University of Southern California
Department of Contracts and Grants
Los Angeles, CA 90089-1147

DIstributed VIRtual System (DIVIRS) Project

formerly

Center for Experimental Research in Parallel Algorithms, Software, and Systems

Semiannual Progress Report #11
November 1993

Principal Investigator:
Herbert Schorr
Co-principal Investigator
B. Clifford Neuman
USC/Information Sciences Institute

Prepared under NASA Cooperative Agreement NCC 2-539
for Henry Lum, Technical Officer
NASA Information Sciences Division 244-7

Semiannual Progress Report

Covers period 1 May 1993 through 31 October 1993

As outlined in our continuation proposal 92-ISI-50R (revised) on NASA cooperative agreement NCC 2-539, we are (1) developing software, including a system manager and a job manager, that will manage available resources and that will enable programmers to develop and execute parallel applications in terms of a virtual configuration of processors, hiding the mapping to physical nodes; (2) developing communications routines that support the abstractions implemented in item one; (3) continuing the development of file and information systems based on the Virtual System Model; and (4) incorporating appropriate security measures to allow the mechanisms developed in items 1 through 3 to be used on an open network.

The goal throughout our work is to provide a uniform model that can be applied to both parallel and distributed systems. We believe that multiprocessor systems should exist in the context of distributed systems, allowing them to be more easily shared by those that need them. Our work provides the mechanisms through which nodes on multiprocessors are allocated to jobs running within the distributed system and the mechanisms through which files needed by those jobs can be located and accessed.

The Prospero Resource Manager

Conventional techniques for managing resources in parallel systems perform poorly in large distributed systems. To manage resources in distributed parallel systems, we have developed resource management tools that manage resources at two levels: allocating system resources to jobs as needed (a job is a collection of tasks working together), and separately managing the resources assigned to each job. The Prospero Resource Manager (PRM) presents a uniform and scalable model for scheduling tasks in parallel and distributed systems. PRM provides the mechanisms through which nodes on multiprocessors can be allocated to jobs running within an extremely large distributed system.

The common approach of using a single resource manager to manage all resources in a large system is not practical. As the system grows, a single resource manager becomes a bottleneck. Even within large local multiprocessor systems the number of resources to be managed can adversely affect performance. As a distributed system scales geographically and administratively, additional problems arise.

PRM addresses these problems by using multiple resource managers, each controlling a subset of the resources in the system, independent of other managers of the same type. The functions of resource management are distributed across three types of managers: system managers, job managers, and node managers. The complexity of these management roles is reduced because each is designed to utilize information at an appropriate level of abstraction.

During the reporting period, we continued development of PRM. We integrated PRM with the Prospero Directory Service, using the Prospero Directory Service to maintain information about

programs for use by the job manager. We also implemented a library that allows existing programs written for the Parallel Virtual Machine (PVM) parallel computing environment from Oak Ridge National Laboratory to run unmodified over PRM. The programs must be relinked with our version of the communication library, but the interface to the library is the same as that for PVM.

The current implementation of the Prospero Resource Manager runs on a collection of Sun3, SPARC, and HP9000/700 workstations running various versions of the Unix operating system, and a single Intel486 personal computer running Mach. Communication between the job, system, and node managers, and between tasks in a job is supported by a reliable delivery protocol based on the user datagram protocol (UDP) running over local and wide-area networks. Heterogeneous execution environments are supported - a system manager may manage nodes of more than on processor type. In the common case there is one system manager for each site. For example, our setup consists of one system manager responsible for a set of SPARCstations on USC's main campus, another managing a collection of Sun3, SPARC, and HP700 workstations at ISI, 15 miles away from the main campus, while a third manages a set of HP700 workstations at MIT, across the country. We have run applications that use processors at all three sites.

Programmers link executables for their tasks with the communication library we provide. Depending on how PRM has been configured users then create a job description file or they make suitable entries in the Prospero Directory Service. To run a parallel application, users invoke the job manager passing it the name of the application. I/O to the terminal and to files that are not otherwise accessible to the application is handled through an I/O task that runs on the user's workstation.

A paper describing the Prospero Resource Manager was presented at the Second International Symposium on High Performance Distributed Computing in July. An advance copy of that paper was included with the previous semi-annual report. This paper has subsequently been invited for publication in a special issue of the journal Concurrency: Practice and Experience.

Our plans for the next year include continued development of PRM. We hope to use PRM to make a prototype embeddable touchstone multi-processor (built by the EV project at ISI) available to Internet users. We have started to develop debugging and performance tuning tools for parallel applications. These tools take advantage of the user level job manager. Work is underway to support suspension and subsequent migration of tasks. Work is planned to allow sequential applications to run remotely by PRM without relinking with our communication libraries.

The Prospero File System and Directory Service

During the reporting period, we continued development of the Prospero File System and Directory Service, a file system and directory service based on the Virtual System Model. In July 1993, an initial version of a menu browser was released together with a gateway that made information from the Gopher service available to Prospero users. Gateways from the Gopher Menu Browser and the X-Mosaic hypertext browser to information exported using the Prospero protocol have been implemented by Pandora systems and Bunyip Information Systems.

We have added support for a new access method based on the CONTENTS attribute that is suited to the retrieval of data from files, where the contents of a file can be sent as an attribute of the file, reducing the number of exchanges needed to access a file. In support of the CONTENTS access method we have extended the Prospero implementation to support binary data values for attributes. This technique is suitable primarily for small files, and access to larger files is best performed using an access method separate from the directory service itself. We have begun implementation of a separate Prospero Data Access Protocol in support of such access. The Prospero Data Access Protocol will provide a common protocol for access to local files and gateway access to remote files using alternative access methods, thus reducing the number of access methods that must be supported by Prospero applications. We have also developed prototype tools to generate transitive indexes. Transitive indexing is a scalable technique for generating high-level indices that direct users to information of interest.

With our recent improvements to the release, commercial organization have started to develop systems based on Prospero. America Online has contracted with Pandora systems, who is developing a system that uses Prospero to make information from the Internet available to users of America Online. Bunyip Information Systems is working with several large publishers to make available information provided by the publisher to Internet users on a subscription basis. This information will be accessed by users of the service using Prospero. We are working with other organizations as well, including the Open Computing Security Group, which is considering the use of Prospero as a component of the internal information systems they are developing for several clients.

During the reporting period, a paper describing the use of Prospero as a base for building information infrastructure was presented at the INET'93 conference and a paper on the use of Prospero in support of location-independent computing was presented at the Usenix Symposium on Mobile and Location-Independent computing. Copies of those papers are attached.

Our plans for the next year include improvement of the Prospero release, including completion of support for the Prospero Data Access Protocol. We will also continue our work on transitive indexing concentrating on methods for querying the indices, and looking at ways to improve our methods for generating the indices.

Security for Distributed Systems

We have continued work to integrate appropriate security mechanisms into Prospero. Support was added for password based authentication for Prospero directory queries. Although Kerberos authentication was already supported, a weaker form of authentication was needed for users at sites that don't run Kerberos, and passwords are better than no authentication at all. Support was also added to the Prospero Directory Service protocol for billing. These changes support a range of billing methods from the inclusion of credit card numbers, to direct billing options, to mechanisms that will be supported in the future such as proxy checks, and anonymous electronic currency. While we discourage the use of techniques that are vulnerable to compromise, such as sending a credit card number unencrypted on the network, we recognize that application developers will use such

methods anyway. By providing a common billing framework within which they can use such methods, the hooks for more secure billing mechanism will already be present in application protocols when those secure billing mechanisms are ready.

The widespread use of the computing and information infrastructure we are developing as part of the DIVIRS project requires an underlying security infrastructure to provide fine-grained access control mechanisms to protect such resources and accounting mechanisms to manage their use. We presented a paper at the 13th International Conference on Distributed Computing Systems discussing the need for such a security infrastructure and describing a possible mechanism to provide it. A copy of that paper was included with the last semi-annual report. We have separately proposed to develop such security infrastructure and have implemented the Prospero Resource Manager and the Prospero Directory Service so that it can take advantage of such infrastructure if it becomes available.

In a paper accepted for presentation at the ACM Conference on Computer and Communications Security we describe how electronic currency can also be implemented on top of such a security infrastructure. A copy of that paper is attached.

APPENDIX A - PAPERS

The following papers were prepared, accepted for publication, or presented during the reporting period. Copies of the first three papers are attached to this report.

Gennady Medvinsky and B. Clifford Neuman. NetCash: A design for practical electronic currency on the Internet. In *Proceedings of the first ACM Conference on Computer and Communications Security*. Fairfax VA, November 1993.

B. Clifford Neuman and Steven Seger Augart. Prospero: A base for building information infrastructure. In *Proceedings of INET'93*. San Francisco, August 1993.

B. Clifford Neuman, Steven Seger Augart, and Shantaprasad Upasani. Using Prospero to support integrated location-independent computing. In *Proceedings of the Usenix Symposium on Mobile and Location-Independent Computing*. Cambridge MA, August 1993.

