The TRADEr: Integrating Trading Into DCE

K. Müller–Jones, M. Merz, W. Lamersdorf

University of Hamburg, Department of Computer Science
Vogt–Kölln–Str. 30, D–22527 Hamburg, Germany
(kmueller,merz,lamersd)@dbis1.informatik.uni-hamburg.de

Abstract

Client support for locating, accessing, and using arbitrary services in open system environments emerges as one of the most interesting, complex, and practically relevant tasks of realising realistic open distributed systems applications. In the context of *Open Distributed Processing* (ODP), current standardisation efforts for a *trading* function play an increasingly important role for open system integration.

In addition to ongoing ODP standardisation activities official and de–facto standard system environments such as, e.g., the OSF *Distributed Computing Environment* (DCE), support efficient development and portability of distributed system applications. Therefore, time seems now ready to analyse and evaluate the use of such platforms also for developing efficient implementations of higher–level system support services, e.g., an ODP trading function.

This paper first elaborates on the specific potential of OSF DCE for supporting *implementations* of ODP trader functions. It then presents an architecture and reports on experiences with such an implementation in the context of system support for general service access, management and coordination in open distributed environments within the *TRADE* project. Finally, the paper draws the attention to still existing *limitations* and deficiencies of OSF DCE for realising ODP trader functions and proposes respective extensions to OSF DCE both at a conceptual and a concrete systems implementation level. According to such prototype implementation experiments, ODP trading functions can be integrated smoothly into a uniform standard system support platform (like DCE) and can be implemented efficiently by extending DCE by additional trading functions which specifically support service management, mediation and access for open distributed applications.

Keyword Codes: C.2.4 D.2.6 H.4.3
Keywords: Distributed Systems, Programming Environments, Communications Applications

1 Introduction

Based on rapid recent developments of telecommunication and networking technologies, users of distributed systems are now increasingly confronted with multitudes and varieties of service offerings in an, in principle, world–wide *open market of services* [MML94]. Faced with the complexity of such open distributed environments, one of the main tasks is to support users and application programs to locate and utilise such services in effective and efficient manners. In this context, one of the most promising efforts is to extend open (operating) system platforms by unified service *trading* or *broking* components as

an important structuring technique for efficient design of open distributed systems. Accordingly, the specification of a trading component is currently — among other issues — subject of the *Open Distributed Processing (ODP)* international standardisation activities as the so–called *ODP trading function* [ISO94].

Beyond ongoing standardisation efforts for a unified trader component specification, it is now increasingly important to also examine possible *implementations* of such trading concepts. Common foundations for various distributed application implementations are existing and evolving distributed systems architectures, as, e.g., proposed and developed by various vendors and consortia like OSF's DCE [Fou92], and OMG's CORBA [OMG91]. In particular, the integration of a trading component into these architectures has to be evaluated. A good starting point for such an evaluation is the *Distributed Computing Environment (DCE)* from the Open Software Foundation (OSF) [Fou92] which has gained wide commercial acceptance and is one of the important system platforms for distributed application development in the future. In particular, this paper focuses on

- an analysis of DCE mechanisms to support service management and service access,
- the limitations of DCE for supporting service mediation,
- a proposal for an architecture of the *TRADEr*, a DCE trading component, which has been developed at the University of Hamburg within the TRADE (*Tra*ding and Coor*d*ination *E*nvironment) project and its co–project COSM (*C*ommon *O*pen *S*ervice *M*arket),
- some details of a smooth integration of the TRADEr implementation into DCE, especially into existing DCE concepts for service management and access,
- extensions of the *service type* abstraction as proposed by ODP which will also be integrated into the TRADEr, and
- current and future work within the TRADE and COSM projects to support client/server mediation in a distributed open systems environment.

2 The ODP Trading Function

The main task of a trader function is the *mediation* and *management* of services in open distributed systems. For this purpose, the trader first offers mechanisms for arranging and categorising various service types, and then supports potential service clients with specific service selection strategies. Thus, the functionality of a trader component can be compared to, e.g., a *yellow pages* service which categorise service kinds and provides service selection support based on different service properties. The most important formal concept underlying such a trading function is the notion of a *service type* [ISO94]. A service type may contain *interface types* which specify the operational service interfaces in terms of operation signatures as well as *service property types* which add additional semantic details to the service type description. A second important mechanism for service structuring in open distributed systems is based on service *contexts* in which service offers can be grouped and located (e.g. in a hierarchical organised name space).

