# Broadening the User Environment with Implicit Trading

Lea Kutvonen and Petri Kutvonen

Department of Computer Science, University of Helsinki
P.O. Box 26 (Teollisuuskatu 23), SF–00014 University of Helsinki, Finland
Lea.Kutvonen@cs.Helsinki.FI, Petri.Kutvonen@cs.Helsinki.FI

**Abstract**

In a distributed system, a trader can be seen as a management tool for providing a convenient and efficient environment to the end users. Trader offers a mechanism for providing access to more services directly from the user's workstation with limited resources. Services can also be arranged to appear fault tolerant and load balanced. This kind of service management imposes new requirements on the trader. In this paper the concept of implicit trading is introduced. With implicit trading old applications can be easily integrated with the trader. Moreover, trading can be considered as a new level of indirect symbolic naming and binding. Thus implicit trading offers a natural way of exploiting the trader.

Keyword Codes: C.2.4, H.2.8, K.6.4
Keywords: Distributed Systems, Database Applications, System Management

## 1. INTRODUCTION

Distributed systems are often abstracted using the client–server model. In this model, a trader is usually considered as a neutral intermediate module between clients and servers [1, 2]. Servers announce their properties to the trader and the clients search through the service descriptions stored by the trader for what they need. Capabilities of servers are described using attribute values that are sent to the trader in export messages. The needs of the clients are passed to the trader in import messages as search criteria consisting of limitations on the attribute values. The trader gets all information needed in export messages from servers and in import messages from clients. When the trader has found a suitable service offer matching the client's request, it passes the address of the server to the client.

Trading can also be considered as an enhanced form of naming and binding. In trading the name of a service may be replaced by a description of the desired properties of the server. The components of the descriptive name could be, for example, type or behavior of the server, descriptions of required or desired properties, constraints on some properties and relative location of the server (relative to the users location). This attribute based naming may be used in a user friendly manner. Examples of this naming scheme can be found in [3]. Traditional binding forms a fixed one–to–one relation between a name and an address. In trading, a "name" may be resolved to many alternative addresses. The resolution rule is fixed but it utilizes dynamic data on addressees. When trading is considered as a form of dynamic binding the addressee may be viewed as a persistent object as in [4] or the binding operation may heavily trust on dynamic, virtual attribute values of a looser group of objects as in [2, 5]. The OMG approach sets up first the group of services to be offered and then creates a persistent distributed object for each service to guarantee the accessibility of the service. The service object may be aware of several server instances. In the latter approach the knowledge of existing server instances in a distributed system is stored directly into the trader. The availability of a service is not guaranteed by the trading mechanism itself.

The custom of using a separate description–to–address translator has become prevalent. It would be more natural if the description–to–address translation were hidden into the server calling modules as it happened with symbolic name handling earlier. In the early days of computing it was customary to refer to devices with absolute hardware addresses. Later on, symbolic names were introduced. Operating systems modules began to operate with symbolic names and hid all handling of hardware addresses. Data was stored into files consisting of data blocks and only the disk controller was aware of the details of cylinders, heads and sectors. Respectively, fixed name–to–address relations are now more and more replaced by name servers and traders. We would like to use the server descriptions like any symbolic names. Even when the client is constructed to use local servers or data files only, the usage of traded execution services or information services can be arranged rather similarly to that of RPC (Remote Procedure Call) [6]. This can be done by a new model of *implicit trading*. In this model, somebody acting on behalf of the client replaces local server calls with remote service calls to the trader. The trader then selects a server according the search criteria given by the acting client. The trader can also implement a policy of load balancing that it follows to select between several suitable servers. After selection the trader invokes the server. The invocation of servers from the trader leads to possibilities to enhance the fault tolerance of services.

In this paper we introduce the trader model used in the project Dryad at the University of Helsinki. We have taken the ODP trader model [2] as a starting point and augmented it. In the following description we concentrate on the features differing from the ODP trader model and thus the description excludes many central features of traders. As one of our goals is to hide the description–to–address translation to operating system modules, we introduce a implementation model of implicit trading. We also give some examples of the applicability of the model.

