A Viewpoints Approach to Designing Group Based Applications

D.H.Akehurst, A.G.Waters, J. Derrick
Computing Laboratory, University of Kent
Canterbury, Kent, CT2 7NF, UK
{D.H.Akehurst, A.G.Waters, J.Derrick}@kent.ac.uk

Keywords: Groups, multicast, Open Distributed Processing, viewpoints, system design.

Abstract
There is increasing interest in group-based applications for video distribution, multimedia conferencing, publish and subscribe etc. Such applications can use networks efficiently by multicasting (supported traditionally at the network layer, but also now at the application layer). Designing such large-scale distributed systems is a complex task that can be aided by using a viewpoint approach to separate out different concerns, for example to separate object interaction from communications support. This paper extends earlier work on the design of distributed systems that use point-to-point communication to propose a framework within which viewpoints can be used to assist the design of complex applications involving groups and multicasting.

1 Introduction

The Internet supports point-to-point communication between two computers placed geographically anywhere in the world. Other applications operate in a distinctly different manner, to support a group of end-users. Rather than employing a one-to-one communications path between two specified users, they operate over a communications model supporting one-to many and many-to-many connections. One-to-many examples include video distribution, News Broadcasts, Advertising and Stock Price Updates. Examples of applications using many-to-many communication are Video Conferencing and Shared Whiteboards.

Group-based applications have a variety of requirements, which we discuss below. A key requirement is that they make efficient use of the network and this can be done by multicasting, which avoids unnecessary duplication of messages across network links. Multicast provision at the network layer is provided on Internet routers through the *host group* model [8]. The model is based on best-efforts delivery and is open and dynamic: the membership is unknown and anyone can transmit to a group. Blocks of addresses for multicasting are provided in both IPv4 and IPv6.

Concerns remain about the scalability and security of the host group model and the lack of provision for sophisticated applications. Many Internet routers are not yet configured to be multicast-capable. For these reasons, the current trend is to implement Application Layer Multicasting (ALM), whereby the application layer itself takes responsibility for organizing multicast trees and forwarding messages. ALM is less efficient, but has the flexibility to support a wide range of group-based services.

Group-based applications raise many design issues and these issues span networking layers. Host groups are ideal for open information services where there are a small number of informally co-ordinated senders. Comprehensive group support requires authenticated membership, group management, identifiable senders, security and support for real-time Quality of Service (QoS) and for mobility.

Such extra support can be offered at the transport or application layer. IETF proposals at the transport layer are most advanced for Reliable Multicast Transport (RMT) [26]. The approach is to use building blocks from which users can construct the required functionality, such as Forward Error Correction or hierarchical acknowledgement trees. The traditional broadcast ordering protocols could also be provided at this layer. Specific group-based applications also have their individual requirements: for example layered video coding, publish and subscribe mechanisms or floor management for real-time conferencing.

It is clear that designing group-based distributed systems is a complex task. We believe that design methodologies can assist with this task and will enable group-based applications to work over different infrastructures and different engineering environments. Additionally we believe that it will encourage more widespread implementation of group-based applications.

Designs should be able to illustrate multiple instances of objects and interfaces, belonging to different capability classes. Groups can be expected to be large, possibly hierarchically organized and their membership may be unknown and may change frequently, as members join and

leave. Designs should be capable of specifying the required support mechanisms. The methodology should offer designs that are easy to specify and understand and are unambiguous. For example, a design might help the implementers to see the consequences of scaling their groups and of implementing reliability and security options.

Most current designs for multicast group applications are very specific e.g. [27]; they take a simple black-box approach in which the use of specific protocols is also indicated. This approach can serve simple applications well, but we believe that a more general, well-defined and integrated approach is needed and are not aware of other work describing such an approach.

To date, most design languages and methodologies have assumed an infrastructure supporting point-to-point communication. The ISO standard Reference Model for Open Distributed Processing (RM-ODP) addresses the issues involved with the complexities of designing distributed applications. By drawing on ideas from the RM-ODP framework, we demonstrate a means to design group-based applications that make use of multicast technology.

An important requirement is to enable a separation of concerns between the issues involved in designing the computational aspects of an application and the issues involved in designing support for group communication.

Specification from different separations of concern is often referred to as viewpoint-based or aspect-oriented design. There is much research surrounding the use of viewpoints [11, 12, 20] and aspects [6, 9, 13] However, these ideas have not yet made it into the standards for mainstream design approaches such as the Unified Modelling Language (UML); nor do these standardised methodologies and languages provide facility for addressing the issues of Group communication.

The research reported in this paper addresses the group support aspects of the Design Support for Distributed Systems (DSE4DS) project [2, 10] which aimed to extend facilities for the design of multimedia distributed systems, to ensure that they can effectively meet the needs of complex systems which will include the use of stream communication, multicasting and Quality of Service (QoS) constraints. The project has defined UML-related notations and methodologies, which are aimed at addressing the complexities of designing distributed systems. We chose UML because it is widely used as a design medium for object-oriented design. The emergence of the Object Constraint Langauge (OCL) has gone some way to help avoid potential inconsistencies in behaviour specification.

The project has also shown how to derive Timed Automata models from the design [4], which are formally analysed to provide verification of QoS specifications against the functional aspects of the design. The results of the verification are fed back to the designer in the context of the original design and design notation; [4] discusses the verification aspects of our work.

