RSVC: A Reliable Distributed Control Software Development Toolkit

Jie Chen
Control Software Group
Thomas Jefferson National Accelerator Facility
12000 Jefferson Avenue
Newport News, VA 23606

Weizhen Mao
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187-8795

Abstract

Modern large accelerator control systems consist of communication software and distributed services that must be reliable, efficient and flexible. These requirements motivate the use of object-oriented technology to develop a toolkit for building reliable and fault-tolerant distributed software. This paper describes a C++ toolkit (RSVC) that reduces effort of development of reliable distributed control software.

Keywords: Distributed Systems, Fault-tolerance, C++, Object-orientation.

1 Introduction

The control system at the Thomas Jefferson National Accelerator Facility (Jefferson Lab) is a distributed system using client/server architecture, name based I/O and TCP/IP network protocol [1]. It consists of networked workstations running different flavors of Unix and m68k based single computers that are connected to various types of hardware control modules, running VxWorks^{tm} real-time kernel. Applications of the control systems must be highly available and scalable to meet their reliability and performance demands. These requirements motivate the development of a flexible toolkit for building reliable and fault-tolerant distributed software.

Developing distributed software is difficult since it requires detailed knowledge of many concepts such as (1) network addressing and remote service identification and (2) creation, synchronization, and concurrency mechanism of processes and threads. In addition, applications are often required to handle termination detection [2], partial failures [3] and automatic client reconnection management.

This paper illustrates a flexible toolkit (RSVC) partially based on Isis [4] model that supports reliable process-oriented distributed system. Moreover, this toolkit supports applications requiring loosely coupled processes that communicate through asynchronous messaging and provides very simple program interface to minimize code changes to existing software.

2 Design and Implementation

2.1 Design and Analysis

Reliability is an important quality in mission-critical distributed applications such as control systems. In many distributed systems, even small amounts of unscheduled downtime can be unacceptable. We define a distributed system as *reliable* if its behavior is predictable despite of partial failure, asynchrony, and run-time reconfiguration of the system. In comparison to stand-alone systems, a distributed system has much more difficulties to achieve high reliability. For instance, connection management from clients to servers are inherent problem in distributed systems. Another inherent problem is that developers must address complex execution states of concurrent programs. Distributed systems consist of processes that run simultaneously on multiple hosts running different operating systems and therefore prone to communication error, message dropping, node failure, and deadlocks. However, distributed systems can make applications more robust by providing redundant services on multiple nodes. Hence a non-robust communication software will lead to fragile distributed systems that require constant supervision and prolonged downtime. In contrast, a sophisticated communication software can produce reliable, fault tolerant, and scalable distributed systems.

2.1.1 Isis Approach

Isis [4] was originally developed by Ken Birman using Virtual Synchrony [3] model which is a distributed execution model. At the core of the Isis model are a *failure suspector service* and a *group abstraction*. The failure suspector service detects faulty objects and guarantees that non-faulty objects have a consistent view of which objects are believed faulty. The failure suspector relies on timeout and underlying network protocol to detect suspicious objects. It can also set up *monitors* acting as notification services for faulty or recovering processes. The Isis also uses *group abstraction* mechanism to achieve fault tolerance by backing one another up within a single group.

Currently the Isis approach has led to various toolkits such as Isis, Horus, Transis [5] all of which are all very well suited for a distributed systems that can be engineered or reengineered to take advantage of technology those toolkits provide. However, reengineering existing systems or forcing every applications using these toolkit involves substantial code changes and presents step learning curve to application programmers. The RSVC toolkit presented here uses object-oriented methodology and C++ to minimize learning curve and reduce code impact of reengineering legacy systems.

2.1.2 RSVC Approach

The high level control system at Jefferson Lab is based on CDEV [6] system which views each process or application as a named entity (device) responding to a set of messages. All I/O requests in the system are in the form of messages to devices. A message to a device can be handled either synchronously or asynchronously and is actually carried out by a request object (*cdevRequestObject*) that is created and registered. A device will dispatch messages to a request object which in turn sends out to its service using service specific protocol. To allow the message based interface to work with different services, a new data type *cdevData* is designed to hold different data types with different tags. In addition, a particular service/server interface is totally separated from applications to hide the representation and internal structure of all I/O requests to/from a service. Applications using CDEV communicate with packages or services through a service layer which contains a C++ class derived from *cdevService* and a class derived from *cdevRequestObject* which handles all I/O requests. Figure 1 illustrates how CDEV system and applications work together as a system.