B. Clifford Neuman and Santosh Rao. Resource management for distributed parallel systems. In *Proceedings of the 2nd International Symposium on High Performance Distributed Computing*. Spokane, July 1993. (copy included with last semi-annual report)

B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*. Pages 283-291. Pittsburgh, May 1993. (copy included with last semi-annual report)

APPENDIX B - GLOSSARY

ACM	Association for Computing Machinery
ARPA	Advanced Research Projects Agency
DIVIRS	Distributed Virtual Systems
EV	Embeddable Variant
INET'93	The 1993 annual conference of the Internet Society
I/O	Input/Output
ISI	Information Sciences Institute
MIT	Massachusetts Institute of Technology
PRM	Prospero Resource Manager
PVM	Parallel Virtual Machine
UDP	User Datagram Protocol
USC	University of Southern California

NetCash: A design for practical electronic currency on the Internet

Gennady Medvinsky B. Clifford Neuman

Information Sciences Institute
University of Southern California

Abstract

NetCash is a framework that supports realtime electronic payments with provision of anonymity over an unsecure network. It is designed to enable new types of services on the Internet which have not been practical to date because of the absence of a secure, scalable, potentially anonymous payment method.

NetCash strikes a balance between unconditionally anonymous electronic currency, and signed instruments analogous to checks that are more scalable but identify the principals in a transaction. It does this by providing the framework within which proposed electronic currency protocols can be integrated with the scalable, but non-anonymous, electronic banking infrastructure that has been proposed for routine transactions.

1 Introduction

As the world becomes more connected, the number and variety of network resources and services requiring monetary payments will grow rapidly. For example, access to online documents might require payment of royalties. Many offline services that formerly relied on cash now use electronic payment methods. More recently, protocols have been proposed [5] to support online payment for such services over open networks. While these protocols are suitable for the vast majority of transactions, most do not protect the identities of the parties to a transaction.

Concern for privacy dictates that it should be possible to protect the identity of the parties to a transaction. This is important to prevent the accumulation of information about the habits of individuals, e.g., the documents they read, or the items they purchase. It is also important to protect parties that receive payment in certain situations, such as rewards. Many protocols have been proposed for anonymous transactions, among them those by Chaum [2]. These protocols typically require a central bank that is involved in all transactions.

In this paper, we present a framework for electronic transactions that combines the benefits of anonymous transactions with the scalability of non-anonymous online payment protocols. The paper begins with a discussion of possible requirements for electronic payment systems, followed by a discussion of related work. We then present a scalable framework for anonymous transactions, discuss the benefits of the framework, and describe how it can be applied to electronic currency protocols. The paper concludes with a discussion of the scope and limitations of the framework.

2 Requirements for electronic currency

Among the desirable properties for an electronic currency system are: security, anonymity, scalability, acceptability, offline operation, transferability, and hardware independence. Some of these requirements are also described in [6].

Security: Forging paper currency is difficult. Unfortunately, electronic currency is just data and is easily copied. Copying or double spending of currency should be prevented or detected. Ideally, the illegal creation, copying, and reuse of electronic cash should be unconditionally or computationally impossible. Some systems rely instead on post-fact detection and punishment of double spending [2].

Anonymity: The identity of an individual using electronic currency should be protected; it should not be possible to monitor an individual's spending patterns, nor determine one's source of income. An individual is traceable in traditional transaction systems such as checks and credit cards. Some protocols are unconditionally untraceable, where an individual's spending can not be determined even if all parties collude [1, 2]. For some transactions, weaker forms of anonymity may be appropriate, e.g. traceability can be made difficult enough that the cost of obtaining such information outweighs the benefit.

Scalability: A system is scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance. The existence of a central server through which transactions must be processed limits the scale of the system. The mechanisms used to detect double spending also affects scalability. Most proposed e-cash protocols assume that the currency server will record all coins that have been previously spent and check this list when verifying a transaction [2, 6, 7]. This database will grow over time, increasing the cost to detect double spending. Even if the life of a coin is bounded, there is no upper bound on the amount of storage required since the storage requirement depends on the rate at which coins are used, rather than on the number of coins in circulation.

Acceptability: Most e-cash proposals use a single bank [2, 6, 7]. In practice, multiple banks are needed for scalability, and because not all users will be customers of a single bank. In such an environment, it is important that currency minted by one bank be accepted by others. Without such acceptability, electronic currency could only be used between parties that share a common bank. When currency minted by one bank is accepted by others, reconciliation between banks should occur automatically. To our knowledge, NetCash is the first system that satisfies this requirement.

Off-line operation: The ability for two parties to make a safe transaction without instantaneously contacting the authority that issued the currency is desirable.

Transferability: The ability of the recipient of electronic currency to spend the currency with a third party without first contacting the currency server is desirable. Such transferability can improve anonymity, but it complicates the mechanism that assures security.

Hardware independence: To prevent double spending during offline operation, some e-cash protocols rely on tamper-proof hardware [4]. A drawback to this approach is that new technology might allow the compromise of such hardware, leaving users vulnerable to double spending.

3 Related work

There have been numerous recent proposals for protocols to support unconditionally untraceable, electronic currency [6, 7]. Many of these proposals are variants of and improvements upon proposals by Chaum [2, 3]. Although these protocols address many of the the requirements from section 2, unconditional anonymity is achieved at the expense of scalability, and acceptability is unaddressed.

NetCash provides scalability and acceptability with weaker anonymity and only a limited form of offline-operation. We believe that for many transactions this is sufficient. Where unconditional anonymity or completely offline operation is required, our framework can be extended to integrate exchanges from other protocols.

Protocols have been proposed that support scalable distributed accounting without anonymity [5]. These protocols provide an accounting infrastructure within which funds can be transferred between clients and servers. Because these protocols do not provide anonymity, they are not by themselves sufficient for our purposes in this paper. They will, however, be used to reconcile balances across currency servers, and to allow users to withdraw and deposit money into existing accounts.

4 Framework

NetCash is designed to support realtime electronic payments with varying transaction anonymity characteristics to geographically dispersed clients in multiple administrative domains. The primary contribution of NetCash is as a framework for integrating anonymous electronic currency into the global banking and accounting infrastructure. Section 5 defines a practical electronic currency protocol that provides weaker anonymity than the unconditional anonymity provided by Chaum [2]. The framework is useful even where unconditional anonymity is required since the protocols implementing Chaum's currency can replace the basic building blocks of the protocol described in section 5, while leaving the basic framework intact.

The NetCash infrastructure is based on independently managed, distributed currency servers that provide a point of exchange between anonymous electronic currency and non-anonymous instruments such as electronic checks. In the framework, checks based on the global accounting infrastructure [5] tie together currency servers in different administrative domains, into a financial federation where currency minted by different servers is accepted.

An organization wishing to set up and manage a currency server obtains insurance for the new currency from an agency similar to federal deposit and insurance corporation; the currency is backed by account balances registered to the currency server in the non-anonymous accounting infrastructure. We will refer to the insuring agency as the federal insurance corporation (FIC). To add a new currency server, an authentication service is used to establish a secure connection between the currency server and FIC. The currency server creates a public key pair and sends the public key to FIC over the secure channel (the corresponding private key is used for signing coins). In return FIC issues a certificate of insurance for producing and managing the currency. Figure 1 shows a certificate of insurance. It includes a unique ID to identify a particular currency server named in the certificate, the public key of the currency server along with the date of issue and an expiration date of the certificate. All the information is sealed with the private key of FIC. Based on this certificate different currency servers and financial institutions will accept the currency of a given server as legal tender. The consequences of a compromise of K_{FIC}^{-1} are severe.

$$\{\text{Certif\_id, CS\_name, } K_{CS}\text{, issue\_date, exp\_date}\}K_{FIC}^{-1}$$

Figure 1: A certificate for minting currency

$$\{\text{CS\_name, s\_addr, exp\_date, serial\_num, coin\_val}\}K_{CS}^{-1},\ \text{Certif\_id}$$

Figure 2: Electronic coin

It is up to the client to select a currency server. A reasonable choice could be based on geographical proximity and the amount of trust the client places in the currency server. A currency server provides the following services to its clients: coin verification (detection of double spending), coin exchange for untraceability, purchasing coins with checks, cashing in coins for checks. The latter two services as well as verification of coins minted by other servers relies on the accounting infrastructure described in [5] and is not further described in this paper. Below, we describe the basic function provided by the currency server to facilitate coin verification and potentially anonymous coin exchanges.

4.1 Functionality and structure of NetCash components

A coin in our protocol (see figure 2) includes among other information a serial number signed with the currency servers private key. This information uniquely identifies the coin to the currency server that issued it. The currency server keeps a list of serial numbers for all outstanding coins[1]. When a participant in a monetary transaction sends a coin for verification, the currency server checks the coin's serial number against the outstanding list. If the serial number is found, the coin is valid (has not been spent before). The serial number is deleted from the list, and a new coin with a different serial number is issued to the client and the new serial number added to the list. If a coin is tendered for which the serial number is not found, an attempt at double spending has been detected and the exchange is refused.

A currency server is implemented as a collection of servers connected on a network. This set of servers has a collective name valid on the Internet. Initially, each server is allowed to create a number of coins based on a policy set by the agency insuring the currency. Each server will manage coins with a range of values.

The structure of an electronic coin is shown in figure 2. The monetary value of the coin is specified in the coin_val field. An internet address is part of the coin, allowing the coin to be sent directly to the server keeping track of it. If the currency server is not reachable at the address in a coin, the name of the currency server (CS_name) is used to find the address by querying a directory server. Time stamps in the coins limit the state that must be maintained by each currency server.