This paper extends the design notations proposed by the project to facilitate specification of group based applications. In particular this paper addresses techniques for designing the communication mechanisms required for supporting ALM.

In Section 2 we introduce the concepts used for our designs. Section 3 introduces the computational design of an example group application, which is extended in Section 4 where we explore alternative designs for a group communication mechanism. Section 5 discusses related work and the paper concludes in Section 6 with a summary and discussion of future work.

2 Viewpoint Based Design

The IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [16] defines the notions of view and viewpoint as important parts of its conceptual framework. Many other works [11, 12, 20] also discus the use of viewpoints and propose viewpoint based approaches to the design of computing systems.

We are specifically interested in the design of distributed computing systems. The ISO/ITU standardisation bodies have published a set of standards called the Reference Model for Open Distributed Processing (RM-ODP) [28]. These standards make use of viewpoints within a framework for addressing the design of distributed systems.

The RM-ODP framework proposes a multi-paradigm specification approach by identifying five separations of concern (viewpoints) and addresses the design of the system from each. Different languages may be used for each viewpoint so that the relative strengths of different specification languages can be exploited.

The five viewpoints are: *enterprise*, *information*, *computational*, *engineering* and *technology*; further information regarding both the reference model and its approach to using viewpoints can be found in [19, 25, 28]. Briefly:

- The Enterprise Viewpoint is concerned with the purpose, scope and policies of the organization of which the system is a part.

- The Information Viewpoint is concerned with information handled by the system.
- The Computational Viewpoint is concerned with the functional decomposition of the system.
- The Engineering Viewpoint is concerned with the infrastructure required to support distribution.
- The Technology Viewpoint is concerned with the choice of technologies with which to build the system.

For the purpose of this paper, we focus on the Computational and Engineering Viewpoints. The concepts addressed within each of these viewpoints are described in the following subsections.

Part 3 of the RM-ODP defines the concepts and structuring rules that define an 'abstract' language for each of the viewpoints. The definitions must be considered abstract, as there is no defined (concrete) syntax specified for any of the viewpoint languages.

The above concepts and rules can be modelled using the OMG's MOF [22] concepts of class, association, generalization etc. (These are a subset of the specification concepts found in the more commonly used UML.) In [3] we model the RM-ODP concepts and propose a concrete notation to be used for the design of concepts within the computational viewpoint. In Section 3 we discuss this approach and extend it by including the Group Management functions offered by an Engineering specification. In Section 4 we propose a UML based notation for the design of concepts within the Engineering viewpoint; focusing specifically on the design of our group communication example.

3 Computational Viewpoint

The Computational Viewpoint is defined in the RM-ODP as being concerned with the logical and functional decomposition of the system into a set of objects that interact at interfaces; this decomposition is to enable distribution. The primary concern of the computational viewpoint is the distribution of processing. However, it should be emphasized that it is **not** concerned with the interaction mechanisms that enable distribution to occur.

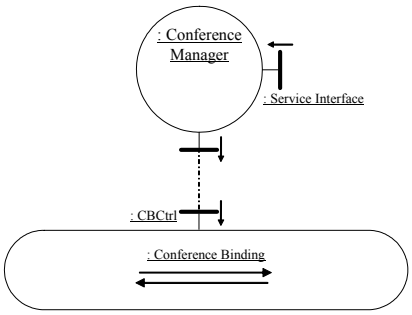

Figure 1 MPM snapshot before users have joined the conference

A computational viewpoint definition of a system should describe an object-based model that defines:

- A set of distributable computational objects
- The interfaces through which an object communicates
- The bindings between interfaces and the interactions they offer
- The actions that a computational object can perform – e.g. creation of objects, interfaces and bindings.

A language enabling the definition of computational viewpoint specifications provides, in essence, a programming model for a generic virtual machine. The mechanisms offered by this virtual machine, model the infrastructure over which the system operates, and is realised by definitions provided as part of the Engineering and Technology viewpoints.

Computational objects (which we abbreviate to compObjects) are self-contained, distributable units of computation – i.e. they exhibit some well-defined independent behaviour. To communicate with other compObjects they require interfaces that depend upon the type of interaction required. The RM-ODP classifies communication interactions into three distinct types – *Operations*, *Signals*, and *Flows* (or Streams). Each interaction occurs between two or more compObjects, and the interface through which the interactions occur indicates the compObject's role as the source or the target of the communication.

In order for the interaction to pass from one compObject interface to another, a communications path is required. Such communication paths are modelled in the computational viewpoint as bindings. The ODP standard defines a binding as follows: "A binding is an infrastructure-provided configuration of network connections and behaviour."