## 2. THE DRYAD TRADER MODEL

We use the following terminology: By a user we always mean a human user. The computing environment is represented by clients and servers sharing the computing tasks given by the user. The trader functions between clients and servers in order to relieve the clients from locating the servers. A client is a user process and a server is a program, a process, a resource or an information object (for example data describing a user). Client and server only denote the roles of application modules with respect to the trader. The same application module can play both roles.

The end user has two different methods of exploiting capabilities of the trader. The first method is interactive usage by means of a specific user interface software. This software can be run on user's own terminal and it interprets the user's choices to a format understood by the trader. Servers may be shown as lists and each server may have its description shown. Also, the user may inquire different attribute values and make search operations. The alternative method is to weave trader calls into applications. The end user may not necessarily know that the used application exploits trading.

The Dryad trader model is illustrated in Figure 1. In the model the requirements on servers and clients caused by trading are minimized. On the server side, servers are categorized either as active or passive, and the responsibility of acquiring the information needed in the trading decision process is moved from the clients and servers to the trading system itself. To the client side, the implicit trading method has been added to move the responsibility of expressing the server selection criteria in some situations from the client to a "secretary process".

On the application level the communication between the application and the trader can be explicit. The application programmer uses direct calls to the trader and can also request the user to collect all information needed before calling the trader. This style is used for example in the interactive user interface mentioned earlier. Other examples of utilizing explicit trading include a desktop publishing system requiring knowledge about available printers and a statistical analysis system searching for free processors for computing intensive tasks. *Explicit trader calls* follow the ODP style of trading and are illustrated by user *E* in Figure 1.

If the application program is unfamiliar with the trading concept or it is desirable to hide the description–to–address translation, the explicit trading method is unsuitable. The programmer can traditionally code references to server names (program, resource names) he presumes to exist into the application. Thus, on the application level there is no knowledge about the trader, but the exploitation of trading takes place on the level of the underlying system. The supposed server names may be viewed as virtual server names. References to virtual servers are interpreted by a "secretary process" (called

Daphne) by translating them onto service request operations for the trader. The Daphne process collects the needed trading information from user specific trading profiles and virtual server specific trading profiles expressing the user and system requirements for the server to be selected. Profiles consist of partial server selection criteria sequences that are passed to the trader in service requests. We call this trading method *implicit trading.* It is illustrated by user *I* in Figure 1.

