The Federative Trader Model of the Multiware Platform

Luiz Augusto de Paula Lima Jr.[1]
Edmundo Roberto Mauro Madeira[2]

Abstract

This work proposes a model of TRADER considering specially the aspects related to the establishment and management of FEDERATIONS OF TRADERS inside the framework of the Reference Model for Open Distributed Processing (RM-ODP). The modules of the proposed model are commented and the operations at the interfaces are listed. Finally, a protocol for the communication between the trader and its administrator is defined and the federation establishment process is discussed. A prototype of this model was implemented inside the Multiware Platform which is being developed at the University of Campinas.

Sumário

Este trabalho propõe um modelo de TRADER considerando especialmente os aspectos relacionados ao estabelecimento e gerenciamento de FEDERAÇÕES DE TRADERS dentro da estrutura do Modelo de Referência para Processamento Distribuído Aberto (RM-ODP). Os módulos do modelo proposto são comentados e as operações nas interfaces são listadas. Por fim, um protocolo para a comunicação entre o trader e o seu administrador é definido e o processo de estabelecimento de federação é discutido. Um protótipo deste modelo foi implementado dentro da estrutura da Plataforma Multiware que está sendo desenvolvida na UNICAMP.

Keywords: open distributed processing, ODP, trader, interworking, federation of traders.

1. INTRODUCTION

The advances in the communication technology with high transmission rates, improved security and low error rates, the future (and present) needs of the users for integration of auto-

1.Institut National des Télécommunications - LOR - 9, rue Charles Fourier - 91011 - Evry Cedex - France
E-mail: lima@hugo.int-evry.fr or lima@dcc.unicamp.br
2.Universidade Estadual de Campinas - DCC - Cx. Postal 6065 - 13081-970 - Campinas - SP - Brazil
E-mail: edmundo@dcc.unicamp.br

mation islands, computer support cooperative work (CSCW) and architectures for open distributed services and the progress in the standardization (RM-ODP) made an environment for open distributed processing not only possible, but also indispensable to attend the new demands of the users (e.g., decision-making systems [1], distributed artificial intelligence (DAI), etc.).

The open environment has the advantage of being unlimited and free to admit any kind of user, component or application. Therefore, this environment is characterized by heterogeneity and decentralization, but, at the same time, it must assure the autonomy of each component or local environment.

In an Open Distributed System, it is highly desirable the existence of a means to make dynamic selection of computational services that satisfy certain properties. For this reason, the *trader* is a key component in such an environment. Its role is to manage the knowledge of the currently available services and to find service offers that match the clients' requirements [2].

The grouping of traders in *Federations* allows that distinct distributed systems work together but, at the same time, keep control of their own domains [3].

In this work, we present a model which can be used as a basis for an implementation of a trader, and we give special attention to the aspects related to Trader Federations. This is done according to the standard of the ISO for ODP[1] [4][5][6][7][8][9][10]. This means that autonomy, decentralization and encapsulation are the fundamental principles that will lead all the following proposals.

Since a number of traders have been implemented in several parts of the world, recently, a project for the interworking of these traders has been launched [11] to allow interaction between these heterogeneous trader implementations. This project will also give participants experiences in interworking for an open environment and in providing an excellent test-bed for interworking across different type systems, different organizations with different policies, different networking, different operating systems and so on.

The work reported in this paper gives contributions to the current international standardization process.

A prototype of the model was implemented inside the framework of the Multiware Platform [12], which aims at providing support for the creation of distributed applications.

2. A MODEL FOR A TRADER

There are basically two groups of activities performed by a trader:

• the management of the data base related to the (static and dynamic) information held by the trader; and

• the execution of the operations using the information.

We propose a model that consists of the refinement of these two "modules", considering separately local functions and those related to the establishment of federations and federated operations, and also putting in different modules the management of the static and dynamic information.

1.The ISO documents for the trader are currently "*committee drafts*". But it is hoped that they will become "*draft international standards*" by July 1995. The last version of the ODP Trading Function Standard was an output of the ODP Trader meeting in Southampton, July, 1994.

Figure 1 presents a model (see [17]) for the trader where we can identify three basic components (agents, from the enterprise viewpoint). They are:

- the client which is the importer or exporter of service offers;
- the administrator - the *local* one that deals with local management and the *federation* one which deals with the establishment of federations (section 3);
- the trader.

