Enabling Interworking between Heterogeneous Distributed Platforms*

B. Meyer, S. Zlatintsis, C. Popien
Aachen University of Technology, Dept. of Computer Science (Coomunication Systems)
Ahornstr. 55, D-52056 Aachen, Germany, Tel.: +49/241/8021415
Fax: +49/241/8888220, bernd@i4.informatik.rwth-aachen.de

Abstract
We present a trader design that is focussed on integrating already existing components. This design has been used for implementing or enhancing traders on CORBA and ANSAware platforms. For CORBA we show, how this design can be used with special focus on type management. Interworking between autonomous distributed platforms requires a contract to be established. Therefore we have developed a new negotiation protocol based on policies, contracts and trader links. Besides establishing federation contracts, gateway objects have to be introduced to enable interworking distributed platforms. Server proxies, group proxies and interceptors are possible ways to realize gateway objects.

Keywords
Trader, Interworking, Federation, Type Management, ANSA, CORBA.

1 INTRODUCTION

Interconnected networks of high performant computer nodes are succeeding large mainframe computer because of the enormous improvement in communication and hardware technology within the last decade. While local area networks connect all computer nodes in one building new high speed networks will extend networks to regional and even to nationwide or global

*This work was supported by the Deutsche Forschungsgemeinschaft under Grant No. Sp230/8-1 structures. Walking along with network size, applications become more and more distributed. In oder to ease the migration from monolothic to distributed applications, distributed platforms, sometimes called distributed processing environments, have been developed. The Common Object Request Broker Architectrue (CORBA) of the Object Management Group, the Distributed Computing Environment of the Open Software Foundation and ANSAware are prominent examples for this kind of platforms. Although these platforms were intended to establish a vendor-independent platform, they do not support all kinds of operting systems. Especially, no platform provides support for interwoking between two heterogeneous distributed platforms. Ongoing with research on distributed platforms ISO started an activity called Open Distributed Processing (ODP), see (ISO ODP 1995). It provides a framework of abstraction, an architecture and a number of functions for distributed platforms. One major function is the trading function, which enables location and server-independent binding of interfaces. All services offered to a trader can be mediated to client requesting a certain service.

The notion of federation has been introduced by (Heimbigner et al 1985) and (Shet et al 1990) in the area of heterogeneoues multi-databases systems. It has been adapted for interworking ODP traders by (Bearman et al 1992). What distinguishes federated form distributed databases is, that the latter have a global data scheme, where the former have to establish a common understanding of data schemata before a federation contract can be signed. For the same reason we are going to distinguish trader interworking between federation and cooperation. In regional, nationwide or global networks, there will be a huge number of different service of fers, so a central trader service is not sensible. In addition, traders belong to different companies each wanting to offer only a small set of service offers to public. Some other user should be completely excluded from accessing a companys service. Therefore the interworking between two traders have to built on a contractual basis. This agreement is called a federation contract. In order to enable interworking between heterogeneous disributed platforms, trading facilities have to be federated. Once all services in a federated system of distributed platforms have been made available, gateway computers (or interceptors in ODP terms) have to be developed for enabling interactions beyond technological boundaries of distributed platforms. Section two provides a new design for a trader component, that can bee implemented on several distributed platforms. As an example, we describe an realisation using a CORBA implementation in section three. This design also has been used to enhance the trader of ANSAware. In order to provide interworking between heterogeneous platforms, a novel federation protocol will be presented in section four. Section four also provides a describtion of a trader gateway connecting an ANSAware and ORBIX distributed platform.

2 IMPLEMENTING TRADER ON HETEROGENEOUS PLATFORMS

In this chapter we propose an architecture for a trader component. It has been developed with focus on integrating existing components like a relational database or an X.500 directory service. Other traders have been developed within the TRADE project at the Univ. Hamburg (Merz et al 1994), the system TBRMS at Univ. of Western Ontario (Pratten et al 1994), the MELODY project at Univ. Stuttgart (Kovács et al 1994), the DRYAD porject at Univ. Helsinki (Kutvonen et al 1994), the system Agora at Univ. Karlsruhe (Keller et al 1995), the trader developed at DSTC in Australia (Beitz et al 1994) and the system X* at Univ. Dresden (Funke 1995). All these trader approaches do not address the topic of trader federation or cooperation.