All information in a coin is sealed with the private key K_{CS}^{-1} of the currency server. A client wanting to decrypt the coin can use the Certif_id, which provides a mapping to an appropriate certificate, thus obtaining the public key K_{CS}. The validity of the coin is proven upon successful decryption

[1] Depending on the characteristics of currency used, this list might be represented as a bit vector or as a list of serial numbers.

of the coin, but the fact that it has not been double spent is not assured until the coin is exchanged for a new coin directly with the currency server.

5 NetCash exchange protocols

In this section, we define protocols for monetary exchanges with provision of anonymity. We will concentrate on protocols providing practical anonymity. In our protocol description, we make the assumption that clients use different currency servers, each of which mints its own currency backed by account balances in the non-anonymous accounting infrastructure, but registered in the name of the currency server itself, not its clients. For the sake of clarity in our protocols, we refer to such non-anonymous transactions using this accounting infrastructure as the transfer of a check. The mechanics of the non-anonymous transfers will not be discussed here but can instead be found in [5].

5.1 Notation

The participants in our protocols are: clients, merchants, and currency servers. The first two are represented as A and B, the currency servers are denoted as CS. Banks are simply merchants in our protocol and are also represented by A and B. X stands for any participant.

The term 'transaction' is used here to mean a monetary transaction between participants. A payment happens in one direction from A to B. Encryption is represented by curly braces {}. A public key is represented as the letter K with a subscript naming the owner of the key. A subscript ending with letter N indicates a newly generated key not advertised anywhere in association with the principal's identity. A private key is the inverse of the public key and is represented as K^{-1}_{key}. Keys for a symmetric encryption system are represented by the letters SK with a subscript.

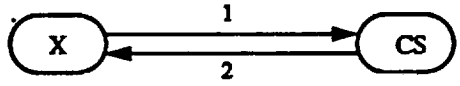
5.2 Basic building blocks

In this section, we describe basic exchanges that serve as building blocks for later protocols. As discussed in section 4, currency servers issue coins that may be used by their clients while preserving their anonymity. Currency servers provide the point of entry to the accounting infrastructure, accepting coins from other currency servers as well as other financial instruments (checks for example).

Anonymity of a client is preserved in a pure currency exchange with the CS for several reasons: the CS exchanges coins with a client by providing new coins having different serial numbers and does not keep records pairing the coins accepted with those issued. The exchange occurs anonymously; the CS does not know who is trading in the coins. It might be the client itself, or an anonymous merchant that just accepted the coins and needs to exchange them to be sure that they are not double spent. Finally, the client selects the CS to be used and is likely to choose one it trusts.

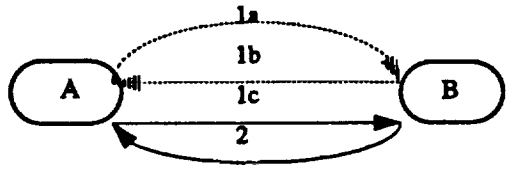
5.2.1 Exchange with the currency server

The basic exchange shown in figure 3, provides the following services: coin verification (detection of double spending), coin exchange for untraceability, purchasing coins with checks, and cashing in coins for checks. Only the latter two operations expose the identity of X. Initially, X is assumed to know the currency server's public key K_{CS}, cached from previous transactions or obtained directly from CS embedded in a certificate (see figure 1). In step 1, X sends a check or a coin (collectively referred to as an instrument), SK_X a newly chosen secret key and an indication of the transaction to be performed (e.g., whether it wants new coins or a check,



1. {instrument, SK_X, transaction}K_{CS}
2. {instrument}SK_X

Figure 3: Exchange with the currency server



1a. K_{AN}
1b. { K_{BN} }K_{AN}
1c. {coins, SK_{AN1}, K_{ses}, S_id }K_{BN}, {Certif_id, K_{CS}, issue_date, expiration_date}K^{-1}_{FIC}
2. {{amount, T_id, date }K^{-1}_{BN} }SK_{AN1}

Figure 4: Simple payment, optional steps: 1a & 1b.

and if a check, the name of the party to which it should be payable), all sealed with the currency server's public key.

If the instrument provided is a coin issued by the currency server itself, the coin is checked for double spending by verifying whether the record associated with the coin exists. If the instrument is a coin issued by another currency server, the local currency server contacts the remote currency server to convert the coin, accepting in return a check payable to the local currency server, which is then cleared through the global accounting infrastructure. If the instrument is a check, the local currency server clears it, depositing the proceeds in its own account.

In the second step, the server returns the desired instrument, either newly issued coins, or a check made payable to the individual named in the transaction. Encryption with SK_X, proves the identity of the CS and prevents the contents of the message from exposure to an attacker.

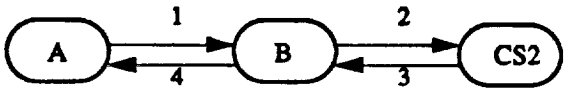
5.2.2 Simple payor-payee exchange

Figure 4 shows a simple payment protocol where A remains anonymous. B has the option to remain anonymous with additional provisions described below. Upon completion of the protocol, B is not protected against double spending and A is not guaranteed a valid receipt.

Initially A is assumed to posses B's address. Messages 1a and 1b are used to obtain B's public key, either one that identifies B, or one generated on the fly if B is to remain anonymous. If A already knows B's public key these messages may be dropped. In step 1c, A sends the coins[2], the identifier of the desired service S_id along with two keys SK_{AN1} and K_{ses}. B uses K_{CS} to verify that a certified currency server minted the coins. In order to pair the coins with a connection, B retains the session key K_{ses}; at the time the service is to be provided, B verifies that A knows the session key. In the last step, B returns a receipt signed with its private key and encrypted with SK_{AN1}, thus preventing the contents of the message from exposure to an attacker. The receipt includes amount paid, date and a unique identifier T_id that will be used along with the session key to obtain the service.

Note if steps 1a and 1b are used to obtain an anonymous public key, the protocol can withstand passive attacks in the

[2]The insurance certificate for the coins can be obtained in one of the following ways: directly from the currency server, sent with the coins as shown in figure 4, or retrieved from a directory service.



1. {coins, SK_{AN1}, K_{ses}, S_id }K_B
2. {coins, SK_{BN}, transaction}K_{CS}
3. {new_coins}SK_{BN}
4. {{amount, T_id, date }K_B^{-1} }SK_{AN1}

Figure 5: Prevention of double spending.

1. { coins, SK_{AN1}, K_B, $date_B$, $date_A$, amount}K_{CS1}
2. {< C_B,C_A,C_X>, <,,C_X>}SK_{AN1}
3. { C_B, SK_{AN2}, K_{ses}, S_id }K_B
4. {{amount, T_id, date }K_B^{-1} }SK_{AN2}

Figure 6: Protection from fraud.

sense that the privacy of the transaction is maintained, but is vulnerable to an active attack where an anonymous attacker impersonates the anonymous service provider.

5.3 Combining the building blocks

Below, we define protocols preserving the payee's and payor's anonymity using combinations of building blocks to provide guarantees against double spending, no receipt, or invalid receipt. To avoid redundancy, we omit the detailed description of steps within a given block; refer to section 5.2 for this information.

5.3.1 Exchange with protection from double spending

We combine the two protocol modules in figures 3 and 4 as shown in figure 5 to protect B from double spending by an anonymous payor A. A is assumed to know B's public key, but the anonymity of B can be protected by adding message 1a and 1b from figure 4. After receiving coins from A, B verifies the coins with the currency server. If the coins haven't been spent already, then B issues a receipt to A. The shortcoming of this protocol is that it only provides protection from fraud by the payor. B could simply spend A's coin without providing a valid receipt. The protocol presented in the next section solves this problem.

5.3.2 Exchange with both parties protected from fraud

We would like to extend the model presented in section 5.3.1 to eliminate B's ability to cheat. The protocol presented in this section preserves A's anonymity, protects B from double spending, and guarantees A a valid receipt or its money back. To make such guarantees possible we extend the definition of a coin. A coin can be customized for a given principal, i.e., it can only be used by that principal for a certain window of time. Thus, the payor A can use the currency server to obtain a coin triplet <C_B,C_A,C_X>. Each coin in the triplet has the same serial number and coin value.

During the first window, C_B is the only coin that can be used with the currency server. In the next window, only C_A can be used, and in the last window the C_X coin is used exclusively. This can be implemented by embedding window boundaries (time stamps) into each coin in the triplet.

The first two coins, intended for B and A respectively have keys embedded in them. If either party wants to use its coin in a transaction with CS1, it must prove knowledge of the embedded key. For example B's public key K_B is encrypted in C_B, during the transaction with CS1 it must prove knowledge of K_B^{-1}. The third coin C_X, does not have a key encrypted in it and can be used by anyone. Additional information embedded in C_A is B's public key, this is done to reduce the amount of state information needed to be maintained by the server in order to issue A a receipt. Also, an additional bit needs to be associated with a coin serial number in the CS1's database to keep track of whether A or B spent the coin.

