12

Achieving Interoperability between CORBA and DCE Applications Using Bridges

Zhonghua Yang and Andreas Vogel
CRC for Distributed Systems Technology
Level 7 Gehrmann Laboratories
University of Queensland
Australia 4072

email: (yang,andreas)@dstc.edu.au

Abstract

This paper identifies a new heterogeneity – the heterogeneity of middleware platforms. To overcome the interoperability problem caused by this heterogeneity we present an application-level approach – bridges. Various kinds of bridges are identified, designed and discussed. A proof of concept is given by a case study.

Keywords

Heterogeneity, Middleware, Platform, Interoperability, CORBA, DCE, Bridge, Type mapping

1 INTRODUCTION

OSF's Distributed Computing Environment DCE [Open Software Foundation (1994)] and OMG's Common Object Request Broker Architecture CORBA [Object Management Group (1995b)] are today established as the two main middleware platforms. The two environments have been developed with different motivations. DCE's primary goal was to overcome the barrier of heterogeneous transport protocols, computer architectures and operating systems. CORBA's motivation was distributed object computing. Although coming from different motivations, DCE and CORBA occupy a very similar technology niche, i.e. middleware platforms, and will continue to coexist. This situation creates a new, higher-level heterogeneity problem, the heterogeneity of middleware platforms.

In particular we see the two following problems:

- transformation and substitutability of interfaces types to advertise and find interface specifications in other middleware domains and
- the binding of and interaction between objects in different environments.

A. Schill et al. (eds.), *Distributed Platforms*

An approach to the first problem has been given in [A. Vogel, B. Grey, and K. Duddy (1995)]. To solve the second problem we see the two following approaches:

- **Infrastructure-level approach:** this approach requires the building of an infrastructure which is independent of the middleware platform and which provides sufficient distribution transparencies (as defined in the Reference Model on Open Distributed Processing [ITU/ISO (1995a), ITU/ISO (1995b)] to overcome the heterogeneity problem.
- **Application-level approach:** this approach is based on application objects, *bridges*, which enable the interoperability between CORBA and DCE objects and vice versa.

In this paper we are investigating the application-level approach to the interworking of DCE and CORBA applications. In Section 2 we will present and discuss the general design and various kinds of bridges. Following that, we will present our case study in Section 3. The case study serves as a proof of concept and as a feasibility study for more generic bridges. This case study is driven by the Interworking Trader (IWT) project [A. Vogel, M. Bearman, and A. Beitz (1995)] which aims to prototype the interworking of traders [ITU/ISO (1995c)] which have been independently developed in DCE as well as in CORBA environments. In Section 4, we present how our prototype can be extended to more generic bridges. Finally, we present related work and conclude the paper.

2 GENERAL DESIGN OF BRIDGES

We have identified the following kinds of bridges, *static*, *on-demand* and *dynamic*. In this section, we present general design ideas of these kinds of bridges. Following that, we will discuss the application of the different kind of bridges in different scenarios. Finally we will identify common problems to all of these bridges and present our approaches to solve them.

A general requirement to all bridges is that they have to reside in a location that provides run-time environments of the middleware platforms to be bridged. In the most general sense one could understand the location as an 'area' and a bridge itself as a distributed application which uses third party communication mechanisms. In this paper, however, we will restrict the concept of a location to one host machine and the bridge to be non-distributed.

The basic mechanism of a bridge is

(1) to receive an operation invocation in one middleware domain,
(2) to transform the parameters and their types in the other middleware's type system,
(3) to invoke an equivalent operation in the other middleware domain,
(4) to transform the output parameters and the results in the originating type system,
(5) to return the output parameters and results to the originating client.

All the identified kinds of bridges will follow this basic mechanism.

2.1 Static bridge

In static bridges, the interface at the both sides of the bridge is statically defined by their respective IDL definitions (ref. Figure 1). The IDL definitions of operations and types are compiled into *stubs* which are linked into the bridge code. A client can only invoke other operations whose translation was determined at compile-time, hence *static*. If the composition of the applications is changed, i.e. adding operations, the new stub definitions have to be generated and recompiled. The type mapper needs also to be extended for the new operations.

