

# Adaptive Group in Open ORB

**Katia Barbosa Saikoski and Geoff Coulson**

Distributed Multimedia Research Group,  
Department of Computing, Lancaster University,  
Lancaster, LA1 4YR, U.K.

E-mail: {saikoski,geoff}@comp.lancs.ac.uk

## Abstract

*This paper describes the research into the design of a configurable and reconfigurable group service for middleware. The main objective is to allow components of a group service to be changed at run time using reflection and component architecture technology.*

## 1 Introduction

Distributed applications such as video-conferencing, video-on-demand, CSCW, web casting, and distributed simulation are demanding new features from the underlying platforms. One example is the support for groups of users where multiple senders and receivers disseminate or share the same data. Improvements in communication technology have brought an efficient solution for dealing with a set of users as a single entity and using multipoint communication (multicast) to send the same data to each member of the group.

Our concern in this paper is related to the complexity involved in the development of group applications. Although a simple multicast facility can be suitable for some cases, more complex requirements can demand a more comprehensive group facility on top of the multicast mechanism. In addition, as a solution for a distributed environment the group service should be independent of the operating system, programming language or network, that is, it is best placed at the middleware level. Middleware provides transparency from several machine-dependent details.

The big problem in supporting groups is the many different possible options in creating, composing and maintaining a group. In addition, communication features such as reliability, ordering, collation, among others, can have a broad set of options. As many combinations of options are possible and sessions (duration of a connection) can vary from minutes to hours, it is important not only to offer configuration at establishment time, but also to provide some way to change the behaviour of the group during a session (reconfiguration).

This paper discusses the development of a configurable and reconfigurable group facility for middleware using techniques to allow *inspection* and *adaptation*

of the group behaviour. Inspection allows the current configuration of the group to be determined at run time, and adaptation allows the group to be reconfigured. This work is part of a research project developed at Lancaster [2, 3, 6, 9] which main objective is to use *reflection* and *component architecture* to provide an open middleware platform.

## 1.1 Problem Description

The main question to be answered by this work is *how to provide a group communication facility for middleware with capabilities to inspect internal details of the group and adapt the group behaviour according to the environment.*

Standards have been proposed for middleware such as *ISO/ITU-T Reference Model for Open Distributed Processing (RM-ODP)* [15], *OMG Common Object Request Broker Architecture (CORBA)* [12], *Java Remote Method Invocation (Java RMI)* [19] and *Distributed Component Object Model (DCOM)* [5]. Although these standards do not yet address group support, it is likely that they will be extended to do so in the near future. The present work should be highly applicable to such future extension.

The key point in extending the current middleware is to introduce configuration and reconfiguration features in order to support the many different options involved in the group creation and maintenance. For instance, imagine a video conferencing session that allows an indefinite number of participants. A significant increase in the number of members in the conference can lead to a scalability problem that can be solved by changing the reliability protocol or message ordering. As a second example, due to scalability problem, a member of a group could change the way it deals with the arrival of requests or replies. Instead of waiting for the requests (or replies) from all members, it waits for the first message or for the majority of the messages. Likewise these examples, many others can arise in different conditions.

The examples presented above show that specific features of the group needed to be changed at run time without the need to stop the whole group. This can be done with techniques that allows the identification of current status of a system and further modification of it.

In this paper we describe the introduction of reflection in a componetised middleware to support such component changes at run time.

## 2 Group Communication

### 2.1 Communication Mechanism

As mentioned before, distributed applications such as video conferencing, video on-demand, software distribution, web casting, and text boards have to support

communication among multiple users. One special requirement is that the same data has to be sent to all the users in the session. In order to achieve this condition, many point-to-point communications would need to be established. This can be done only if each member knows all the other members in the group. Furthermore, the network would be being used in a very inefficient way.

So, instead of using many point-to-point communications, the idea of multicast (or multipoint communication) arises, that is, a set of users or objects is managed as one single entity and the data is sent to the group. IP Multicast is an example of multicast protocol widely used.