In order to make control applications more reliable, less manual supervision and have capability of automatic reconnection to servers recovering from node crash or hardware failure, the RSVC toolkit has been developed. This toolkit contains a set of servers similar to *failure suspector* of Isis toolkit and a set of Application Program Interfaces (APIs) using message based interface. The RSVC servers can be identified by name and respond to a set of messages. Using message based interface definitely minimizes the code change impact to existing software, simplifies APIs and enables easy integration of RSVC into the CDEV system by providing a simple CDEV service layer.

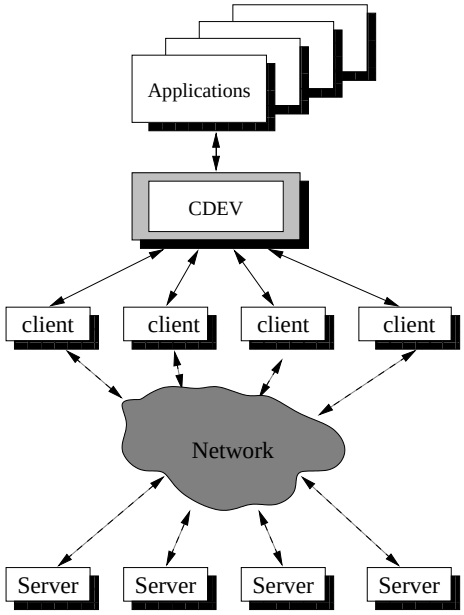

Figure 1: Architectural overview of CDEV. Clients may communicate with servers using different protocols.

A RSVC server actually consists of two parts: one is the naming service and the other is life cycle service. The naming service holds all crucial information about processes that wish to be managed. The life cycle service offers notification mechanism for any configuration change events such as process termination, partial failure and node crash. Figure 2 presents the architectural overview of RSVC integrated with CDEV system.

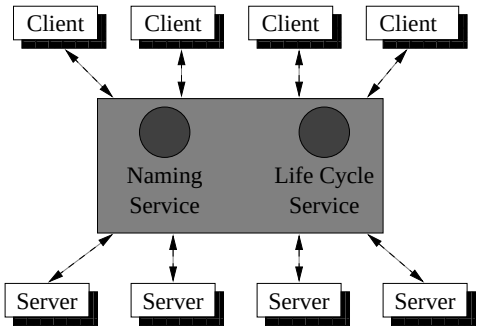

Figure 2: Architectural overview of RSVC.

2.2 Implementation

The RSVC toolkit consists of one or more network servers and one set of APIs for communication software. A distributed service like RSVC is often significantly more difficult to design, implement, and debug. To handle the requirements of distributed applications, developers must address many topic (such as connec-

tion management and partial failure). Object-oriented design and implementation techniques provide various methods and principles such as system toolkits, development framework and usage of design patterns to simplify the development of distributed applications.

2.2.1 Reliable RSVC Server

In order to provide services to other communication software, the RSVC servers must be reliable, responsive and easy to maintain. To meet these requirements, a RSVC server is constructed as a concurrent, on-demand invocation, message based, I/O non-blocking and configurable network daemon. Several well established Object-oriented design patterns such as reactor[7] are also deployed in the design and implementation phase. In addition, primary-backup model [8] is adopted to enhance fault tolerance. At any given time, there could be two servers running on different nodes holding same information. One server is primary and the other is secondary. In case of crash failure of the primary server, the secondary server will serve as the primary server and a new secondary server will be spawned on a different node.

2.2.2 Network Protocol and Naming Service

The type of data used to send/store information about processes that are managed by RSVC is crucial to the flexibility and performance of the RSVC toolkit. Usually an application needs certain information such as host, port, and protocol to locate processes stored in the naming service by names. However, different distributed systems may need different information stored and exchanged from applications to the naming service. Using a C or C++ data structure with several predefined fields prevent systems from adding new fields which may be important. In order to let applications and systems store and exchange information of any type without sacrificing the performance, a new type of data *rsvcData* is implemented in C++ to serve as a dynamic structure that contains multiple data fields, accessed by tagged values, of different types and sizes. Currently *rsvcData* can hold any data of type integer, pointer to character string, char, short, ushort, uint, long, ulong, float, double, time_val structure or arrays of these.