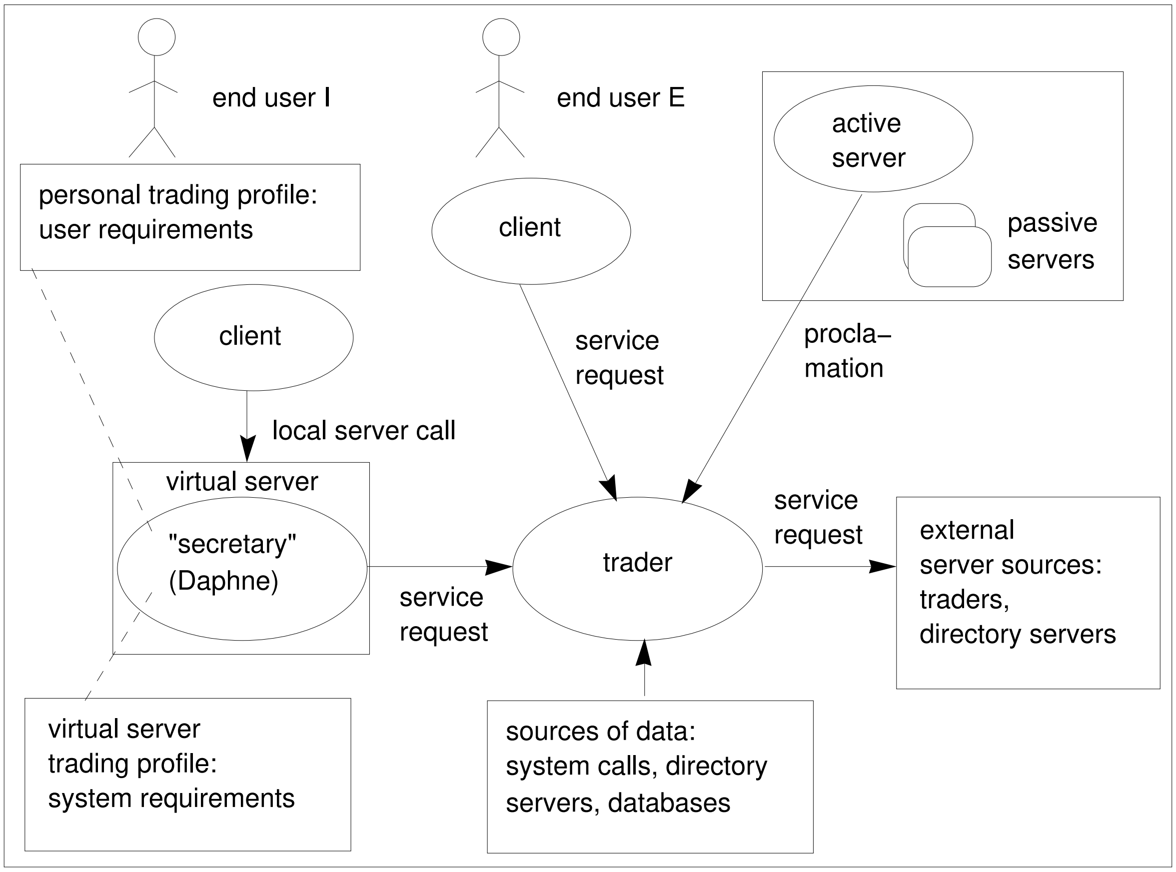


Figure 1. Overall system structure of the Dryad trader.

As discussed above, servers can be categorized either as *active* or *passive servers*. Active servers are continuously running servers, i.e. processes that usually start at the system startup time. Passive servers are activated through active servers. For example, a database server is an active server and a normal program file is a passive server. Active servers *proclaim* themselves to the trader when they are started. They may also proclaim the group of passive servers they are responsible of (Figure 1). When an active server is closing down it withdraws itself and its group of passive servers from the trader.

The proclamation of servers differs from the export operation in ODP trader model [2]: the proclamation only contains addresses and types of the servers, but no attribute values. It is not necessary for the servers to be able to declare their attribute values. Instead, rules for the evaluation of these attributes are stored in the trader database. These rules may include for example normal operating system calls. By using proclamations instead of exports we allow servers that are passive or not knowledgeable enough to cooperate with the trader. We also gain access to more up–to–date attribute values with less overhead, as the trader is able to re–evaluate the values only on demand.

## 3. STRUCTURE OF THE DRYAD TRADER

The Dryad trader consists of *an object database, a monitor* and *a concierge,* as shown in Figure 2.