2.1 The trader federation model

Traders collect service offers exported by its users. If an user wants to import a certain service, the trader checks whether or not it can find a matching service offer. Import and export operations are provided at the Trader Service Interface, see (ISO Trader 1995) and Figure 1. An import request describes the required service by its type and properties. Some porperties of a service are frequently changing, whereas others have constant or rarely changing values. They are called dynamic respectivly static properties. Whereas static property values will be stored by the trader, dynamic properties will be evaluated for every request. Therefore every server offers an evalution operation at its Service Offer Evaluation Interface. Besides service offers a trader keeps information about other traders it knows, stored as so-called links, trader properties and other management data. Access is given to these information using operations of the Trader Management Interface. In the following we will call each trader in a federation a trader component. Trading federations will be established based on the policies of each participating trading component. Since trader administrators are in charge of enforcing a trader policy, they are also those objects negotiating a contract for trader components, see (Meyer et al 1994a). Negotiations will be performed using operations offered at the Federation Service Interface, that has not yet be defined by ISO. We will present an new approach to federation negotiation in section 4.1.

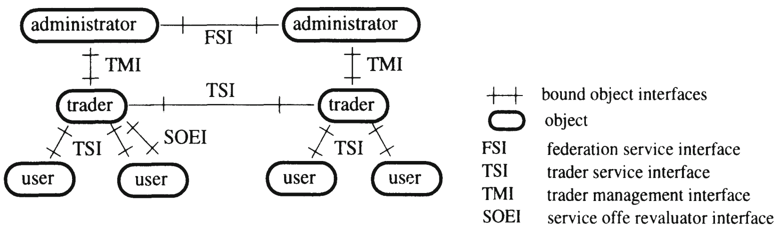
administrator
FSI
administrator
TMI
TMI
TSI
trader
trader
TSI
SOEI
TSI
user
user
user
user
bound object interfaces
object
FSI federation service interface
TSI trader service interface
TMI trader management interface
SOEI service offe revaluator interface

Figure 1 Object interfaces in a trader federation

2.2 A common trader architecture

An overall architecture of a trader component, possibly integrated within a federation, is given in Figure 2. It consists of an number of different modules and edges connecting these modules representing a usage relation. With usage is meant, that one module uses the other one for implementation of the module body. By convention, modules located on a higher level always use modules on lower levels.

The following design has been established according to the ODP trader specification (ISO Trader 1995), with some slight modifications and extensions. The core of the architecture is the (service) offer database which serves as storage for service offers of one trader. It encapsulates the structure of a service offer and the way the service offer space is structured for efficient access. Its has been found that the service type is a good primary key for service offers. In order to compare a requested service type to offers in the database, the offer database falls back on the `filter` module. This module includes concepts for matching constraints, selection criteria and scope restriction. In addition, it defines a query language using these concepts, see (Popien et al 1995). Based on user requirements it controls whether or not a property is evaluated dynamically. The offer selection gets more complex and complicated if more than one property is to be optimzed. In general, properties are not independent, so optimizing one property conflicts with a best selection of another property, e.g. looking for a printer service with a minimum cost per page and maximum printer resolution cannot be solved best with respect to both properties. Therefore, a trade-off has to be fixed based on the importer's preference. This preference can be specified in qualitative or quantitive terms. A qualitative preference can be expressed by an ordering of the importance of properties, e.g. costs per page could be of more importance than a printer's resolution. For quantitative preference the user must specify which amount of variation of one property is of equivalent to the variation of a second property, e.g. a variation of 0,05 DM for `cost_per_page` is of equal value as a variation of 100 dpi for `resolution`. Assuming linear continuation of this preference, a offer *Printer1* with (`cost_per_page` = 0,10 DM; `resolution` = 360 dpi) must be selected compared with an offer *Printer2* with (`cost_per_page` = 0,20 DM,`resolution` = 500 dpi).