The bridge acts as a server in the one middleware domain and as a client in the other. Both interfaces have to provide equivalent interfaces.

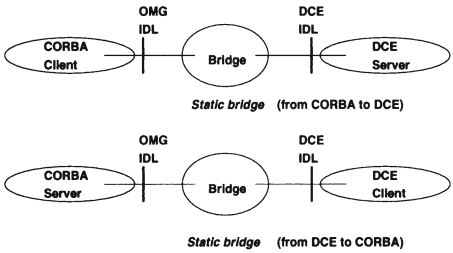

Figure 1. Static bridges.

Within the bridge, the mapping of the IDL definitions (including the semantics of types and operations) is necessary for bridging two different domains.

2.2 On-demand bridge

On-demand bridges can be considered as an extension to static bridges. The idea is to provide a *bridge factory* which automates the code-generation of a static bridge. The inputs to a bridge factory are representations of a CORBA interface and a DCE interface. The mapping rules for the respective types, which are generalisations of the ones found in static bridges, must be coded into the bridge factory (compare the type mapper in Section 3). A scenario for the CORBA to DCE bridging is illustrated in Figure 2.

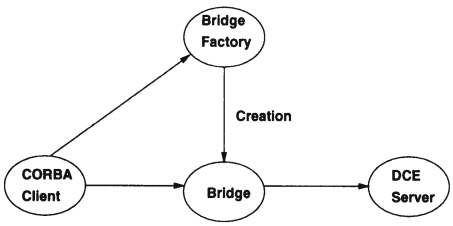

Figure 2. On-demand bridge.

The invocation of the bridge factory can be part of the binding process which would make the creation of the bridge transparent to a client. This idea is presented in greater detail in Sections 2.5 and 4.1.

2.3 Dynamic bridge

Dynamic bridges support any IDL interface, i.e. the interface type does not have to be known at compile time. To enable this dynamic nature of the bridge, the concept of dynamic invocation interfaces is required. The dynamic invocation refers to constructing and issuing a request (or operation) whose signature is possibly not known until run-time.

The CORBA *Dynamic Invocation Interface* is specified in the CORBA specification [Object Management Group (1995b)] and is implemented in most of the available ORBs. DCE does not provide such a facility. However, the feasibility of a DCE dynamic invocation interface has been shown [David Arnold, Andy Bond, and Alexei Vovenko (1995)]. The mechanisms required by a dynamic bridge are illustrated in Figure 3.

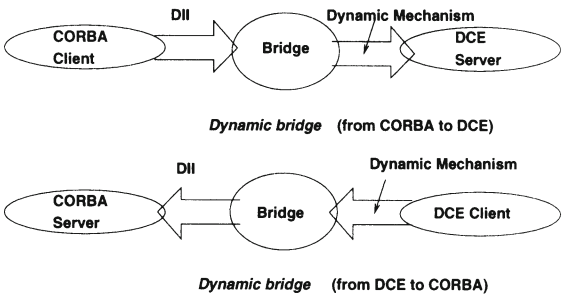

Figure 3. Dynamic bridges.

2.4 Discussion

The static approach is appropriate in an environment where the interface specifications are stable, e.g. when embedding legacy applications. From the server point of view, a static bridge can be seen as a proxy for another middleware domain.

When interface specifications are likely to change or services are frequently created, the static bridge approach becomes a programming bottleneck. This can be overcome by using either on-demand bridges or dynamic bridges.

Using an on-demand bridge requires code generation for a particular bridge and its compilation. However, once the bridge is compiled it can be reused, e.g. using a type manager [W. Brookes, J. Indulska, and A. Vogel. (1995)] to locate a template of the bridge or a trader [A. Beitz and M. Bearman (1994)] to search for an appropriate bridge instance (see also Section 2.5).

