Group Communication Support for Dependable Multi-User Object Oriented Environments*

Hugo MIRANDA
Universidade de Lisboa
hmiranda@di.fc.ul.pt

Miguel ANTUNES
INESC
Miguel.Antunes@inesc.pt

Luís RODRIGUES
Universidade de Lisboa
ler@di.fc.ul.pt

António Rito SILVA
INESC
Rito.Silva@inesc.pt

Abstract

Distributed multi-user interactive systems have a rich and complex set of requirements including the need for dependable operation. A promising approach to tackle the complexity of these systems is to rely on configurable architectures that are able to support component re-utilization and composition.

The MOOSCo project, Multi-user Object-Oriented environments with Separation of Concerns, addresses the difficulties in applying a component-based approach in a vertical and integrated manner, from analysis to implementation, to the design of this class of systems. To support communication among distributed entities, the project will use a configurable group communication system called Appia. The paper discusses the role of Appia in the MOOSCo architecture.

1. Introduction

Distributed multi-user interactive systems are an extremely relevant application area. Applications such as virtual environments, distributed simulation, computer supported collaborative work (CSCW), multi-user games or dungeons (MUDs), and multi-user object-oriented environments (MOOs) are becoming increasingly pervasive. These applications pose a very rich and complex set of requirements from the analysis, software engineering and system support point-of-view. A promising approach to tackle the complexity of these systems is to rely on configurable architectures that are able to support component re-utilization and composition.

*This work was partially supported by Praxis/C/EEI/12202/1998, TOPCOM.

The MOOSCo project, Multi-user Object-Oriented environments with Separation of Concerns, addresses the difficulties in applying a component-based approach in a vertical and integrated manner, from analysis to implementation, to the design of this class of systems. The project will define an architecture that will be applied in the MOOs context. MOO environments constitute a challenge for object-oriented distributed systems theory and practice due to its unique requirements for dependability, scalability, adaptability, usability, dynamic changes, non-functional domains to be considered, and efficiency.

The project intends to design and implement an architecture to the support of multi-user object-oriented environments. The architecture is based on component composition and addresses three abstraction layers: user models, middleware abstractions, and infrastructure communication protocols. This last building block will be based on a configurable protocol kernel supporting group communication, called *Appia*. Due to the compositional characteristics of the architecture, including the *Appia* system, it is possible to use middleware abstractions and communication protocols tailored to the specific user models needed in each case.

2 Related work

Compositional approaches are attractive and becoming more and more fashionable. However, it is dangerous to have a naive approach to development with components. There are very hard open problems that current research results have identified but that are far from being solved. The main problems are related to the composition of non-orthogonal components, components which composition semantics is more than the sum of the parts. Existing systems, such as AVIARY[7], MASSIVE [13] and DIVE [7], fail, from a software engineering perspective, to provide a

fully-fledged compositional, reusable and customizable approach to the design and implementation of MOO environments. As result of its monolithic structure these systems are restricted to a single user model, to a restricted set of middleware abstractions.

All the existing systems provide support for information sharing. DIVE [7] and SPLINE [2] use a replicated database approach. All user's and application's interaction is done through the replicated database. Although they offer a clean separation between the application and the replicated database, applications have little control over the replication issues. For instance, dead-reckon algorithms are used to reduce position updates. Nevertheless, none of these systems allow the application developer to specify customized algorithms. Furthermore, from the point of view of communications support existing MOO systems are usually tied to a single quality of service. For instance, NPSNET [11]) only uses unreliable communication while DIVE only use reliable communication. SPLINE support both reliable and unreliable with ordering for messages regarding the same object. However in some situations it could be useful to force message ordering for a particular set of objects. In all systems existing systems there is no support for quality of service adaptation that takes in consideration application specific requirements.