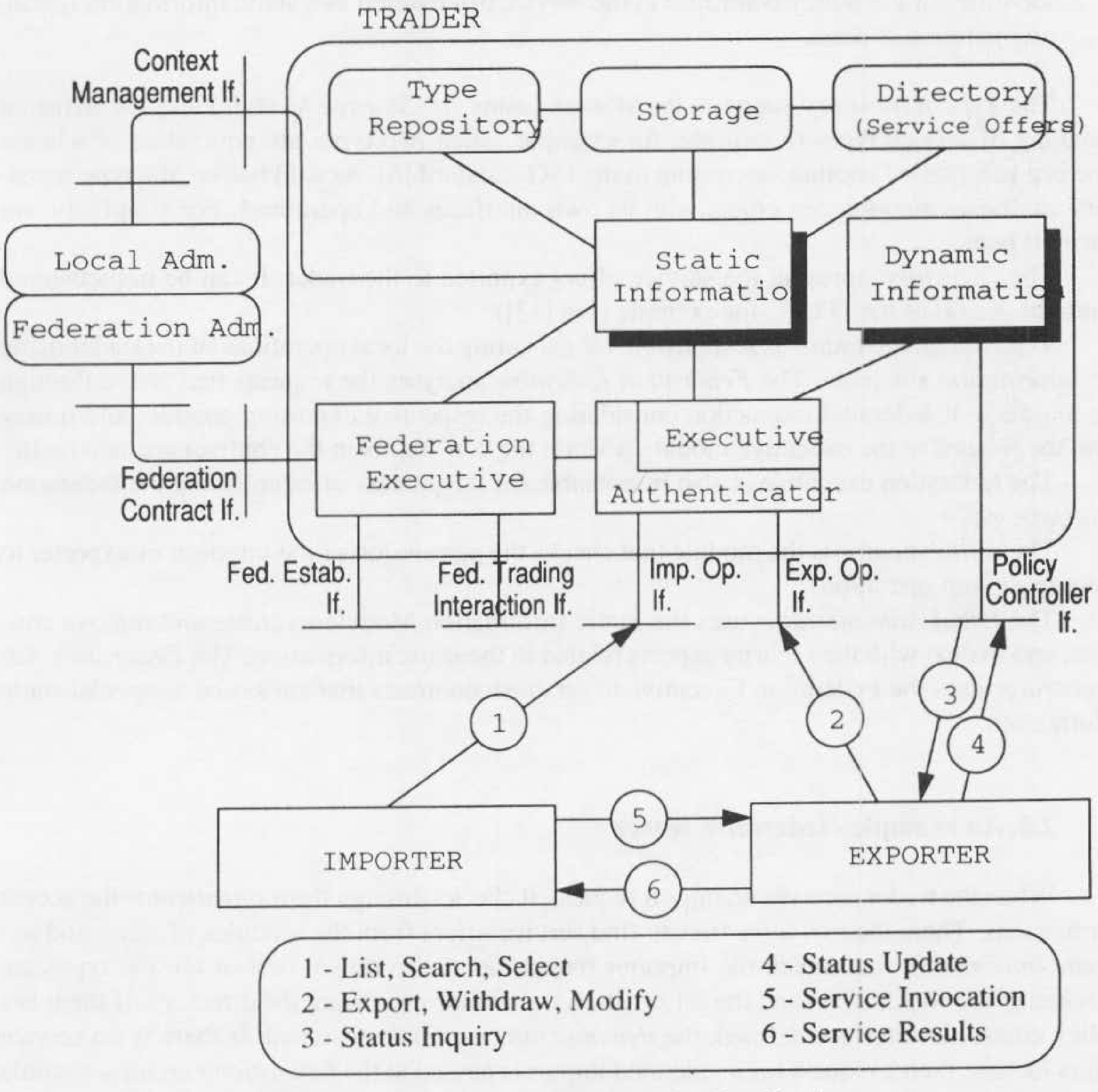

Figure 1 : A model for a trader. (if. = interface)

2.1. The description of the modules

Two modules are responsible for storage and information retrieval in the trader. They are:

- *Static Information Module*

This module concentrates the operations that handle the static informa:'on which corresponds to service types (in the Type Repository[1]) and service offers (in the Directory).

- *Dynamic Information Module*

It is responsible for updating the dynamic properties of a service offer. This module contacts the *Policy Controller* of the exporting object to obtain this information. If the identifier of the policy controller in the service offer stored as a static information is null, no operation is done.

The *Type Repository* supports the storage (using the *Storage Module*) and the dynamic matching of service types to indicate, for example, when two types are equivalent or when a type is a sub-type of another, according to the ISO standard [6]. As said before, the type repository can be an autonomous object with its own interfaces and operations. For simplicity, we placed it here.

The *Directory* stores all the service offers exported to the trader. It can be implemented using the X.500 of the ITU-T, for example (see [13]).

The *Executive Module* is responsible for executing the local operations of the trader using the information available. The *Federation Executive* analyses the requests that arrive through the interface of federated interaction considering the respective exporting contract and it may send the request to the executive module (in case the constraints in the contract are satisfied).

The federation executive is also responsible for the process of establishing the federation contracts.