A *primitive binding* ties one interface directly to another (of the same type), enabling interactions to occur between the two compObjects. However, in many situations, a more complex *compound binding* is necessary. For example, to involve more than two compObjects in an interaction, or to offer application based control over the communications path (e.g. to manipulate the QoS

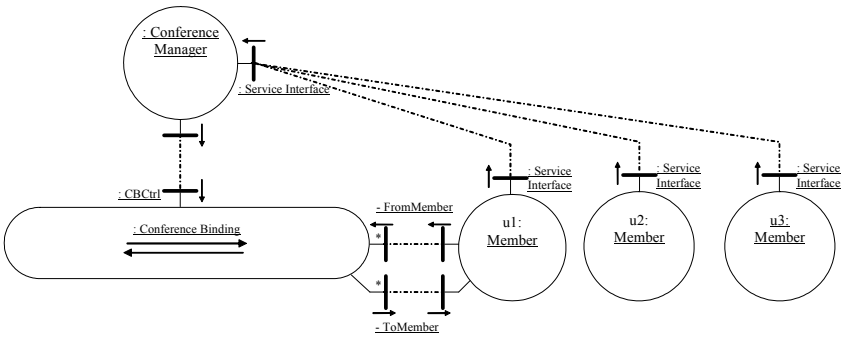

Figure 2 MPM snapshot showing one user having joined the conference

characteristics), or perhaps to facilitate monitoring of the communication.

For modelling system applications that involve group communication, compound bindings are of significant interest and importance. Compound bindings model a communications path between two or more, different, compObject interfaces. Thus, as illustrated by examples in the RM-ODP standard, a compound binding can be used to model the presence of a multicast communications path between the source of a multicast interaction and a number of destination compObjects.

The Multicast functionality is an infrastructure provided facility, where that infrastructure is defined by the Engineering and Technology specifications, which detail the particular multicast implementations to be used. Thus, the multicast functions become part of the Computational Language virtual machine and can be treated as atomic actions. However, it is still necessary to provide interfaces in a computational viewpoint description that enable compObjects to invoke the multicast facilities. This is achieved by providing a 'Control' interface on a multicast binding, that offers (as an example) operations for adding or removing members of the multicast target group.

To illustrate the group-specific facilities of our design approach and languages we present, in the following subsections, the computational design of a multi-party messaging system.

3.1 Design of a Multi-Party Messaging System

Our example of a Multi-Party Messaging (MPM) System enables members to join a conference in which each participant can send text messages to all other participants. Members can join and leave the conference by informing the conference manager. The example has been kept small for the purposes of explanation of the design concepts.

We start by showing some Snapshots that illustrate possible configurations of objects in the system. These highlight the facilities for representing groups of objects in the diagrams. Following this we show the specification of object templates and interface signatures from which the system can be instantiated, showing how groups are facilitated in such specifications.

The final part of the example illustrates our specification technique for the behaviour of the object components in the system; this raises some issues (which are discussed) regarding the use of Group functions offered by the underlying Engineering environment. Our work in [3, 4] show how QoS specifications are added to a design.

3.2 Using the System (Computational Snapshots)

To make use of the messaging system, each user must instantiate a Member object; this compObject enables connection to a ConferenceManager, which provides an interface to which members are bound; that interface supports operations to join the conference. Figure 1 shows a snapshot of a system before any users have joined.

Each Member object instantiates a client *ServiceInteface* which must be bound to the conference manager's server *ServiceInterface*. After users have made a connection to the conference manager, the next action is for them to join the conference; by sending a message to the conference manager – via its *ServiceInterface*. This results in bindings to the *FromMember* and *ToMember* interfaces of the conference binding object.

Figure 2 shows that the *u1 Member* object has joined the conference, with users' *u2* and *u3* bound to the conference manager. If we were to depict many more users within this snapshot, it would soon become overwhelmed with *Member* objects. Additionally we may wish to depict the group of *Member* objects without specifically

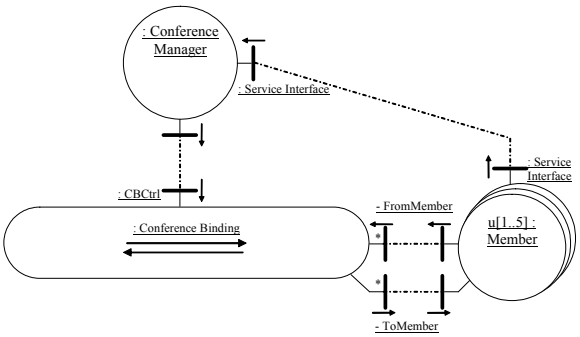

Figure 3 MPM snapshot showing 5 users having joined the conference

identifying them or the size of the group. To achieve this we facilitate the use of a stacked circle to represent a *group* of objects. Figure 3 shows a Computational Snapshot, showing five users as a *group*, participating in the conference

Each group member supports a *ServiceInterface*, bound to the conference manager, and both a *FromMember* and *ToMember* interface, bound to the conference binding. The number of group members may be shown (as in Figure 3); or a group of unspecified size can be defined using a '`[0..*]`' label.

The interface symbol attached to the group of objects represents a group of interfaces (each member of the group has its own interface). The dashed line representing a primitive binding may represent either:

- that each of the interfaces at the group end are bound to a single interface at the opposite (conference binding) end; or
- that each interface at the group end is bound to a different interface instance at the opposite end.

This difference is distinguished by adding a '`*`' to the opposite end of the binding line if there are a group of interfaces at that end, as in the case of the *FromMember* and *ToMemeber* interfaces; or by omitting the '`*`' if there is a single interface, as in the case of the *ServiceInterface*.

3.3 Computational Templates

We can construct a computational viewpoint template diagram (similar to a UML class diagram) that defines the component types for our example application, Figure 4. In a similar fashion to class diagrams, Interface Signatures and compObject Templates are defined using compartmentalised boxes. The top compartment illustrates the name of the template type, and the bottom compartment shows the properties (attributes or operations) of the type. We use an icon in the top right of the box and a label enclosed in guillemots («..») to aid the visual distinction between interface, compObject and binding object templates.