In the transaction with B, A will keep coins C_X and C_A and pass C_B to B. If B does not give a receipt to A, A can query the currency server and check whether B spent the coin. If B spent the coin, the currency server will issue A a receipt specifying the coin value and B's public key. Otherwise, A can obtain a refund during the window in which C_A is valid. B should keep track of C_B until it expires in case A attempts to double spend C_B with B. C_X is provided for additional flexibility in monetary transactions when A does not ultimately spend the coin with B. Figure 6 shows the steps of the enhanced protocol. In step 1, A sends coins to its currency server to obtain a coin triplet[3] ($date_A$ and $date_B$ denote expiration dates for A's and B's window of operation). The currency server creates a coin triplet and embeds the information in the coins as described above. CS1 returns the triplet, along with possible change <,,C_X> if the amount specified was less than the total value of the coins sent in step 1. In step 3, A passes C_B to B. B must convert the coin while it's valid, during the first interval. In the next step B returns a valid receipt to A. In case it doesn't, during the second time interval, A sends C_A to CS1. CS1 then checks whether the coin was spent in the first window of time. If it was, CS1 returns a receipt specifying B's key and the value of the coin all signed with CS1's private key. In case the coin was not spent, CS1 will issue a new coin to A.

It should be noted even though B is a client of CS2, it can still accept coins minted by other authorities because of the accounting infrastructure on which NetCash is based.

The anonymity of the payee can be achieved by combining steps 1a and 1b of figure 4 with the protocol presented in this section. In the resulting protocol, the reciept provided to the payor is not very useful since the payee is anonymous. The details of the protocol are left as an exercise to the reader.

5.4 Off-line protocols

In an offline transaction, it is desirable to prevent double spending while preserving the anonymity of the participating parties. Transactions conducted in the offline mode where neither party contacts the currency server during the exchange can be supported in NetCash by several means.

The protocol shown in figure 6 can be used as follows: If A knows ahead of time that it is going to conduct business with B, steps 1 & 2 can be done in advance. At a later time, A & B go through an exchange, using steps 3 & 4, where upon completion, double spending is prevented and payor's anonymity is maintained. A drawback of this protocol is the payor has to know in advance with which particular party a transaction will be performed.

Another approach to offline transactions is to use the protocol shown in figure 4 in conjunction with tamper-proof electronic wallets. Double spending is prevented by properties of the hardware. The problems with this approach was described in section 2.

Currently, we are looking into incorporating Chaum's post-fact punishment scheme[2] into NetCash. Double spend-

[3] In case different coin denominations are desired, A could specify several amounts, and obtain a number of triplets each having a particular value.

ing makes it statistically possible to determine the identity of a dishonest client. The drawback with this approach is that post-fact punishment may be unacceptable to financial institutions due to the complications in tracking and punishing potential violators.

6 Discussion

NetCash combines the benefits of anonymous transactions with the scalability of non-anonymous online payment protocols. It is secure, scalable, valid across administrative domains, and provides some assurance of anonymity for the parties to a transaction. In this section, we discuss the benefits and drawbacks of NetCash, revisiting some of the requirements from section 2.

Where it is possible for at least one party to interact with a currency server at some point during a transaction, NetCash is secure. Double spending is either detected at the time the recipient verifies or exchanges coins with the currency server, or the coins can only be spent by the recipient during an initial time window, allowing the recipient to cash them in before they can be double spent.

Because independent currency servers exist NetCash is more scalable than other e-cash proposals. When coins are exchanged with remote currency servers, the balances of the currency servers (the backing of the currency) are adjusted through the scalable, but non-anonymous, accounting infrastructure proposed in [5]. The anonymity of the client is not jeopardized because only the currency servers themselves are identified in the non-anonymous transaction.

The anonymity provided by NetCash is weaker than the unconditional anonymity provided by Chaum. In particular, at the point that a client purchases coins from a currency server by check, or cashes in coins, it is possible for the currency server to record which coins have been issued to a particular client. It is expected that currency servers will not do so, and it is likely that the agreement with clients will specifically preclude it. Additionally, the client can choose its own currency server, and will choose one that it feels it can trust.

Once coins have been purchased, they can continue to circulate without identifying the intermediaries. Although the currency server is involved each time a coin changes hands, and could conceivably track which coins are exchanged for others though prohibited from doing so, it will not know the identity of the intermediaries until one of the parties chooses to identify itself when converting in coins. The longer the chain of intermediaries, the less information that is available about who made purchases where.

Although coins may be transferred in our scheme without interaction with the currency server, when coins are used in this manner, no assurances exist that a coin has not been double spent. Thus, among a group of individuals that trust one another (or each others tamper-proof hardware), coin transfer is possible. Parties to a transaction would need to eventually verify and exchange their coins to limit their vulnerability to double spending.

Our approach supports partially offline operation, where the parties are offline during the final exchange; secure operations do require that at least one party interact with a currency server at some point during a transaction.

Where unconditional anonymity or completely offline operation is required, our framework can be extended to support exchanges from Chaum's protocol or from other electronic currency mechanisms. Such exchanges could be applied to only those transactions that require them, while still providing scalability, acceptability, and interoperability across mechanisms.

7 Conclusion

This paper presents a framework for electronic transactions that combines the benefits of anonymous transactions with the scalability of non-anonymous online payment protocols. Our framework is secure, scalable, acceptable across administrative domains, and provides some assurance of anonymity for the parties a transaction. Our approach supports partially offline operation, where the parties are offline during the final exchange; secure operations do require that at least one party interact with a currency server at some point during a transaction.

Where unconditional anonymity or completely offline operation is required our framework can be extended to support exchanges from other electronic currency mechanisms for those transactions that require them, while still providing scalability, acceptability, and interoperability across mechanisms.

Acknowledgments

We would like to give a special thanks to Yacov Yacobi for his insightful comments regarding e-cash. We would also like to thank Celeste Anderson, Deborah Estrin, Katia Obraczka, Barry Perkins, Jon Postel, Stuart Stubblebine, and Peter Will for discussion and comments on drafts of this paper. A great thanks to Shai Herzog for coming up with the name NetCash.

References

[1] D. Chaum, B. Boer, E. Heyst, S. Mjolsnes, and A. Steenbeek. Efficient off-line electronic checks. In *Proceedings of Eurocrypt '89*, 1989.

[2] D. Chaum, A. Fiat, and N. Naor. Untraceable electronic cash. In *Proceedings of Crypto '88*, 1988.

[3] Chaum D. Security without identification: Transaction systems to make big brother obsolete. *Communication of the ACM*, 28(10), October 1985.

[4] S. Even, O. Goldreich, and Y. Yacobi. Electronic wallet. In *Proceedings of Crypto '83*, 1983.

[5] B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[6] T. Okamoto and K. Ohta. Universal electronic cash. In *Proceedings of Crypto '91*, 1991.

[7] B. Pfitzmann and M. Waidner. How to break and repair a 'provably secure' untraceable payment system. In *Proceedings of Crypto '91*, 1991.

This research was supported in part by the Advanced Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies. Figures and descriptions in this paper were provided by the authors and are used with permission.

The authors may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511, email ari@isi.edu, bcn@isi.edu.

Prospero: A Base for Building Information Infrastructure

B. Clifford Neuman Steven Seger Augart

Information Sciences Institute
University of Southern California

Abstract

The recent introduction of new network information services has brought with it the need for an information architecture to integrate information from diverse sources. This paper describes how Prospero provides a framework within which such services can be interconnected. The functions of several existing information storage and retrieval tools are described and we show how they fit the framework. Prospero has been used since 1991 by the archie service and work is underway to develop application interfaces similar to those provided by other popular information tools.

I. Introduction

The past several years has brought the introduction of a large number of information services to the Internet. These services collectively provide huge stores of information, if only one knows what to ask for, and where to look. Unfortunately, knowing what and where to ask is a major problem; available information is scattered across many services, and different applications are needed to access the data in each.

While there have been many attempts to create gateways between these services, such gateways have not been as useful as needed. Gateways have typically taken one of two forms. The first is a portal between two services: one service is used to find an instance of a second service, to which the user is then connected, leaving the user to figure out how to query the second service. The second form of gateway translates queries from one service into those of another, and returns the result of the query in a form that a user of the first service can understand. The second kind of gateway is easier for the user, but it is still limited since, as typically implemented, such gateways do not allow complete integration of the two information spaces.

This paper shows how the Prospero Directory Service provides a framework for interconnecting information services. The paper begins by dividing the functions of information systems into four categories: storage, retrieval, organization, and search. The role of existing services in this taxonomy is described, and the role of Prospero in interconnecting existing information services using this taxonomy is presented.

II. The Four Functions

The services of existing Internet information retrieval tools can be broken into four functions: storage, access, search, and organization.

The storage function is the maintenance of the data that may subsequently be provided to and interpreted by remote applications. File systems support storage, providing a repository where data may be stored and subsequently retrieved. Document servers including the Wide Area Information Service (WAIS) [5], menu servers including Gopher [7], and hypertext servers including World Wide Web [1] also provide the storage function since they manage documents that are subsequently retrieved and displayed by their clients.

Whereas storage is primarily a function of the server, access involves both the client and the server. The access function is the method by which the client reads and possibly writes the data stored on a server. The access method is typically defined by network protocols including the File Transfer Protocol (FTP) [11], Sun's Network File System (NFS) [12], the Andrew File System (AFS) [4], and the application specific protocols used by WAIS, Gopher, and World Wide Web. The client and the server must use the same protocol.