The *Authenticator* is the module that checks the permission of the importer or exporter to execute a given operation.

The *Local Administrator* uses the Static Information Module to create and remove contexts, and to deal with the security aspects related to the static information. The *Federation Administrator* uses the Federation Executive to establish contracts that are stored as special static information.

2.2. An example - federative search

When the trader receives an import request, it checks through the *authenticator* the access permissions. Then, the *executive* tries to find service offers from the modules of *static* and *dynamic information* that match the importer requirements. To do so, first of all, the types are checked (*Type Repository*) and the information is then retrieved from the directory. If there is a policy controller interface defined, the *dynamic information* is searched. If there is no service offers locally, then a request for a federated import is passed to the *federation executive* module that checks the existence of suitable importing contracts in the search scope and then sends the request for a federated search to the remote trader.

In the remote trader, the search request arrives in the *Federation Trading Interaction* interface. Then, the *federation executive* applies the restrictions, rules and mapping functions which are specified in the respective export contract to the parameters received with the remote

1.The *Type Repository* can be located outside the trader and this module can be used to deal with the efficient communication with the "real" type repository.

search. Once all the constraints are satisfied, a new operation with possibly new (transformed) parameters is issued to the *executive* module that performs the normal operation and returns the found values (if any) to the *federation executive* module.

For example, when an operation

```
search (..., service_type, matching_criteria, scope, ...);
```

is performed in some *Federation Trading Interaction* interface, the *federation executive* applies the rules in the associated export contract transforming (if necessary) some parameters to meet the requirements specified in the contract and to make them understandable for the local trader. Then an operation

```
search (..., service_type', matching_criteria', scope', ...);
```

is issued to the *executive* module which goes on with the normal procedure.

2.3. Interfaces

From this model, we identify a set of interfaces that are needed to allow trader management and to attend the functionality that is required. The interfaces with the respective operations are presented in figure 2.

The interfaces for *importing* and *exporting operations* correspond to the *trading interface*. This is the most used interface in a *Trading Community* because it contains the basic operations.

The interfaces of *Context Management* and *Federation Contract* are used only by the local and federation administrators, respectively.

The interface of the *Policy Controller* allows the trader to obtain the dynamic information of a service offer.

The interface of *Federation Establishment* offers to other traders the possibility of establishing federation contracts between them.

Finally, the interface of *Federated Interaction* receives requests of federated imports and exports from other traders that are acting on behalf of their clients and with whom there is a federation contract established. There is a distinct interface of Federated Interaction for each exporting contract of the trader. All the operations in this interface have the same semantics and the same parameters that are found in the operations of the *Export* and *Import Operations* interfaces. But here all of these operations are submitted to the restrictions specified in the associated export contract.

3. THE ADMINISTRATOR

The local and federation administrators are responsible for the definition and the enforcing of the trading policies in the local and federation levels, respectively. The *local administrator* adds new service types to a given context, creates, destroys and renames service offer contexts and authorizes clients to use (for a search or an export) the several existing contexts. The role of the *Federation Administrator* is to prepare the *catalogue*, to request a catalogue from other trader, to decide when and with whom it should establish a federation, to accept, refuse or make proposals for federation contracts defining the policies to establish such federation. A *catalogue* contains information on the service types and on the extension of the service offer data

Interfaces	Operations
Export Operations Interface	EXPORT, WITHDRAW, REPLACE
Importer Operations Interface	LIST_OFFER_DETAILS, SEARCH, SELECT
Policy Controller Interface	STATUS_INQUIRY, EXPORTER_POLICY
Context Management Interface	CREATE_CONTEXT, DELETE_CONTEXT, LIST_CONTEXT, LIST_CONTEXT_CONTENT, AUTHORIZE
Federation Contract Interface	ESTABLISH_FEDERATION, DISTRIBUTE_CATALOGUE, REQUEST_CATALOGUE
Federation Establishment Interface	EXCHANGE_CONTRACT
Federated Interaction Interface	EXPORT, WITHDRAW, REPLACE, SEARCH, LIST_OFFER_DETAILS, SELECT

Figure 2 : Table of interfaces and their operations.

base that the trader will make available to other traders. It also specifies the permitted operations through federation (exporting trader policy) and the linking extension to other traders, i.e. if the exporting trader will pass on requests to other traders or not.

The administrators are located outside the trader to add the possibility of several traders being managed by only one administrator, which is desirable if we consider several traders in a same enterprise, for example. In other words, a trader must have only one administrator, but an administrator can manage several traders (as suggested also in [14]).

Our administrator is divided into two parts as can be seen in figure 3.

• The *Server Part*, that offers the following services to the trader :

- *send_catalogue :* using this operation the trader can inform the administrator about a received catalogue;
- *request_catalogue :* asks the administrator to create a catalogue;
- *evaluate_contract :* asks the administrator to evaluate a proposal of a federation contract.