The UML symbol for composition (a line with a solid diamond at the tail) shows that a compObject or binding object template offers a particular interface type in the role of producer/consumer, client/server, or initiator/responder, as indicated by a label in guillemots attached to the composition line.

This specification defines two computational object templates, *Member* and *ConferenceManager*. These two compObjects both support instances of the *ServiceInterface* interface signature, one in the role of server the other as a client. The *ServiceInterface* signature defines two operations, one for a member to join the conference, the other enabling a member to leave the conference. Each of

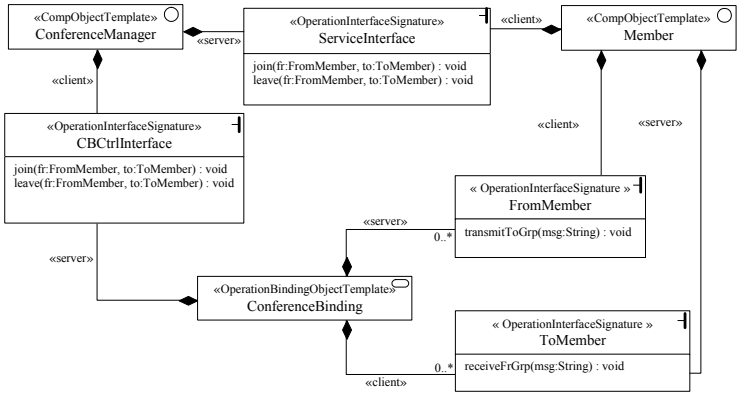

Figure 4 Computational Template Diagram

these operations takes a reference to an instance of the *FromMember* and *ToMember* interfaces as parameters; this enables the specific member object to be identified and facilitates binding of those interfaces to interface instances of opposite role on the conference binding object.

The *ConferenceBinding* specification defines a template for an operation binding object; it supports a *CBCtrlInterface* interface signature in the role of server, used by a conference manager object to communicate joining and leaving of members. In addition the *ConferenceBinding* template supports a group of *FromMember* interfaces in the role of server, from which messages are received; and a group of *ToMember* interface signatures, to which messages are sent.

3.4 Computational Behaviour

The functional behaviour of the system can be defined using the Statechart [15] notation. Events are caused by each interaction that occurs at a particular interface. These events are used as the semantic mechanism to trigger transitions in a Statechart.

Actions caused by a transition are either operations called on management object interfaces (supported by the 'engineering virtual machine' – ODP management functions) or the activation of an interaction at a connected output interface.

The behaviour of our Conference Binding object is illustrated by Figure 5. This shows two threads of behaviour. The top thread shows that the binding object must always forward messages transmitted by any group member to all group members. The bottom thread indicates that binding handles requests from members to join and leave the group.

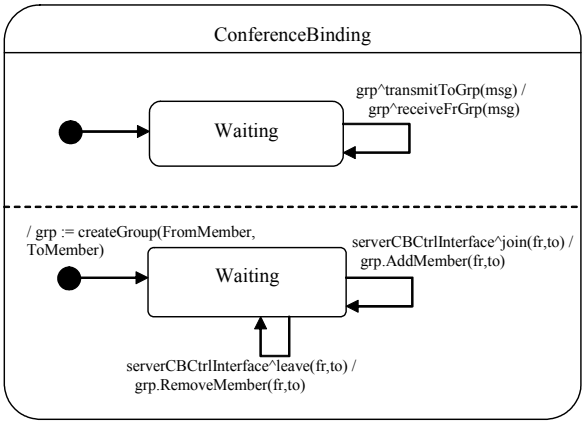

Figure 5 Computational Behaviour of ConferenceBinding

This thread initially creates a group identifier, to which further group interactions are sent (as described below).

3.5 Events and Messages

Interaction between computational objects is via the interfaces supported by each object. An object may respond to messages or events received at a specific interface and may cause events or message to be sent at other interfaces. Sending or receiving a message (at an Operational interface) can be treated in the same manner as sending or receiving an event (with the name of the operation) at that interface.

In general an object will respond to events at server, consumer and responder interfaces, and raise events at client, producer and initiator interfaces. Events may also be raised at server interfaces in order to respond with results or exceptions to client-server interactions. An event is identified jointly by the identification of an interface and the name of an Operation, Signal or Flow defined for that interface. Interfaces can be identified using explicitly given names or by using a combination of the role and interface signature name; e.g. in our example, a conference manager object can refer to the event caused by receiving a join operation message with the following expression:

```
serverCBCtrlInterface^join(fr,to)
```

The first part of the expression gives the role and signature name of an interface, in the context of a conference manager object there is only one such interface, thus the identifier is unambiguous and sufficient. The last part refers to the *join* operation defined for *ServiceInterface* interfaces, the parameter values of which can be identified using the names *fr* and *to*. The two parts are separated using a carat '^', this is the OCL [23] notation for referring to messages; we use this notation in order to stay with a notation familiar to the UML family of languages.