Variations of this simple approach can include different ordering protocols, different solutions for reliability (such as SRM [10]) and many other issues such as state control and feedback suppression. A good review of issues can be found in [27].

## 2.2 Policies and Group Semantic

While group communication at the network level relies on mechanisms to control the delivery of messages to every member of the group, many details about membership and group design should be handled in an upper level. This can include simple operations allowing members to join and leave the group or more complex operations related to membership control, such as the cardinality of the group. Even choices related to the communication itself can be described in the group creation. At this level, the semantic of the group is defined and the group can be configured and reconfigured.

Note that in most current group support system, the semantic of the group is a static association of policies such as creation of the group, collation and message ordering. A combination of policies is established in the beginning of a session (configuration) and it remains until the session finishes. In other words, reconfiguration is not provided.

In our work we separate policies into different groups to have a better view of the system and for each group of policies we identify the different possible policies. The most important set of policies are:

- Group creation — The creation of the group involves the definition of features such as *who can send messages to the group (members vs non members)*, *can the group be split or not*, *cardinality of the group*, *when the group is destroyed* and many other issues.
- Join/leave operation – Users (objects) can join and leave the group based on the group creation policy. Security is an example of a constraint for a join operation.
- Reliability — Delivery of data to a group can be reliable or not. Reliability means that all members of the group receive each message exactly once.

Reliability is associated with mechanisms to control the delivery and reception of messages. For instance, ACK or NACK based reliability protocol.

- Collation — describes how to get a consistent view in a request/reply communication, that is, how to reduce multiples requests or replies in only one request or reply: (interpret all requests or replies, interpret first request or reply or interpret k requests or replies).
- Message ordering — Four basic models can be used to deliver messages among members of a group: arbitrary (no ordering), FIFO (messages sent by the same member are delivered in the same order), total (messages sent by one member are delivered in the same order to all members of the group) and causal (messages respect the notion of dependability (*what happened before*)) ordering.

Many combinations of policies can be created and only some of them have useful meaning in terms of group creation and maintenance. It is part of our work to identify the relationship between the set of policies and policies to provide a basic group model with a reconfigurable approach. Section 3 discusses this issue in more details.

### **3 Proposed Approach – Towards An Adaptive Middleware with Group Support**

This section describes a middleware architecture with group support using reflection and component technology. The basic architecture was proposed in [3] and some implementation details can be found in [6, 9].

#### **3.1 Middleware**

The purpose of middleware is to hide the underlying network and operating systems platforms by providing a set of generic services and interfaces, facilitating the development of distributed applications. But the complexity of applications has grown due to the use of multimedia resources, mobility, group communication and real-time requirements. Therefore, middleware platforms with new features are now required.

One important issue in the design the new middleware is how to control the complexity of these emerging applications such as a networking application running on a mobile computer or a video-conferencing system with a large number of participants. Mobile computers are subject to many different network conditions according to their geographical position. Therefore, the conditions offer to the applications change during the connection and the application should be able to adapt itself to the new environment. In the second example (video-conferencing)

the increase in the number of participants in a conference can influence the behaviour the whole group in terms of performance.

So, it is important to consider that the support platform (middleware) must provide some configurable and reconfigurable features. Configuration is applied to select the options related to the installation of the system. Once the system is running, reconfiguration can take place to change the selection made at installation time.

### **3.2 Component Architecture**

Component architecture [22] is the concept of building an application using a set of components. It is an evolution from the object oriented paradigm where abstraction, inheritance, encapsulation and reusability are key words for software development.

In object oriented languages, most of the program is managed at source code level. This means that any change to one object leads to the compilation/linking process. Using a different approach, component-based development uses pieces of software in binary form that can be plugged into any application.

Component software allows the definition of rules describing how components interact. A *plug-and-play* interoperability is achieved since the system is not aware of internal details of the component such as programming language, implementation details and so on. This means that a component from any developer can be integrated into the system as long as it respects the rules defined.