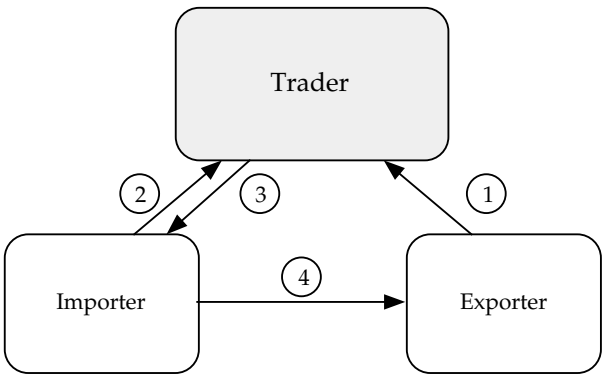

Figure 1: The ODP Trader and its Users

Possible interactions of clients and servers with a trader component in an open distributed environment are shown in Figure 1.

According to the ODP trader function, service provider, a service *exporter*, first registers its service by supplying the service type, the current service property values, and a context in which the service shall be exported (step 1). Then a service client, the *importer*, may ask for servers offering a specific service (step 2). Such an inquiry contains — among other things — the desired service type, the desired service properties, and a search context. Based on this information, the trader then determines appropriate service providers, and selects — if necessary — the best matching service offer. Subsequently, the necessary binding information is returned to the client (step 3), and the client is finally able to execute remote operations offered by the service provider directly (step 4).

3 DCE Service Concept and Support Infrastructure

The OSF Distributed Computing Environment (DCE) provides an integrated set of support services and application programming interfaces (so–called *middleware* [Ber93]) for the development of distributed applications in open heterogeneous environments. The main goal of DCE is to provide users and applications of distributed computing environments with a *homogeneous* view which hides much of the complexity of the underlying hardware and system software components. Therefore, DCE provides ways to develop platform–independent distributed applications, based on de–facto standardised application programming interfaces. Basic DCE services are the *Remote Procedure Call (DCE RPC)*, the *Thread Service*, the *Cell Directory Service (CDS)*, the X.500 compliant *Global Directory Service (GDS)*, the *Security Service*, and the *Distributed Time Service (DTS)*. Additional services are the *Distributed File Service (DFS)* and the *Diskless Client Support*. Based on standard protocol implementations, each of these services provides a single unified application programming interface (with the exception of the Cell Directory Service which offers two such interfaces).

3.1 DCE Support for Service Management and Access

The basic unit of service management and structuring in DCE is the so–called *cell*. Each cell includes its own Security Service, a Cell Directory Service, and a Distributed Time Service. The Cell Directory Service plays a central role for storing all information concerning actual services of the DCE cell (e.g. configuration and binding information). Access to information of foreign cells is provided by an X.500 compliant Global Directory Service which offers a world–wide available name space connecting several different organisation domains.

DCE service structuring is conceptually based on service *interfaces*, i.e. sets of service operations. In order to offer a service interface in a distributed environment, a service provider has to first describe its interface in terms of an abstract *interface definition language (IDL)*. An IDL service description includes information about the offered operation types as well as data types for parameters and results. *Universal Unique Identifiers (UUIDs)* are used for unique identification of interfaces and have to be included in the interface description. They basically provide a very simple type system based on interface names[1] as a basis for managing service providers in the DCE Cell Directory Service. In addition to the interface, a UUID *version number* can be used in order to describe relationships between different versions of an interface (with a given UUID). In this way, support for *inclusion polymorphism* [CW85] can be provided, for example for expressing service evolution (e.g. by step–wise addition of new operations to an interface). Since there is no central type management component included in DCE, however, the DCE application programmer stays solely and fully responsible for the correct definition and use of such interface relationships and compatibilities.