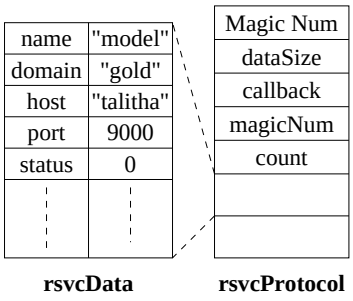

Figure 3: Illustration of *rsvcData* and *rsvcProtocol*.

A network protocol is a set of rules that dictate how data and control information is exchanged between communication entities. The network protocol of RSVC defines how communications are carried out between processes using RSVC and RSVC as well as among RSVC servers. Figure 3 presents the data and protocol used in the RSVC system.

2.2.3 Life Cycle service

Each process using the RSVC toolkit registers itself using the protocol described above into the life cycle service of the primary server which in turn could propagate this information to the secondary server. The information stored in the service then is updated periodically by the correspondent process. A separate task checks time stamp of information of the processes periodically and sends out timeout notification to interested parties if a time stamp has not been updated for a predefined time interval. Furthermore a termination from a process or a node crash triggers different notifications. The life cycle service also offers capability of restarting a crashed server from registered server information. Finally a different set of notifications are generated when a new process joins RSVC or a dead process recovers, which enables applications to reestablish connection to those servers automatically. Figure 4 shows the architecture of life cycle service of RSVC.

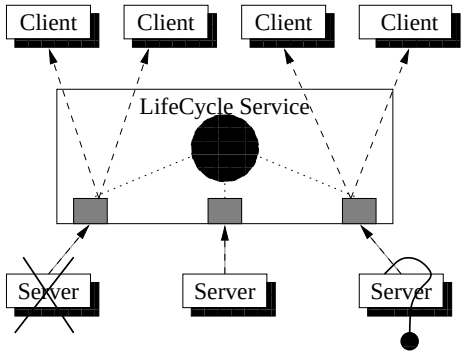

Figure 4: Illustration of life cycle service.

2.2.4 Program Interface

A set of APIs is defined not only to simplify the usage of RSVC but also to allow applications to use their own network protocol to communicate among themselves. The APIs are organized into a single C++ class called *rsvcClient*. Both synchronous and asynchronous communications with RSVC servers are possible. Applications using the life cycle service have to provide callbacks to handle different notifications from RSVC. In addition a CDEV service for *rsvcClient* is provided, which enables any CDEV applications to refer RSVC by name (rsvc).

3 Conclusions

The RSVC toolkit is a new C++ toolkit for developing reliable and fault tolerant distributed systems. It consists

of a set of network servers implemented in C++ along with a very simple set of APIs to reduce development costs. The network protocol and data used to store information in the servers are flexible C++ data objects that allow a different system to tailor RSVC to a new environment. The RSVC toolkit offers naming service in addition to life cycle service. Applications using RSVC benefit from better reliability and less manual supervision.

Currently the RSVC system has been used in the control system at Jefferson Lab. The RSVC server has been tested on Hewlett-Packard machines running hpux-9 and hpux-10, Sun workstations running Solaris 2.x and Intel based PCs running Linux 2.0.x. The APIs have been tested on mv162, mv167 running VxWorks real-time kernel in addition to the above platforms.

4 Acknowledgment

Special thanks to Walt Akers and Johannes Van Zeijts in the control software group of Jefferson Lab for their valuable suggestions and timely integration of RSVC into the control system at Jefferson Lab.

References

[1] Leo R. Dalesio, et. al., The Experimental Physics and Industrial Control System Architecture: Past, Present, and Future, *International Conference on Accelerator and Large Experimental Physics Control Systems*, Oct. 1993.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, N.J.

[3] K. P. Birman and R. van Renesse, eds., *Reliable Distributed Computing using Isis Toolkit.* IEEE Computer Society Press, 1994.

[4] K. Birman and R. Cooper. The ISIS Project: Real Experience with a Fault Tolerant Programming System. In *Workshop on Fault-tolerance in Distributed Systems.* ACM, 1990.

[5] "Special section on group communication". Communications ACM, 39(4).

[6] J. Chen, G. Heyes, W. Akers, D. Wu and W. Watson, CDEV: An Object-Oriented Class Library for Developing Device Control Applications, *Proceedings of ICALEPCS 1995*, pp.97-105.

[7] Douglas C. Schmidt, Reactor – An Object Behavioral Pattern for Event Demultiplexing and Event Handler Dispatching. In *Pattern Languages of Program Design*, Addison-Wesley, 1995.

[8] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. The Primary-Backup Approach. In Sape Mullender, editor, *Distributed Systems*, pp.199-216. ACM Press, 1993.