• The *Client Part* that uses some operations in the trader's interfaces. These operations are :

- in the *Type Repository Interface :*

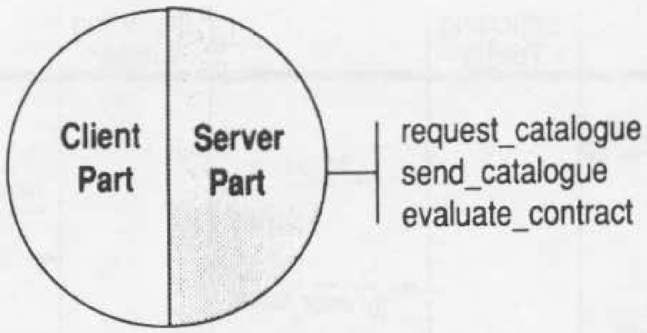

Figure 3 : The administrator.

• *add_service_type :* adds a service type to the type repository;
• *display_types :* shows the service types in the type repository;
• (other operations needed to manage the type repository).

- in the *Context Management Interface :*

• *create/delete_context :* creates/destroys a given context in the directory of service offers;
• *authorize :* defines the set of available interfaces, operations and contexts for a client.

- in the *Federation Contract Interface :*

• *distribute_catalogue :* asks the trader to advertize its catalogue to some other trader.
• *request_catalogue :* asks the trader to obtain the catalogue from some other trader.
• *establish_federation :* asks the trader to try the establishment of a federation contract with other trader providing a contract proposal.

3.1. Establishing federation contracts

To establish a federation contract it is necessary the intervention of the Federation Administrator, because the decisions to create a federation and to evaluate and propose contracts are taken by this object.

In figure 4 it is represented the scenario for negotiation and establishment of a federation contract between two traders, where one performs the role of *exporter* and the other the role of *importer* of services.

Beforehand, in order to establish a federation contract, it is necessary that the importing trader receives the catalogue of the exporting trader. And there are two ways to do this. In the first one, the *administrator 1* decides to send its catalogue to a potential importing trader. When the exporting trader receives the request to distribute the catalogue (*distribute_catalogue* operation) from its administrator, it acts as a normal client of the importing trader using the "*export*"