Dynamic bridges are more flexible and do not need re-compiling. However, they are complex to implement.

2.5 Common problems

Problems which are common to all kind of bridges are caused by the heterogeneity of the middlewares crossed by the bridges. We have identified the following problems:

Type mapping The type mapping problem can be divided into two sub-problems. The first problem is the creation of equivalent interface types specified in OMG IDL and DCE IDL, respectively, or the validation that two interface types specified in different IDLs are equivalent. This problem has been approached in [A. Vogel, B. Grey, and K. Duddy. (1995)] resulting in implementation of IDL transformation tools [A. Vogel and B. Grey (1995)].

The second problem is the transformation of types on the programming language level. Our approach is presented in the context of our prototype in Section 3.2.

Addressing and naming An address provides the necessary information so that one object can bind to the addressed object. DCE addresses are called binding handles, CORBA addresses are called object references. To avoid the low-level concept of addresses, DCE as well as CORBA provide naming services. DCE's naming service is know as *Cell Directory Service (CDS)*. CORBA's naming service is defined as one of the *Common Object Services* [Object Management Group (1994)].

Neither addresses nor names specific to particular middleware are appropriate when crossing middleware boundaries. This problem can be solved by using *federated names* and a *locator* as suggested in [A. Beitz, M. Bearman, and A. Vogel (1995)]. A locator resolves a federated name into an address. A federated name has three components, a *naming system type*, e.g. CDS, a context reference, e.g. a DCE cell, and a name, e.g. a local CDS entry name "/.:/examples/example1". Depending on the type of the naming system, the locator selects an appropriate name server which resolves the name into an address.

Applying this procedure in a bridge would result in the following scenario. A client approaches the bridge with a federated name for the requested server. The bridge would use a locator to resolve the name into an address suitable for a binding in the server's middleware.

Binding Our solution to the naming and addressing problem requires corresponding extensions to the middleware binding concepts. DCE provides functions to establish bindings which are based on the direct or indirection provision of a binding handle. In CORBA, each object which is defined in the interface specification provides a binding function. This function also relies on the provision of at least parts of an object's reference.

To accommodate our naming solution, higher-level binding functions are needed which support federated names. The semantics of these operations are that they behave as the local binding function if the specified object is in the same name space, i.e. using the same naming systems. If the requested object is in a different name space, the binding function selects an appropriate bridge, binds to the bridge and passes the federated name to the bridge.

Trading The selection of an appropriate bridge involves the use of a type manager [J. Indulska, M. Bearman, and K. Raymond (1993)] and a trader [ITU/ISO (1995c)]. The appropriate service type is selected from the type manager. Based on the service type, the trader selects an instance of such a server, i.e. bridge. The trader itself works with federated names to identify the bridge.

Within the DSTC, an infrastructure has been implemented which provides all the supporting objects as required above: a locator [A. Beitz, M. Bearman, and A. Vogel (1995)], a type manager [C.J. Biggs, W. Brookes, and J. Indulska (1994)] and a trader [A. Beitz and M. Bearman (1994)].

We explain the concept of high-level binding using the scenario illustrated in Figure 4:

(1) The client requests a binding to a server identified by a federated name.
(2) The binder determines that the requested object is outside the name space of the client using the locator. Hence the binder needs a bridge to establish a connection to the server.
(3) The binder determines the necessary type of the bridge, i.e. service type, using the type manager.
(4) The binder selects a bridge using the trader based on the bridge service type. If it can't find an appropriate bridge, the binder can instantiate a bridge template (which is also held in the trader).
(5) The binder binds to the bridge (local binding) and passes the federated name of the requested server.
(6) The bridge resolves the federated name of the server using the locator.
(7) The bridge binds locally to the server.
(8) The binder establishes a binding between the bridge and the client; the steps 2-7 are hidden from the client.