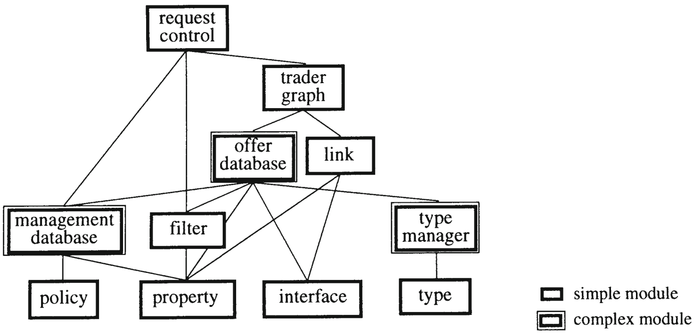

Figure 2 Modular architecture of a trader component

In order to improve the efficiency of the evaluation of dynamic property values, it should be possible to group all service exports e.g. of a computer node and let a certain manager object maintain the dynamic property values. Consistently a trader performing a lookup only has to contact these managers for obtaining dynamic property values of several exporters. Therefore it is necessary to evaluate dynamic properties afterwards searching with respect ot service type and static properties, because the trader must know, which exporters within a group it needs to contact for import query processing. More details can be found in (Küpper et al 1995).

Different traders will be connected by links, which together establish an arbitrary structured trader graph. Another database, the management database, stores and maintains trader properties, trader policies and related kinds of management information. More detailed informattion on the notation used for policy representation can be found in (Meyer et al 1994b). All these modules are under control of the trader query manager. Taking the import operation as an example, the trader query manager checks the request against the trader policy and forwards requests to the local offer databse or to federated traders.

2.3 Using standard components

The most important database of each trader is the service offer database. It might contain hunderts or even thousands of service ofers. Usually not all service offers can be stored in the main memory of a computer node, so a database system as secondary memory is very useful. Whereas an object-oriented data model seems to be more natural for modeling the offer database, it is possible to model all information in relational database schemata. Therefore requests to the service offer database have to be transformed into standard SQL (structured query language). It seems to be sensible to design a query language for the service offer database, that extends SQL with attributes for properties (static/dynamic, mandatory/optional) and selection of an optimal service with respect to one (or more) properties. Another approach uses the X.500 Directory Service for storing service offers, service types, links, properties or policies, see (Popien et al 1993) and (Waugh et al 1995), which is a comprehensive summary of annex B of the ISO trader document.

3 TRADER ON A CORBA PLATFORM

3.1 The Object Request Broker

The Object Management Architecture (OMA) has been designed to integrate objects from heterogeneous systems into a single application. Therefore objects offer their services as operations of an interface defined by an OMA-specific interface definition language (IDL). Thus an object's implementation is hidden from the clients, while a homogeneous view to all objects is provided. Interaction between objects are performed by a dedicated component, the Object Request Broker (ORB), see (OMG CORBA 1993). It allows to access objects independent from their implementation or their location in the environment and allows the handling of objects and their corresponding references in an OMA platform. To offer its functionality, the ORB has two repositories at its disposal, the interface repository and the implementation repository. Interactions via an ORB follow a typed approach. This requires a type checking mechanism to ensure that an invocation is provided with legal parameter types. Therefore interface definitions can are stored in the interface repository, along with their corresponding operations, parameters and exceptions, where they can be accessed for type checking purpose. To make type description comparable, an ORB system offers the concept of type codes. Type codes describe datatypes used at an interface in a uniform way, thus allowing any constructed type description to be passed as a parameter in an invocation. The implementation repository stores information concerning an object\'s implementation, i.e. an object's reference, name, pathname of the executable or activation mode of the object. Whenever the ORB comes across a dynamic object invocation, it retrieves the appropriate reference from the implementation repository.

3.2 Type management based on CORBA

Besides service offers, the trading system needs to store service types and relationships between them. Relationships between types can be subtyping or equivalence. To provide type compatibility in the sense of ODP subtyping rules for interfaces should be met, see (ISO ODP

1995). For being a subtype of another type, it allows parameters or properties to be subtypes, but requires name equivalence for all named items in an service type, such as operation, parameter or property names. Type equivalences can be used to relate types, that have different names, data structures or attribute domains, but the same semantics. Therefore type transformations have to be defined, see (Meyer 1995).