In the following, we give a concrete example of how to establish a binding between a client and a service provider using the DCE Cell Directory Service. Figure 2 shows the involved DCE system components and denotes the necessary execution steps for establishing such a binding between a DCE client and a remote DCE server. In a DCE based open systems implementation environment, a new service provider has to execute the following steps in order to register a new interface[2]:

1. The first step is to inform the RPC runtime system about the offered interface type using an *interface handle*. In addition, a *type UUID* for local type identification, an *object UUID* for unique interface access, and an *Entry Point Vector (EPV)* which serves as a local pointer to the server operations, have to be provided.

2. Subsequently, *binding vectors* are generated which contain binding information (e.g supported communication protocols and dynamic communication endpoints). These along with the interface type and the object identification are then registered at the local host's *Endpoint Mapper*. It manages the mappings from interface and object identifiers to communication endpoints of current running service providers at the local host.

[1] In contrast, ODP defines the notion of an *interface type* for this purpose.

[2] In principle, a server can provide several interfaces at a time.

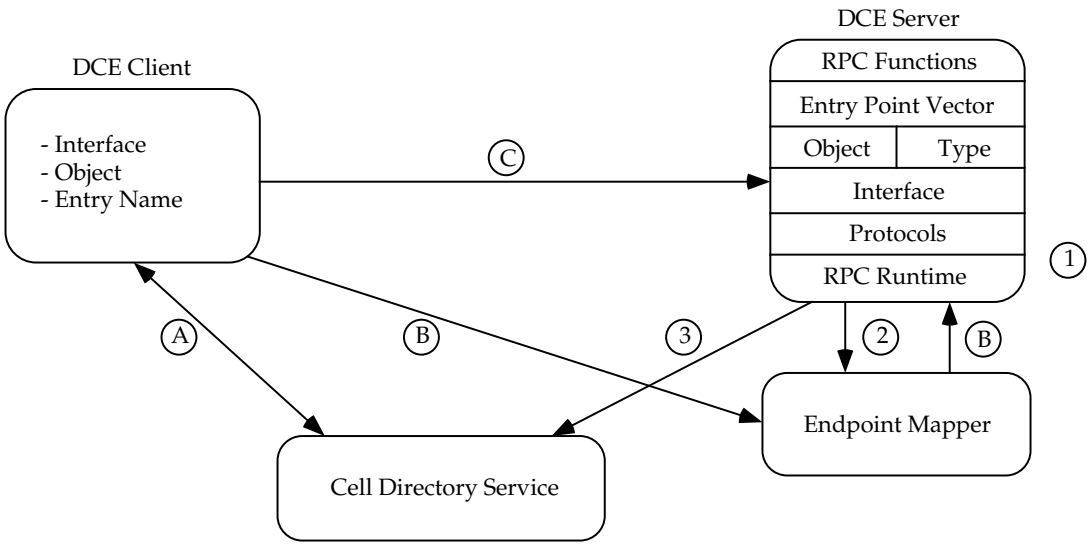

Figure 2: CDS Based Binding

3. In the last step, the same information (except for the dynamic communication endpoints) are registered in the DCE Cell Directory Service under a distinct entry name.

In order to obtain the necessary binding information for a server offering a desired interface, a client has to execute the following steps:

A. First, the DCE Cell Directory Service must be called in order to obtain server binding handles of a distinct server instance. For this service instance, the distinct server entry name, the desired interface identifier, and the object identification have to be supplied.

B. Using these binding handles, the client can then bind to the server and start to execute its remote procedure call. Since the DCE Cell Directory Service manages only incomplete bindings without dynamic communication endpoints, the first call is directed (by the RPC runtime system) to the Endpoint Mapper located at the server's host. There, the dynamic communication endpoints are added to the binding handle, and the call is forwarded to the actual service provider.

C. Subsequent remote procedure calls can then be transferred directly to the corresponding service provider.

3.2 DCE Support for Service Mediation

Although DCE — as briefly reviewed above — provides several useful prerequisites for vendor–independent implementations of distributed open system applications, it still lacks some important features, especially in order to support a *trading function* as, e.g. specified by ODP. Currently, there is no support for service mediation in DCE, i.e. the clients (resp. application programmers) are entirely responsible for the selection of appropriate service offerings.