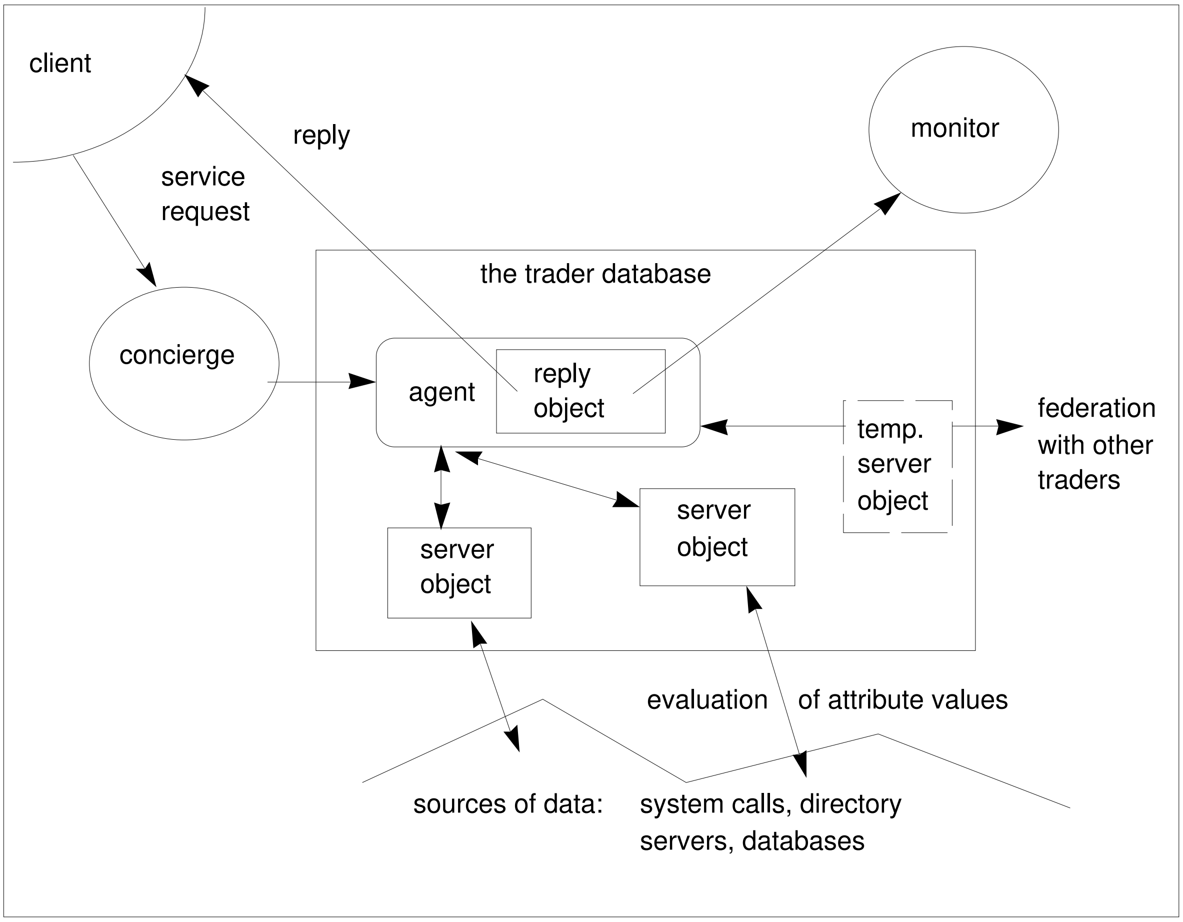


Figure 2. The building blocks of the Dryad trader.

The database is a simple relational database that includes objects. Each object contains private data, private data evaluation rules and public methods for other objects to obtain information from it [7]. The basic object type describes a server. Each server object includes a server description, attribute evaluation rules and temporarily stored attribute values. For each incoming message an active agent object is created by the concierge to process the request. These agents may run in parallel and each of them may start parallel attribute evaluations in several server objects by calling server methods. Agents must be able to handle missing and undefined attribute values in a way suited to the request.

It is characteristic of the Dryad trader to acquire information from external sources, such as federating traders and databases or directory servers. To accommodate results of these queries, temporary server objects are introduced. Temporary objects are created only on demand and are deleted through timeouts. For example, it is not feasible to store information about all devices into the trader database. Instead temporary objects may be created on demand to represent the group of devices under inspection. The trader database includes rules for requesting information from the information repository and for creating temporary objects based on the response.

The trader monitor (Figure 2) is responsible for invoking the selected servers. As the server selection mechanism of the trader selects only servers that are in a working condition at the time of invocation request, some fault tolerance is achieved by using the trader. Furthermore, the invocation mechanism may be structured so, that the invoker monitors the execution of the server until it is successfully done. It is even possible to invoke a group of servers instead of one, in order to make sure that at least one of them succeeds. The client accepts the first result or requests the group to vote for the result.