The same notation is used both to refer to *events* and *actions*. The context of the expression determines the intended meaning. In a Statechart, such expressions would occur in the context of a transition as either:

- the event part, for identifying events to respond to; or
- the actions part, for causing operations and signals to be sent

3.6 The Group Management Function

The RM-ODP does not give much detail regarding the group management function; it simply states that this

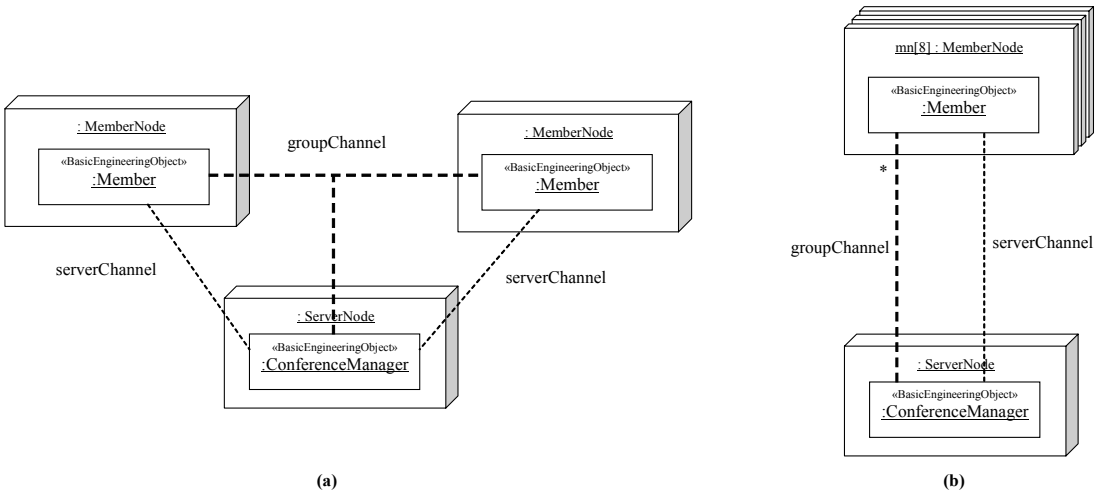

Figure 6 Engineering snapshots of the Multi-Party Messaging System

function "provides the necessary mechanisms to coordinate the interactions of objects in a multi-party binding."

The group function requires the definition of an Interaction Group and manages:

- Interaction – deciding which group members participate in which interactions.
- Collation – derivation of a consistent view of interactions (including failures).
- Ordering – ensuring that interactions are correctly ordered.
- Membership – dealing with addition, removal, failure and recovery of members.

In our approach, to provide reference in the computational viewpoint, to engineering support of the group function we provide the following set of functions, available for use within behaviour specifications:

- createGroup(sig:InterfaceSignature) – returns an identifier for a group; each member of the group must support the specified interface signature.

On each group identifier we can call the following operations:

- groupId.AddMember(ref:InterfaceReference) – adds a member to the group; the member is referred to using the passed interface reference which must be a reference to an interface signature of the correct type (defined when the group was created).
- groupId.RemoveMember(ref:InterfaceReference) – removes a member from the group.
- groupId^<DefinedOperation> - Any operation defined on the interface signature can be sent as a message to every member of the group by calling that operation on the group identifier.

The '.' notation is used for adding and removing members as we are calling an operation for interpretation by the interaction group concept itself, rather than intending a message to be sent to each member of the group.

4 Engineering Viewpoint

An Engineering viewpoint design describes the infrastructure and resources required to support interaction between objects; and rules for structuring the communication channels over which interactions occur.

Processing resources are known as *Nodes*. Through a hierarchy of *Clusters* and *Capsules* (not discussed here, see [28] for details) *compObjects* execute behaviour on *Nodes*, and communicate through a configuration of other, engineering, objects that form a communications channel. *Channels* are constructed from configurations of *stub*, *binder*, *protocol* and *controller* objects. The channel objects execute on the Nodes linked by the channel; the subset of objects executing on each particular node are know as channel endpoints and are linked by a *communications domain* (network).

4.1 Design of Multi-Party Communication Mechanisms

The design of the communication channels requires us to identify the structure and behaviour of the channel endpoints for each different channel type.

In our example system there are two communication paths: communication between the conference manager and members of the conference (*serverChannel*); and the multicast communication between groups of members participating in a conference (*groupChannel*).

This is illustrated in Figure 6(a) and (b). Figure 6(a) shows an engineering snapshot of three processing nodes. The two MemberNodes have a Member object running on them, whereas the ServerNode has a ConferenceManager object running on it. The dashed lines indicate communication channels between the objects. The groupChannel links all the Member objects and the Conference manager. The serverChannel provides communication between the Member objects and the Conference Manager (i.e. for registration, joining, leaving, etc.). It is obvious that if we were forced to explicitly represent every member of a group in the snapshot, our diagrams would be unusable for large group sizes. Hence, in a similar way to the mechanism provided in Computational Viewpoint diagrams, we provide the alternative notation shown in Figure 6(b); this shows a group of 8 MemberNodes each with a Member object that is connected to the groupChannel. The groupChannel is marked at the Member object end with a '*' to indicate multiple endpoints. Conversly, the serverChannel has no additional marking, and therefore represents 8 separate channels from the ConferenceManager to each Member object.