In recent years, there has been a significant progress in the development of group communication infrastructures. The latest systems offer a very impressive range of configuration facilities. For instance, Horus [14] allows communication stacks to be changed in runtime; BAST [8] allows different protocols to be selected to implement the same services under different usage patterns; Coyote [3] allows the same message to be processed by different protocols in parallel. Unfortunately, few of these new advances are currently used in the MOO design, eventually due to a phenomenon of interface mismatch that has not yet been cleanly understood.

The MOOSCo project is trying to build an integrated middleware solution that tackles composition in an integrated manner, from the application to the group communication support.

3 Configurable middleware

In the MOOs context there is not a unique and best solution. Solutions should be contextual. On the other hand, the complete satisfaction of MOO requirements for consistency, adaptability, scalability, and efficiency, is not easy and may result in conflicting and inconsistent solutions. For instance, due to latency, messages might arrive in different orders at different machines. This results in a consistency problem, different users get different views of the environment. A simple solution uses a centralized server to serialize messages. However, there will be a problem scaling to a large number of users because of the burden of centralization. Another solution can be based on infrastructure protocols that provide total ordering and causal ordering for messages in the system, but efficiency may be compromised because of the required number of messages that needs to be exchange. In addition, domain-specific requirements may consider different levels of consistency and even their change at runtime. For instance, in a virtual space, as a user is approaching a group that is holding a conversation his awareness of the conversation increases as he approaches the group. This means that the users perception of the conversation is not consistent with the perception of another user that it is in the group. Moreover, both perceptions become "more" consistent has they get closer.

Due to these requirements MOOs design and development will profit from an approach that allows the customization of contextual solutions by the tuning and composition of predefined reusable components. The MOOSCo project addresses the several concerns involved in the development of MOOs, such as object interaction, awareness management, distributed communication, information sharing and so forth. For a preliminary discussion of how these concerns can be composed see [1]. In the following text, we focus on the issues related to the composition of the distributed communication and replication concerns provides support for replicating MOOs across different processes.

The following elements are defined in this composition:

- **Distributed Consistency Protocol.** It represents a distributed protocol that enforces a particular consistency criteria so that shared state can be maintained consistent among all distributed replicas. Each particular protocol will be implemented using a multicast channel with the proper quality of service in order to support a particular consistency criteria. For instance, strong consistency protocols may use channels with reliable, and total order message transmission. Weak consistency protocols may use reliable or even unreliable channels.

- **Distributed Replica Manager**. This component ensures that every state update and shared behavior events are sent to an object shared state through one or more distributed consistency protocols.

- **Distributed Membership Protocol.** Represents a distributed protocol for managing participants in the application. It uses the notion of **Member**, a concept from the distributed communication concern, to manage space members as they join and leave a shared space. It coordinates with Distributed Replica Managers to transmit the shared state of the replicated objects, to every new member that has joined the shared space according to a particular newcomers policy.

The above definitions of protocols express some natural dependencies: the replica managers rely on membership information to ensure consistency. These dependencies exist also at the underlying group communication level. In models such as virtual synchrony [5], which are fundamental to provide dependable implementations of the distributed consistency protocols, reliability of communication is related to the notion of membership information in the form of *group views*.

More subtle dependencies are derived from the simultaneous and concurrent use of different consistency criteria at the application level. In a naive implementation, each replicated object would use its own consistency criteria, which would be supported in run-time by a independent protocol stack. Unfortunately, such approach would make very hard to implement global consistency constraints across different objects. For instance, it might be interesting to use the same causal communication layer for all stacks such that causal dependencies are preserved across objects. Global reconfiguration procedures, that require all stacks to be in a quiescent state, can also be strongly simplified by having different communication stacks sharing common synchronization layers.

4 *Appia*

Appia [1] is a communication architecture that allows different communication channels, each with its own QoS, to be integrated in a coherent multi-channel protocol stack [12]. Furthermore, the architecture allows the application designer to specify the protocol stack that meets her/his QoS requirements through the composition of micro-protocols. A requirement constraining a single channel is called *intra-QoS* requirement. The integration of different media contents in a single application also imposes *inter-QoS* requirements.