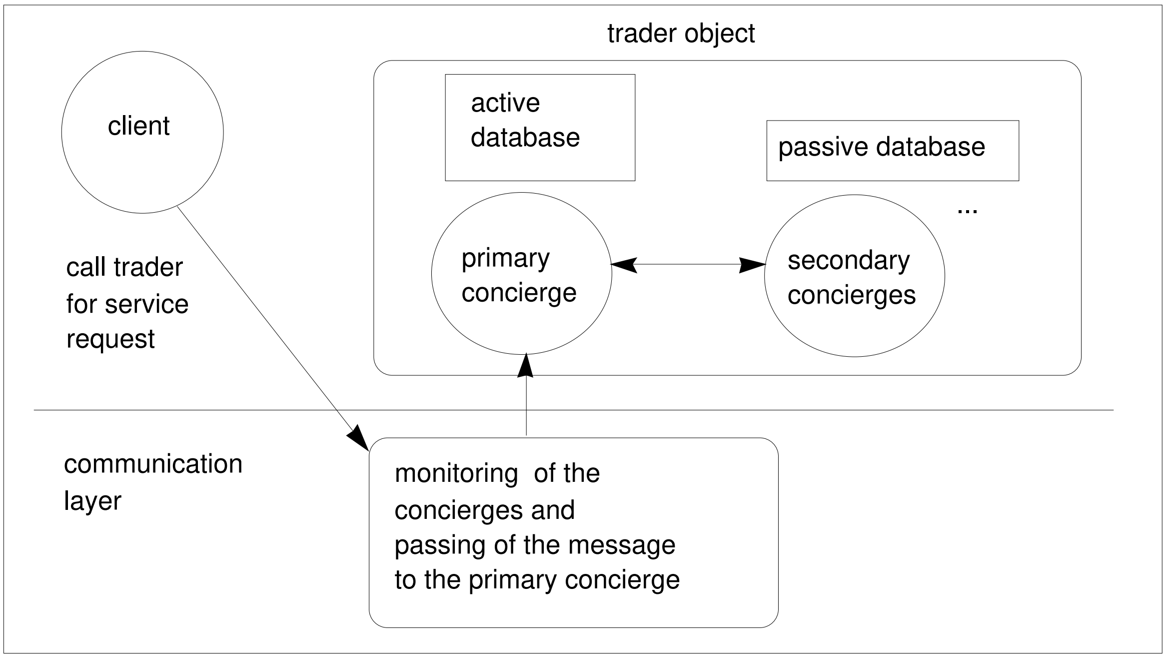


Figure 3. Replication mechanism for the trader.

Fault tolerance of the trading service is provided by using replication as shown in Figure 3. A replicated concierge process is resident in all hosts that are able to take the responsibility of trading service. The concierges negotiate which of them will be in charge of the trading service and the address of the selected primary concierge is passed to the communication layer. The selected concierge offers a service interface to clients from this announced host. In case of failure the concierges elect a new primary that starts a new trader database incarnation. Each concierge has a copy of the basic trader database that consists of static data. All the dynamic data in an active database lives only a very short time, thus the replication of dynamic data is not feasible.

## 4. IMPLICIT TRADING MECHANISM

The idea of implicit trading implementation is based on the fact that all references to servers can be reduced to references to files. Alternatively, a request for a file can be viewed more abstractly to mean a request for service. Service can be either execution of a program file or retrieval of information from a (output) file. Before a program or a remote call can be executed an executable file must be read by the operating system. Resources, for example devices or even processes, are frequently represented as files in the operating systems [e.g. 8]. Thus, mediation of servers through a trader can be viewed as a new kind of file system.

Except the files themselves, an essential part of a traditional file system is the process interpreting the symbolic file names and managing the file storage [9]. All references to files included in the file system are passed to the file system process. The directory system of a computer may contain several subtrees, each managed by a different file system process. The file system processes are often external to the operating system kernel and thus changeable without operating system modifications.

The main function of the file system process is to interpret the names and provide the requesting process the contents of the named file. It uses an internal policy, which can be chosen freely to satisfy the function of the file system. Traditionally this policy is used to show to the users exactly what has been stored, but the stored information may as well be the description of data that should be generated and shown.

We have created our own file system manager process (the "secretary" process Daphne) to take care of the conversion between servers and contents of files. To the clients of Daphne a consistent view to servers is offered by setting up a group of virtual server files, representing either processing servers or information objects. Thus the user sees names of program files and data files.

