A Composable Reflective Communication Framework

Sebastian Gutierrez-Nolasco Nalini Venkatasubramanian
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
{seguti,nalini}@ics.uci.edu

Abstract

A communication service is described by an abstract protocol that specifies a set of roles to be played by participants, the requirements on role players and installation information. The (dynamic) installation of a protocol requires no knowledge or modification of the component itself; it is sufficient to encapsulate each component in a layer that implements the role it is to play, affecting only the communication between participants of the particular communication protocol. However, preservation of guaranteed behavior from the communication service may crucially depend on the composition order of the protocols and their possible interactions. Furthermore, Quality of Service(QoS) specifications have been usually incorporated as part of the communication service specification, further complicating the validation of the composition process. This document describes a communication layer within the CompOSE|Q middleware framework that addresses this problem.

Introduction

Recent advances in technology make it feasible to develop large scale concurrent distributed applications such as distributed multimedia that require flexible interaction mechanisms. The communication architecture must be able to represent and express diverse communication models in a transparent way, while ensuring system consistency and correctness. There is interest in developing dynamic and selective composable communication services at run-time to obtain safe flexibility. Safe flexibility is required to protect the system from security threats and failures while ensuring high performance; policies to enforce this are often hard-wired across different parts of the system.

Flexible middleware platforms incorporate the notion of reflection in order to provide the desired level of configurability and openness in a controlled manner [9][10]. Research in distributing computing [1][2][3] [4] and formal methods [5][6][7] have used the concept of reflection to provide modular and adaptable solutions to deal with the problems posed by composition of distributed communication services. However, the goal of achieving truly modular and composable communication services, while ensuring correctness in the compositions of such services is quite subtle. Former approaches to dynamic installation of communication services [2][8] do not impose formal restrictions on the structure and semantics of basic communication primitives. Here, multiple protocols may be applied (installed) to a single component by *stacking* metalevel objects which implement each protocol. An issue of concern is the composition order of these protocols and services; a simplistic implementation can lead to an inconsistent configuration state which may violate the semantics of the basic primitives. Furthermore, many approaches assume that communication is point-to-point. Wide-area applications will benefit heavily from flexible group-communication services (*e.g.* reliable multicast, secure broadcast) that provide customizable communication among members of the group; this implies integration of customized broadcast and multicast protocols among objects in the group. The work described here proposes a reflective communication architecture that integrates with QoS-enabled customizable middleware frameworks such as ComPOSE|Q [11], currently being developed at the University of California, Irvine.

A Two Level Reflective Communication Framework

Meta-level facilities are especially useful in an open distributed system as it allows us to specify, control and reason about the composability of multiple resource management services[6a]. In order to provide flexibility in composition of communication protocols while ensuring correctness of basic middleware services in a meta-level architecture for distributed resource management (*e.g.* garbage collection, remote creation), it is necessary to develop a framework that can distinguish and handle different types of messages (and communication protocols) among groups of objects. Furthermore, multimedia applications require the enforcement of QoS guarantees; therefore, the middleware communication architecture must also integrate QoS parameters, such as timing constraints (*e.g.* end-to-end delay, jitter and synchronization skew) into the resource management and message handling processes. Since the actor model of computation [12] incorporates the notion of encapsulation and interaction only via message passing, it is a natural model to use as a basis for interactive distributed applications[1]. Multimedia applications can be described as a collection of media-actors [13], which interchange information in a precise order with timing constraints.

[1] Actors are concurrent objects that contain state and behavior; buffered asynchronous communication is implemented via mailqueues in each actor.

In order to provide communication services to QoS-based applications in a transparent fashion, the communication framework is designed as follows (see *Figure 1*). There is one communication manager in every node of the distributed system, which implements and controls the correct composition of communication services provided. The reflective architecture consists of base-level actors that implement the application and meta-level actors, which customize the base-level execution. As in [6], each base-level actor has a meta-level actor, called messenger, which serves as the customized and transparent mail queue for its base-level actor. Furthermore, the communication between the actor and its messenger is synchronous; the messenger has four message queues: the **up** and **down** queues are used to communicate with its base-level actor, serving as its send buffer and customized mail queue respectively, while the **in** and **out** queues are used for interaction with the node communication manager, requesting communication services that satisfy QoS constraints. In short, the **up** and **down** queues hold raw messages from and to base-level actors, while the **out** and **in** queues hold messages with the required protocols attached. *i.e.* processed messages.

Efficient Implementation of Flexible Communication

In purely reflective architectures, reasoning about the semantics of correct communication composition may be complicated; moreover, its implementation may be inefficient. In order to maintain accurate semantics and provide an efficient implementation of the architecture, the communication manager consists of a set of communication protocol actors. Each communication protocol actor implements one particular communication service (*e.g.* in-order protocol, fault-tolerant protocol). This scheme allows us to abstract a core set of communication services and share it between the different messengers on a node through meta-level representatives in the communication manager, called pool-actors. The concept of pool-actors is an efficient way to handle the service request of each messenger without having to pay the bottleneck associated with the centralization of the services in the node communication manager. Furthermore, the communication protocol actors have, like the messengers, four queues, but they are used differently: the **up** and **out** queues are used to communicate with the pool-actors, simplifying the synchronization and composition process, while the **in** and **down** queues are used to communicate between the communication protocol actors. This subtle differentiation enforces QoS constraints and encourages separation of concerns in the process of message transmission and reception.