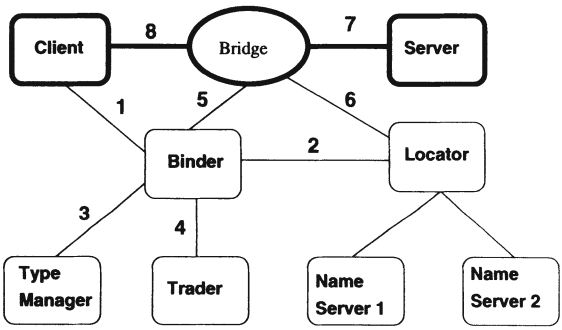

Figure 4. High-level binding based on bridges.

Security Currently DCE provides security services but OMG has yet to adopt a security service. Therefore, this problem cannot currently be addressed.

3 A PROTOTYPE

In this section, we describe our prototype implementation of a CORBA-DCE bridge for the IWT project. This bridge is static and provides two-way interworking functionality between DCE and CORBA. We have used Digital's DCE [Digital Equipment Corporation

(1992a), Digital Equipment Corporation (1992b)] and IONA's Orbix [IONA Technologies (1995)] on Digital Alphas for the prototyping. In the remainder of the section, we present our case study by illustrating CORBA-to-DCE bridging; the other direction bridging (DCE-to-CORBA) is similar.

3.1 Functionality and Design

The prototype of our CORBA-to-DCE bridge requires both a DCE and a CORBA environment (ref. Figure 5). From the point of view of the CORBA environment, the CORBA-to-DCE bridge is implemented as an CORBA server. CORBA clients use the CORBA binding mechanism to bind to the bridge and then use CORBA mechanisms to invoke operations.

To the DCE environment, the bridge appears as a DCE client. The bridge uses the DCE mechanism to bind to the server.

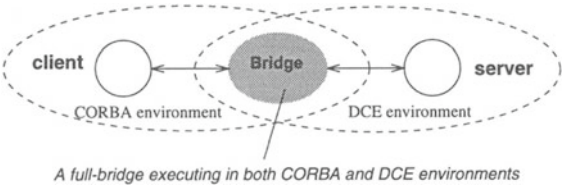

Figure 5. The execution environment for a full-bridge.

When receiving an operation invocation from the client the bridge translates the operations parameters and their respective types from the CORBA language binding into the DCE language binding (see Section 3.2). The bridge then invokes the equivalent operation at the server.

A complementary process takes place for passing the results of the operation.

The structure of the implementation of the CORBA-to-DCE bridge is illustrated in Figure 6. There are three main components: the CORBA server, the DCE client and the type mapper that links them.

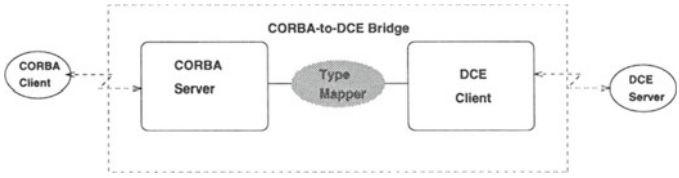

Figure 6. A CORBA-to-DCE Bridge.

3.2 Type mapping

In order to make a meaningful invocation of a DCE service by a CORBA client, the interface specification of the CORBA server and the DCE server have to be equivalent. The equivalence of OMG IDL types and DCE IDL types has been investigated [A. Vogel, B. Grey, and K. Duddy (1995)] and tools which transform IDL specifications (according to this equivalence) have been implemented [A. Vogel and B. Grey (1995)]. In our prototype for the IWT CORBA-to-DCE bridge, we started with a DCE IDL specification and have generated the corresponding OMG IDL specification with the above mentioned tools. We illustrate below the translations of types by a list type from the IWT specification, and the equivalence of two example types. The pointer based list structure in DCE IDL is recognised by the transformation tool and transformed into an OMG IDL sequence.

```
/* DCE IDL type */                                  // OMG IDL type
typedef struct PropertyValueListType {
    PropertyValueType *propertyValue;        ⟹    typedef sequence <OMG_PropertyValueType>
    [ptr] struct PropertyValueListType *next;       OMG_PropertyValueListType;
} PropertyValueListType;
```