The search function involves the iterative or recursive retrieval and analysis of information about data (meta-information), possibly across multiple repositories, with a specific goal of identifying or locating data that satisfies the search criteria. A search heuristic defines the method and strategy used to conduct the search. Examples of search tools include Knowbots [6] and NetFind [13]. Though searches are initiated by a client when the need for specific data is realized, parts of the search may execute remotely. For example, the lookup of an entry in a remote index or directory on a WAIS, Gopher, WWW, or Prospero server constitutes a search. The Dynamic WAIS interface to NetFind [3] takes this a step further; the server initiates and manages, in real-time, a search across multiple hosts on behalf of the client.

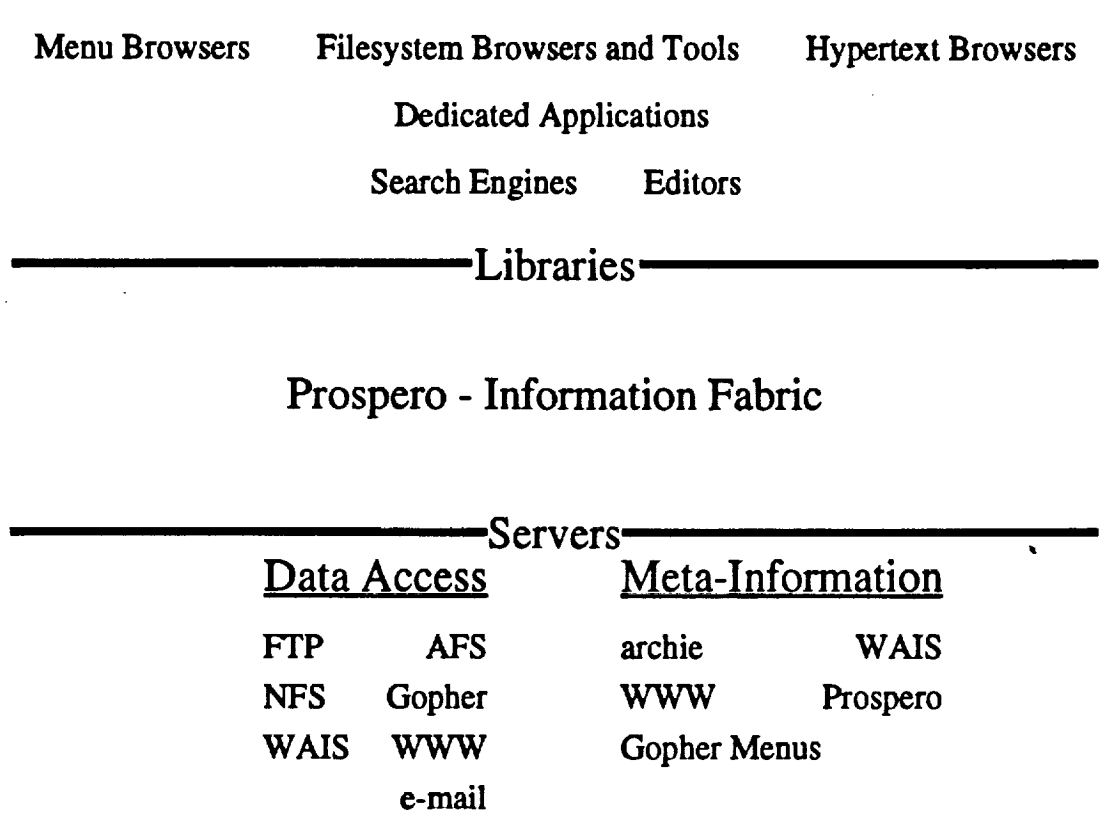


Figure 1: A framework for network information services

Brute force search in a system as large as the Internet is not practical. To be effective, search heuristics must be applied. These heuristics rely on meta-information describing the data that is available. The more structured this meta-information, the more useful it is in directing searches. The organization function involves the collection, maintenance, and structuring of such meta-information. Examples of organization mechanisms include directories in Prospero, menus in Gopher, links in hypertext documents, indices of data local to a WAIS server, and the file name index maintained by archie [2] for files scattered across the Internet.

The difference between search and organization is that a search is initiated when specific data is needed, whereas data is organized in advance to support subsequent searches. Search mechanisms play a role in the organization of data when the results of possibly complex or expensive searches are recorded for subsequent use by applications.

III. A Flexible Framework

Most information services presently available on the Internet provide their own mechanisms for each of the four functions, whereas their novel features are usually confined to their user interfaces and/or at most one of the four functions. As a result, gateways are required to allow applications from one service to use information maintained for another.

A well defined interface is needed that will allow information maintained by one service to be used by another. This interface would appear between search and organization, providing a common method for applications to query the meta-information maintained by other services. A similar interface would separate storage and access, providing access by all applications to data maintained by different services.

Figure 1 shows the location of existing services in such a framework. Meta-information available from archie, WAIS, World Wide Web, Prospero, and Gopher menus are available through a common query mechanism. Menu browsers, file system tools, hypertext browsers, and dedicated applications such as the command line archie client and NetFind use this mechanism to query services. Update of meta-information is also supported, providing a common interface between hypertext and menu editors and repositories. Search engines can use the query mechanism to identify objects of interest. They can then record the result of the search for subsequent use by using the update mechanism to add new links to the information fabric. Section IV discusses how Prospero provides this functionality.

The separation of storage and access is a bit more complicated. There are already numerous access methods in use, and many of the files of interest in the Internet exist on servers that will only support a single protocol. The problem can

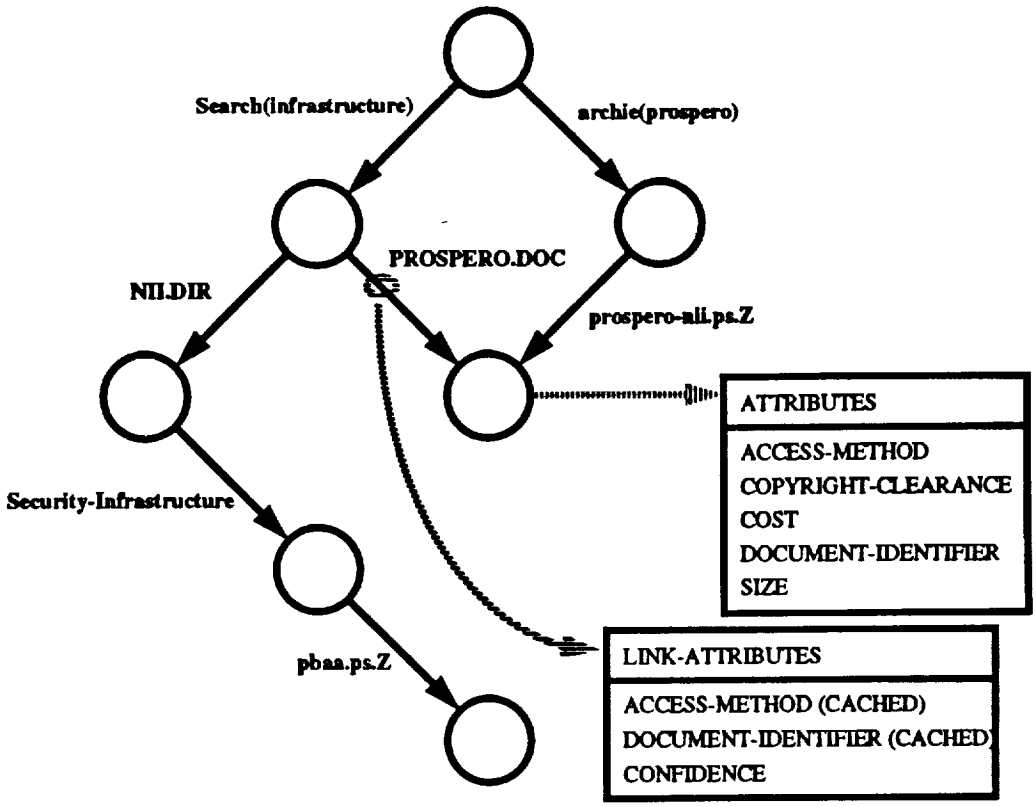


Figure 2: The Prospero naming network

initially be addressed by supporting a common application library that accepts a reference to an object obtained from the meta-information services, and automatically invokes the appropriate access method to retrieve the object. Prospero currently uses this method. While not described in this paper, we are also working on a common data access protocol for information services. That protocol will support gateway services to existing data access protocols.

IV. The Model

The information services available on the Internet each have their advantages. In order to accommodate the needs of each, a mechanism to integrate these services must be flexible. However, to be practical and efficient, such a mechanism must not become bogged down providing functionality that will only be used occasionally. The solution to these competing goals is to provide a simple but extensible framework on which the mechanisms specific to individual systems can be built.

These characteristics are present in the Prospero Directory Service [8]. We chose to concentrate initially on integrating tools for organizing information, applying Prospero as a common protocol for access to meta-information from multiple information services.

Meta-information is represented by Prospero in a *naming network*, a directed graph with labeled edges (shown in figure 2). This naming network provides the fabric through which applications navigate. Each node in the graph represents an object. An object can be a file, a directory, both, or neither (if neither, it only has attributes). Each object has associated with it a set of user extensible attributes. These attributes can have intrinsic meaning, such as the length of a file, or they can be application specific.