When developing a trader based on an ORB, the implementation repository and the interface repository seemed to be a good basis. First, the implementation repository stores reference information and could provide these information to the trader. Unfortunately, there is no public class allowing to access information stored in the implementation repository, so object references can only be accessed ORB-internal mechanisms. For this reason service offers and object references have to be stored in a separate database. This is feasible since an object´s reference is known to the object itself and can be accessed and/or made available by it.

On the other side the interface repository is provided a public calss interface that allows to store and view interfaces types by retrieving them from the repository. Especially the type codes offered by the ORB were a great ease, because they can be generated on request by IDL compiler along with type descriptions of interfaces, their operations and exceptions. The user does not have to deal with the construction of type codes, the concerning mechanisms are already provided. The obtained types can easily be brought into the typemanager's space. Only type information concerning service types requires a little more expense, because service types are not in the scope of an ORB. In this case information according service properties, service offer properties and the service's semantics have to be provided by the exporter.

```
interface TypeManagerServiceInterface:
void AddType(
        in TypeSig new_type,
        in TypeIDList subtypes,
        in TypeIDList supertypes,
        out TypeID identifier);
void DeleteType(
        in TypeID identifier);
void SetEquivalenceRelation(
        in TypeID type_1,
        in TypeID type_2,
        in Transformation 2_to_1);
void SetSubtypeRelation(
        in TypeID supertype,
        in TypeID subtype,
        in Transformation sub_to_super);
void PlaceType(
        in TypeSig type,
        out TypeIDList subtypes,
        out TypeIDList supertypes)
void IsSubtype(
        in TypeDesc supertype,
        in TypeDesc subtype,
        out Boolean ok)
void IsEquivalentType(
        in TypeDesc type_1,
        in TypeDesc type_2,
        out Boolean ok)
void GetSubtypes(
        in TypeID type,
        out TypeIDList subtypes)
void GetSupertypes(
        in TypeID type,
        out TypeIDList supertypes)
void GetEquivalentTypes(
        in TypeID type,
        out TypeIDList equivalent_types)
```

Figure 3 Service interface for a type manager

Figure 3 gives an overview of the service interface of the type manager. Type relations the type manager supports cover subtype and equivalence relations. The type manager allows users to add and delete types from the type repository. Adding types requires to give supertypes and subtypes of the new type. In return, the user gets an unique type identfier. Relations between two types of the repository can be introduced by dedicated operations for both, subtype and onequivalence relation. Furthermore, ther are operations for checking subtype and equivalence relation between two types. Therefore, the types can be described by a type identifier or a type signature depending whether or not the types are part of the type manager. Another three operations return all subtypes, supertype or equivalent types with respect to a given type. It is important to mention that syntactic subtyping is not equivalent to semantic subtyping. Syntactic subtyping can be automated whereas semantic subtyping needs support by the user and can thereby carried out only computer-aided or manuall. Related work on type management can be found in (Indulska et al 1994) and (Brookes et al 1995).

4 ESTABLISHING AND PERFORMING INTERWORKING

The following section presents a new approach to the federation of trader components based on the notions of policies, contracts and links. Therefore a new protocol for the negotiation of a federation contract, called Federation Negotiation Protocol will be introduced. After establishing a federation, one trader can forward e.g. import requests to the partner trader. In general. this interworking requires crossing technology boundaries of the involved ditsributed platforms. Therefore we propose a trader gateway to be introduced. For the distributed platforms ANSAware and ORBIX describe a prototype implementation of a trader gateway.

4.1 The Federation Negotation Protocol

In the follwing we describe a new federtion protocol called the Federation Negotation Protocol and concepts used for its realization. A related approach can be found in (Beasley et al) and (Lima et al 1995). But contrary to our approach Lima assume a contract to negotiated between two traders. We believe, that federation negotiation is a management tasks and should not reduce a trader´s performance, see (Meyer et al 1995).