The DCE IDL and OMG IDL definitions are compiled into the respective programming language mappings. In our case, these are C for DCE and C++ for Orbix. It is the main task of the type mapper to translate the arguments of a CORBA operation into the corresponding ones of a DCE operation. The following illustrates the C/C++ types of the above shown IDL types:

```
/* DCE C type */                                    // Orbix C++ type
typedef struct PropertyValueListType {              struct IONAMC_IDL_SEQUENCE_iwt_types_OMG_PropertyValueType {
    PropertyValueType *propertyValue;                   unsigned long _maximum;
    struct PropertyValueListType *next;      ⟹         unsigned long _length;
}PropertyValueListType;                                 iwt_types::OMG_PropertyValueType *_buffer;
                                                        // C++ member functions omitted...
                                                    };
```

For the above example the type mapper counts the number of elements in the DCE based list and sets `_length` accordingly. It also maps each element of the DCE based list into `_buffer[i]`, where `i = 0, ..., _length-1`.

The type mapper handles the translation of the respective types preserving their semantics.

4 EXTENDING THE PROTOTYPE

In this section we present two possible extensions to our prototype.

4.1 Towards bridges on-demand

As explained in Section 2.2, the difference between static bridges and on-demand bridges is the run-time construction of the on-demand bridge.

The first extension is to develop a bridge factory. Figure 7 illustrates the design of a

bridge factory for CORBA-to-DCE bridges. A factory for DCE-to-CORBA bridges would be similar.

The bridge to be created must have the following three main components: the two interfaces and a type mapper.

The interfaces are created by using the respective IDL compilers which generate the stub code. To ensure the substitutability of the interface [A. Vogel, B. Grey, and K. Duddy (1995)], we will use our cross-IDL translators [A. Vogel and B. Grey]. Figure 7 illustrates the transformation of DCE interface specification into an equivalent specified in OMG IDL.

Additional code is required for the implementation of the operations as specified in the interface type. This code calls the operations in the other domain after transforming the parameters according to the type mapper rules. The type mapper rules can be provided by a library.

The transformation tools are implemented using the compiler generator [P. van Eijk and A. Belinfante (1990)]. Kimwitu supports abstract syntax trees (AST) and functions to manipulate and transform these trees. Using this technology, the remaining code can be easily generated from the AST (refer to dotted parts in Figure 7).

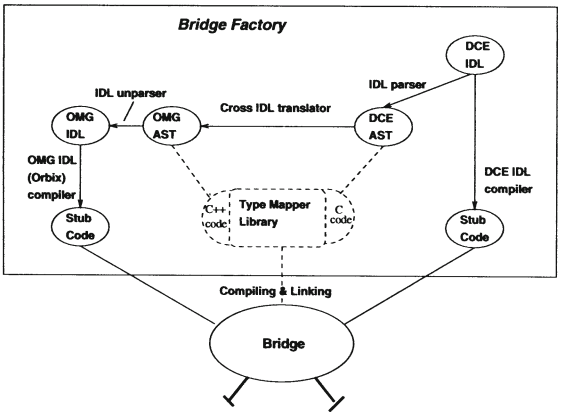

Figure 7. A bridge factory for CORBA-to-DCE bridges.

Finally, the code for the server main loop can be predefined, although it requires some specific parametrisation.

4.2 Towards dynamic bridges

The most demanding task, when implementing a dynamic bridge, is the *dynamic invocation interface* for DCE. An initial prototype has been already implemented [David Arnold, Andy Bond, and Alexei Vovenko (1995)]; however its extension to handle all DCE data types requires a major implementation effort.

The mapping of the operations parameters would use again a type mapper library.

5 RELATED WORK

While there appears to be no reported work on bridging DCE and CORBA applications, there are some related work on supporting application integration and interoperating among technology domains.