The serverChanel is a one-to-one operational binding, enabling members to request that they join or leave a particular conference. The design of the engineering channel that supports this binding is not complex. An implementation of the channel could be by using simple Remote Procedure Calls, CORBA method invocation, or a Web Services framework. In each case the channel components are pre designed and can be used 'out of the box'.

More interesting is the design of the groupChannel components, which must address the flowing issues.

1. Each endpoint is both a source and a destination.
2. There are potentially multiple (>2) end points.
3. The channel has some specific computational behaviour of its own.

Each of these issues means that a more complex set of channel components is required. In addition, there is a choice to make; does the targeted communications domain support group communication (network-layer multicast) or do we design the channel components to provide the group support (application-layer multicast)?

We explore a number of different design options, showing the design of the group channel endpoint in each case. The computational aspects of the design are not affected by the choice of option made in the engineering viewpoint. The options, illustrated in Figure (a-d), are as follows:

a. The communications domain supports network-layer multicast.
b. We provide ALM by creating a communication path between each pair of group members.
c. We provide ALM by providing a group server.
d. We provide ALM by providing a hierarchy of group servers.

There are two types of endpoint within the group channel, distinguished by their interface type and functionality: those that connect the member objects to the channel and the one that connects the conference manager to the channel. We call the endpoints connecting members "group channel endpoints"; and the endpoint connecting the conference manager a "group channel controller endpoint".

4.2 Network-layer Option

If our network supports multicast routing, Figure (a), we design the group channel endpoint to make use of it. A client and server stub are required to marshal and un-marshal the sent messages and pass them to and from the Member objects.

A protocol object is used to send and receive the messages to and from a multicast group address. The controller object sends appropriate messages to the network (via the protocol object) to add and remove members from the group. This structure is illustrated in Figure 7.

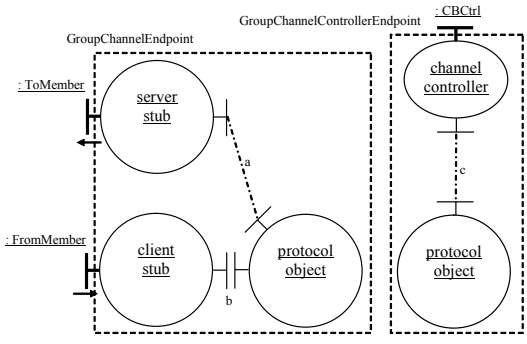

Figure 7 Network-layer Option Endpoints

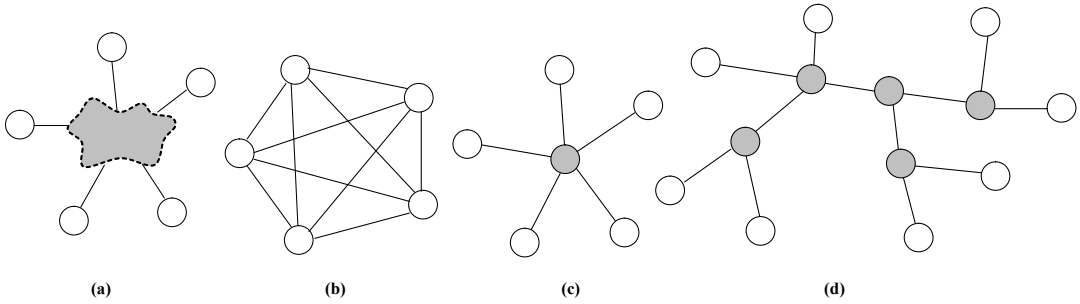

Figure 8 Alternative Group Configurations

We do not include a binder object, as it is not necessary in this case, although one could be added between the protocol and stub objects. The interfaces illustrated with thick lines correspond to the interfaces for the binding object defined in the computational viewpoint design. The interfaces and primitive bindings (labelled a-c), which connect channel components, require further specification; this would be achieved using a Template Diagram, similar to Figure 4.

4.3 Multi-path Option

If network-layer multicast is not available, then we must provide the group support explicitly. A simple (but inefficient) approach would be to provide binder objects within the channel (as shown in Figure). These binder objects would have the responsibility of recording all of the members of the group, copying outgoing messages to each member, and receiving incoming messages from each member.

Multiple protocol objects would be required, each providing the connection to every other member in the group, Figure (b). However, there would only be one channel controller object and its add and remove requests can be propagated around the group like other messages, although they would be filtered out and used by the binder object rather than being forwarded to the stub for processing by the connected member object.

4.4 Group Server Option

A more efficient option for the design of the group channel endpoints is to provide ALM by adding a specific group server at a 'well-known' address in the network, Figure (c); probably provided on the same node as the conference manager (though not required to be).

The protocol object is configured to communicate point-to-point with a group server as the means to provide group based communication. All messages are tagged with the identity of a particular group; it is subsequently the responsibility of the group server to communicate the message to all members of the group. (See Figure 9.)

This mechanism is offered by the JGroups library [17], which we have used to investigate implementation of the various channel endpoints.