If an object serves as a container for data, then it is a file and has one or more ACCESS-METHOD attributes associated with it. Each ACCESS-METHOD attribute encodes the information needed to retrieve the data from a data repository using a particular access method.

If an object is a directory it contains a set of named links to other objects. The labeled edges in the naming network correspond to these links. The names on the links serve as sign posts to help the user, or the user's application, navigate through the naming network in search of the desired information. Applications that treat the naming network as a filesystem use the names of the links as components of filenames, allowing files in subdirectories to be named by the concatenation of the names of the links that are followed to reach them.

When searching for information, users and applications also use attributes. In addition to object attributes, attributes can be stored with links. Such link attributes either store additional information about the link such as annotations, or they cache information about the object to which the link refers. Cached attributes allow an application to retrieve attributes for the objects in a directory in a single query, rather than making individual requests for the attributes of each object referenced from the directory.

A filter can also be associated with a link. A filter, and a second kind of link called a union link, allow a view of a directory to be created that is a function of another view. Filters and union links provide a mechanism for users to encode, on a link, methods to be applied when searching. Because they are applied when a directory is listed, the resulting view is kept up-to-date, even when the underlying information changes. Filters and union links are described in greater detail elsewhere [9].

V. Implementation Notes

Prospero was designed to be simple but extensible. The protocol is simple, supporting stateless queries for attributes and the contents of directories. The protocol is layered on top of a reliable delivery mechanism implemented using UDP. In the normal case, a query requires a single packet for the request and a single packet response, eliminating the cost associated with setting up and breaking down a connection. This low cost is particularly important for clients that contact multiple servers in series as is often the case when resolving names.

Despite the simplicity of the protocol, Prospero is extremely flexible. Functions specific to an application rely on the extensible attribute mechanism. Applications not needing that functionality ignore the additional attributes, while applications that know about it can take action based on the value of the attributes.

VI. Integrating Services

This section describes how the meta-information maintained by several Internet information services can be represented using the model from section IV.

Archie [2] constructs an index of files from Internet anonymous FTP sites and makes the index available to applications using the Prospero protocol. A query to the archie database corresponds to a Prospero directory query for a directory whose name includes the arguments of the query. These arguments include a string that is matched against the filenames in the index. The result of a query is represented as the contents of the Prospero directory. Upon receipt of the directory query, the archie server extracts the arguments, performs a query on the archie database, and returns the results to the client. Each file matching the query is represented as a link in the directory to the file on the remote FTP site. The information needed to retrieve the matching file is returned as part of the ACCESS-METHOD attribute associated with the link. Other attributes are also returned, including file size and the time of last modification. One can apply a Prospero filter to a directory query to restrict the links that are returned to those for files on hosts in a particular part of the network.

Gopher [7] allows users to browse menus configured by Gopher service providers. By selecting menu items, users are able to run applications, search local databases, select and display files and sub-menus, and connect to other services available on the Internet. A Gopher menu can be represented as a Prospero directory. The individual items in the menu correspond to links in the directory. Submenus are links to other directories. Links to files have an associated ACCESS-METHOD attribute that provides the information needed to retrieve the file. Attributes associated with links also specify how the data in a file should be presented, for example how to display images or play back an audio recording. Links to Internet services reachable by telnet have an ACCESS-METHOD attribute of type TELNET; this ACCESS-METHOD contains the information needed to telnet to the service.

The WAIS server [5] maintains a full text index to a set of documents on a single system, allows that index to be queried remotely, and allows remote read access to the documents. The WAIS client provides a common front end for queries and access to documents on multiple WAIS servers. A WAIS query can be represented as a Prospero directory in much the same manner as an archie query. Like archie, the result of the query would be represented by the links in the directory. Each of these links would be a reference to a document on the server itself and the ACCESS-METHOD associated with the link would specify the WAIS access method. Other attributes of the link could include the relative confidence in the match. To retrieve a matched document the application would pass the selected link to the Prospero library function `pfs_open()` which would automatically invoke the WAIS access method to retrieve the document from the WAIS server.

World Wide Web is a hypertext browser that supports the interconnection of collections of documents with links that originate within the documents themselves. Links to non-hypertext documents are also supported. A hypertext document may be represented in Prospero as a node that is both a file and a directory. The document that is displayed to the user would be retrieved according to the ACCESS-METHOD attribute associated with the node. The links in the directory would represent links to other documents. The name of each link would contain the text of a tag within the document from which the link is to originate. Offsets in the target document can be represented as attributes of the link.

Prospero allows users to create their own directory hierarchies from which they can make links to directories and objects of interest. Since queries to other services and the objects stored by them are represented as nodes in the Prospero naming network, users can create directories in which the linked objects reside in different services, but can be accessed using a common protocol, thus supporting the integration of information across services.

VII. Security

Security is an important feature that is missing from most information systems on the Internet. Without fine grained control for access to information, the owners of certain information will not make it available using such systems. Since our goal is to allow the integration of information from all sources on the Internet, security must be an important consideration. Security is especially important in our model since we support the remote modification of information.

In Prospero, access control lists (ACLs) may be associated with directories in the naming network, and with individual links within a directory. The permissions supported include read, modify, insert, delete, list, and administer. Prospero requests are authenticated and the identity of the client is used to determine the access permissions that apply. Several authentication methods are supported including authentication based on the client's Internet address, the use of passwords, and Version 5 of Kerberos.

Hooks are also present to support distributed authorization and accounting mechanisms [10]. Support for accounting will become critical as new for-hire information services are introduced.

VIII. Status

Prospero has been available since December 1990. Recent revisions to the protocol allow more flexible integration of information services. Prospero has been used since Spring of 1991 to make available meta-information maintained by archie. Work is underway to develop application interfaces for Prospero that provide the functionality of Gopher, WAIS, and World Wide Web. To find out more about Prospero, or for directions on retrieving the latest distribution, send a message to info-prospero@isi.edu.

IX. Summary

A huge amount of information is available on the Internet. Unfortunately, this information is scattered across many services and different applications are needed to access the data in each. A framework was defined within which such information services can interoperate. By providing a common method for applications to query the meta-information maintained by the services that organize data on the Internet, information maintained by one service can used by all. Prospero provides such an interface. A service that uses Prospero to export meta-information about the data it provides will be usable by many applications. Similarly, an application that uses Prospero to find files will be able to access information from many services.

Acknowledgments

Many individuals contributed to the design and implementation of Prospero. Ed Lazowska, John Zahorjan, David Notkin, Hank Levy, and Alfred Spector helped refine the ideas that ultimately led to the development of Prospero. Kwynn Buess, Steve Cliffe, Alan Emtage, George Ferguson, Bill Griswold, Sanjay Joshi, Brendan Kehoe, Dan King, and Prasad Upasani helped with the implementation of Prospero and Prospero-based applications. Gennady Medvinsky and Stuart Stubblebine commented on drafts of this paper.

References

[1] Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. World-wide web: The information universe. *Electronic Networking: Research, Applications and Policy*, 2(1), Spring 1992.

[2] Alan Emtage and Peter Deutsch. archie: An electronic directory service for the Internet. In *Proceedings of the Winter 1992 Usenix Conference*, pages 93–110, January 1992.

[3] Darren R. Hardy. Scalable internet resource discovery among diverse information. Technical Report CU-CS-650-93, Department of Computer Science, University of Colorado, Boulder, April 1993. M.S. Thesis.

[4] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[5] Brewster Kahle and Art Medlar. An information system for corporate users: Wide area information systems. Technical Report TMC-199, Thinking Machines Corporation, April 1991.

[6] Robert E. Kahn and Vinton G. Cerf. The Digital Library Project; Volume 1: The world of Knowbots (draft). Corporation for National Research Initiatives, 1988.

[7] Mark McCahill. The Internet gopher: A distributed server information system. *ConneXions - The Interoperability Report*, 6(7):10–14, July 1992.

[8] B. Clifford Neuman. Prospero: A tool for organizing Internet resources. *Electronic Networking: Research, Applications and Policy*, 2(1):30–37, Spring 1992.

[9] B. Clifford Neuman. The Prospero File System: A global file system based on the Virtual System Model. *Computing Systems*, 5(4):407–432, Fall 1992.

[10] B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 283–291, May 1993.

[11] Jon B. Postel and J. K. Reynolds. File transfer protocol. DARPA Internet RFC 959, October 1985.

[12] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network File System. In *Proceedings of the Summer 1985 Usenix Conference*, pages 119–130, June 1985.

[13] Michael F. Schwartz and P. G. Tsirigotis. Experience with a semantically cognizant internet white pages directory tool. *Journal of Internetworking: Research and Experience*, 2(1):23–50, 1991.

Author Information

Clifford Neuman is a scientist at the Information Sciences Institute of the University of Southern California. After receiving a Bachelor's degree from the Massachusetts Institute of Technology in 1985 he spent a year working for Project Athena where he was one of the principal designers of the Kerberos authentication system. He holds M.S. and Ph.D. degrees from the University of Washington, where he initially developed Prospero as part of his dissertation. His research focuses on problems of system organization and security in distributed systems.