There are some approaches, e.g. [J. Dilley (1995), E. N. Elnozahy and V. Ratan (1995)], which use DCE for communications among CORBA objects to take advantage of the secure and vendor-independent DCE RPC. These approaches include an OMG-IDL-to-DCE-IDL mapping. These approaches could be extended to enable interaction between CORBA clients and DCE servers. This extension is equivalent to our CORBA-to-DCE bridge factory.

Object Linking and Embedding (OLE) is Microsoft's vision of object technology [Kraig Brockschmidt (1993)]. OLE Network Portal is a bridging mechanism built in Digital's ObjectBroker V2.5 [Digital Equipment Corporation (1994)]. ObjectBroker is Digital's implementation of CORBA 1.2 and additionally contains Network Portal. OLE Network Portal implements a set of standard Microsoft OLE V2 interfaces, and provides a bridge for data objects from remote CORBA server to the Microsoft OLE container application. In our terminology OLE Network Portal is a static bridge.

Another approach which should be mentioned in the context of this paper is OMG's *Universal Networked Objects Specification* [Object Management Group (1995a)]. Although it tackles inter-ORB interoperability, similar problems are addressed. However, the solution belongs to the infrastructure-level approaches.

6 CONCLUSION

In this paper we have presented our approach to application-level interoperability between DCE and CORBA application using bridges. We have identified three kind of bridges, static, on-demand and dynamic bridges. The design for these kind of bridges have been presented and common problems identified. We have presented our approaches to solve these problems including type mapping, addressing and naming, binding,

The feasibility of the bridge approach has be demonstrated by our prototype implementation, static CORBA-to-DCE bridge. This bridge is used in the IWT project. We have also demonstrated how the prototype can be extended towards on-demand bridges and dynamic bridges.

Since our approach does not rely on any specific characteristics of DCE or CORBA, the concepts and design decision should be applicable to the interworking of other middleware platforms.

Acknowledgements

This research has been funded in part by the Cooperative Research Centres Program through the department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

We would like to thank our colleagues in the Architecture Unit at the DSTC, in particular Mirion Bearman and Kerry Raymond for their comments on this paper, and our partners in the IWT project.

REFERENCES

David Arnold, Andy Bond, and Alexei Vovenko. (1995) A Dynamic Invocation Interface for DCE. In *the Proc. of DSTC Symposium*, Brisbane, July 1995. DSTC Pty Ltd.

A. Beitz and M. Bearman. (1994) An ODP Trading Service for DCE. In *Proceedings of the First International Workshop on Services in Distributed and Networked Environments (SDNE)*, pages 34–41. IEEE Computer Society Press, Los Alamitos, CA, 1994.

A. Beitz, M. Bearman, and A. Vogel. (1995) Service Location in an Open Distributed Environment. In *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE)*, pages 28–34. IEEE Computer Society Press, Los Alamitos, CA, 1995.

C.J. Biggs, W. Brookes, and J. Indulska. (1994) Enhancing Interoperability of DCE Applications: A Type Manager Approach. In *Proceedings of the First International Workshop on Services in Distributed and Networked Environments (SDNE)*, pages 42–49. IEEE Computer Society Press, Los Alamitos, CA, 1994.

Kraig Brockschmidt. (1993) *Inside OLE 2.* Microsoft Press, 1993. ISBN: 1-55615-618-9.

W. Brookes, J. Indulska, and A. Vogel. (1995) A Type Management System Supporting Interoperability of Distributed Applications. In *The Proc. of DSTC Symposium*, Brisbane, July 1995. DSTC Pty Ltd.

Digital Equipment Corporation. (1992a) *DCE Application Development Guide.* Maynard, MA, 1992.

Digital Equipment Corporation. (1992b) *DCE Application Development Reference.* Maynard, MA, 1992.

Digital Equipment Corporation. (1994) *ObjectBroker Reference Manual.* Digital Equipment Corporation, Maynard, MA., August 1994.