Daphne process constructs the contents of the virtual information files with the help of the trader. Daphne interprets the service references to trader calls as shown in Figure 4. When a client makes a reference to a file, for example calls "read F3" (see arrow 1 in Figure 4), the manager process sees the name F3 (2) and starts to interpret the name. After reading the user (3) and virtual server profiles (4) containing some selection criteria, the Daphne process generates the service request message to the trader (5) and finally, after receiving the response (6) passes it to the client as if it were the contents of the file F3 (7).

Constructing the contents of virtual processing server files is a simpler procedure. Daphne generates a program code that includes steps of formulating a service request message for the trader, sending the message and processing the results. Thus Daphne forces the executor of a file to execute the service request.

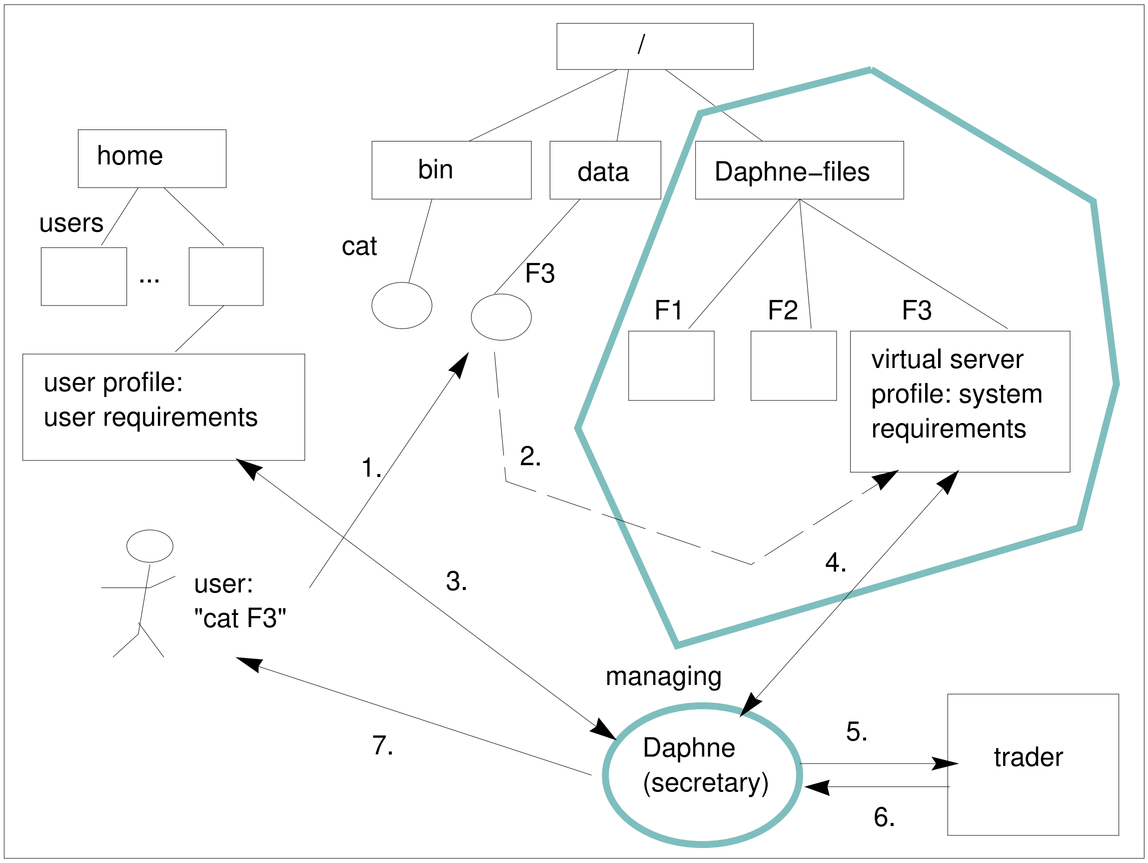


Figure 4. An example of implicit trading.