3.2.1 DCE Cell Directory Service

A simple basis for such support can, as described above, be realised by the DCE Cell Directory Service. The corresponding *NSI application programming interface*, specifically designed for registration of RPC interfaces with the Cell Directory Service, provides simple access and management functions for storing service offers in a hierarchical organised name space. Since there is no direct support for search functionalities the usage of the NSI interface is mainly restricted to simple name–based lookup operations[3].

3.2.2 DCE Global Directory Service

In addition to the Cell Directory Service, the programmer could also use the more powerful functions of the X.500 compliant DCE Global Directory Service. In contrast to the Cell Directory Service, however, there is no direct support for RPC interface registration. Therefore, application programmers have to write their own registration functions. One of the main advantages of the DCE Global Directory Service is the ability for attribute–based searching in the name space. It also provides a world–wide accessible name space which facilitates interworking capabilities, e.g. within trader federations. Section 4.2 explains the use of the DCE Global Directory Service for implementing the TRADEr's service offer management capabilities.

4 The TRADEr: Service Mediation on Top of DCE

This section describes a prototype implementation of a trading component based on DCE, called the *TRADEr* (i. e. TRADE trader), developed recently in the *TRADE* project at the University of Hamburg, Germany. In the TRADE project, prototype trader functions are implemented and step–wise extended as part of a general system support environment for service access, management and coordination in open distributed systems [MJM94, MJML95]. The corresponding *COSM/TRADE* prototype system is currently realised on a heterogeneous cluster of interconnected Sun SPARC and IBM RS/6000 workstations; the *TRADEr* implementation has been developed on IBM RS/6000 workstations with AIX and DCE.

Figure 3 gives an architectural overview of the main TRADEr components. As shown, the TRADEr is structured into several sub–modules which — in turn — realise the main sub–tasks of a composite trading function. According to the trader's role in service management, selection, and access in open distributed systems, the TRADEr's core components are the *service offer manager*, the *service selection manager*, the *trader interworking manager*, and the *type manager*. Additionally, *access control management* is provided as an extended option. As an orthogonal extension of OSF DCE functions, the TRADEr is basically realised as a DCE RPC server using authenticated *Remote Procedure Calls* as interface both for service exporters and importers as well as trading administrators. In addition, the TRADEr prototype implementation uses DCE *Threads* for efficient execution of the different TRADEr functions. DCE *Cell Directory Service* and *Global Directory*

[3]CDS groups and profiles provide simple additional name space organisation, they can be used for user–provided search routines (see section 4.2).