J. Dilley. (1995) Using OMG IDL to write OODCE applications. Technical Report HP Document Number NSA-94-025, Hewlett-Packard, 1995.

E. N. Elnozahy, V. Ratan and Mark E. Segal. (1995) Experiences Using DCE and CORBA to Build Tools for Creating Highly-Available Distributed Sytems. In K. Raymond and E. Amstrong, editors, *Open Distributed Processing III*, pages 488-499. London, 1995. Chapman & Hall.

Open Software Foundation. (1994) *OSF DCE Application Development Guide.* Open Software Foundation, 11 Cambridge Center, Cambridge, MA., revision 1.0, update 1.0.2 edition, 1994.

Object Management Group. (1994) *Common Object Services Specifications*, volume 1. John Wiley & Sons, Inc, 1994.

Object Management Group. (1995a) *Universal Networked Objects.* Object Management Group (OMG), Framingham, MA, March 1995.

Object Management Group. (1995b) *The Common Object Request Broker: Architecture and Specification* (Revision 2.0). Object Management Group (OMG), Framingham, MA., July 1995.

J. Indulska, M. Bearman, and K. Raymond. (1993) A Type Management System for an ODP Trader. In J.De Meer, B. Mahr, and S. Storp, editors, *Open Distributed Processing, II*, pages 169–180. North-Holland, 1993.

ITU/ISO. (1995a) Basic Reference Model of Open Distributed Processing — Part 2: Descriptive Model. International Standard 10746-2, ITU-T Recommendation X.902, 1995.

ITU/ISO. (1995b) Basic Reference Model of Open Distributed Processing — Part 3: Prescriptive Model. International Standard 10746-3, ITU-T Recommendation X.903, 1995.

ITU/ISO. (1995c) Reference Model of Open Distributed Processing - Trading Function. Draft International Standard ISO/IEC DIS 13235, June 1995.

IONA Technologies. (1995) *The Orbix Programmer's Guide (V 1.3.1)*. IONA Technologies Ltd., Dublin, Ireland, February 1995.

P. van Eijk and A. Belinfante. (1990) The Term Processor Kimwitu, Manual and Cookbook. Technical Report INF-90-45, University of Twente, 1990.

A. Vogel, M. Bearman, and A. Beitz. (1995) Enabling Interworking of Traders. In K. Raymond and E. Armstrong, editors, *Open Distributed Processing III*, pages 185-196, London, 1995. Chapman & Hall.

A. Vogel and B. Grey.(1995) Translating DCE IDL in OMG IDL and vice versa. Technical Report 22, CRC for Distributed Systems Technology, Brisbane, 1995.

A. Vogel, B. Grey, and K. Duddy. (1995) Overcoming the Heterogeneity of Middleware Platforms. Submitted for publication, June 1995.

About authors

Zhonghua Yang is currently a senior researcher at the Cooperative Research Center for Distributed Systems Technology (DSTC), Australia. Prior to his current appointment, he spent two years in University of Alberta as a visiting professor (1991–1993), and a few years with the Distributed Software Engineering group at the Department of Computing, Imperial College (London) (1980–1982); He spent a significant part of his professional career with the Beijing Institute of Information & Control, under the Ministry of Aerospace, China.

His research interests include: distributed systems, distributed platforms and their interworking, distributed software architecture, and system integration.

Andreas Vogel is senior research scientist with the CRC for Distributed Systems Technology (DSTC) in Brisbane, Australia. Prior to this appointment, Andreas worked in the CITR project "Broadband Services" at University of Montreal, Canada, and at Humboldt-University Berlin, Germany.

His research interests are in the area of (heterogeneous) distributed systems, in particular, open distributed processing including service trading and type management, distributed middleware, formal description techniques, distributed multimedia systems, quality of service and the world wide web.

More detailed information is available from his personal WWW service:
http://www.dstc.edu.au/AU/staff/andreas-vogel.html