This technology will be applied in the proposed architecture in order to have a middleware composed of components.

### **3.3 Basic Reflective Architecture - The OpenORB proposal**

Our approach is to introduce *reflection* in a componentised middleware to achieve an open model capable of adapting, inspecting and extending the components of the middleware.

The architecture brings a new perspective for application developers. Instead of using an approach that hides the internal details of a system, an open implementation is provided. In this case, some mechanisms are introduced to provide hooks in the system allowing the inspection of internal behaviour and appropriate adaptation. This is achieved by the use of reflection that provides a way for a system to *do computation about itself* [17]. Reflection provides a way for a system to access, reason about and alter its own interpretation.

The use of reflection introduces a distinction between the aspects related to the system and the aspects related to the representation of the system. These are called the *base* and the *meta* level, respectively.

The OpenORB architecture adopts a component approach, a per component meta-space (as opposite to a global meta-space as is used by some reflective system) and a meta-space with four meta-models. Per component meta-space means that every component in the system can have an individual meta-space (set of 4 meta-models). The four meta-models are *environmental*, *encapsulation*, *composition* and *resource*. The environmental meta-model denotes the execution environment for each interface. It describes functions such as message arrival, marshalling, scheduling among others. The encapsulation meta-model allows the representation of each interface (methods and attributes). For example, the set of available methods can be determined or new methods can be added. The composition meta-model represents components in terms of their internal composition. It describes, using an component graph, the components inside the composite component. Finally, the resource meta-model models resources and tasks per component.

Our approach is to introduce *reflection* in a componetised middleware to achieve an open model capable of adapting, inspecting and extending the components of the middleware.

### 3.4 Groups in Open ORB

The basic mechanism for group support in middleware is a multi-party communication, known as *multi-party binding*<sup>1</sup>. Options related to communication mechanism and group management should be available as configuration items that could be reconfigured during the operation of the group. Component technology brings a new approach to the implementation of a group system that can be described as a set of management and communication components that are plugged together. A basic group service is provided and components are included in this model to embody policies within the group. When a reconfiguration action takes place, components can be replaced by other components with the same interface.

Introducing this concept in the open middleware architecture, the reconfiguration action could be a call to the composition meta-level in order to change the configuration of the components.

### 3.5 Adaptation Effects in the Group Behaviour

The policies chosen at configuration time can be changed, denoting a reconfiguration. Change in some of the policies can have different effects both on the group and on the members of the group.

We can classify these situations as *local* and *global* effects:

---

<sup>1</sup>Based on the RM-ODP Computational Model, a distributed system is composed by objects that have well-defined interfaces and the interaction is done through bindings (communication path).

- *local effect* – a local effect means that one member can require a reconfiguration that is not going to reflect in other members. For example, a collation policy.
- *global effect* – a global effect means that if one member requires a reconfiguration, other members have to perform a related reconfiguration in order to maintain consistency. A distinction can be made between effects that are applied to members already joined to the group (e.g., ordering) and effects applied only to users that want to join the group (e.g., cardinality of the group).

### 3.6 An Example of Configuration and Reconfiguration

So far we have presented some issues related to groups and the need for configuration and reconfiguration. The next step is to describe how configuration and reconfiguration can be applied to a distributed application that uses group. We consider four main actions:

**Set up a group:** In order to create a group, an element in the middleware accesses a template<sup>2</sup> and selects the components that implements the behaviour specified. This implies the creation/loading of group management and group communication components according to the policies specified for the group.

**Group join:** Whenever a user wants to join a group, the following actions take place: (1) the user has to check the possibility to join the group by accessing the management components and check the policies such as security, cardinality and so on (2) a communication path has to be created, that means, it is necessary to check the components being used in the communication by *inspecting* the graph of components and build the user's communication path. In terms of multi-party binding, a join operation is a reconfiguration action because an interface is being added to a binding already established.