In the previous section, we have already identified some inter-QoS requirements in the MOOs design such as the need for preserving causal order across different stacks or to perform global operations on all stacks. Similar examples have been identified by other research teams. The work of CCTL [15] also uses different communications channels which are managed by a single control channel. The work with Maestro [4] illustrates the difficulties of maintaining consistent failure detection when channels with diverse characteristics are used concurrently. *Appia* addresses these problems by providing a stack composition model that allows to express *inter-QoS* requirements. For instance, one can build a stack where several channels share common *causal* and *failure detection* state.

Stack composition in *Appia* relies on a clear separation between two related concepts: *layers* and *sessions*. We define a *layer* as the implementation of a protocol. All protocols implement the same *event interface*, which defines the types of events each one is able to consume and produce. The format and semantics of these events is irrelevant to our exposition. Typical examples of events are data transmission requests, indication and confirmations. Examples of layers are “datagram transport”, “positive acknowledgment”, “total order”, “checksum”, etc. Good examples of relevant layers and events in the context of fault-tolerant applications can be found in [9]. We define a *session* as an instance of a layer [10]. The session maintains state that is used by the protocol code to process events. A protocol that implements ordering may keep a sequence number or a vector clock as part of the session state. In connection oriented protocols, the session also maintains information about the endpoints of the connection. Note that it is often useful to maintain several active sessions for the same layer even when they share the same endpoints: for instance, one might want to have different FIFO channels for different priority streams.

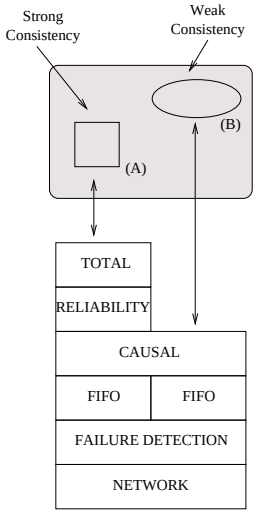

Figure 1. Two objects with different distributed consistency requirements in *Appia*

Figure 1 shows how *Appia* can coordinate different objects with independent consistency requirements. The square object (A) requires totally ordered updates upon every group member. The elliptical object (B) does not need to enforce such a strong property. The figure illustrates two inter-stack requirements: (1) updates on any of the objects should respect causal ordering and (2) both channels share a common failure detector module; this way, inconsistencies motivated by the unreliability of failure detection are avoided. Additionally, the event propagation mechanisms for coordination among layers allows the upper layers to

[1] *Appia* was started in the context of the previous project, TOPCOM.

configure the failure detection module according to application specific requirements as suggested in [6].

Appia handles this sort of requirements in a clean way: property sharing is achieved by allowing protocol instances to be present in the required channels. The figure presents two distinct *Appia* channels, one for object (A) and another for object (B), and behaving as such for the application programmer. Despite the flexibility of the model, developing protocols for *Appia* is not harder than for previous protocol frameworks. Depending on protocol behavior, participation of an instance in several channels can be transparent to the implementation. The causal protocol, for instance, can be coded without concern with the number of channels that may share a single session.

It is our belief that the *Appia* model may simplify the development of complex dependable systems: coordination tasks that are usually implemented by the application programmer, namely those that impose constraints on multiple channels, can now be easily encapsulated in the communication stack, sometimes in a transparent way. Given its rich set of requirements, the MOOSCo project is an excellent target to assess the effectiveness of the approach.

The type of protocols that can be supported by the *Appia* system is not restricted to group communication protocols, even if these were the first to be implemented. Since *Appia* uses an open event mode, letting the user to create the set of events more suitable for a given application class, it is possible to define a broad class of protocols. Furthermore, the *Appia* kernel is independent of the way the events are processed by each layer. The use of *Appia* for other application areas is currently under evaluation, so we expect that the number of available protocols continues to grow.

