2012 International Conference on Computer Technology and Science (ICCTS 2012)
IPCSIT vol. 47 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V47.42

Publisher/Subscriber InterProcess Communication

Nidhi Sharma Kaushik[1], Saurabh Singh Sisodia [2] and R.N.Rajotiya [1+]

[1]Student, Advanced Institute of Technology, Palwal

[2]Senior software Designer (Rates Technology), Royal Bank of Scotland IDC, Gurgaon

[1+]Head of Computer Science Department, Advanced Institute of Technology & Management, Palwal

Abstract. In the present time computers are penetrating into more fields every day. As the complexity and scale of tasks to be performed to carry out any business is increasing exponentially it is has become a necessity that the tasks be automated to the greatest level possible. The possible solution is that we break the task into subtasks. As expected all these subtask are solving a common problem so would require sharing data. This project is an attempt to implement a mechanism for this data sharing using the Publisher/Subscriber model. In this model the basic concept is the one component publishes the data and other component(s) subscribe to it. All the subscribers receive updates to the information they are interested in. The main aim of the project was to develop a protocol to ease the movement of data within a system that has many components interacting amongst themselves with a many to many relationship. So there are many instances in such system where a particular component provides information which is of interest of one or many other components of the system.

The also describes a reference implementation of framework using the above described protocol. The implementation for the system is done in C++ using the concepts of Object Oriented design. This is a major improvement over most of the similar systems existing, as most of them are implemented in C and do not have a proper object oriented design. The framework consists of a library and a central component called master. There can be multiple master applications running in a system which will support load balancing and fault tolerance. The system as a whole has been designed to avoid any single point of failures so that it can be used in a 24x7 environment.

Keywords: Computer Networks, Distributed Computing, InterProcess Communication, Publisher, Subscriber

1. Introduction

There are many business applications which are based on a collection of functionalities. Each of the functions is best understood by experts in that area. This implies each of the function should be best implemented in a separate module so that each team can be assigned to implement and maintain a single module. The complete business application needs all these applications to interact with each other by passing information amongst each other. This type of application is very commonly used in financial applications. The solutions developed in financial institutions are in general small applications which provide a small part of the functionality (basically process the data) and pass it on to another application. The interesting point in all these applications is that the data is structured and follows a common format. So basically what we can define is a communication engine which takes care of the data transmission and the structure of the data can be defined as per the application logic.This paper is an attempt to define such an engine which comprises of the inter process communication protocol, a central application which is responsible for bookkeeping and

[+] Corresponding author. Tel.: + (91- 8901520088);
E-mail address: nidhisharma.pwl@gmail.com

status management of applications in a system and a library which can be used by applications to use the engine. One practical example of such a system is: In a trading system the trade is done on multiple exchanges with each of them implementing it's own communication and business rules. Also for correct pricing traders need information from different markets, with control over the information. This means they need to be able to know the average, best and maybe filter out prices based on user defined rules. The aim of this paper is to design a system which acts like an engine for such communication and also provide a reference implementation for the same.

2. Architectural Design

Components

The design of the system is such that any implementation will have to have the following components for it to work correctly. These are the following types of components that will exist in the system:

Master:

This is the central component managing the complete system. It will be responsible to perform the following tasks:

- State management
- Authentication
- Published type and functions
- Subscriptions

Publisher:

This type of component will be the publisher of information or the handler for remote procedure calls. The component needs to register the type(s) of data it will be publishing with master before starting to publish information. Similarly it needs to publish the function details (name, arguments and return type) before it can handle any incoming function calls. Also there can be changes requested by subscribers for a particular record which is called transaction for the record.

Subscriber:

This type of component will subscribe with a publisher to receive data either of a particular type, for a particular record or for a group of records of a particular type. There will be a 2 step process for subscription. The first step will be retrieve information about the type from the master and then send the request for subscription to the publisher of the type. In case of function call the details about the function will be retrieved from the master and then the call will be made to the function call handler.

3. Types of Communication

The various components described above will communicate amongst themselves to make the system useful for solving business problems. The components in the system will communicate with each other for the following purpose:

master and master

In case if there are 2 masters in the system then they will communicate for:

- Fault tolerance
- Synchronising
- Load balancing

master and generic component

This describes the generic communication between the master and other components of the system. This is mostly the communication to facilitate state management and security of the system by the master. These messages are generic between master and other data components of the system, i.e not specific to the publisher or the subscriber.

master and publisher

This section describes the specific messages that will be exchanged between a publisher component and master.

master and subscriber

Subscribe: The subscriber gets the information about the record(s) from the master. This information will then be used to subscribe for the record updates from the publisher. The information about the type will be cached with the subscriber so that overhead is not added due to refetching of type information for each subscription.

publisher and subscriber

Subscribe: The subscriber requests for subscription for information on record(s).