As the role of profiles representing the user and virtual server requirements is essential, the representation format and contents of these files must be designed carefully. The files include server selection criteria to be passed to the trader. We have allowed the server selection criterion to be split into several sequences, each labeled with a priority. If a criterion has a high priority, the selected server must fulfill it. And, if a criterion has a low priority, it is interpreted as a preference and it is used only to select between several servers filling the high priority criteria. The criteria are interpreted in priority order. If there are available servers filling all the *requirements* (high priority criteria) the selection of servers is continued with *preferences* (low priority criteria) as long as they leave at least one server available. With careful selection of priority levels in user and virtual server requirement criteria different *trader roles* [5] are achieved.

The Daphne process may also be contacted remotely, in the same way as it is possible to utilize other existing file system processes remotely. This means that the Daphne files can be accessed from remote computers. The assortment of virtual servers in different computers may differ. And if all the Daphne processes are unreachable there can be a small local assortment of nontraded servers available.

## 5. APPLICABILITY OF IMPLICIT TRADING

Implicit trading is an easy way of utilizing old applications with traders and bringing them into new environments. It also offers a natural model of indirect, symbolic naming and binding. Load balancing and fault tolerant services may also be easily arranged with all kinds of traders. However, we would not recommend trading in a real time system because the related goals are incompatible. Late binding does not allow scheduling with hard deadlines.

Via trading, especially via implicit trading, non−native services can be instantiated in desktop computing equipment (multi−purpose workstations, personal computers, microcomputers). As the Daphne files are made visible within the desktop computer's file system, all services in the network can be arranged to appear as local services. Only the ability of accessing remote file systems over a network is required. The access method may vary (e.g. NFS or Novell). To test the applicability of the implicit trading method we took a PC equipped with PC−NFS network package and Windows 3.1. We added the capability of making queries to X.500 [10] running elsewhere to the Notepad application within Windows. The products involved in the test were regular commercial software and we modified neither the source code nor the object code.

Another practical usage of implicit trading is the variation of the user environment. When a user has a personal workstation, with some effort one can arrange that all services needed are easily accessible from the workstation by editing the session startup scripts somewhere in the system. When the user for some reason uses a different kind of computing equipment, he has to arrange everything again. The screen may have different size and the window manager has to be started with different type of parameters. Maybe the new equipment utilizes loudspeakers and microphone or colors, but the environment is not tuned to handle these features. If the session startup scripts are generated with the help of the trader, the session properties can be automatically adjusted to the capabilities of the equipment. Also, in failure situations some basic services may be absent, but the user can be allowed to use the system with a reduced setup. For example, if a mail server is down, users should be allowed to log in into the system without the mailing capability and not be forced to hang in the login script forever as it happens in many systems.

Trading is an effective tool for system management. The trader may be used to impose policies on usage of the services. For example, a set of internal selection criteria may force the trader to search the least loaded CPU to run a program on. Also, trading profiles may contain information on the resource requirements of the server referred. By combining these, managing a processor farm in a supercomputing environment with large, time–consuming tasks [11] could be conveniently achieved.

With implicit trading the users can be offered a more fault tolerant feel of the system. If they use virtual servers instead of physical servers, their work is not hindered when a physical server fails. Via virtual servers always an operational physical server is selected. Also, the system manager is able to hide most of his management operations. When a physical server is about to be removed, it can be replaced with a virtual server. Thus the references to the server can be redirected and the users can continue their regular routines undisturbed.

## 6. THE DRYAD PROJECT

Dryad is a research project working on the trading model [12]. In our development work we have followed the lines of ODP trading model [2] whenever our goals have not been contradictory. We feel that a trader standard is essential in the cooperation between organizations. We also feel that we have to respect the autonomy of each organization and their system management. Organizations have different needs for trading and this should be reflected by the service types and properties known of the services by the trader. We have tried to isolate the trader matching mechanism and the data on servers as far as possible. We have also tried to isolate the applications in the system from the data known by the trader. This means that application programs preserve their portability in spite of differences in traders. It also leads to some difficulties in federation because service types may differ between traders and matching these may need human intervention at the time when the cooperating organizations formulate contracts about letting each other to exploit their services and resources.

A practical goal of the project is to construct a prototype trader that can be utilized in our departmental network which consists of workstations, terminals and PC's. In this environment we want our students and researchers to be able to effectively and easily access even the infrequently used services from any workstation they choose. The implicit trading model is just one aspect necessary in this task.