**Changes in the management component:** Changes in the management components allow the group to be reconfigured in terms of its internal behaviour, that is, behaviour related to the policies identified as *group management policies*.

**Changes in the communication path:** Changes in the communication path are related to changes in the components that compose the graph. This is done by modifying the composition meta-level, i.e., the graph of components. An example could be the replacement of a reliability component.

---

<sup>2</sup>The template is a specification that describes the behaviour of the group in terms of components.

In order to show how configuration and reconfiguration occur in a real distributed application, consider the CSCW tool developed in the context of the Upper Atmospheric Research Collaboratory Project (UARC) that uses the Corona communication service ([14]). In summary, this system provides a groupware tool with features to disseminate data generated by remote instruments and facilities to scientists to access and manipulate this data.

One important application requirement pointed out by the Corona authors is the *dynamic nature of the application*. For instance, the transmission of images generated by remote instruments does not usually require strong reliability since the user can miss one of the many samples generated. But, if this image is the centre of a discussion, it is important that all the users receive it. Once the discussion has finished, the reliability applied to the image transmission can be changed back from strong to weak. As another example, in Corona users do not have to be aware of other users accessing one specific image unless the image is being updated.

So, consider that at creation time (configuration) there is no need for reliability in order to transfer the images. This is a specific policy chosen when the group is created. When it is necessary to have a reliable image transfer, the component (or components) that implements this policy has to be changed. While the group is relatively small, an ACK based reliability scheme is chosen to favour low latency on retransmissions over usage of bandwidth. Such a scheme would require that the underlying communications mechanism is aware of each group member in order to track who had acknowledged which messages. This implies that some kind of admission control is required.

As the group continues to grow the ACK based reliability scheme may congest the network and endpoints. A further adaptation to a NACK based scheme, trading off latency for scalability, may become necessary. Furthermore, with this new change admission control may no longer be required.

In summary, all the policies chosen at creation time are considered for the configuration of the group and the changes performed during the course of the session are considered reconfiguration.

## 4 First Prototype - GASP

The first prototype is a group facility for middleware. It was developed on top of GOPI (Generic Object Platform Infrastructure) [7]. GOPI is a distributed object-based middleware platform with multimedia support designed in modules that represents independent parts of the system.

The important module for the group facility is the **Comm Module** that provides a stack of transport protocols and application specific protocols (ASPs). ASPs are used to provide new functionalities to the system since they can be included on top of a stack. Each layer has a standard set of functions that acts as entry points and provide fully independent layers (cf. components). Following

this idea, GASP (Group ASP) introduces groups in GOPI as a feature on top of a IP multicast ASP.

In GASP a server represents the group manager and clients represent members of a group. When the server starts, it sets up the group (configure) based on certain parameters specified as quality of service (QoS) entries and waits for members to join the group. When a user (client) wants to join the group it establishes a binding with the server and gets the information related to the group communication (multicast address, port, ...). The group to which the client wants to join and its member name are specified again in the QoS description for that member.

An example of a server (group manager) QoS specification is:

```
gasp_buildqos(&cs, gr_name, gr_member, max_members, gr_openess,  
             gr_life, asp)
```

where, `&cs` is the structure that stores the QoS parameters; `gr_name` is the group name; `gr_member` is the identification of the member in the group; `max_member` is the maximum number of members in the group; `gr_openess` describes if a group is open or closed; `gr_life` describes if a group is destroyed in case of 0 members and, `asp` is the transport layer below GASP.

This prototype is a basic group service to be the starting point for the group service in the open middleware. In this implementation, policies are described in the QoS specification and some more development has to be done in order to provide adaptation. Another prototype will be started to get some benefits from reflective languages. In this case, the development of a prototype has already started in a middleware platform based on Reflective Python [26].

## 5 Related Works

Group communication is not a new field of research. The first approach was to include groups in operating systems to manage groups of processes. So, in 1985, Cheriton introduced the V kernel group communication [4] and in 1993, Amoeba [16] was presented.