Unpublished: A record has been unpublished

4. The Protocol

The header for Publisher/Subscriber protocol is defined as per the image below:

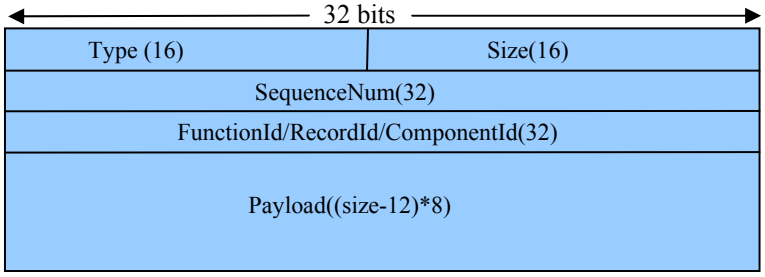

Figure 1: Packet structure of the PubSub protocol

This protocol will basically have the following category of packets:

Data packets: Used to transfer record updates and transactions

Function packets: Used to make function calls and get results

Control packets: Used to perform control activities

Notification packets: Used to notify about state changes

These categories are further divided into following packet types:

Data

Record updates and Transaction.

Function

Function call and Function Return

Control

Register, Deregister, Publish Type, Publish Function, Unpublish Function, Publish record, Unpublish record, Subscribers, Unsubscribe, Refresh Component, Suspend Component.

Notification

Unpublish Function, Unpublish Record, Component down, Component up, Error.

The Size of field specifies the total size of the packet including the header. Payload will have the following structure depending on the type:

- Function calls and Control actions: List of arguments containing the following information for each argument: size, argumentId and value
- Record updates: List of attribute values containing following information for each attribute: size, attributeId, value
- Function Return: List of return values containing following information for each value: size, sequenceNum, returnId, value
- Control actions return and error status: Error status in the following format size, sequenceNum, ErrorCode, Description

5. Low Level Design

5.1. Modules

SocketLib

This is the communication layer of the system. It is basically a C++ wrapper over the C socket layer. The main concepts used for this layer is the use of abstract class factory for socket creation. This helps to make the communication layer extensible so that we can replace TCP/IP by any other protocol suite if so desired.The second use is the Interface oriented design. We just expose the interface to the upper layer and the implementation details are completely hidden. This leads to light cohesion and light coupling between the layers

UtilityLib

This is a collection of utility classes which has helper classes and functions defined.

ApplicationLib

This is the library which implements the framework which will be used to develop publishers and subscribers.

Master

This is the central application. The management and bookkeeping of the system is responsibility of this application

5.2. Class design

SocketLib

Socket: This class is the root of socket classes. It just stores the identifier of the socket. In the case of TCP/IP which is the integer value returned from the socket call.

CommonLib

- Component: This class stores information about the component.
- ComponentEvent: Subclass of Notification event specific for component related events like component registered/deregistered. Important member is getComponent
- FunctionEvent: Subclass of Notification event specific for function related events like type published/unpublished. Important member is getFunction
- AttributeDetails: The details of an attribute. This attribute defines either a type attribute or a function argument.
- Type: The type object. This defines the type of record that can be published. Important members are name, id, attributeDetailsList.

ApplicationLib

Application: The root object which needs to be implemented in any component that needs to use the framework.

Cache:A template class which will be used store information that are retrieved from master so as to reduce the round trip.

Master

This is the main class which is contains the core logic of the master application.

6. Sequence Diagrams

The sequence diagram below shows the sequence of action performed for a record subscription and function call.