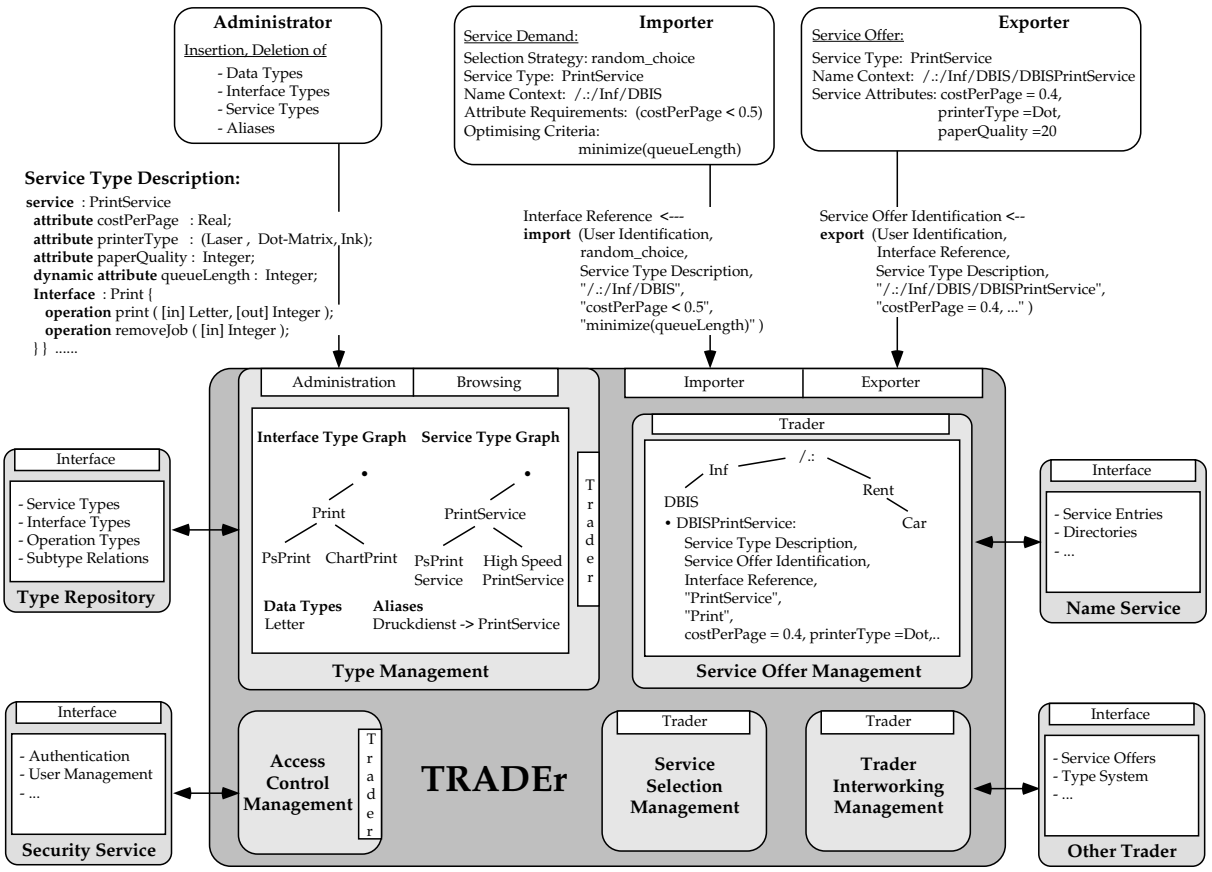

Figure 3: TRADEr Components and Implementation Architecture

Service together provide the basis for realising the service offer management as one of the main functional units of the TRADEr. Figure 3 also gives some implementation details of, e.g., the TRADEr's *export* and *import* functions used by service offering and accessing nodes respectively. (The example specifications given as part of the exporting and importing functions in Figure 3 refer to a basic *Printing Service*).

The following subsection concentrates specifically on the trading functions of *type* and *service offer management* in order to explain some details of design and implementation decisions of the current TRADEr's prototype implementation as realised on top of OSF DCE.

4.1 Extending Type Management within the TRADEr

One of the most important components of a composite trader function is the *type manager*. It provides the basis for a common understanding and for comparison of services types as a main structuring technique for service requests and offerings in open distributed environments. As shown above, different notions of *service types* serve in this context as formal abstractions of *service characteristics*, i. e. common properties of classes of service instances of a distinct service type. Because of the significance of the service type concept, standard typing mechanisms, as known from modern programming languages,

and further extensions to such service type concepts have to be evaluated for the trader's type manager component. Therefore, we briefly mention in the following some alternative forms of well understood type mechanisms which are capable to fulfill the requirements for service typing and trading in general, and address extensions to such basic type management techniques. Such type extensions are main candidates for being integrated into an extended type manager component of a trader and, accordingly, into future releases of the TRADEr prototype implementation.

In general, several levels of type management can be distinguished within a trader's type manager (based on very simple up to very rich and complex service type descriptions). Some possible steps in such a continuum from simple name–based service (type) descriptions up to (ideal) full formal semantic specifications of a service type are listed below. They are used as a basis for step–wise extension of type description and managemenent functions as already available in, e.g., DCE, up to what is really needed for a future trader's type manager components.