Following the evolution, ANSA [20], xAMP [21] and Ensemble [24] (based on Isis [1] and Horus [25]) were important research on group communication. Another important contribution is BAST [11] that describes protocols in terms of reusable objects.

With the emergence of CORBA, groups were introduced in systems such as Electra [18], Object Group Service [8] and Chorus ORB [23]. They describe group as an aggregation of objects addressed as a single interface. Membership control and communication policies are provided in these systems, but they do not provide mechanisms to flexibly configure and reconfigure behaviour that is the subject of this paper.

Another example of groups in CORBA is emerging from the new RFP (Request for Proposal) on Fault Tolerance [13] issued by OMG. Group is used to address a set of replicated objects that are supporting fault tolerance. But in this case, the main point is fault tolerance and it limits the usage of groups to this purpose. Other potential applications of groups, such as data dissemination or interactive communication, are not addressed.

Although these researches propose a solution for groups in CORBA, they do not address the issue of flexible configuration and reconfiguration of groups. In this aspect, the work presented in this paper improves the proposals presented so far.

## **6 Final Considerations and Further Work**

Group communication has become a very complex part of a distributed system due the wide range of different requirements involved. Current standards and research efforts do not view groups as potentially reconfigurable systems. This results in a lack of support for distributed applications such as video-conferencing, collaborative work and video-on-demand that requires groups but cannot rely on a static operation of the group.

In this paper we have discussed issues involved in the design and development of a configurable and reconfigurable group facility for an open middleware. We have seen that many aspects are involved such as the identification of policies for the group, definition of a basic group facility for middleware respecting the many different requirements, mapping of policies into components, and applying techniques to change components at run time.

With the group support offered, it will be possible to design an application considering that during run-time many policies can be replaced. An important issue is that the middleware will act to perform these changes on behalf of users/applications.

Much work has still to be done in order to complete the system but the first steps have already been taken. Currently we are working on the definition of a set of generic components that can be used to support a wide variety of group semantics. The interfaces are under definition and some experiments have already been done on how to connect the components to build a basic group service.

## **Acknowledgments**

Katia Barbosa Saikoski is a PhD student sponsored by CAPES and PUCRS/Brazil. Thanks to Gordon Blair, Fábio Costa, Nikos Parlavantzas, Lee Johnston, Hector Duran Limon and Michael Clark from the Reflection Group at Lancaster.

## References

- [1] Kenneth Birman, André Schiper, and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transaction on Computer Systems*, 9(3):272–314, August 1991.
- [2] Gordon Blair. Notes on Reflective Middleware. Technical report, Computing Department, Lancaster University, Bailrigg, Lancaster, LA1 4YR, UK, 1997.
- [3] Gordon Blair, Geoff Coulson, Philippe Robin, and Michael Papathomas. An Architecture for Next Generation Middleware. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, 1998.
- [4] David Cheriton and Willy Zwaenepoel. Distributed Process Groups in the V Kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.
- [5] Microsoft Corporation. Distributed Component Object Model Protocol - DCOM/1.0. Internet Publication - <http://www.microsoft.com/com/>, January 1998.
- [6] Fábio Costa, Gordon Blair, and Geoff Coulson. Experiments with Reflective Middleware. In *Proceedings of ECOOP'98 Workshop on Reflective Object-Oriented Programming Systems*, Brussels, Belgium, 20, July 1998.
- [7] Geoff Coulson. GOPI: A Distributed Object Platform Infrastructure for Multimedia Applications. Internet Publication: <http://www.comp.lancs.ac.uk/computing/users/geoff/GOPI/>, 1998.
- [8] Pascal Felber, Benoît Garbinato, and Rachid Guerraoui. The Design of a CORBA Group Communication Service. In *15th Symposium on Reliable Distributed Systems*, Niagara-on-the-Lake (Canada), October 1996.
- [9] Tom Fitzpatrick, Gordon Blair, Geoff Coulson, Nigel Davies, and Philippe Robin. Supporting Adaptive Multimedia Applications through Open Binding. In *Proceedings of 4th International Conference on Configurable Distributed Systems (IC-CDS'98)*, Annapolis, Maryland, US, May 1998.
- [10] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Session and Application Level Framing. *IEEE/ACM Transactions on Networking*, December 1997.
- [11] Benoît Garbinato, Pascal Felber, and Rachid Guerraoui. Modeling Protocols as Objects for Structuring Reliable Distributed Systems. In *Communication Networks and Distributed Systems Modelling and Simulation Conference (CNDS'97)*, Phoenix (Arizona), January 1997.
- [12] Object Management Group. CORBAservices : Common Object Services Specification. OMG TC Document 97-12-02, OMG, November 1997.
- [13] Object Management Group. Fault Tolerant CORBA Using Entity Redudancy. <http://www.omg.org>, April 1998. Request for Proposal orbos/98-04-01.