Steven Seger Augart is member of the research staff at the Information Sciences Institute of the University of Southern California. He received a Bachelor's degree from Harvard University in 1989 and an M.S. from the University of California at Irvine in 1992, with various bouts working as a systems programmer along the way. His work at ISI has focused on the development of Prospero as a scalable information infrastructure for large distributed systems.

This research was supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Centers, Digital Equipment Corporation, and the Advanced Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies. Figures and descriptions in this paper were provided by the authors and are used with permission. The authors may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511, email bcn@isi.edu, swa@isi.edu.

Using Prospero to Support Integrated Location-Independent Computing

B. Clifford Neuman Steven Seger Augart Shantaprasad Upasani

Information Sciences Institute
University of Southern California

Abstract

As computers become pervasive, users will access processing, storage, and communication resources from locations that have not been practical in the past. Such users will demand support for location-independent computing. While the basic system components used might change as the user moves from place to place, the appearance of the system should remain constant.

In this paper we discuss the role of, and requirements for, directory services in support of integrated, location-independent computing. We focus on two specific problems: the server selection problem and the user location problem. We present solutions to these problems based on the Prospero Directory Service. The solutions demonstrate several unique features of Prospero that make it particularly suited for support of location-independent computing.

1 Introduction

As the use of computers becomes pervasive the distinction between computer networks and computer systems will blur. In the ideal world, users will think of a computer network and the systems connected to it as a single system, rather than as a collection of systems connected by networks. Users will not want to use a different system each time they change their location. Although users will want to see a single system, they won't want to see the same system as every other user. Each user will want a system that is tailored to his or her particular needs.

This paper begins with a discussion of the characteristics of and requirements for what we call pervasive computing. We examine two problems that arise in such systems, the server selection problem and the user location problem, discussing the role played by a distributed directory service in their solution. In so doing, we describe the Prospero Directory Service, highlighting important features, and describing how it can be used to solve these problems.

2 Pervasive Computing

Pervasive computing combines aspects of ubiquitous computing with the integration of information and resources from many sources, within a single system tailored to the needs of a particular user. Whereas the focus of ubiquitous computing has been on the devices and the communication infrastructure, allowing the use of large and small computing devices from many locations, the focus of pervasive computing is on mechanisms that allow the pieces to be tied together to form a coherent whole. The two areas are not disjoint; each includes the other, only the perspective is different.

There are several characteristics to pervasive computing that place new demands on system organization and structure. One of the primary characteristics is mobility. The term mobility applies even to systems that don't support wireless communication; it is the mobility of users that is critical to pervasive computing. Users interact with the system from more than one location. We already see this on large university campuses where students log in from public terminal clusters.

Proceedings of the Symposium on Mobile and Location-Independent Computing, Cambridge, MA, August 1993.

The use of portable computers while traveling provides another example. In the future, users will be able to interact with the system through whatever I/O device is within reach as they travel from location to location.

A second characteristic of pervasive computing is scale. The number of objects and services to be managed can easily overwhelm the user, the geographic expanse of the system adds constraints to be considered when selecting servers, and the lack of a single organization that controls the system makes organization of these resources difficult. These characteristics can be addressed in part through support for customization. Users should be able to choose the resources and objects of interest, and treat the selected resources as a single system [5].

The directory service will play a critical role tying together the components of future systems. The requirements for such a directory service are greatly affected by the scale of the system, and the mobility of its users. One of the biggest problems to be addressed is support for transient information. The transient information in such a system comes in two forms: first, the choice of servers for certain operations may change as the user moves from location to location; and second, information about users and other mobile objects needs to be maintained.

3 The Server Selection Problem

We begin our discussion by considering the server selection problem. Selecting resources for use in a centralized system is straightforward: users choose from among the resources available and select defaults which rarely change. In traditional distributed systems, the selected resources are then located through remote name maps or directory services such as Sun's Network Information Services (formerly known as Yellow Pages) [6] and Hesiod [3].

The mobility of users complicates server selection; the user's choice of resources will often vary according to location. For example, while at home a user might want to use a printer at home and while at work, one down the hall. While traveling the user might want output faxed to the hotel's front desk, but only if there is no per-page charge for incoming faxes. Alternatively, a user might want output sent to the printer at home so that it is waiting upon return.

It should be possible for users to specify defaults in such a way that the binding is determined dynamically, when the service is needed, and based on a combination of user specified factors (e.g., cost and reliability), application requirements (e.g., support for PostScript), and transient factors (e.g., load and proximity).

3.1 Using Prospero to Solve the Server Selection Problem

With Prospero, users define a virtual system which, among other things, specifies the mapping of names to servers. This mapping is used to select the servers used by applications. Figure 1 shows how a virtual system is represented using Prospero. In the figure, the link labeled ROOT is a reference to the root of the user's file system through which the user sees the same files regardless of the location from which logged in. The Prospero File System is described elsewhere [4]. The directory referenced by the SESSIONS link identifies the locations from which the user is logged in and is described in Section 4.

In this virtual system, the server selection criteria are encoded in the CONFIG/SERVERS directory. In a traditional directory service, such a mapping would not provide the flexibility that is needed for pervasive computing. However, several features of the Prospero Directory Service allow the mapping to be determined dynamically. These features include virtual system aliases, union links, and filters.

3.1.1 Virtual system aliases

Prospero maintains several virtual system aliases that provide well defined starting points from which names may be resolved. These aliases end with the string #: and identify the virtual systems associated with the user, the home processor for the current login session[1] (the session), the processor on which an application is running (the platform), and open file descriptors.

[1] The virtual system for the session is associated with the workstation or I/O device through which the user is interacting with the system.

Proceedings of the Symposium on Mobile and Location-Independent Computing, Cambridge, MA, August 1993.

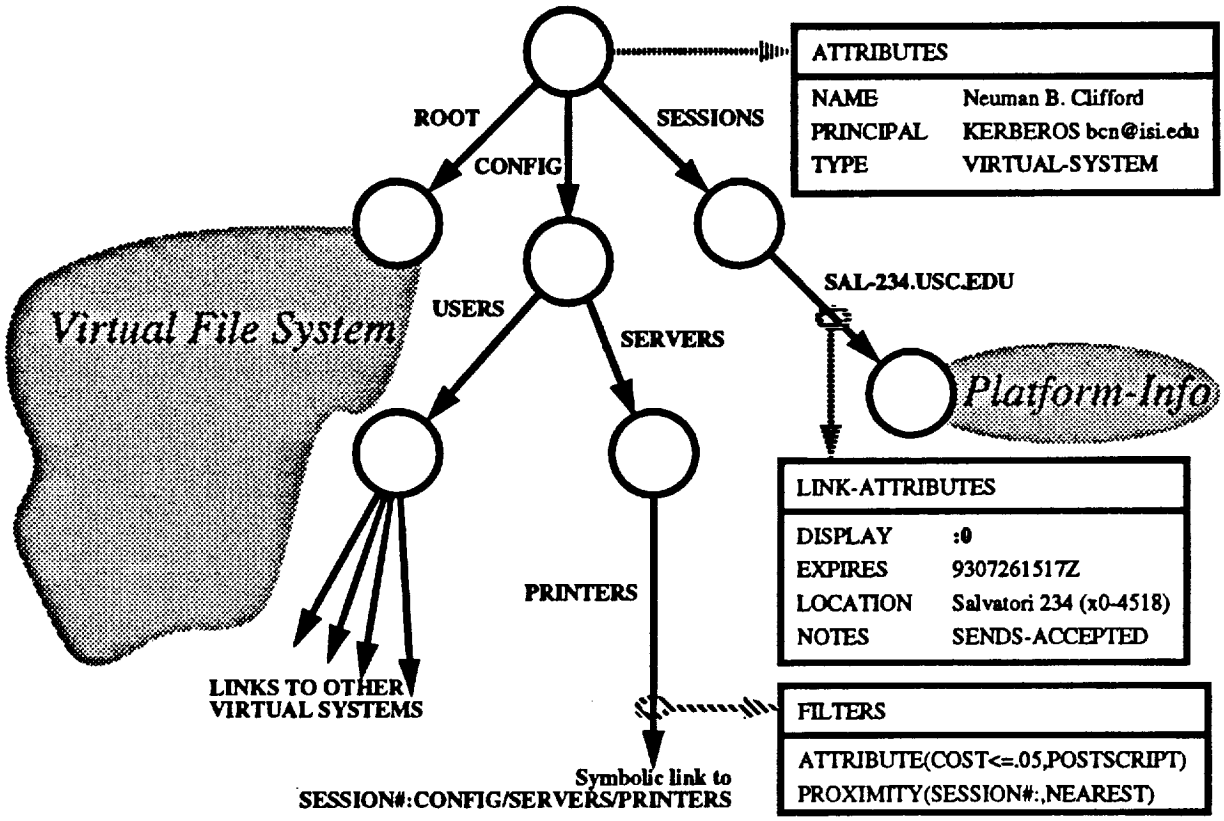


Figure 1: Structure of a Virtual System

In the figure, the directory CONFIG/SERVERS/PRINTERS defines the printers available to the user. If the user wanted to use the same printers always, the link would refer to a directory that explicitly named the printers to be used. Here, the user chose to define the available printers as a symbolic link to the directory of printers for the session. Because the target of the symbolic link begins with the string SESSION#: the rest of the name is resolved in the virtual system for the session, which is reachable through the node for the login platform and is part of the cloud labeled *Platform-Info* in the figure.