1. In its simplest form, classification ("typing") of services is based on type *names*. Limited type flexibility can be achieved using *subtype polymorphism* as one kind of type polymorphism as, e.g., explained in [CW85]. Subtype polymorphism means that a type A which is a subtype of type B can also be used in a type safe way when B is required. For example, a service S offering one additional operation at its interface as compared to a service K can then also be used by a client instead of K. Since this kind of subtyping is based on names only, subtype relationships have to be defined *explicitly*, i.e. whenever new interface types are inserted, their respective sub- and supertypes must be listed explicitly by, e.g., the trader administrator. As explained in the previous section, DCE provides only very little support for subtype polymorphism (based on interface type name, and version numbers for interfaces) but no support for checking the defined relationship between interfaces automatically.

2. Adding explicit *attributes* to the interface type is one way to enrich service decriptions which are to be matched in an open distributed trading environment. Service attributes play an important role to extend the specification of semantics of a given type by including some selected, predefined properties into service (type) decriptions. This allows further discrimination between service types offering similar operations. Subtype polymorphism for attributes can then be extended using *semantic substitutability* as explained in, e.g., [IBR93]. Currently, however, there is no support for service attributes in existing releases of OSF DCE. Service attributes can only to be *simulated* using explicit "get" and "set" operations, similar to the operations defined in the CORBA standard [OMG91] for attribute access.

3. As a subsequent step in generalising and enriching service type descriptions, simple name based subtyping mechanism can be replaced by a *structural* subtyping mechanism. Here, decisions of type relationships can be made *automatically* by the system based on a structural analysis of the corresponding service type descriptions. This concept is known as *type conformance* [BHJ+87] and enforces an *implicit* (or

automatic) style of subtype checking which frees the application programmer from this failure prone standard programming tasks.

4. Experiences with the development of system support for open distributed applications at our group have shown that additional service description techniques are required in order to express semantical aspects of services which go beyond standard (i.e. programming language) type concepts. As one such extension, for example, *finite state machines*, have been introduced into COSM service (type) description as a first step into protocol specifications and are used to support users of a so–called *Generic Client* in accessing "unclassified" services which are not known in advance [ML93, MML94]. They provide a way to formally express how to use a service, e .g. which sequence of operations may be executed by a remote user, e. g. via a Generic Client component, as part of its service description.

5. A final extension to service (type) descriptions considered in the TRADE and COSM projects so far is concerned with coordination of complex distributed services which are comprised of a set of more basic ones. It uses *Coloured Petri Nets* as a formal description technique of such coordination problems and is currently evaluated for supporting workflow modeling and execution within complex open distributed client/server environments [MMML94, MJML95]. Here, Coloured Petri Nets are used for describing service coordination in order to support the execution of workflows in open service environments in an adequate manner.

4.1.1 A Name Based Type Manager with Explicit Subtyping

The current TRADEr prototype provides dynamic type management based on explicit subtyping as mentioned in items 1 and 2 above. Internally, all type information is managed within the type manager component as two directed acyclic type graphs which represent the type relationships between interface and service types. At the current prototype state the administrator still has to explicitly list all supertypes when inserting new interface and service types. Simple type checking is supported by an internal interface comparing service type descriptions as offered by exporters with existing previously defined service types. The trader's external *administration interface* and *browsing interface* provide functions for insertion, deletion and browsing of service and interface types at runtime.

Currently the TRADEr's type manager is extended to support implicit type checking as mentioned in item 3 above as well. In this context, the CORBA type model (respectively the CORBA IDL) is evaluated for its capability to describe services more adequately. Further extensions (see item 5) shall be integrated in the future, if possible.

Future versions of the TRADEr prototype implementation will also comprise an *interface repository* which is developed currently. It will be used to store all kinds of service type descriptions. In the mean time, the TRADEr prototype still stores type management information as local data in volatile memory.