- [14] Robert Hall, Amit Mathur, Farnam Jahanian, Atul Prakash, and Craig Rasmussen. Corona: A Communication Service for Scalable, Reliable Group Collaboration System. In ACM Press, editor, *Conference on Computer Supported Cooperative Work (CSCW'96)*, pages 140–149, Boston, Massachusetts, USA, November 1996.
- [15] ISO/IEC. Open Distributed Processing Reference Model, Part 1: Overview. ITU-T Rec. X.901 — ISO/IEC 10746-1, ISO/IEC, 1995.
- [16] M. Frans Kaashoek, Andrew S. Tanenbaum, and Kees Verstoep. Group Communication in Amoeba and its Applications. *Distributed Systems Engineering Journal*, 1:48–58, July 1993.
- [17] Pattie Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of OOPSLA'87*, volume 22 of *ACM SIGPLAN Notices*, pages 147–155. ACM Press, 1987.
- [18] Silvano Maffei. Adding Group Communication Fault-Tolerance to CORBA. In *Proceedings of USENIX Conference on Object-Oriented Technologies*, Monterey, CA, June 1995.
- [19] SUN Microsystems. Java Remote Method Invocation - Distributed Computing for Java. Internet Publication - <http://www.sun.com>, 1998. White Paper.
- [20] Ed Oskiewicz and Nigel Edwards. A Model for Interface Groups. Technical Report 1002.01, APM Ltd., Poseidon House, Castle Park, Cambridge, CB3 ORD, UK, May 1994.
- [21] Luis Rodrigues and Paulo Veríssimo. xAMP: A Protocol Suite for Group Communication. Technical Report RT/43-92, INESC, 1992.
- [22] Tom Spitzer. Component Architectures. *DBMS On-Line*, 10(10), September 1997. <http://www.dbmsmag.com>.
- [23] Chorus Systems. COOL-ORB Tutorial. Technical Report CS/TR-96-4.1, Chorus Systems, 1996.
- [24] Robbert van Renesse, Ken Birman, Mark Hayden, Alexey Vaysburd, and David Karr. Building Adaptive Systems Using Ensemble. Technical Report TR97-1619, Cornell University, February 1997.
- [25] Robbert van Renesse, Kenneth P. Birman, Brad Blade, Katie Guo, Mark Hayden, Takaro M. Hickey, Dalia Malki, Alex Vaysburd, and Werner Vogels. Horus: A Flexible Group Communications Systems. Technical Report TR95-1500, Cornell University, March 1995.
- [26] Guido van Rossum. Python Tutorial - Release 1.5. Technical report, Corporation for National Research Initiatives CNRI, Preston White Drive, Reston, VA 20191, USA, December 31 1997.
- [27] Zheng Wang, John Crowcroft, Christophe Diot, and Atanu Ghosh. Framework for Reliable Multicast Application Design. Internet Publication: <http://www.cs.ucl.ac.uk/external/zwang/hipparch97.html>, 1997.