Available online at www.sciencedirect.com

SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 88–101

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

A middleware architecture to facilitate distributed programming DAROC: Data-Activated Replicated Object Communications

Brian M. Stack, Gene Hsiao, Stephen F. Jenks*

University of California, 444E Engineering Tower, Irvine, CA 92697-2625, USA

Received 6 December 2003; accepted 6 December 2003
Available online 25 December 2004

Abstract

Programming distributed computer systems is difficult because of complexities in addressing remote entities, message handling, and program coupling. As systems grow, scalability becomes critical, as bottlenecks can serialize portions of the system. When these distributed system aspects are exposed to programmers, code size and complexity grow, as does the fragility of the system. This paper describes a distributed software architecture and middleware implementation that combines object-based blackboard-style communications with data-driven and periodic application scheduling to greatly simplify distributed programming while achieving scalable performance. Data-Activated Replication Object Communications (DAROC) allows programmers to treat shared objects as local variables while providing implicit communications.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Middleware; Distributed system; Data-activated; Blackboard

1. DAROC at a glance

The Data-Activated Replication Object Communications (DAROC) approach empowers programmers to be able to write components of distributed systems without being distributed systems experts, while leaving the configuration of mission-critical systems to architects educated in design complexities and pitfalls of such systems. When writing an application in DAROC, programmers are able to simply access the objects they want without having to be concerned with the objects' locations. Programmers have access to both the data members and the member functions associated with the object. For example, the following simple code fragments show how to access a member value in a temperature object in a simulation:

Data member usage:
$\$temperature \rightarrow current\_temp;$
Member function usage:
$\$temperature \rightarrow convert\_to\_celsius(\,);$

* Corresponding author. Tel.: +1 949 824 9072;
fax: +1 949 824 3203.
E-mail addresses: stackbr@hotmail.com (B.M. Stack), gene@ucsd.edu (G. Hsiao), sjenks@uci.edu (S.F. Jenks).

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2003.12.025

Logic usage:
if(\$*temperature* $\rightarrow$ *currrent_temp* > *100*)
  \$*temperature* $\rightarrow$ *current_temp* = *100*.

The '\$' symbol is used by the DAROC parser to wrap objects in DAROC code. Code written to use DAROC is normal C++ code that accesses local variables and distributed objects in similar ways. The addition of the '\$' notation tells the DAROC toolkit to add code to manage the distributed nature of the object without letting that distributed nature affect the programmer. The DAROC middleware handles distribution of objects and activation of subscribed applications without any explicit communications or scheduling operations in the application code.

2. Motivation

Distributed software is normally inherently more complex than normal, sequential, monolithic software. The complexity arises from local issues, like buffer and concurrency management, to network-centric problems of naming, addressing, and message passing. Middleware solutions, like CORBA and others, have been developed to abstract or hide some of the communication details, but the distributed nature of the system is often still exposed and some complexity and overhead are generally associated with such approaches. Developers must consider aspects such as communications and scheduling in addition to the functional algorithm of the code they are writing. For example, if a middleware solution provides a remote procedure call (RPC) mechanism, the developer must choose between blocking and non-blocking calls, and if non-blocking, worry about buffer usage and synchronization upon completion. If, instead, the communication approach is message based, the programmer must compose the message, send it, receive the reply, and decode the reply message. All of this work is in addition to the required implementation of the function the program or module is intended to perform.

The concepts behind DAROC arose from the goal of eliminating the burden of communications and scheduling from the application programmer while maintaining performance and scalability of the system. By giving programmers the view of communicating through shared objects, very natural interactions between programs are facilitated. With the introduction of the single-writer rule and activation based on data changes from data-flow and data-driven systems, distributed systems issues such as race conditions and explicit scheduling are eliminated. While the single-writer rule for data objects (DO) may be seen by some as a significant constraint on expressiveness, it eliminates behavior that limits scalability or would cause significant confusion or overhead to manage.

Because the application programmers are simply reading and writing data objects, their programs are logically decoupled from one another through their interfaces. If timing constraints require tight temporal coupling between two programs, the system engineer can allocate both to the same node, where their shared objects are implemented efficiently through shared memory. If they can execute on separate nodes, the DAROC runtime system handles object replication transparently over the network. Neither case requires any modification to the application program; thus, the functional behavior of the programs in the system remains the same.

DAROC borrows the central concept from the blackboard model, specifically that programs communicate through shared data (objects, in this case) on a logically shared medium. DAROC differs from conventional blackboards because of the addition of data-driven activation (or periodic activation, if appropriate to the application). This eliminates any notion of and need for a centralized controller and greatly enhances the scalability of the system. By pushing scheduling decisions out to the local processors and as close to the distributed programs as feasible, there is no central bottleneck. If programs are activated by changes in data, they will activate on appropriate state changes. If they are periodic, they see the current system state on the blackboard when they run. This combination supports the composition of significant distributed systems.

These specific goals help DAROC reach the overarching goals below:

- Reduce software development costs, error-proneness, and inflexibility associated with distributed systems applications.
- Reduce code complexity so that the "average" programmer is able to build distributed systems.
- Provide a tool to better prepare students for the ever-growing need in the job market for experienced distributed systems programmers.