 DFS Design Issues

Issues involved with the design of a DFS include: file directory service, file service interface, replication, and caching. The file directory service provides a mapping of symbolic names to universal file identifier (UFID). The file directory service is the user's

interface to the file system. Historically, files have always been named with a hierarchical organization. It is convenient if all machines have a uniform view of the DFS. Having the file directory service on a separate machine from the file service provides flexibility but performance is enhanced if they coexist on the same server. The file service interface issues involve whether the entire file is cached from the server or whether most accesses go to the server. The former scheme provides simplicity, scalability, and performance. Caching can be used to enhance both response time and availability and can be applied to the client or the server. Most DFS's apply it to both. Client caching poses the bigger problem - cache coherence. Replication can be used to enhance response time, availability, throughput via parallelism, and reliability. Note that replication is not used only for files. It can be used for directory services, key management services, time services, etc. to provide all the above requirements. Because replication produces consistency problems we must find ways of dealing with it. Methods range from primary/slave (the simplest) to quorum consensus (the most complex).

6.2 Recommendation

We highly recommend a DFS for NASA in general, and KSC in particular. A DFS simplifies the way remote files are handled. With NASA and its partners spread across five to seven continents in 15 time zones, with potential in the future of large volumes of data flowing between countries involved with the international space station, a DFS will make file transfer more efficient. For example, most data transfer today is done by e-mail or by FTP. Consider using FTP for a large file which you might decide to make changes to before sending it back to the remote location. A convenient way of handling such large files is to use a DFS. Instead of FTPing you can use a copy command. You can create a file remotely without too many problems. You can also update remote files easily. In three to five years ease of availability of data might be important because the international space station will be in place. A distributed file service will certainly help. The biggest problem NASA might have to face with distributed file service is security.

We recommend using OSF DCE's DFS. This DFS is similar to Andrew File System (AFS), developed at CMU and currently marketed by Transarc. The nice thing about OSF DFS is that it can be integrated within DCE. That is, DCE security can be used to provide authentication and authorization. This DFS also is scaleable (designed to support thousands of workstation) in an international scale. It uses X.500-like naming scheme, provides a uniform view of all files for all clients, provides replication, caching, and can be fully integrated into other components of OSF DCE such as the distributed time service.

7 DISTRIBUTED TRANSACTION SERVICE (DTRS)

A distributed transaction service (DTRS) is a service that supports atomic transactions involving multiple servers. In this service a client specifies the sequence of operations that are to comprise the transaction. The service guarantees to preserve the atomicity of the whole sequence. Typical examples of this include electronic funds transfer, on-line databases, distributed diaries, distributed editors, etc. Note that transactions are not only

needed for conventional bank-oriented databases but also for designing certain reliable distributed systems.

7.1 DTRS Design Issues

A transaction has the so called ACID characteristics of *Atomicity* (all or nothing), *Consistency* (stable states), *Isolation* (no interference), and *Durability* (permanence of commitment). The main issue that needs to be resolved with transactions is transaction management. There are two parts to it. The first is *recovery* which means that in the face of server or client crashes, the atomicity commitment will still be met. The second is *concurrency control* which deals with making sure that interleaved transactions produce the correct result even though they may be concurrently accessing common data. Recovery is handled by using the distributed two phase commit protocol. Concurrency control is handled by using locking, optimistic schemes, and timestamps. Details of each of these issues are covered in chapter 7.

7.2 Recommendation

Although transaction processing can be used for on-line applications such as handling POs, travel vouchers, time cards, etc., they are not required for these applications unless these applications involve multiple concurrent read/update. We do not see this in these applications. Where they will most likely be used is client server database type applications. We have not done enough studies to figure out where these applications are but we suspect that there are numerous applications where they could be used. In general, any application where multiple servers are solving a problem which is of atomic nature (reliable distributed system) will involve use of a transaction manager. If such a need arises we recommend using Transarc's Encina. Encina is an On Line Transaction Processing (OLTP) middleware designed to run top of OSF DCE. That is its major advantage. For example, it uses DCE directory and security services. It provides locking and two phase commit protocols for transaction management. Encina is marketed by most of the major OSF companies such as IBM, HP, etc.

8 CONCLUSIONS

This report developed a COTS-based DCE for NASA in general, and KSC in particular. We found out that based on current and future needs of KSC, OSF DCE holds the best promise of satisfying NASA application requirements. Given that NASA is interested in a commercial-off-the-shelf, comprehensive, open, secure, integrated, and multi-vendor distributed computing environment, OSF DCE is the only product we know of that comes close to meeting those requirements. Where necessary, we suggested modifications to DCE to satisfy NASA needs. This was done in the area of directory and security services.

Officially, NASA has not committed itself to any specific vendor's products. However, unofficially, there is a defacto commitment to Microsoft due to the fact that Microsoft

systems are pervasive. This was not necessarily done intentionally. For example, around 1994 or so a NASA-wide poll showed that about 80% of systems were Microsoft.
It is true that Microsoft products are cheaper than UNIX-based systems. For example it costs less to manage an NT server that a UNIX server. In addition, it is understood that Microsoft is one of the few vendors which has committed itself to providing cross-platform systems for applications such as spreadsheets. On the other hand, Microsoft products are not necessarily open, scaleable, or secure. We have provided OSF DCE as an alternative. Indeed, in reading Microsoft white pages about future trend of their products, we see Microsoft providing OSF DCE-like solutions. For example, although Windows NT is supposed to have a time service, there is no mention of the time service protocol although we suspect that something similar to Berkeley algorithm (implemented as timed in 4.3BSD) is being used. The security system is different from Kerberos and it uses an unscalable key management scheme. The directory service is similar to DNS. Yet, the claim in their white pages seems to indicate that future systems will have X.500 and Kerberos. NASA does not have to wait for the future Microsoft release. OSF DCE is already here.

References

[CHA96] Chadwick D., *Important Lessons Derived from X.500 Case Studies*, IEEE Network, Mar/April 1996 pp. 22-34.

[COU94] Couloris G., Dollimore J., and Kingberg T., *Distributed Systems: Concepts and Design 2ed*, 1994, Addison Wesley.

[DIL95] Dilley, J., *Experiences With the OSF Distributed Computing Environment*, Proceedings of the 3rd IFIP TC 6/WG 6.1 International Conference on Open Distributed Processing, Brisbane, Australia, February 1995, pp. 465-475.

[KAU95] Kaufman C., Perlman R., and Speciner M., Network Security, 1995 Prentice Hall

[LIN95] Linington, P., *Reference Model for Open Distributed Processing: The Architecture*, Proceedings of the 3rd IFIP TC 6/WG 6.1 International Conference on Open Distributed Processing, Brisbane, Australia, February 1995, pp. 15-33.

[MIC1] Microsoft White Paper, *The Microsoft Strategy for Distributed Computing and DCE Services*, 1995, Microsoft Corporation.

[MIC2] Microsoft White Paper, *The Microsoft Directory Services Strategy*,, 1995, Microsoft Corporation.

[NAS96] NASA MSFC, *NASA Windows New Technology (NT) Enterprise Domain Strategy*, Revision 2, June 1996

[ROS90] Rose, M., *The Open Book*, 1990, Prentice Hall.

[ROS92] Rosenberry W., Kenney D., and Fisher G., *Understanding DCE*, 1992, Oreilly and Associates

[ROS93] Rosenberry W., Teague J., *Distributing Applications Across DCE and Windows NT*, 1993, Oreilly and Assoc.

[SCH96] Schneier B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2 Ed.*, 1996 Wiley and Sons.