

Implementing Integrated Fine-Grain Customizable QoS using Cactus

Matti A. Hiltunen, Richard D. Schlichting, and Gary Wong
Department of Computer Science, The University of Arizona
email: hiltunen/rick/gary@cs.arizona.edu

1. Introduction

The concept of quality of service (QoS) has traditionally focused on performance-centric metrics such as throughput, delay, jitter, and loss rate (e.g., [2, 8]) primarily in the field of data communication. Recently, however, the use of the concept has extended to all types of services in computer systems, and the scope of the QoS concept has expanded to include other attributes of service quality such as fault-tolerance, availability, reliability, timeliness, consistency, accuracy, and security. Such an integrated QoS concept provides users the ability to specify exactly what aspects of the service quality are important, and—in case of trade-offs between different QoS attributes—what balance between the different attributes should be chosen. For example, strengthening communication security through cryptographic methods typically reduces throughput and increases latency. Providing users the power to specify the desired service requires a methodology that allows the software providing the service to be customized so that it satisfies these requirements. The goal of our work is to provide a framework that allows such fine-grain customization through configuration.

Different protocols and algorithms have been proposed to provide various QoS guarantees, and many systems even allow some degree of customization. For example, the degree of process replication can often be changed to adjust availability and reliability, while the size of read and write quorums can be modified to adjust read and write availability for replicated data. However, our goal is to provide fine-grained customization of each attribute, and in particular, to allow customization between attributes in tradeoff situations.

2. The Cactus approach

The Cactus approach to constructing highly-customizable middleware services is based on implementing QoS guarantees and other service properties as separate modules that interact using an event-driven paradigm as illustrated in Figure 1. The basic building block of this model is a *micro-protocol*, a software module

that implements a well-defined property of the desired service. A micro-protocol is, in turn, structured as a collection of *event handlers*, which are procedure-like segments of code *bound* to be executed when a specified *event* occurs. Events are used to signify state changes of interest, e.g., “message arrival from the network”. When such an event occurs, all event handlers bound for that event are executed. Events can be *raised* by micro-protocols or be raised implicitly by the runtime system. Execution of handlers is atomic with respect to concurrency, i.e., each handler is executed to completion without interruption. The binding of handlers to events can be changed at runtime.

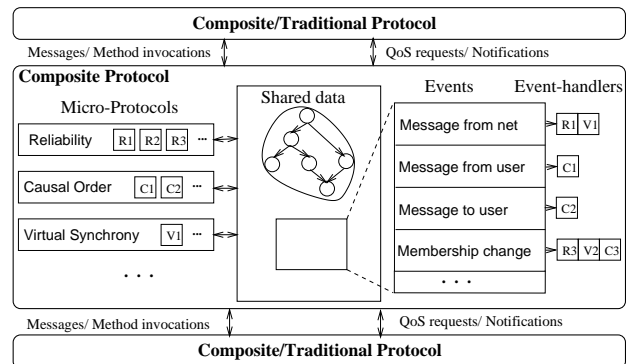


Figure 1: Two-level composition model

Event handler binding and invocation are implemented by a standard runtime system or *framework* that is linked with the micro-protocols to form a *composite protocol*. The framework also supports shared data (e.g., messages) that can be accessed by the micro-protocols configured into the framework. Once created, a composite protocol can be composed with other protocols in a traditional hierarchical manner using a system like the *x-kernel* [7] to construct a middleware service.

A number of prototypes of the Cactus model have been implemented, and these prototypes have been used to design and implement a variety of highly-configurable middleware services. The services range from various communication protocols with fault-tolerance, real-time, and se-

curity properties, to distributed file service and distributed shared memory supporting multiple consistency properties. The current Cactus prototypes range from Cactus++, a C++ version on Linux and Solaris, and Cactus/J, a Java 1.2 version, to a real-time version based on the MK 7.3 operating system and CORDS communication subsystem from Open Group/RI [6].