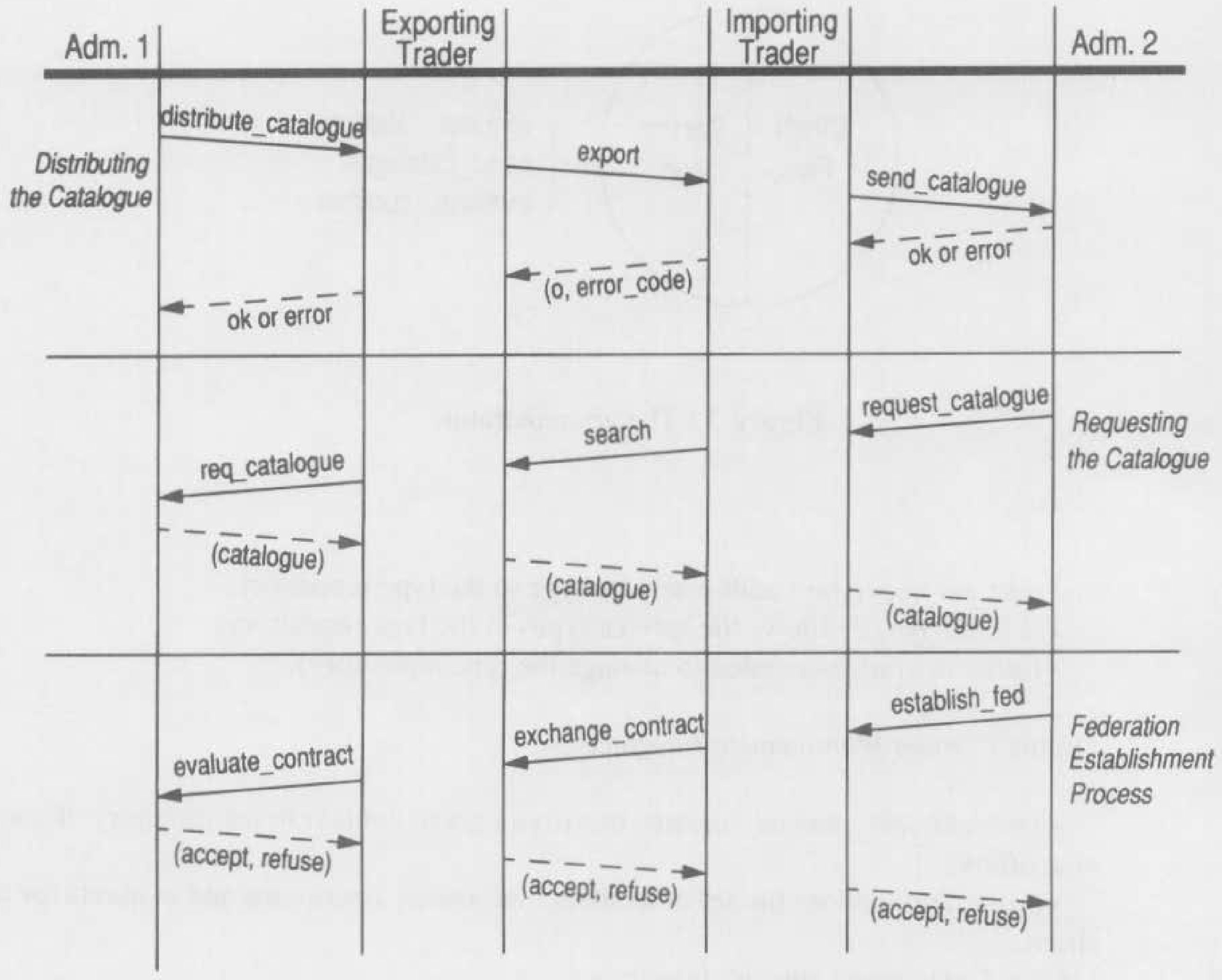

Figure 4 : Scenario to establish a federation.

operation to send the catalogue as service properties and to announce the identifier of its *Federation Establishment* interface. As soon as the importing trader identifies an export request with the standard service type of *federation establishment*, it warns its administrator (*send_catalogue* operation) about the arrival of a catalogue. What is returned to the importing trader, to the exporting trader and to the *administrator 1* is only an indication that the *administrator 2* is informed about the exporting trader's catalogue. The other way to notify the *administrator 2* about the catalogue of the exporting trader consists of simply requesting the catalogue and the process is similar.

Once the catalogue of the exporting trader is known to the *administrator 2*, it can now begin the negotiation of the federation contract. If it is interested in the entries in the catalogue, it can start the contract establishment process. The *administrator 2* creates a federation contract proposal (based on the information held in the catalogue) and sends it to the *administrator 1* (*establish_fed*, *exchange_contract* and *evaluate_contract* operations). The *administrator 1* may accept, refuse, or refuse the proposal and suggest a reduced contract. When a federation contract is agreed, a new *Federated Interaction* interface associated with the exporting contract is created in the exporting trader and the federated operations can be now executed.

The operations that allow a trader to communicate with its administrator are presented in the table of figure 5.

Operation	Parameters	Return Values
distribute_catalogue	location of the imp. trader, catalogue	ok or error
send_catalogue	catalogue, fed. establishment interface id.	ok or error
request_catalogue	location of the exp. trader	catalogue (possibly empty)
req_catalogue		catalogue
establish_fed	fed. establishment interface id., location of the exp. trader, proposal of a federation contract, local context to store the imp. contract	accepted or refused
evaluate_contract	proposal of a federation contract	accepted or refused

Figure 5 : Operations between the trader and its administrator.

3.2. Interworking considerations

In our work, we have identified basically two phases involved in the interworking of traders :

- the phase of "federation establishment" (or "link establishment", or "contract establishment");
- and the phase when the federated operations are performed.

There are also two approaches in attempting to solve the problem of *service types* crossing type domain boundaries : either we leave the "heavy work" to be made in the importing phase (when a federated search is performed), or we leave it to the federation establishment phase.

From our point of view, it is better to do all the "heavy work" in the federation establishment phase for the following reasons :

1. the trading operations through federation become lighter, that is, faster, and performance aspects are more important here than in the federation establishment phase;

2. we will not need a universal standard language for expressing service types, but just some (type description) language agreed between each pair of traders in the federation establishment phase;

3. we will not need to know, for example, the interface identifier of the remote type repository.

3.3. Implementation aspects

The Multiware Platform includes most already known ideas and functions of RM-ODP. This platform is composed of three layers: Basic Software/ Hardware, Middleware and Groupware (Figure 6).