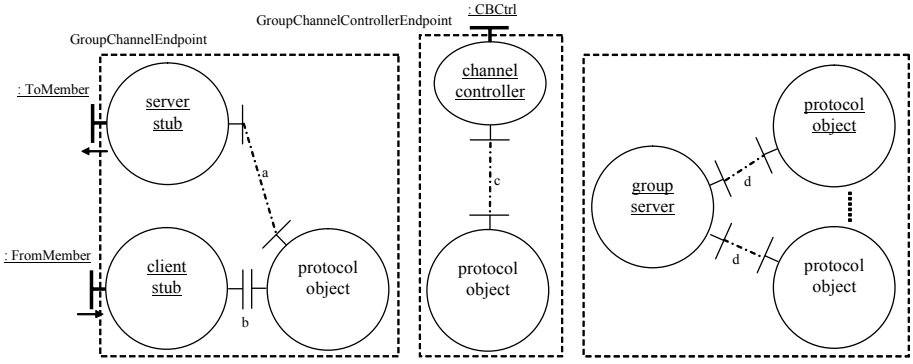

Figure 9 Group Server Option

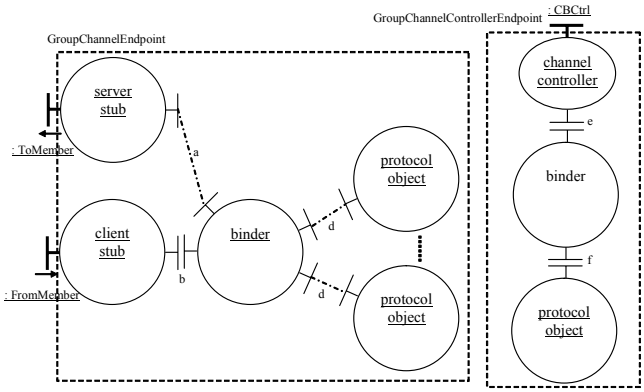

Figure 10 Multi-path Option

4.5 Hierarchy of Group Servers

The group server option does not scale well (see [7]). A more scalable option is to provide a hierarchy of group servers; with each server responsible for a subset of the whole group.

To achieve this, we must allow group servers to be members of groups themselves. When they receive a message that is to be sent to the whole group, they send it to all of the subgroup members serviced by this server, including members of that subgroup that are themselves group servers. The group server hierarchy should be set up to provide the most efficient coverage of the overall group members. Typically group servers would execute on the Node of one of the group members, but they need not necessarily do so.

5 Related Work

CoSMIC [14] is a project which is developing a Model Driven Architecture (MDA) based tool suite for generating, configuring and assembling distributed real-time and embedded applications.

This work focuses on: the MDA aspects of generation from a high level model onto a variety of middleware platforms; and on the aspect-oriented cross-cutting of QoS requirements into the specification and resulting implementation.

The tool and results produced by the project are highly relevant and complementary to our work; addressing similar challenges, but do not focus on groups.

Oldengarm and Haltern [21] propose a multi-viewpoint architecture for defining distributed systems; based on the RM-ODP viewpoints. This work focuses on using the viewpoint framework to visualise what is happening within a program; for purposes such as debugging. They suggest a mapping from CORBA concepts to the concepts of the RM-ODP engineering viewpoint. This mapping is too simplistic for our purposes in the suggestion that an Engineering Channel maps to a CORBA TCP/IP connection.

There is also work within the Object Management Group (OMG) regarding the specification of an Enterprise Distributed Object Computing profile for UML [24]. This work does not sufficiently address issues regarding engineering viewpoint design and, specifically, does not facilitate a mechanism for designing complex bindings such as that required for group based applications.

6 Conclusion and Future Work

This paper concentrates on the multicast and group support mechanism aspects of the DSE4DS project; the project advocates a viewpoint-based approach to distributed system design, built on the ISO standard RM-ODP framework. This paper has illustrated how to make use of group management functions within the computational viewpoint design and proposed a (UML based) notation for designing the engineering viewpoint communication channels. Specifically, this notation facilitates the design of group-based applications, which are not addressed by standard design notations such as the UML.

This approach enables computational aspects of the design to be separated from the design of mechanisms to support the group-based communication. In fact we have illustrated that multiple different designs of the communication mechanism may be provided without affecting the computational design. The key to scalability is in providing notations for specific components and concise representation for multiple instances of these components.

6.1 Future Work

We plan to extend the notation described in this paper to cover designs that include more comprehensive support mechanisms for groups, as discussed in the introduction. Providing a framework for the definition of components and their interactions enables other support mechanisms to be defined, such as a compObject that offers membership authorization or, in the Engineering viewpoint, protocol objects that allow designated receivers to take part in a scalable acknowledgement scheme for reliable multicast transport.

We also intend to model the transformation from designs to Timed Automata (TA) using the techniques

described in [1, 5] and making use of MDA technology and tools, such as the Kent Modelling Framework (KMF) [18], to generate tools that automate the design-to-TA transformation. Additionally, again using an MDA approach, we believe that by drawing on design information from multiple viewpoints it will be possible to generate system implementations from designs such as those presented in this paper. This concept is supported by similar suggestions in the UML profile for Enterprise Distributed Object Computing [24].

We also intend to extend existing QoS verification techniques to verify systems based on information from multiple viewpoints.

Acknowledgements

The DSE4DS project was supported by the UK Engineering and Physical Sciences Research Council (GR/M69500/01).