In what regard to the validation compositions of channels, *Appia* provides a limited support, in the form on checks on the consistency of the set of events that are generated, subscribed, and required by each layer. At this point, it is the responsibility of the application to ensure that the combination of different QoS are consistent.

5. Conclusions

MOOSCo is a recently approved project that will explore the difficulties in applying a component-based approach to Multi-user Object-Oriented environment. To address the dependability requirements of this type of environments, MOOSCo relies on a configurable group communication infrastructure, called *Appia*. This framework allows the design of compositional and customizable infrastructure protocols, offering different qualities of service. *Appia* makes possible to implement, in a clean way, inter-QoS constraints. This is a key mechanism to support the construction of dependable MOO applications that are easily tuned and can evolve.

References

[1] M. Antunes and R. Silva. Using separation and composition of concerns to build multiuser virtual environments. In *Proceedings of the 6th International Workshop on Groupware - CRIWG'2000*, Madeira, Portugal, 2000. IEEE.

[2] J. Barrus, R. Waters, and D. Anderson. Locales: Supporting Large Multiuser Virtual Environments. In *IEEE Computer Graphics and Applications*, pages 16(6):50–100, Nov. 1996.

[3] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, Nov. 1998.

[4] K. Birman, R. Friedman, and M. Hayden. The maestro group manager: A structuring tool for applications with multiple quality of service requirements. Technical report, Cornell University, Ithaca, USA, Feb. 1997.

[5] K. Birman and R. van Renesse, editors. *Reliable Distributed Computing With the ISIS Toolkit*. Number ISBN 0-8186-5342-6. IEEE CS Press, Mar. 1994.

[6] F. Cosquer, L. Rodrigues, and P. Veríssimo. Using tailored failure suspectors to support distributed cooperative applications. In *Proceedings of the 7th IASTED/ISMM International Conference on Parallel and Distributed Computing and Systems*, pages 352–356, Washington (DC), USA, Oct. 1995.

[7] E. Frcon and M. Stenius. Dive: A Scalable Network Architecture for Distributed Virtual Environments. In *Distributed systems Engineering Journal(Special Issue on Distributed Virtual Environments)*, number Vol. 5, No 3, pages 91–100, September 1998.

[8] B. Garbinato and R. Guerraoui. Flexible protocol composition in Bast. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS-18)*, pages 22–29, Amsterdam, The Netherlands, May 1998. IEEE Computer Society Press.

[9] M. Hayden. *The Ensemble System*. PhD thesis, Cornell University, Computer Science Department, 1998.

[10] N. Hutchinson and L. Peterson. Design of the x-Kernel. In *Proceedings of the SIGCOMM'88: Communications Architectures and Protocols*, Stanford, USA, Aug. 1988. ACM.

[11] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, and P. Barham. Exploiting Reality with Multicast Groups. In *IEEE Computer Graphics and Applications*, pages 15(5):38–45, September 1995.

[12] H. Miranda and L. Rodrigues. Flexible communication support for CSCW applications. In *5th Internation Workshop on Groupware - CRIWG'99*, pages 338–342, Cancún, México, Sept. 1999. IEEE.

[13] J. Pubrick and C. Greenhalg. Extending Locales: Awareness Management in MASSIVE-3. In *URL:http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3*, September 1999.

[14] R. V. Renesse, K. P. Birman, B. B. Glade, K. Guo, M. Hayden, T. Hickey, D. Malki, A. Vaysburd, and W. Vogels. Horus: A flexible group communications system. Technical Report TR95-1500, Cornell University, Computer Science Department, Mar. 23, 1995.

[15] I. Rhee, S. Cheung, P. Hutto, and V. Sunderam. Group communication support for distributed collaboration systems. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 43–50, Balitmore, Maryland, USA, May 1997. IEEE.