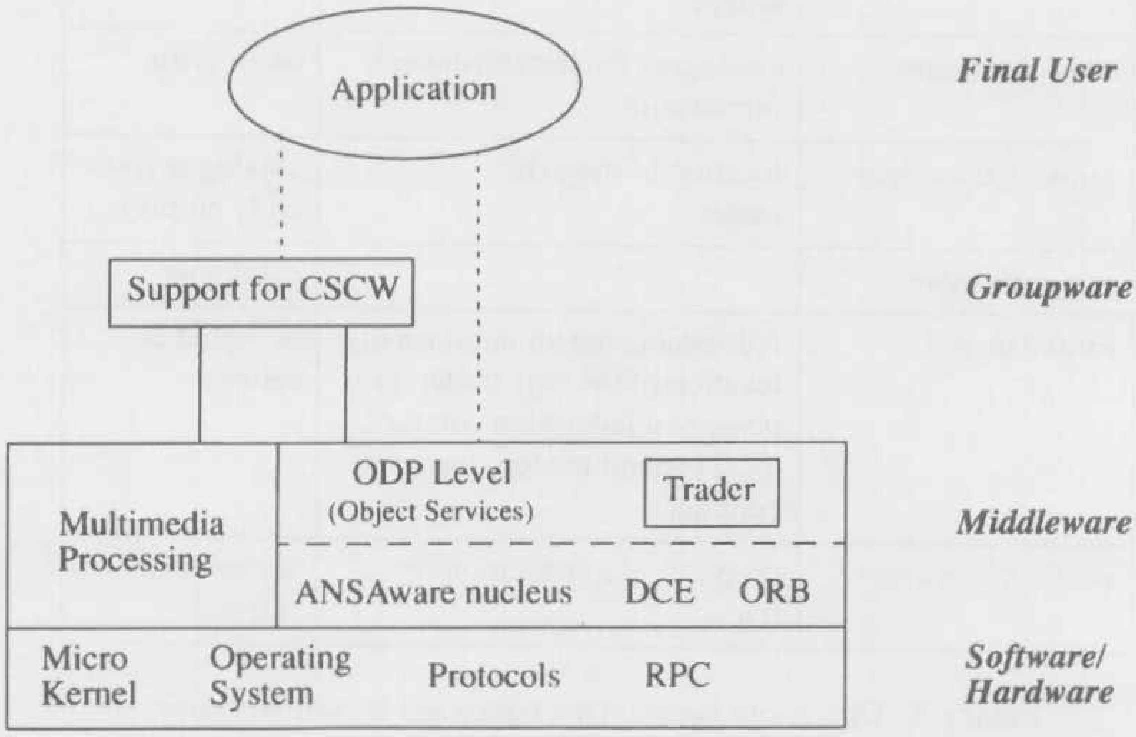

Figure 6 : The Multiware Platform.

The Basic Software/ Hardware is composed of an operating system (eventually built above a microkernel), communication protocols, and so on. This layer provides no distributed system support.

The Middleware layer is responsible for providing distributed processing facilities to the Groupware layer and to the applications. The Middleware layer is composed of two sublayers:

1. Multimedia Processing sublayer - allows the exchange of real-time multimedia information with a specified quality of service; and
2. ODP sublayer that is composed of two levels:
   - 2.1. Commercial Distributed Systems - like ANSAware, ORB (Object Request Broker) [18], and DCE (Distributed Computing Environment) [19]; and
   - 2.2. ODP-level - aggregates ODP functionalities to the commercial distributed systems.

The Groupware layer provides the functionalities demanded by different classes of application, like CSCW (Computer Support Cooperative Work). Typical services supported by this layer are: dialogue management, interaction protocols and handling of multimedia documents.

In the current implementation, CORBA (Common Object Request Broker Architecture) is used as the basic infrastructure. Object Services can be put upon an ORB. The Object Services constitute a set of services (interfaces and objects) that provides the basic functions to use and to implement objects.

The reasons for the choice of the ORB are: ORB is a simple platform, it covers the main concepts proposed by the ODP specification, and it allows adding of new objects with ODP functionalities (Object Services).

In the Multiware Platform, the Trader is an Object Service that is located in the Middleware layer, in the ODP-level. It is responsible for providing the trading functions to the upper layers. In this level, there are other Object Services, like the group support and transaction support, among others.

Since the Multiware platform [12] is in process of design and initial implementation, we had to implement the trader prototype directly over the SunOS using Remote Procedure Calls (RPCs). To do so, a module called "Broker" was implemented aiming at providing communication facilities between the trader and its client (importer or exporter) and to provide the intermediary functions. The Broker corresponds to the result of the compilation of the interface definitions written in some Interface Definition Language (IDL).

The concept of an interface was implemented in order to group operations which are likely to be used by some specific client. Doing so, we become capable of controlling more efficiently the access and the availability of the operations.

The Trader interfaces were defined using the OMG-IDL to facilitate the migration of the Trader to the ORB platform. The repository and storage functions were implemented and incorporated to the Trader.

To transport the prototype to the Multiware Platform, that initially uses the ORB, it is necessary to substitute the compilation of the OMG-IDL interfaces with the developed "Broker". Then, this new broker is composed of stubs that call the ORB to manage the communications among clients, traders and administrators.

