Object-Oriented Open Implementation of Reliable Communication Protocols

José Orlando Pereira Rui Oliveira

Departamento de Informática
Universidade do Minho, Portugal
{jop,rco}@di.uminho.pt

Abstract

This paper presents a novel architecture for communication protocols that takes advantage of object mobility, allowing applications to specify the behavior of protocols for each individual message. This is achieved by opening the implementation of protocols and letting the application program associate a set of possibly customized meta-objects to every message, separately adapting different aspects of the service provided by the protocol.

Keywords: reliable distributed systems, object-oriented communication protocols, open implementation.

1 Introduction

A simple definition of a distributed system is a collection of independent computers communicating by message passing and working together. As a direct consequence, the most conspicuous characteristic of distributed programs is the code that deals with communication related aspects. Most notably, the recovery from network faults and the detection of remote computer crashes account for most of the added complexity [1].

This complexity can be addressed by developing communication protocols that provide powerful abstractions such as view synchronous process groups and totally ordered multicasts, which the application can use through a simple message passing interface [2, 10].

However, this simplicity reduces the freedom of the application programmer to customize the system for optimum performance, leading to the rejection of reliable communication toolkits in favor of *ad-hoc* solutions that often neglect reliability aspects.

This results in a growing necessity to rethink the loose relationship between application programs and communication protocols and, as a consequence, to improve protocol implementation techniques. The reasons for this necessity comes both from developer demand as well as from a technological push.

The demand for communication protocols that are easily modified to accommodate new situations and are dynamically adaptable to network conditions and application requirements, is present in a series of proposals in the area, such as: active networks [21, 22, 16], very thin protocol layers [23], event-driven micro-protocols [11, 5], protocol specialization by inheritance [7] and protocol composition with the strategy pattern [6, 8].

The technology push comes from the wide-spread availability of networks that are able to transfer objects, as opposed to networks which convey byte strings. The most ubiquitous is the coupling of object streams [19, 20] with sockets and portable code on the Java platform [15].

In this context, this paper presents a novel architecture that takes advantage of object mobility provided by object networks to allow applications to configure the behavior of communication protocols for each individual message. This is achieved by an object-oriented open implementation [14] of protocols, allowing customizable meta-objects that describe each aspect of communication to be associated to each individual message object.

The remainder of this paper is organized as fol-

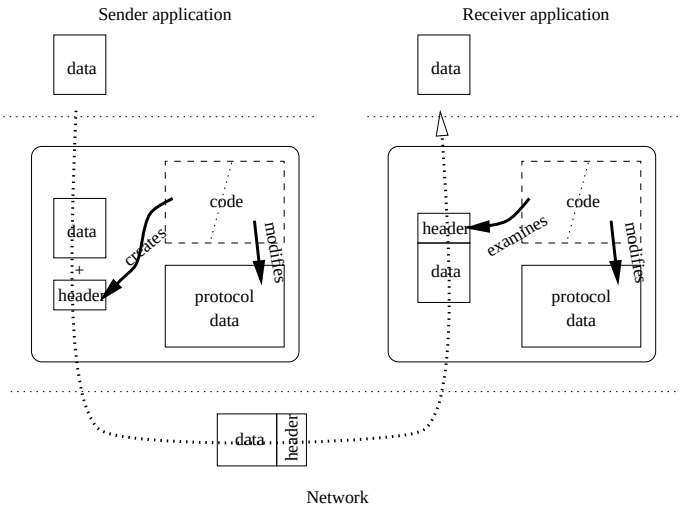

Figure 1: Traditional protocol implementation.

lows: In Section 2 we present a short overview of traditional protocol implementations. Section 3 introduces the general architecture of object-oriented open protocols. Section 4 briefly describes an application of the proposed architecture. Section 5 concludes the paper and speculates about future research directions.

2 Protocol stacks

The most widely used architecture for reliable communication protocols is the *protocol stack*, adopted from point-to-point protocols such as TCP/IP [3], the OSI reference model [17, 4] and the x-Kernel [12].

In these systems, each protocol layer accepts data packets and prefixes them with headers which are used to convey meta-information about the message to the receiver (e.g. sequence numbers for ordered delivery). Upon reception, the header is examined, in order to decide what to do with the packet (e.g. delay or drop it), and discarded when the packet is delivered. Figure 1 depicts such a system, restricted to a single layer for simplicity and clarity of presentation.

The problem with this approach is that there is not a clear separation between the code that deals with message headers and the code that maintains protocol state, which can not be separately customized or replaced. This is the consequence of both entities being hidden inside the same *black box*.

This excessive encapsulation is a source of trouble for application developers that wish to customize protocols and are confined to using them through very simple message passing interfaces.