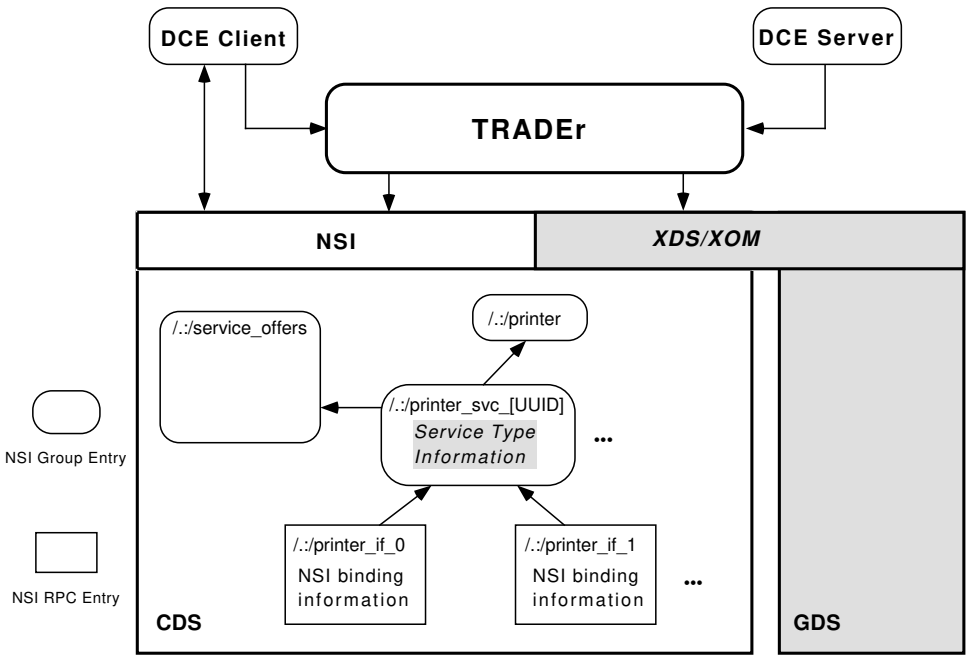

Figure 4: Use of DCE CDS and GDS for Service Offer Management

4.2 Service Offer Management

Another important core component of the TRADEr is the *service offer management*. In the current TRADEr prototype, service offer management is entirely based on the DCE Cell Directory and Global Directory services. The respective interface contains operations for a variety of service offer management functions, e.g. inserting, deleting, reading, and modifying of service offers. Also attribute–based search operations are provided which facilitates the implementation of the TRADEr's service selection strategies. For a service provider to advertise a service offer in the TRADEr, it is possible either to use CDS entry names, such as /.:/services/Laserprinter_1, or to use X.500 entry names, such as /.../C=DE/O=Hamburg University/OU=services/CN=Laserprinter_1. Service offers are stored in a special format representing the offered service type and the corresponding interface types, the current values of the static service attributes, the interface reference for server binding, and, optionally, a service type description. This service type description can be used by programmers to develop corresponding DCE client applications and to generate the RPC stubs necessary for communicating with the DCE server.

The current TRADEr prototype implementation of the service offer management operations is based on the two DCE name service application programming interfaces, namely the *Name Service Interface (NSI)* and the *X/Open Directory Service and X/Open OSI–Abstract–Data Manipulation (XDS/XOM)* interface. Using both of these programming interfaces allows a smooth integration of the TRADEr's trading functions into the existing DCE RPC/CDS interface registration concept. Figure 4 shows the usage of the two programming interfaces in some more detail.

In order to enable former Cell Directory Service based clients to access servers advertising their service offers in the TRADEr, service offers exported into the CDS name

space are at first generated using the NSI interface. This is necessary because the NSI interface expects a special internal format for name service entries. Subsequently, the service attributes used by the TRADEr are added to the CDS entry via the XDS/XOM interface. Therefore, former CDS based servers are able to use the capabilities of the more powerful TRADEr functions without affecting existing distributed applications.

In addition to standard name service operations, the XDS/XOM interface provides powerful functions for searching the X.500 name space. The corresponding search operation, namely the ds_search function, is extensively used for the realisation of the service selection strategies offered by the TRADEr. In the current DCE release, however, the ds_search function is restricted to the X.500 name space and can not be used for searching the CDS name space as well. Because of this limitation, the TRADEr simulates searching the CDS name space by using the CDS group concept by collecting all service offers in a well–known group entry for each CDS directory.