Each trader holds a policy determining its behaviour. Two parts of this trader policy are called the Federation Export Policy and the Federation Import Policy, that determines which behaviour a trader offers to other traders and what kind of behaviour a it requests from federated traders to be offered. The behaviour described within these policies might depend on certain conditions or states of the trader, e.g. a very busy trader will have a very strict export policy allowing only a small number of federted traders or restricting the shared service offer space. A certain company might grant guest traders only a smaller service offer space than associated companies. Both federation partner trader will store their part of a contract. We have developed a formal notation for policies based on the notion introduced in the Reference Model ODP, see (Meyer et al 1994b). (Anstötz et al 1995) show, how these policies can be interpreted by a rule-based agent. A contract builds a directed relation between traders giving either an importer or exporter role to each one of them. Each contract contains an identification, a specification of the shared interface operations and the shared information. In Figure 4, we give an example of an importer´s part of a federation contract between two trader. The contract identifier consists of a combination of both partner´s IDs and a sequence number to allow more than one contract between two traders. The shared information specification can be used to reduce the accessible service offer space of the exporting trader. It is possible to restrict to service offer space to certain nodes, contexts, service offers of certain type or property values. If the result offer space does not match with physical service offer partitions, it can be cached to achieve performance gain. The transformation of a policy into a contract is not only a syntactical process but also a semantical one.

```
contract TraderI4&TraderI3&1
with TraderI3 as exporter
for interface TSI_TraderI3:=UIID
allowing operations {import, listOfferDetails}
require
maxPropagationDepth = 0;
maxNumberOfCheckedOffers = 1.000;
transportProtocol = UDP
giving access to
node = I3_Public;
serviceType in {Drucker, DruckFormatKonverter}
where
resolution >= 360 dpi
```

Figure 4 Sample federation import contract

This is because knowledge of a trader´s system state or of the membership of the requesting object in a certain domain might have influence on the charateristics of the contract being created. In general, a contracts are static, which means, that shared interfaces and information do not change over the time. But it can be fixed in a contract, that it includes dynamic change without modifcation. This requires a policy to be included within a contract. There exists an federation export policy on the accepting side, which is matched with the federation import policy of the requesting side in order to build a federation offer. Due to assuring a maximumautonomity of the accepting side, the federation export policy takes preceedence over the federation import policy.

As mentioned in section two, links between traders are used for storing the knowledge one trader holds about service interfaces of other traders within a federation. In correspondence with the standard (ISO Trader 1995), a link contains an identification to distinguish it from other links, a name for the accessible service offer partition of this link, the reference for the remote trader service interface and a number of properties concerning the link. Transforming a contract into a link can be done by syntactical means, in contrast to the transformation of a policy into a contract, that requires trader state knowledge. All information except name for the service offer space and the interface identification will be merge into a single property list.

A federation contract between two traders will be established by negotiation between the corresponding administrators, see Figure 5. If both traders are linked to the same administrator, there is no need for inter-administrator cooperation. Federation negotiations are directed inthe sense that one administrator takes the contract requesting role whereas the other has theaccepting role. The federation negotiation protocol works as follows. The importer forwards a contract request to the exporter. This contract request will be created from the importing trader´s federation export policy. Depending on the contract content, the exporter checks, whether it can accept it or not. Therefore it matches the contract offer with the federation export policy of the exporting trader. If it is not accpetable, it is possible for the accepting administrator to weaken the contract requested and return it to the requestor as a contract offer. The importer has to investigate, if the contract offer, which might be a modification of the contract request, is acceptable. In this case is sends a confirmation. Both sides can send a reject to abort the negotiation. After federation has been established both administrators transform the contract into a trader link, which is sent to the importing respectively exporting trader.