The implementation of the Dryad trader takes place in a UNIX environment. We have based the implementation of trading protocol directly on RPC [13]. The trader information storage is implemented with a locally developed special−purpose database tool called Debbie [14]. The implicit trading has currently been built using NFS protocol [15] and automounters [16]. As both of these are RPC based, our only requirement for the distributed computing platform is RPC. The user interfaces follow the OpenLook recommendations [17]. The implementation of the trader has begun and a prototype should be operational in the autumn of 1993. Beside the trader prototype we also plan to implement network management software cooperating with the trader.

## 7. CONCLUSION

We consider trader as a management tool that offers flexible means to provide services to end users. With trading, some fault tolerance and load balancing can be achieved, as well as an illusion of a centralized, homogenized system. In this paper, we have introduced the new concept of implicit trading. Implicit trading gives an easy way of delivering services over the network even to the smallest workstations with limited resources. It also allows descriptive naming and utilizes hidden dynamic binding. In our project we have implemented implicit trading. Our experience shows that implicit trading is both feasible and useful.

## ACKNOWLEDGMENTS

Dryad research project is supervised by professor Martti Tienari at the University of Helsinki. The project was started in the autumn of 1992 and it has financial support for two years from Finnish Technology Development Center TEKES and four Finnish industrial partners.

## REFERENCES

1. Architecture Projects Management Limited, An Overview of ANSAware 4.0. Document RM.099.00. March 1992.
2. ODP Trader Rappourteur Group, ISO/IES JTC1/SC21 N7047 Basic Reference Model of Open Distributed Processing. WG 7: ODP−Trader. Revised text. Ottawa, May 1992.
3. Neuman, B. C., The Prospero File system. A Global File System Based on the Virtual System Model. Workshop on File Systems, May 1992.
4. Digital Equipment Corporation, Hewlett−Packard Company, HyperDesk Corporation, NCR Corporation, Object Design, Inc., SunSoft, Inc., The Common Object Request Broker: Architecture and Specification. OMG Document Number 91.12.1. Object Management Group and X/Open, 1991.

5. Wolisz, A., Tschammer, V., Dynamic Binding of Service Users and Providers in an Open Services Environment. Computer Networks, 1991.
6. Birrell, A. D., Nelson, B. J., Implementing Remote Procedure Calls. ACM Transactions on Computer Systems, Vol. 2 No. 1 (February 1984), pp. 39–59.
7. Cattell, R. G. G., Object Data Management. Object Oriented and Extended Relational Database Systems. Addison–Wesley, 1992.
8. Presotto, D., Pike, R., Thompson, K., Trickey, H., Plan 9, A Distributed System. Proceedings of the Spring 1991 EurOpen Conference, Tromsø, May 1991.
9. Satyanarayanan, M., Distributed File Systems. In Mullender, S., Distributed Systems. ACM Press Frontier Series, 1989.
10. Kille, S. E., Implementing X.400 and X.500: The PP and QUIPU Systems. Artech House, Inc., 1991.
11. Koski, K., Lindberg, K., Saarinen, S., The CSC Metaflops Project. /csc/news (a publication of the Centre for Scientific Computing, Finland), Vol. 4 No. 3 (October 1992).
12. Kutvonen, L., Overview on the DRYAD Trading System. Internal Report of Project DRYAD. Department of Computer Science, University of Helsinki, 1993.
13. Sun Microsystems, RPC: Remote Procedure Call Protocol Specification. RFC–1050. Sun Microsystems, April 1988.
14. Kutvonen, P., Kutvonen, L., Debbie, A Database Management System. Internal Report of Project DRYAD. Department of Computer Science, University of Helsinki, 1993.
15. Sun Microsystems, NFS: Network File System. RFC–1094. Sun Microsystems, March 1989.
16. Pendry, J.–S., Amd – An Automounter. Department of Computing, Imperial College, London. January 1990.
17. Sun Microsystems, Open Look Application Style Guidelines. Sun Microsystems, 1989.