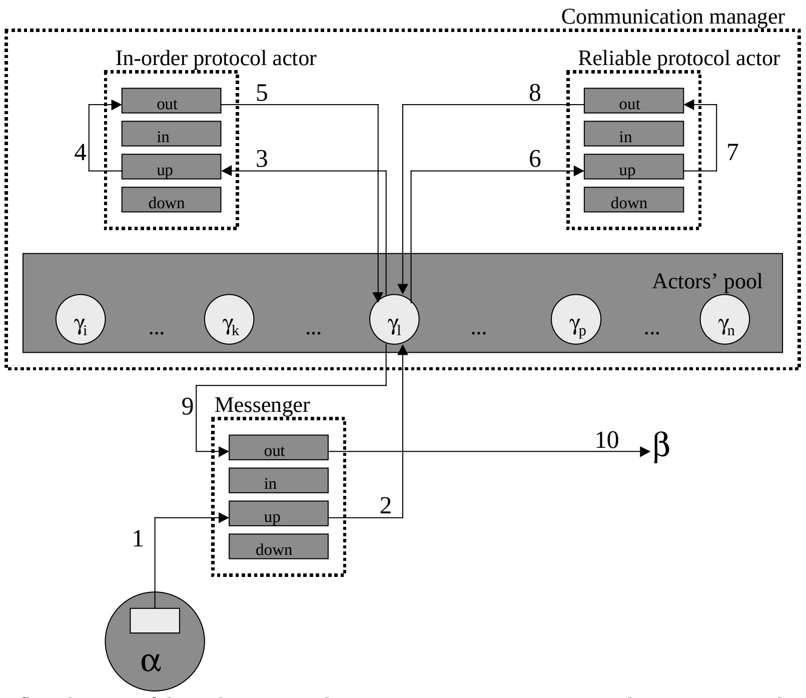

Figure 1. Message flow diagram of the architecture. When actor a sent a message to actor b, a's messenger determine the communication services required to achieve the proper communication and request them to the messenger manager.

Future Work

The designed communication framework provides flexibility in composition of communication services, incorporates end-to-end QoS enforcement as an integral component and provides a basis to reason about the interrelationship between them. The work developed so far is the first step in an effort to provide a cost-effective communication framework capable of addressing adaptive environments with evolving policies and communication services, while ensuring end-to-end QoS. The future work will refine the presented communication architecture and integrate specific protocols (*e.g.* atomic group communication), in order to provide customizable QoS in heterogeneous environments and applications such as mobile multimedia.

References

[1] Mark Astley and Gul Agha, "Customization and Composition of Distributed Objects: Middleware Abstractions for Policy Management", in 6th International Symposium on the foundation of Software Engineering, 1998.

[2] Daniel Sturman, "Modular Specification of Interaction of Interaction Policies in Distributed Computing", PhD thesis, University of Urbana-Champaing, May 1996.

[3] Gregor Kiczales, John Lamping, Anurag Mendhecar, Chris Maeda, Cristina Videira Lopes, Jean-Mar Lointtier and John Irwin, "Aspect Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming, Finland, 1997.

[4] Jeff McAffer, "Meta-level Architecture Support for Distributed Objects", Technical report, Department od Information Science, The University of Tokyo and Object Technology International, 1996.

[5] Rushby J, "Combining System Properties: A Cautionary example and formal examination", Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1995.

[6] Jose Meseguer, Carolyn Talcott and Denker G, "Rewriting Semantics of Meta-Objects and Composable Distributed Services", March 1999.

[7] Lynne Blair, and Gordon Blair, "The Impact of Aspect-Oriented Programming on Formal Methods", Proceedings of the European Conference on Object-Oriented Programing, Aspect-Oriented Programming Workshop, Brussels, 1998

[8] Mark Astley, Daniel Sturman and Gul Agha, "Customizable Middleware for Distributed Software", to apear Communication of the ACM, 2000.

[9] Fabio Costa, Gordon Blair and Geoff Coulson, "Experiments with Reflective Middleware", Internal Report MPG-98-11, Computing Department, Lancaster University, 1998.

[10] Ashish Singhai, Aamod Sane and Roy Campbel, "Reflective ORBs: supporting robust, time-critical distribution", Proceedings of the European Conference on Object-Oriented Programing, Finland, 1997.

[11] Nalini Venkatasubramanian, "ComPOSE|Q - A QoS-enabled Customizable Middleware Framework for Distributed Computing", Proceedings of the IEEE International Conference on Distributed Computing Systems, June 1999.

[12] Gul Agha, "Actors: A model of Concurrent Computation in Distributed Systems", MIT Press, 1986.

[13] Shangping Ren, Nalini Venkatasubramanian and Gul Agha, "Formalizing Multimedia QoS Constraints using Actors", Proceedings of Second International Conference on Formal Methods for Open Object Based Distributed Systems, September 1997.

[14] Brian Nielsen and Gul Agha, "Semantics for an Actor-based Real-time Language", Proceedings of the 4th International Workshop on Parallel and Distributed Real-time Systems, Honolulu, April 1996.

[15] Richard Hayton, "FlexiNet Open ORB Framework", Technical Report 2047.01.00 APM, APM Ltd, UK, 1997.