5 Conclusions and Outlook

Work reported in this article is based on the importance of early evaluations of prototype implementations of trading concepts in the context of existing distributed systems platforms as, for example, the OSF Distributed Computing Environment (DCE). A concrete implementation goal is an orthogonal and smooth integration of basic trader functions into, in particular, already available service registration and management mechanisms.

Related experiences from the TRADEr prototype implementation as available so far are twofold: On the one hand, DCE already provides powerful functions for distributed application programming, many of which could also be used beneficially for the TRADEr's trading function implementations. On the other hand, however, DCE concepts already available for service (type) description are still unsatisfactory and corresponding DCE functions still lack important (especially: type management) mechanisms which are necessary for service management and mediation in open distributed environments. Therefore, extended type management functions have to be developed separately, based on modern programming language (polymorphic type) concepts and on specific service description extensions (as, e. g., for protocol and workflow management specifications) as needed in open distributed environments. In summary, experiences with the TRADEr prototype implementation have shown that trading can be smoothly integrated into DCE, but DCE functions have to be extended substantially in order to support and use, in particular, realistic service descriptions based on modern type management functions.

In the TRADEr prototype implementation, such extensions currently concentrate on more elaborate *type management* functions (as described above), support for distributed *trader interworking*, and support for inclusion of *dynamic* (i. e. time varying) *attributes* into extended service descriptions. In particular, techniques to extend service descriptions with additional semantic information are still evaluated and will be integrated step–wise into the COSM/TRADE prototype implementation. In this context, early experiences with service type extensions like finite state machine description of service protocols [ML93] have motivated recent work on using of more powerful description methods also for coordinating *composite* services (aspects of *workflow management*) based on Petri nets

[MMML94, MJML95] as mentioned in section 4.1 of this paper.

References

[Ber93] P. A. Bernstein. Middleware – an architecture for distributed system services. Technical Report CRL 93/6, Digital Equipment Corporation, Cambridge Research Lab, March 1993.

[BHJ+87] A. Black, N. Hutchinson, E. Jul, H. Levy, and L. Carter. Distribution and abstract types in Emerald. *IEEE Transactions on Software Engineering*, 13(1):65–76, 1987.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[Fou92] Open Software Foundation. *Introduction to OSF DCE*. Prentice–Hall, Englewood Cliffs, New Jersey, 1992.

[IBR93] J. Indulska, M. Bearman, and K. Raymond. A type management system for an ODP Trader. In *Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing*. North–Holland, Elsevier Science Publishers B.V., September 1993.

[ISO94] ISO/IEC JTC 1/SC 21. Recommendation X.9tr/Draft ODP Trading Function ISO/IEC 13235: 1994/Draft ODP Trading Function, July 1994. ISO/IEC JTC 1/SC 21 N9122.

[MJM94] K. Müller, K. Jones, and M. Merz. Service mediation and management in open distributed systems. In B. Wolfinger, editor, *Innovationen bei Rechen– und Kommunikationssystemen — Eine Herausforderung für die Informatik*, Informatik Aktuell, pages 219–226. Springer–Verlag, Berlin,Heidelberg, August 1994. in German.

[MJML95] K. Müller-Jones, M. Merz, and W. Lamersdorf. Cooperative applications: Integrated application process control and service mediation in open distributed systems. University of Hamburg, Germany, submitted for publication, in German, 1995.

[ML93] M. Merz and W. Lamersdorf. Cooperation support for an open service market. In *Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing*, pages 329–340. North–Holland, Elsevier Science Publishers B.V., 1993.

[MML94] M. Merz, K. Müller, and W. Lamersdorf. Service trading and mediation in distributed computing environments. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS '94)*, pages 450–457. IEEE Computer Society Press, 1994.

[MMML94] M. Merz, D. Moldt, K. Müller, and W. Lamersdorf. Workflow modeling and execution with coloured petri nets in COSM. University of Hamburg, Germany, submitted for publication, 1994.

[OMG91] The Common Object Request Broker: Architecture and Specification. Digital Equipment Corporation, Hewlett–Packard Company, HyperDesk Corporation, NCR Corporation, Object Design Incorporated, SunSoft Incorporated, 1991.