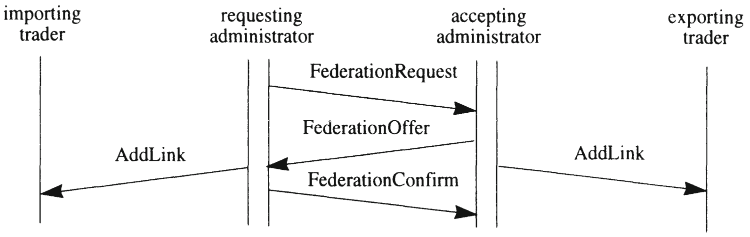

Figure 5 Protocol of successful federation negotiation

4.2 Bridging distributed platforms

Interworking of distributed platforms deals with different kinds of heterogenity. Each platform uses its own format for object references or inter-object cooperation is realized by different remote procedure call (RPC) mechanisms. In addition, interface types are described by different notations called interface or type definition languages. Whereas these kinds of heterogenity are related to the platform technology, another source for heterogenity are user-defined structures within a distributed platform. Therefore, the service type hierarchy is a good example. Assuming two distributed platforms are using different type hierarchies, interwoking requires an integration of both hierarchies. Otherwise a federation between them has nosense at all, because services offered by a federated platform could not be used instead of a local service. Integration of type hierarchies on its side requires bridging of name, structural and semantic heterogenity. All these above mentioned kinds of heterogenity have to be brigded in order to provide federation transparency to the user. Federation transparency is the property of a system to hide technological and administrative boundaries from the user. All mechanisms supporting federation transparency have to be scalable, which means that they are able to mask the integration of a new kind of distributed platform without requiring are compilation of all applications. Consistently, they have to be realized within the run-time system of a distributed platform.

In the following we will focus on solving heterogenity of object references and cooperationmechanisms. Type integration mechanisms are part of the type manager and an approach extending interface definition languages with type integration be found in (Meyer 1995). To provide federation transparency, mechanisms have to be integrated to the the run-time system of each computer node or a special gateway object has to be introduced. Gateway objects can be realized by proxy objects or by an interceptor. A server proxy stubstitutes a certain foreign server, whereas a group proxy represents a group of foreign server objects offering services of thesame type. In contrast to proxies, an intercepetor is a gateway, that transforms operation calls between two platforms. The main diference between a proxy and an interceptor is, that an interceptor is a generic object dealing with operation calls of any service type, whereas proxies only deal with operations of a certain service type. In order to forward an operation to the server, an interface reference is needed to be given to the RPC run-time system. In case of a foreign object, the local run-time system cannot interprete the foreign object reference, and has to be transformed into the foreign object reference format. The proxy interface reference can be used in case of a server proxy as gateway, whereas group proxies and interceptors require more information in addition to the gateway interface reference. This might be a server identification or reference, that is valid in the foreign domain.A more elegant solution is the introduction of a universal interface reference, that can be interpreted in all distributed platforms. Therefore, all platforms have to agree to a universal description for interface references or a union data type has to be defined for all involved distributed platforms. The disadvantage of the universal interface reference approach is, that all distributed plat forms have to be extended in order to use this kind of reference. This is possible for platforms delivred with its source code, but not for commercial platforms. In the following discussion of the three gateway approaches we will concentrate on the e transformation approach. One possible gateway is server proxy object for each object. A client wishing to invoke an operation at a foreign object calls the corresponding proxy object which does the invocation of the real server, receives the invocation\'s result and passes them back to the client. Service offers of foreign server will be stored in the local trader with the reference to the proxy object instead of the foreign reference. The mapping onto the foreign reference is going to be done within the proxy object implictely. The client in this case is not aware of whether the invoked object is a foreign one or not, since the reference it uses belongs to its own system. A disadvantage of this scenario is the overhead of proxy objects that exist in both the environments since not all of the exsting objects need a proxy object as not every object is invoked by foreign objects. The call mechanism for a group proxy is similar to the one for a server proxy except that the proxy interface reference is not enough to uniquely identifying a certain server. Although the client invokes an operation at the proxy interface, it also has to pass the foreign address (or an identifier) of the server to call. It is the task of the group proxy to forward the call to the corresponding server. The addressing problem in the local domain is the same as with group proxyies mentioned above. The major drawback of the interceptor approach is, that this object is likely to become a performance bottleneck, even if it is located on special computer node. In order to improve performance, several equivalent interceptors might be created. This raises the problem that interceptors have different interface references, so it must be known in advance which server will be handled by which interceptor, or group addresses for interceptors must be supported.