In our model, a generic object can:

- create and destroy interfaces dynamically;
- check the authorization of clients to use a given operation which belongs to a given interface;
- define which clients are allowed to use a given interface;
- associate a service type to a given interface (which is particularly useful when we associate an exporting contract to a given interface).

To the normal (non-federated) operations of the trader, there are two interfaces:

- the importer operations interfaces and;
- the exporter operations interface;

which contain the corresponding operations as shown in figure 2. By now, we have implemented the operations:

- export;
- withdraw;
- search;
- list_offer_details;

among other managing/administrating operations. We will comment briefly some implementation aspects of each one, namely:

1. Export

According to the model of figure 1, after checking the authorization of the exporter to perform the operation, the executive verifies that the service type requested for exporting exists and the service properties are correctly given. This information is retrieved from the Type Repository. If everything is all right, it "packs" the received parameters in a "service offer" and stores it in the proper context of the trader's directory. If no errors were found during this process, the trader returns to the user an identifier of the service offer (which is unique in a given context). Otherwise, an error code is returned.

2. Withdraw

This operation simply removes the service offer from the directory. It is performed by the exporter which is the "owner" of the offer.

3. Search

With this operation, the importer can specify the service type it wants, the desired properties (according to the type) and the scope of search. After checking the authorization, the trader verifies the correctness of the required properties according to the type (with the *Type Manager*) and performs the search on its own directory. During the search, every time it finds an import contract inside some context that belongs to the searching scope, the trader puts that contract in a list. In the end of the search, if no service offer is found locally, the contract list is used to perform federated trading. Otherwise, the contract list is simply discarded. A list of matching service offers (that can be empty) is returned to the importer.

4. List_Offer_Details

This operation is normally executed after a "generic" search operation to look into some specific service offer. The complete list of the properties of that offer is returned to the importer, if it has authorization to do such an operation.

In our prototype, all of the operations of figure 5 were implemented together with those necessary to the federation establishment process (e.g., *export*, *search*, *exchange_contract*, etc.) and other security functions. The prototype will be soon migrated to the Multiware Platform which is based on the ORB.

Our approach in attempting to solve the "interworking problem" is to rely on the federation establishment phase to define the equivalence between service types that belong to different "service type domains" (this is done using *mapping functions*) and to define the searching scope by means of the agreement of a number of available contexts via federation. This information is stored in the *exporting contract* in the exporting trader side and in the *importing contract* in the importing trader side.

The contexts of the remote (exporting) trader are associated to the local context space through the importing contract which is stored in some local context. An importing contract contains a list of the accessible contexts (for reading and/or writing) of the remote trader and it always belongs to at least one context of the importing trader. The appropriate context where it is stored is specified by the administrator in the federation contract establishment process. Therefore, the relation between the local and remote context spaces is only "logical", it is neither reciprocal nor hierarchical and, in fact, this relation is very "free", in a sense that it "associates" a context in one trader to a *list* of contexts in the remote trader, which may be or not part of a directed acyclic graph structure.

4. RELATED WORK

Many traders are currently being implemented around the world (DSTC - Australia, University of Hamburg, the trader of the BERKOM project, etc.) and some approaches have been proposed (integration with X.500, implementation over DCE/OSF, etc.). Here, we will comment briefly some aspects related to using the X.500 directory with the trader and some features of the trader of the BERKOM project.

The Directory presented in figure 1 can be implemented using the X.500 directory, as suggested in [13], but in our implementation we did not use X.500. Instead, we built a simple directory using C++ subroutines. As pointed out in [13] there are certain problems which cannot be solved inside the X.500 service (e.g., type checking) and it is impossible to model some relations between traders forming a federation. For this reason, in our model, the X.500 is suitable only with a limited "power", that is, only to store the (possibly distributed) static information which concerns only one trader. The fact that the structure of the X.500 directory is hierarchical can impose some undesirable constraints, since the directed graph required by the ODP documents [15] is a more flexible structure. To sum up, our trader is not based in the X.500 ideas because we think they may be restrictive in some cases, but we agree that the X.500 could be used to implement the directory depicted in figure 1.

In the trader of the Y Platform [16] the imports always return only the best service offer. In other words, their *search* corresponds to our *select*. This means that the trader must verify the availability of the found service in every import in order to assure that the client will not need another option (one that it cannot give, since it returns just "the best" one). Performance aspects are seriously considered [2] to reduce the importing time (e.g., *cache* memory). They will be added into our prototype that was implemented using the model of figure 1. There is no mechanism for federation in the analysed version of the trader of the Y Platform.

5. CONCLUSION

The model described in this work is suitable for an Environment for Open Distributed Processing, where autonomy, decentralization and encapsulation are basic principles. To sum up, the main contributions of this work are:

- the clear specification of the modules inside the trader leads to an easier implementation;
- the definition of the basic protocol between the trader and its administrator is very important (it shall be improved in the future) to allow interworking;
- one administrator may manage several traders in different sites (but inside a certain domain);
- a scenario for the federation contract negotiation phase was proposed.