3. Implementing QoS attributes

The event-driven approach used in the Cactus project was originally developed for the construction of configurable fault-tolerant group communication protocols, including atomic multicast, group RPC, and group membership [5, 1]. In these services, the main emphasis is on reliability, availability, consistency, and performance properties. The approach was designed to allow a natural way to implement many well-known algorithms for the particular properties as separate modules. For example, reliability properties have been implemented to address both communication failures (through retransmissions) and site failures (through support for replication). Consistency has been addressed in the form of various message ordering properties for communication, and membership services that make it possible to maintain a chosen form of consistency for replicated objects or processes. Techniques for ensuring other consistency properties, such as failure and concurrency atomicity, can be implemented in the same way. Finally, performance issues have been addressed by allowing the services to be configured to provide only the required properties, without unnecessary overhead. We also provide alternate implementations of micro-protocols, each of which is optimized for various system conditions.

Different security properties for data communication security can also be implemented in this framework. For example, it is easy to implement different cryptographic methods for confidentiality, authenticity, and integrity as micro-protocols [4]. It would also be easy to implement many other security services, such as access control or key distribution, as configurable composite protocols. Note, however, that the Cactus approach does not provide the means for implementing protection properties such as memory protection. Such properties must be provided by the underlying operating system.

Real-time properties differ from the above in the sense that such guarantees cannot be added to an existing implementation by providing additional micro-protocols. Rather, these properties must be implemented by controlling system execution through admission control, resource allocation, and scheduling mechanisms. Thus, the real-time version of Cactus has a separate admission control module that translates the real-time service requirements and characteristics into thread priorities on the MK operating system [6]. The

configurability aspect provides some additional challenges, since each service configuration may have its own execution time characteristics and may potentially send different number and type of messages.

The micro-protocol approach provides unparalleled facilities for fine-grain customization of QoS attributes. It is easy to implement micro-protocols that realize different tradeoffs in different situations. For example, when a system cannot ensure both timeliness and consistency for a message, we can implement one micro-protocol that sacrifices timeliness and another that sacrifices ordering. Another example is a group RPC service that allows configuration time choice of the failure model to be tolerated, and thus allows customization of the tradeoff between reliability, performance, and efficiency [3].

Acknowledgements

This work has been supported in part by the Office of Naval Research under grant N00014-96-0207, the Defense Advanced Research Projects Agency under grant N66001-97-C-8518, and the National Science Foundation under grant CDA-9500991.

References

- [1] N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Trans. on Computer Systems*, 16(4):321–366, Nov 1998.
- [2] A. Campbell, G. Coulson, and D. Hutchison. A quality of service architecture. *ACM SIGCOMM Computer Comm. Review*, 24(2):6–27, Apr 1994.
- [3] M. Hiltunen, V. Immanuel, and R. Schlichting. Supporting customized failure models for distributed software. TR 99-04, Dept. of Computer Science, Univ. of Arizona, Feb 1999.
- [4] M. Hiltunen, S. Jaiprakash, and R. Schlichting. Exploiting fine-grain configurability for secure communication. TR 99-08, Dept. of Computer Science, Univ. of Arizona, Apr 1999.
- [5] M. Hiltunen and R. Schlichting. An approach to constructing modular fault-tolerant protocols. In *Proc. of the 12th Symposium on Reliable Distributed Systems*, pp. 105–114, Oct 1993.
- [6] M. Hiltunen, R. Schlichting, X. Han, M. Cardozo, and R. Das. Real-time dependable channels: Customizing QoS attributes for distributed systems. To appear in *IEEE Trans. on Parallel and Distributed Systems*, Jun 1999.
- [7] N. Hutchinson and L. Peterson. The *x*-kernel: An architecture for implementing network protocols. *IEEE Trans. on Software Engineering*, 17(1):64–76, Jan 1991.
- [8] K. Nahrstedt and J. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, 1995.