We are implementing a gateway for the distributed platforms ANSAware and the CORBA software ORBIX. It consist of a server proxy. As already mentioned a uniform interface reference is required to allow clients to invoke foreign objects. Both platforms offer tools to transform an object reference or an ansa_interface_reference into a string and back. A client wishing to invoke an operation of a certain object passes the previously received uniform interface referenece to the server proxy of its distributed platform along with the ope ration's name and the list of parameters the operation requires. The interceptor then receives the request and does for its part the real invocation, afterwards passing the invocation's result back to the client.

These trader implementations on ANSAware and ORBeline and the gateway between them will also be used in the IWT project started at the DSTC in Australia, see (Vogel et al 1995).)

5 CONCLUSIONS

Connecting heterogenenous distributed platforms can be easily achieved by trading components. Therefore common service interfaces are necessary. We have presented a modular trader architecture that can be used to implement trader components on different distributed platforms. We have used trader design by enhancing or establishing a trader component on the ANSAware and ORBIX distributed platform. For enabling interworking between between heteroegenous, autonomous platforms, a federation contract between its trading components has to be established. Therefore we have presented a protocol called the Federation Negotation Protocol, that is based on the notions of policies, contracst and links. It is realised by a three-way communication assuring both parties can quit the negotiation process if the contract is not satisfactory. Once a federation has been established operations can be forwarded to remote traders and servers. Because of heterogeneous object references, cooperation protocols, interface and type descriptions, gateway objects have to be introduced. We have discussed several gateway approaches like server proxies, group proxies or interceptors. For enabling interworking between the ANSAware and ORBIX distributed platform we described concrete implementation work going on at Aachen University of Technology.

6 REFERENCES

Anstötz, F.; Meyer, B. (1995) Towards implementing Flexible Systems Management - A policy-based approach. *In: Sloman, M. (ed.): International Workshop on Services for Managing Distributed Systems*, Karlsruhe 1995

Bearman, M. and Raymond, K. (1995) Federating Traders: An ODP Adventure. *In: Meer, J. de; Heymer, V.; Roth, R. (eds.): Open Distributed Processing*, North Holland 1992, pp. 125-141

Beasley, M.; Jane Cameron, J.; Gray Girling, G. et al (1993) Federation Manifesto. *Documentt APM.1193.01*

Beitz, A.; Bearman, M: An ODP Trading Service for DCE. *Proceedings of First International Workshop on Services in Distributed and Networked Environments*, IEEE Computer Society Press 1994, pp. 34-41

Brookes, W.; Indulska, J, Bond, A. et al (1995) Interoperabilty of Distributed Platforms: A Compatibilty Perspective. *In: Raymond, K.; Armstrong, L. (eds.): Open Distributed Processing: Expereinces with distributed environments*, Chapman & Hall 1995, pp. 67-78

CORBA (1993) Object Management Group: The Common Object Request Broker Architecture: Architecture and Specification. Revision 1.2, December 1993

Funke, R. (1995) X* - a DC++ based Trader (in german). *In: Mittasch, C. (ed.): Anwendungsunterstützung für heterogene Rechnernetze*, Workshop Proceedings, Freiberg 1995, pp. 51-58

Heimbigner, D.; McLeod, D. (1985) A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, Vol. 3, No. 3, July 1985, pp. 253-278

Indulska, J.; Raymond, K.; Bearman, M. (1994) A Type Management System for an ODP Trader. *In. Meer, J. de; Mahr, B.; Storp, S. (eds.): Open Distributed Processing II*, North Holland 1994, pp. 169-180

Keller, L.; Grosse, A. (1995) Mediation of Reliable Services in Trader-based Systems (in german). *In: Mittasch, C. (ed.): Anwendungsunterstützung für heterogene Rechnernetze,* *orkshop Proceedings*, Freiberg 1995, pp. 41-50

Kovács, E.; Wirag, S. (1994) Trading and Distributed Application Management: An Integrated Approach. *Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management*, Toulouse, October 1994

Küpper, A.; Popien, C.; Meyer, B (1995) Service Management using up-to-date quality properties. #