Indeed, by now we cannot evaluate how expensive our proposal for interworking can be. Future work is needed to find out if this approach is reasonable. It will certainly require more "human interaction" (by means of the administrator). However, some operations of the administrator can be automated.

The problem of making *new service types* available to the remote trader is the same problem of the contract federation updating and in this stage of the work we do not have a proposal for a solution.

Following this discussion, there are some aspects that need a more detailed definition and can be subjects for future research :

- formalizing the "mapping function" concept;
- deciding on how to update a federation contract;
- deciding on how to negotiate the agreement of types, services, scope, etc. in an efficient way (defining the scope of "human interaction").

Our approach to interworking requires a greater participation of the trader administrator because most of the details for interworking are agreed in the federation establishment phase.

The Multiware Platform will incorporate the prototype that was implemented following the concepts of the proposed model.

6. ACKNOWLEDGMENTS

The work reported in this paper was funded by CNPq and FAPESP.

7. REFERENCES

[1] *Mendes, M. J.; Loyolla, W. P. D. C.; Madeira, E. R. M.* - "Demos: A Distributed Decision-Making Open Support System" - 4th IEEE Workshop on Future Trends in Distributed Computing Systems - Sept. 1993 - Lisbon - Portugal - pp 208-214

[2] *Tschammer, V.* - "Trading in Open Distributed Systems" - Proceedings of the Invited Papers - XI SBRC - Campinas - Brazil - May 1993

[3] *Bearman, Mirion; Raymond, Kerry* - "Federating Traders: An ODP adventure", IFIP 1992, pp 125-141.

[4] "ISO/IEC JTC 1/SC 21/N 7053" - Basic Reference Model of ODP - Part 1: Overview and Guide to Use

[5] "ISO/IEC DIS 10746-2 - ITU-T Draft Rec. X.902 - Feb. 1994" - Basic Reference Model of ODP - Part 2: Descriptive Model

[6] "ISO/IEC JTC 1/SC 21/N 10746-3.2 - ITU-T Draft Rec. X.903 - Feb. 1994" - Basic Reference Model of ODP - Part 3: Prescriptive Model

[7] "ISO/IEC 21/N 7056" (Recommendation X.905) - Basic Reference Model of ODP - Part 5: Architectural Semantics

[8] "ISO/IEC JTC 1/SC 21/N 7047, 1992-06-30" - Working Document on Topic 9.1 - ODP Trader

[9] "ISO/IEC JTC 1/SC 21/N 7057, 1991-06-20" - WG7 Project Management Document: ODP list of open and resolved issues - June 1992

[10] "ISO/IEC JTC 1/SC 21 - Nov. 1993" Information Technology - ODP Trading Function

[11] *Vogel, A; Bearman, M.; Beitz, Ashley* - "Enabling Interworking of Traders" - ICODP'95 - IFIP International Conference on Open Distributed Systems 95, Brisbane, Australia, February 1995 (accepted paper)

[12] *Loyolla, W. P. D. C; Madeira, E. R. M.; Mendes, M. J.; Cardoso, E.; Magalhães, M. F.* - "Multiware Platform: an Open Distributed Environment for Multimedia Cooperative Ap-

plications" - COMPSAC'94 - IEEE Computer Software & Application Conference, Taipei, Taiwan, Nov. 1994

[13] *Popien, C.; Meyer, B.* - "Federating ODP Traders: An X.500 Adventure" - Proceedings of the International Conference on Communications 1993 (ICC'93), Geneva, Switzerland, pp. 313-317

[14] *Popien, C.; Heineken, M.* - "Object Configuration by ODP Traders" - Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing, Berlin, Germany, 13-16 Sept. 1993, Publ. North-Holland, pp. 406-408

[15] *Bearman, M.* - "Tutorial on Trading Function" - Part of ISO/IEC 13235: 1994 or ITU-TS Rec X.9tr - WG7 Committee Draft ODP Trading Function Standard - Sept. 1994

[16] *Popescu-Zeletin, R.; Tschammer, V. and Tschichholz, M.* - "Y Distributed Application Platform" - Computer Communications, vol. 14, 6 - July/August 1992

[17] *Lima Jr., L. A. P.; Madeira, E. R. M.* - "A Model for a Federative Trader" - Participant's Proceedings of ICODP'95 - IFIP International Conference on Open Distributed Processing 95, Brisbane, Australia, February 1995 - pp. 155-166

[18] Object Management Group, "ORB Architecture", July 18, 1993, OMG TC Document 93.7.2

[19] Open Software Foundation, "Distributed Computing Environment" - Sept. 1990