The same is true for system programmers developing new entities that must deal with the same messages, such as gateways, which is in part solved by strict standardization of message formats. However, this solution is a further limitation to the possibility of configuring and adapting protocols, even if the application programmers interface is enriched.

With complex reliable communication protocols that have not stabilized around a few well defined standards, these limitations are particularly restraining of their wide acceptance and evolution.

3 Object-oriented open protocols

A significant improvement of protocol implementation techniques can be achieved by replacing the data network by an object network and by acknowledging the fact that headers are suitably modeled as meta-objects of messages, as they define the be-

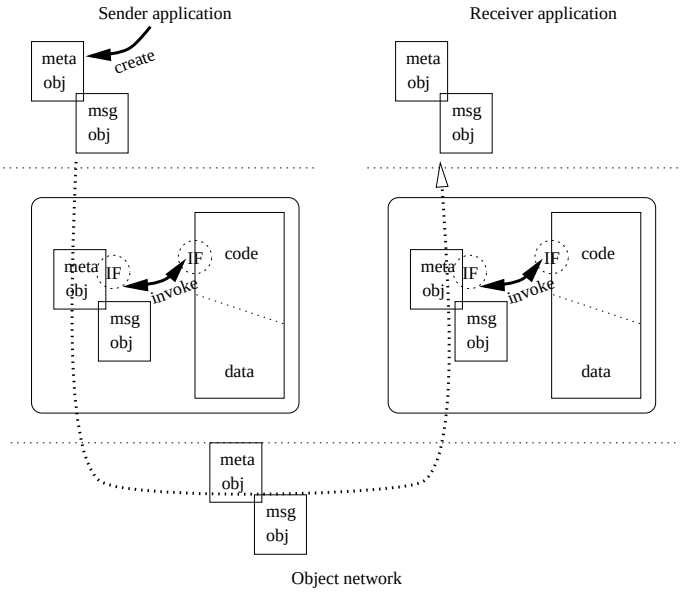

Figure 2: Object-oriented open protocol implementation.

havior of messages when being transfered. Figure 2 presents an overview of this approach.

As a consequence, and in deep contrast with classical implementation of protocols, the code that manages protocol state is clearly separated from code that deals with message meta-information. These two entities, that were previously tightly coupled, become now independent objects related only by a set of abstract interfaces that describe client-server interactions.

This separation eases both development of new protocol layers as well as the configuration of protocols by applications. Different implementations of protocols, as long as they comply with the agreed interfaces, are able to operate on different kinds of meta-information because it is encapsulated behind agreed interfaces.

Configurability is then achieved by opening the implementation of protocols and allowing the application itself to create and associate different meta-objects to different messages, instead of letting the protocol code to always use the same, as happens with traditional protocols.

As a consequence, this proposal is an improvement over other protocol configuration techniques, as it allows the application programmer to specify different qualities of service for different messages within the same message stream, and not only over unrelated streams.

4 Case study: *Groupz*

This architecture has been implemented in *Groupz*, a toolkit of configurable communication protocols built to support reliable process groups over large-scale networks [18]. *Groupz* is written in Java and is structured as four object-oriented open protocol layers on top of the object network provided by the Java platform. The layers correspond to four separate concerns found in distributed programming:

Dependable delivery: Buffer and retransmit messages until a safe reception condition is met, to cope with faults.

Order: Delay messages until an arbitrary precondition on previous deliveries is met, possibly implying agreement.

Membership: Maintain an agreed view of the current set of correct processes.

Hierarchy: Support large hierarchical networks, by transforming messages when gateways are traversed.

Taking dependable delivery as an example, the information collected by the corresponding the protocol layer in *Groupz* is an approximate history of message receptions by foreign hosts. This information is gathered from messages which can supply it in a variety of ways, in contrast to the fixed sequence numbers or version vectors in traditional protocols. It is then used to evaluate conditions for message buffering and retransmission that are associated to messages.

By programming the delivery of reception information and the conditions that act upon it, it is possible to implement a wide range of services for each individual message. For instance, a message can discard all its predecessors, as in stubborn channels [9], or only some of its predecessors, being the equivalent of multiple stubborn channels that have the advantage of being able to merge and split. The same techniques can be exploited at the ordering layer to relax and combine traditional ordering classes. As a consequence, it is possible to configure protocols that take advantage of redundancy and commutativity to relax reliability and order constraints.

Another interesting example is the group membership change message under a virtually synchronous environment. This event often means that some messages are discarded from retransmission buffers, even if not received by the members that are leaving the group. To accomplish this, traditional group protocols usually have specific control operations. In *Groupz* this is not necessary, as the group membership change message itself, acts as an universal acknowledge from failed processes, discarding messages that are no longer needed. It is also possible to consider changing the policy for triggering view changes without stopping the system, as it is defined by the view change message itself that can be customized by the application.