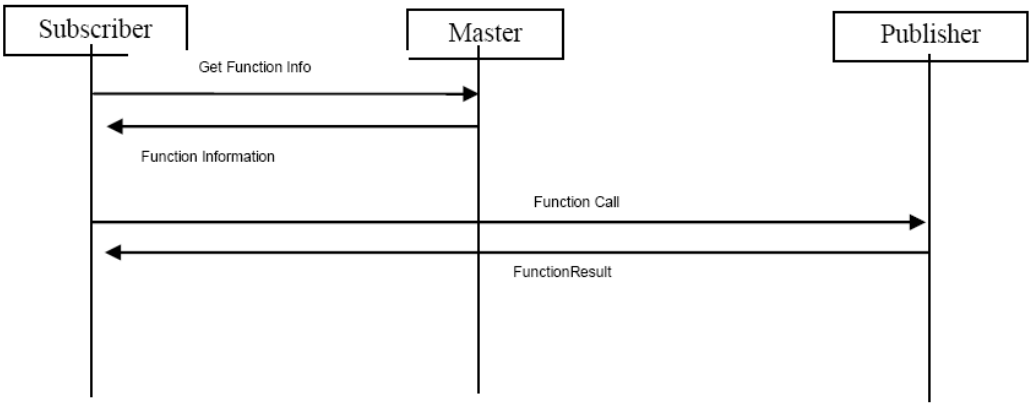

Figure 2: Function call Sequence Diagram

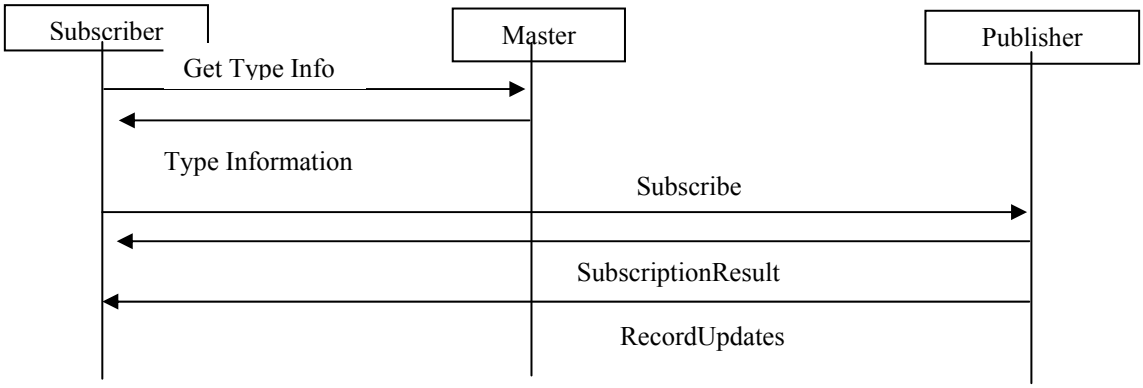

Figure 3: Subscription Sequence Diagram

6.1. State Diagram of a component

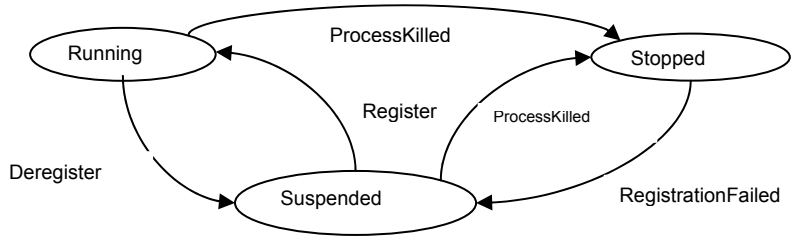

Figure 4: State diagram of component

There are three states in which a component can be in:

Running: The component is executing and is registered with the master so that it can call a function, publish a function, publish a type or subscribe for information. This state can be reached when component calls a successful register either from suspended state.

Stopped: The component is not executing. This is the initial state and can be reached if the program is stopped (process killed)

Suspended: The component has just started or has not been able to register then it will be in suspended state.

7. Acknowledgements

I would like to extend my gratitude towards god, who guided us through the way.To Saurabh Sisodia Singh and Dr. R.N.Rajotiya, for his great efforts of supervising and leading, to accomplish this fine work.To every person gave us something to light our pathway, we thank them for believing.

8. References

[1] Andrew S. Tanenbaum: Computer Networks, Prentice Hall 2002

[2] Boost Library : www.boost.org documentation for boost library to be used by development.

[3] Douglas Comer: Internetworking with TCP/IP Volume I: Principles, Protocols and Architecture, 5th edition Prentice Hall 2006

[4] Douglas Comer (with D. Stevens): Internetworking With TCP/IP Volume II: Design, Implementation, and Internals, 3rd edition Prentice Hall 1999

[5] Douglas Comer (with D. Stevens): Internetworking With TCP/IP Volume III: Client-Server Programming and Applications, Linux/POSIX Socket, Prentice Hall 2000

[6] Linux Socket Programming by Example

[7] Lker-Level Interprocess Communication for Shared Memory Multiprocessors BRIAN N. BERSHAD Carnegie Mellon University THOMAS E. ANDERSON, EDWARD D. LAZOWSKA, and HENRY M. LEVY ,University of Washington

[8] publish/subscribe interprocess communication package e. Wolin, d. Abbott, v. Gyurjyan, g. Heyes, e. Jastrzembski, d. Lawrence, c. Timmer, jefferson lab, newport news, va 23606, u.s.a.

[9] TIBCO Design Patterns: www.tibcommunity.com