References

[1] Akehurst D. H., "Model Translation: A UML-based specification technique and active implementation approach," thesis, Department of Computing, University of Kent at Canterbury, Canterbury, 2000

[2] Akehurst D. H., Bordbar B., Derrick J., and Waters A. G., "Design Support for Distributed Systems: DSE4DS," in J. Finney, M. Haahr, and A. Montressor (eds) proceedings 7th Cabernet Radicals Workshop, Bologna, Italy, October 2002.

[3] Akehurst D. H., Derrick J., and Waters A. G., "Addressing Computational Viewpoint Design," in proceedings Enterprise Distributed Object Computing Conference, EDOC 2003, Brisbane, Australia, pp. 147, September 2003.

[4] Akehurst D. H., Derrick J., and Waters A. G., "Design and Verification of Distributed Multi-media Systems," in E. Najm, U. Nestmann, and P. Stevens (eds) proceedings Formal Methods for Open Object-Based Distributed Systems, FMOODS 2003, Paris, pp. 276-292, November 2003.

[5] Akehurst D. H. and Kent S., "A Relational Approach to Defining Transformations in a Metamodel," in J.-M. Jézéquel, H. Hussmann, and S. Cook (eds) proceedings The Unified Modeling Language 5th International Conference, LNCS, Springer, 2460, Dresden, Germany, pp. 305-320, 2002.

[6] Aldawud O., Kande M., Booch G., Harrison B., and Stein D., "Proceedigns of Third International Workshop on Aspect-Oriented Modeling," 2003, http://lglwww.epfl.ch/workshops/aosd2003/papers/Schedule.htm

[7] Banerjee S., Bhattacharjee B., and Kommareddy C., "Scalable Application Layer Multicast," in proceedings ACM Sigcomm, pp. 205-220, 2002.

[8] Cain B., Deering S., Kouvelas I., and Fenner B. T., " Internet Group Management Protocol, Version 3," *IETF, RFC 3376*, 2002.

[9] Constantinides C. A., Bader A., and Elrad T., "An Aspect Oriented Design Framework," *ACM Computing Surveys*, March 2000.

[10] DSE4DS-team, "Design Support for Distributed Systems (DSE4DS) Project Home Page," 2000, http://www.cs.ukc.ac.uk/projects/dse4ds/index.html

[11] Emmerich W., Arlow J., Madec J., and Phoenix M., "Tool Construction for the British Airways SEE with the O2 ODBMS," *Theory and Practice of Object Systems*, vol. 3, pp. 213-231, 1997.

[12] Finkelstein A., Kramer J., Nuseibah B., Finkelstein L., and Goedicke M., "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, pp. 31-58, March 1992.

[13] Glandrup M., Clarke S., Tarr P., and Akkawi F., "Proceedings of Aspect Oriented Design (AOD) 2002 Workshop on Identifying, Separating & Verifying Concerns in the Design," 2002, http://www.iit.edu/~akkawif/workshops/AOSD2002/AOSD1.html

[14] Gokhale A., Natarjan B., Schmidt D. C., Nechypurenko A., Wang N., Gray J., Neema S., Bapty T., and Parsons J., "CoSMIC: An MDA Generative Tool for Distributed Real-time and Embdedded Component Middleware and Applications," in proceedings OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture, Seattle, WA, November 2002.

[15] Harel D., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.

[16] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," Institute of Electrical and Electronics Engineers, Inc., IEEE Std 1471-2000, ISBN 0-7381-2519-9, September 2000.

[17] JGroups-team, "JGroups - A Toolkit for Reliable Multicast Communication," 2002, http://www.javagroups.com/javagroupsnew/docs/index.html

[18] KMF-team, "Kent Modelling Framework (KMF)," 2002, www.cs.kent.ac.uk/projects/kmf

[19] Linington P. F., "RM-ODP: The Architecture," in K. Raymond and E. Armstrong (eds) proceedings Open Distributed Processing: Experience with Distributed Environments, 3rd IFIP TC 6/WG 6.1 International Conference on Open Distributed Processing, Chapman and Hall, February 1995.

[20] Nuseibah B., Kramer J., and Finkelstein A., "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering*, vol. 20, pp. 760-773, October 1994.

[21] Oldengarm P. and van Haltern A., "A Multiview Visualisation Architecture for Open Distributed Systems," in proceedings 22nd International Computer Software & Applications Conference (Compsac'98), Vienna, AUSTRIA.

[22] OMG, "Meta Object Facility (MOF) Specification, Version 1.4," formal/2002-04-03, April 2002.

[23] OMG, "Response to the UML 2.0 OCL Rfp (ad/2000-09-03), Revised Submission, Version 1.6," Object Management Group, ad/2003-01-07, January 2003.

[24] OMG, "UML Profile for Enterprise Disributed Object Computing," Object Management Group, ptc/02-02-05, Febuary 2002.

[25] Putman J. R., *Architecting with RM-ODP*: Prentice Hall, ISBN 0-13-019116-7, 2001.

[26] Whetten B., Vicisano L., Kermode R., Handley M., Floyd S., and Luby M., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer," *IETF, RFC 3048*, 2001.

[27] Willebeek-LeMair M. H. and Shae Z.-Y., "Videoconferencing over packet-based networks," *IEEE Journal on Selected Areas in Communication*, vol. 15, pp. 1101-1114, 1997.

[28] X.901-5, "Information Technology - Open Distributed Processing - Reference Model: All Parts," ITU-T Recommendation, 1996-99.