5 Conclusions

A major goal of this work is to build protocols that are configurable on a per-message basis, which is considered to be necessary to adapt complex reliable communication toolkits to real life problems. The proposed solution achieves it in two steps: first, by the use object-orientation techniques to design and implement protocols, clearly identifying the entities involved, and then, by opening these implementations to application programmers.

Some proposals in active networks also allow this degree of configuration, but do not make a clear separation between the protocol as the mechanism and the message as the policy. As a consequence, by shifting most of the code to the message, active networking protocols are subject to many of the same limitations of traditional protocols. For instance, by having complete knowledge about what actions to perform at each concrete node hard-coded in the message, the introduction of new and unforeseen types of nodes is restricted. It also is harder to separately develop and reuse different aspects of a protocol.

Other architectures that allow configuration of protocols, rely on the composition of protocol building blocks which after configured act mostly as black boxes, making them useful only when configuring unrelated communication sessions with different qualities of service for independent streams of messages.

One final conclusion is that open protocol layers in *Groupz* separately address different *emerging entities* in distributed programs and as such are good candidates for corresponding aspects to be composed [13]. In this context, libraries of message meta-objects can be seen as aspect libraries, allowing the developer to separately program different aspects of the same problem, providing valuable information and experience for aspect languages to be researched.

References

[1] K. Birman and B. Glade. Consistent failure reporting in reliable communication systems. Technical report, Cornell University, Dept. of Computer Science, May 1993. TR 93-1349.

[2] Kenneth Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[3] David Clark. The design philosophy of the DARPA internet protocols. In *ACM Sigcomm Symposium*, 1988.

[4] J. Day and H. Zimmermann. The OSI reference model. *Proc. of the IEEE*, 1983.

[5] Henrique Fonseca. Ambientes de suporte para modularização, concretização e execução de protocolos de comunicação. Master's thesis, Universidade Técnica de Lisboa, Instituto Superior Técnico, 1994. (In Portuguese).

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] B. Garbinato, P. Felber, and R. Guerraoui. Protocol classes for designing reliable distributed environments. In *Proceedings of ECOOP'96*, July 1996.

[8] B. Garbinato, P. Fleber, and R. Guerraoui. Using the Strategy pattern to compose reliable distributed protocols. In *Proceedings of the 3rd Conference on the Patterns Languages of Programs (PLoP'96)*, September 1996.

[9] R. Guerraoui, R. Oliveira, and A. Schiper. Stubborn communication channels. Technical report, LSE, EPF Lausanne, December 1996.

[10] Vassos Hadzilacos and Sam Toueg. Fault-Tolerant Broadcasts and Related Problems. In Sape Mullender, editor, *Distributed Systems*, chapter 5. Addison Wesley, second edition, 1993.

[11] Matti Hiltunen. *Configurable Fault-Tolerant Distributed Services*. PhD thesis, Department of Computer Science, The University of Arizona, Tucson, Arizona 85721, July 1996.

[12] N. Hutchinson and L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *ECOOP'97 Object Oriented Programming, Proceedings*, number 1241 in Lecture Notes in Computer Science, page 220. Springer-Verlag, 1997.

[14] G. Kiczales and Xerox Parc. Beyond the black box: Open implementation. *IEEE Software*, 13(1):8–11, January 1996.

[15] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. The Java Series. Addison-Wesley, 1996.

[16] G. Di Marzo, M. Muhugusa, C. Tschudin, and J. Harms. The messenger paradigm and its implications on distributed systems. In *Proceedings of ICC'95 Workshop on Intelligent Computer Communication*, 1995.

[17] NN, ISO, TC97, SC16, and ANSI. Data processing - open system interconnection basic reference model. *Computer networks and: ACM CCR 11, April 1981*, 5:81–118, 1981.

[18] J. Pereira and R. Oliveira. On stacks and russian dolls: Mobile objects in configurable communication protocols. In *Proceedings of the 3rd. ECOOP Workshop on Mobile Object Systems*, Jÿvaskÿla, 1997.

[19] R. Riggs, J. Waldo, A. Wollrath, and K. Bharath. Pickling state in the Java system. *Usenix Computing Systems*, 9(4):291–312, Fall 1996.

[20] Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043. *Java Object Serialization Specification*, December 1996. 1.2.

[21] D. Tannenhouse, J. Smith, W Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications*, 35(1):80.

[22] D. Tannenhouse and D. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26(2), April 1996.

[23] R. van Renesse, K. Birman, 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, March 23 1995.