Kutvonen, L.; Kutvonen, P. (1994) Broadening the User Environment with Implicit Tradding. *In: Meer, J. de; Mahr; B.; Storp, S. (eds.): Open Distributed Processing II*, North Holland 1994, pp.157-168

Lima, L.; Madeira, E. (1995) A Model for a Federated Trader. *In: Raymond, K.; Armstrong, L. (eds.): Open Distributed Processing: Expereinces with distributed environments*, Chapman & Hall 1995, pp. 173-184

Merz, M.; Müller, K.; Lamersdorf, W. (1994) Service Trading and Mediation in Distributed Computing Systems. *Proceedings of 14th International Conference on Distributed Computing Systems (ICDCS'94)*, IEEE Computer Society Press, 1994, pp. 450-457

Meyer, B.; Popien, C. (1995) Performance Analysis of Distributed Applications with ANSAmon. *In: Raymond, K.; Armstrong, L. (eds.): Open Distributed Processing: Expereinces with distributed environments*, Chapman & Hall, pp. 309-320

Meyer, B.; Popien, C. (1994) Object Configuration by ODP Traders. *In: Meer, J. de; Mahr, B.; Storp, S. (eds.): Open Distributed Processing II*, North Holland 1994, pp. 406-408

Meyer, B.; Popien, C. (1994) Defining Policies for Performance Management in Open Distributed Systems. *Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management (DSOM'94)*, Toulouse 1994

Meyer , B. (1995) Integration of Heterogeneous Interfaces in Distributed Systems (in german). *In: Mittasch, C. (ed.): Anwendungsunterstützung für heterogene Rechnernetze, Workshop Proceedings*, Freiberg 1995, pp. 25-32

ISO ODP (1995) ISO/IEC IS/DIS 10746-1/2/3: IT - Open Distributed Processing - Reference Model, 1995

ISO Trader (1995) ISO/IEC DIS 13235: IT - Open Distributed Processing - ODP Trading Function -Editors Draft DIS text. 19 May 1995

Popien, C.; Meyer, B. (1995) A service request description language. *In: Hogrefe, D.; Leue, St. (eds.): Formal Description Techniques VII*, Chapman & Hall 1995, pp. 37-52

Popien, C.; Meyer, B. (1993) Federating ODP Traders: An X.500 Approach. *Proceedings of Interbnational Conference on Communication (ICC'93)*, IEEE Computer Society Press 1993, pp. 313-317

Pratten, W.; Hong, J.; Bennett (1994) A trader based resource management. *Proceeding of 5th IEEE/IFIP International Workshop on Distributed Systems: Operation & Management (DSOM'94)*, Toulouse 1994

Shet, A.; Larson, J. (1990) Federated Database Systems. *ACM Computing Surveys*,Vol. 22, No. 3, Sptember 1990, pp. 185-236

Vogel, A.; Bearman, M.; Beitz, A. (1995) Enabling Interworking of Traders. *In: Raymond, K.; Armstrong, L. (eds.): Open Distributed Processing: Expereinces with distributed environments*, Chapman & Hall 1995, pp. 185-196

Waugh, A.; Bearman, M.: Designing an ODP Trader Implementation using X.500.*In: Raymond, K.; Armstrong, L. (eds.): Open Distributed Processing: Expereinces with distributed environments*, Chapman & Hall 1995, pp. 133-144

7 BIOGRAPHY

Bernd Meyer studied computer science at University of Karlsruhe and Aachen University of Technology. 1994 he received his diploma and then became a research assistant at the Department of Computer Science at Aachen University of Technology. His research topics are trading, distributed platforms and managment of distributed systems.

Stefan Zlatinstis studies computer science at Aachen University of Technology. He received his pre-diploma in 1993. Since 1994 he works at the Department of Computer Science where he is involved in distributed systems espcially trading and type management. He has submitted his diploma thesis entitled "Design and evaluation of a trader gateway between ANSAware and ORB Systems".

Claudia Popien studied mathematics and theoretical computer science in Leipzig, Diploma 1989. After a research work at Technical University of Magdeburg she became an assistant at Aachen University of Technology,the Department of Computer Science in 1991. She finished her Ph. D. thesis entitled „Service trading in distributed systems - service algebra, service management and service request".