3.1.2 Union links

A user who wanted to choose from among the printer at home, the one at work, and those nearby, could use a union link to create a dynamic view of available printers. The CONFIG/SERVERS/PRINTERS link would refer to a directory with references to the printers at the user's home and office. That directory would include a symbolic union link to the directory of printers for the session. The resulting directory would appear to contain the union of the two directories.

3.1.3 Filters

Users should be cautious when allowing the system to select servers for use in unfamiliar locations. Business travelers have been taught this lesson by unscrupulous alternative telephone operator services that charge exorbitant fees for long distance phone calls. Using Prospero, users are able to constrain the selection of servers by applying a filter to a directory. A filter is a function attached to a link that modifies the result of a directory query. In figure 1 the user applied the **attribute()** filter to restrict the selection of printers to those charging no more than 5 cents per page, and supporting

PostScript. Of the printers selected by the **attribute()** filter, the **proximity()** filter selects the closest, provided that the implementer of the **proximity()** filter devised an appropriate heuristic.

A filter can also change the order in which servers are presented to the user setting the COLLATION-ORDER attribute of the links it returns. This allows filters to sort servers according to user specified criteria.

Prospero supports two kinds of filters: loadable and predefined. Loadable filters are dynamically loaded and executed during name resolution (their present implementation is not portable and presents security problems). Predefined filters are compiled into the name resolution library. Predefined filters have registered names and must be present in the name resolver (or on the server for some filters). Predefined filters must be widely supported to be useful; therefore they usually provide general operations such as selection based on the values of attributes.

3.1.4 Supporting mobile platforms

Our discussion so far has assumed that a stable platform exists for each session, and that a server directory has been defined for each platform. The discussion has ignored problems related to mobile platforms. The mobile platform problem can be addressed by another layer of indirection: the platform might itself use a filter in the definition of its server directory. Such a filter might locally broadcast a query in search of nearby platforms willing to provide such a directory. The resulting directory would include links for each server that responds.

3.1.5 Presenting selections to the user

When selecting a server, an application can indicate additional constraints to be applied by specifying additional filters. Once all filters have been applied, if the result is a single link, the referenced server can be used. For example, the NEAREST argument to the **proximity()** filter in the figure might result in the automatic selection of the nearest printer.

If the directory contains multiple links after the application of all filters, then the user could be prompted through a dialog box to select one. This might occur if the argument to the **proximity()** filter were specified as NEAREST 5, resulting in the return of the 5 nearest printers. Once selected by the user, the choice should remain in effect until some event specified by the user, such as a change in location, or a change in the list of available servers.

4 The User Location Problem

The user location problem is a second problem that illustrates the requirements imposed on a directory service by user mobility. In centralized systems, locating an active user is easy; users are either logged in, or they aren't. If logged in, the system records the the terminal in use and makes this information available to applications such as `finger`, `write`, and `send`. In a distributed system, the problem is considerably more complex.

A common approach to the user location problem is to replicate the data on all hosts. This is the approach taken by `rwho`. Systems broadcast the names of the users that are logged in, and others store this data locally where it can be searched by the user or application. The primary drawback of this approach is that it doesn't scale very well. A second concern is privacy; users might not want others to know where they are logged in, or that they are logged in at all.

A different approach is taken by the Zephyr [2] system at MIT's Project Athena [1]. Zephyr provides a single database that may be consulted when a user's location is needed. This database is replicated for reliability (technically, Zephyr provides a notification service that relies on this database, but it is the database that is of interest here). Zephyr addresses privacy because each user decides whether to register a session with Zephyr, and to what classes of other users the login location is to be visible. Zephyr does not provide fine-grained control over access to user location data. Though suitable for a large campus, the use of Zephyr as a user location database does not scale across administrative domains.

4.1 Using Prospero to Solve the User Location Problem

A third approach is to use a directory server to store user location information. Such a directory server would have to tolerate frequent updates. If a user is to be able to specify the principals who can obtain his or her location, then support for fine-grained access control is also necessary. Finally, it must be possible to authenticate both updates and queries.

The Prospero Directory Service already maintains information about a user's virtual system. This information has been extended to include information about login sessions. At login, an entry is added to a list of sessions, and at logout the entry is removed. By associating an expiration time with the entry, it is possible to detect sessions that are not properly terminated. By placing an access control list on the list of sessions, a user can specify on a per-principal basis the individuals to which the session is visible. The SESSIONS link in figure 1 points to the directory that maintains a list of sessions.

Prospero is a distributed directory service, and it is likely that user information will be distributed across a large number of systems, maintained by multiple organizations. Each directory server enforces its own access control. As such, if a user's directory information is stored on a trusted server for the user's organization, the user's privacy depends only upon the security of that server.

Through its support for customization, Prospero allows a user to define a set of colleagues, and the name used to refer to each. This set initially contains the names of other users in the local organization, with users beyond the organization named hierarchically based on the name of the organization to which they belong. Users can define their own short names for remote colleagues, after which they are referred to no differently than if they were local. This is represented in figure 1 by the CONFIG/USERS directory. In fact, this customization mechanism allows a user to define the set of colleagues considered local for use by `finger`; when run with no arguments it displays the locations of users currently active and identified as colleagues by the user.

When using modified versions of commands such as `send`, `talk`, or `finger`, the name of the target (another user) is specified relative to the CONFIG/USERS directory. The command consults the directory server to determine the location of the target user, and if found, performs the requested operation.

5 Establishing a Session

The solutions to both the server selection and the user location problems depend on several operations being performed when a new login session is established. This section describes what happens when a user logs into a system supporting the Prospero login program. In this discussion, the platform is the processor on which the application (in this case the login program) runs.

When a user attempts to log in to a system running the Prospero login, the system uses the name of the user to find the user's virtual system. For local users the virtual system is found in the directory of virtual systems for the local site. For remote users the virtual system is found starting from a directory that identifies other sites. The PRINCIPAL attribute associated with the user's virtual system is examined and used to select an appropriate authentication method. The user is authenticated and the login program determines whether the user is authorized to use the resources of the platform.

Once logged in, the namespace is defined by the user's virtual system and the virtual system alias SESSION#: is defined as the virtual system of the platform on which the user logged in. Next, if the user was suitably authenticated, an entry is made in the SESSIONS directory of the user's virtual system recording the platform, the login time, an expiration time, and other information associated with the session. During a session, name resolution occurs in the name space defined by the user's virtual system. When the session is terminated, the entry in the SESSIONS directory is removed.

6 Status

The Prospero Directory Service has been available since December 1990 and has been used to support the integration of information services on the Internet. The recent release of Version 5 of Prospero allows it to be more easily integrated with other applications. This paper described some of the ways that Prospero can be used to support integrated location-independent computing. Prospero presently provides the basic mechanism needed to address the problems discussed in this paper. We have started work on the user location problem as part of the implementation of a Prospero-based login program. We have not yet started work on the server selection problem, but plan to do so in the near future. To find out more about Prospero, or for directions on retrieving the latest distribution, send a message to info-prospero@isi.edu.

7 Summary

Pervasive computing places new demands on directory services. Among the demands is a need to support transient data, fine-grained authorization for queries and updates, and the ability to support dynamic (functional) bindings from names to servers and objects. These requirements are met by the Prospero Directory Service, which can play a role in support for truly integrated, location-independent computing.

Acknowledgments

Many individuals contributed to the design and implementation of Prospero. Ed Lazowska, John Zahorjan, David Notkin, Hank Levy, and Alfred Spector helped refine the ideas that ultimately led to the development of Prospero. Kwynn Buess, Steve Cliffe, Alan Emtage, George Ferguson, Bill Griswold, Sanjay Joshi, Brendan Kehoe, and Dan King helped with the implementation of Prospero and Prospero-based applications. Celeste Anderson, Sio-Man Cheang, Gennady Medvinsky, Santosh Rao, Eve Schooler, and Stuart Stubblebine commented on drafts of this paper.

References

[1] George A. Champine, Daniel E. Geer Jr., and William N. Ruh. Project Athena as a distributed computer system. *IEEE Computer*, 23(9):40–51, September 1990.

[2] C. Anthony DellaFera, Mark W. Eichin, Robert S. French, David C Jedlinsky, John T. Kohl, and William E. Sommerfeld. The Zephyr notification service. In *Proceedings of the Winter 1988 Usenix Conference*, pages 213–219, February 1988.

[3] Stephen P. Dyer. The Hesiod name server. In *Proceedings of the Winter 1988 Usenix Conference*, pages 183–189, February 1988.

[4] B. Clifford Neuman. The Prospero File System: A global file system based on the Virtual System Model. *Computing Systems*, 5(4):407–432, Fall 1992.

[5] B. Clifford Neuman. *The Virtual System Model: A Scalable Approach to Organizing Large Systems.* PhD thesis, University of Washington, June 1992. Department of Computer Science and Engineering Technical Report 92-06-04.

[6] Sun Microsystems. *Yellow Pages Protocol Specification*, February 1986. In Networking on the Sun Workstation.

This research was supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Centers, Digital Equipment Corporation, and the Advanced Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies. Figures and descriptions in this paper were provided by the authors and are used with permission.

The authors may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511, email bcn@isi.edu, swa@isi.edu, prasad@isi.edu.