Communications and Network, **2013, 5, 34-38**
doi:10.4236/cn.2013.51B009 Published Online February 2013 (http://www.scirp.org/journal/cn)

Scientific Research

A Communication Middleware with Unified Data Transmission Interface

Yu Zhang, Yuanbing Zhang, Xiangyuan Bu, Chao Zhang
School of Information and Electronics, Beijing Institute of Technology, Beijing, China
Email: zhangyb0108@gmail.com

Received 2012

ABSTRACT

In a Modern communication network, data exchange become a complex problem because there usually exists a various data formats due to the diversity and complexity of communication devices. In this paper, a communication middleware with unified data transmission interface is introduced, which acts as the abstract communication layer in the heterogeneous distributed private communication network. Devices within the network only need to interact with the middleware, instead of directly communicate with each other by various protocols and interfaces, which can reduce the complexity and diversity of the heterogeneous network. This article describes the structure and features of the middleware, and analyses the use of "buffer pool" in message sending and receiving process. The applications of this middleware can reduce the complexity of heterogeneous network interoperation, and improve communication efficiency.

Keywords: Communication Middleware; Buffer Pool; UDP; XML

1. Introduction

With explosive development of telecommunication technology, a modern communication network is becoming more and more convenient and powerful. Meanwhile, it becomes more and more complex, which brings additional technical difficulties. For a communication network in a local area which integrates multi abilities, including the telecommunication, localization and navigation, we must make the data transmission and exchange between any two devices smoothly and reliably [1]. Since the network is composed of various devices with different protocols and physical layer specifications, to provide the direct connection between devices through the physical layer becomes very complicated and costly. Also, because of different data formats, the quick and correct data format translation is crucial for the network. To solve this problem, a communication middleware with unified data transmission interface is introduced in this article. Combined with the hardware adaptors, the middleware acts as an abstract communication layer, and provides an effective and flexible centralized message service. It can receive messages from source devices, and transfer them to the destinations accurately. As a result, multiple devices within the network only interact with the middleware through messages, and do not need considering the format converting or data translating process [2]. The use of this middleware can greatly reduce the complexity of device interoperation, and furthermore improve communication efficiency.

*This work was supported in part by the Ministry of Science and Technology of the People's Republic of China under Grant 2011BAH05B08.

2. Architecture of the Network

In this article, the communication middleware with unified data transmission interface makes the information exchange between various devices of the network based on a logical software bus, so the communication-related devices only need to establish a direct connection with the middleware, and can achieve communication with other ones. Thus, the coupling degree between devices within the network can be significantly reduced, and data can be transmitted in a unified way. The architecture and data flow of the entire communication network are shown in **Figure 1**. System data are preserved in XML format, and then transmitted through UDP protocol [3]. The information of various devices is saved in a system database, so the middleware can obtain message configuration information by querying the database, and then transmit messages.

3. Basic Function of the Middleware

3.1. Centralized Message Service

It is the key service of the middleware. As a transmission middleware of a large number of received messages, it

cannot transmit them directly at the same time. Therefore, the middleware creates a cache folder in the local server, which is used to save the received messages temporarily, and transmit them from here as soon as immediately. The establishment of cache folder is automatic and immediate, that is, a message is automatically saved in the temporary folder when sent to this middleware, and its forwarding and query are both from here.

3.2. XML Format Messages

If the identifiable message formats of various devices are not the same, a destination must translate data from source terminals into a format that it can understand, and the translation process will cost a lot of system resource. So the messages within network are all defined in XML format, which simplifies the complexity of the transmission and exchange [4]. The XML file is represented by some hierarchical elements. An element is defined by a pair of tags, called the start and end tags. Content between the pair is the element body, which may contain a set of child elements. The unified-format message is not only easy to be transmitted, but also conveniently parsed out according to certain rules when arriving at destination, and the message itself will not be changed.

3.3. Loose Coupling

This middleware is loosely coupled with other devices, that is, it has no knowledge of other separate devices' definitions. Devices and the middleware do not know each other's working process, and the communication between them is completely achieved by the messages [5]. As long as the message accords with the structure regulation, the clients' requests or the servers' feedback service can be realized. Also, should the receiver application fail for any reason, the senders can continue unaffected, as the messages they send will simply accumulate in the message queue for later processing when the receiver restarts. Loosely coupling middleware platform can make full use of existing system resource and construct flexibly and effectively. Under the premise of no more than the equipment load, the middleware can access a large number of devices, in order to meet the requirement of large communication networks; it also can provide a powerful global information organizational ability, to reduce the error rate of the network.

3.4. Real-Time Transmission and Real-Time Response

This middleware uses the UDP protocol to transmit messages. UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity, that is, a kind of connectionless-oriented data transfer protocol [6]. Compared with TCP, it has higher transmission efficiency, and is more applicable to the real-time system. Also, UDP's stateless nature is also useful for terminals answering small queries from huge numbers of clients. And it supports packet broadcast (sending to all on local network) and multicasting (send to all subscribers), which apply to the system with various message types and large transmission capacity.

3.5. Persistent Messages Guarantee Reliable Transmission

As we know, UDP provides an unreliable service and datagram may arrive out of order, or go missing without notice. Therefore, both the sending devices and the middleware itself, use the programmable interval timer (PIT) when sending messages. The timer's interval is set as the repeat sending time of message, so the same message can be sent periodically after the timer starts, until the receivers return a finishing flag, then an interrupt is triggered, and the next message can be transmitted. As a result, a small amount of message loss will not affect the information transmission.

4. Architecture of the Middleware

In this communication network, each device can be used as a message sending or receiving terminal. These terminals are able to widely listen for the location information, their own state information, and the unexpected information in emergency, and then transmit the information to this middleware periodically. The middleware will send them promptly, so as to ensure that the destination terminals can receive updated information. The key architecture of the middleware is shown in **Figure 2**.

Accurate and timely sending and receiving of mass data is still the main problem the communication middleware faced with. To solve the problem, the technology of "buffer pool" is used in this article, combined with multicast to send and receive messages [7]. It contains three key services as follows:

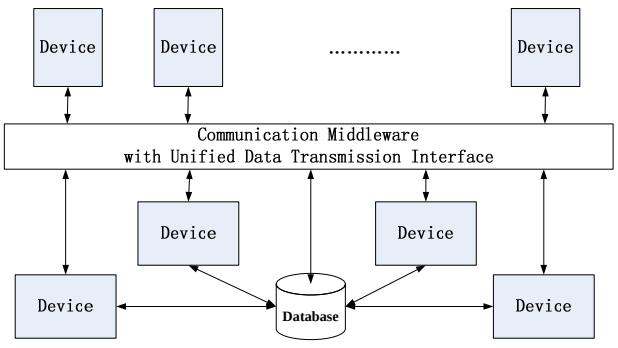

Figure 1. The achitcture and data flow of the communication network.

4.1. Monitoring Control

This module completes the monitoring and receiving functions. Ensuring to receive messages uninterruptedly, the receiving port of middleware is always on. Messages will be automatically received to the local server when coming to the middleware. In the communication system, the number of message is large while the sending time is uncertain, so the middleware cannot send all messages at the same time. Therefore, it is necessary to design a message receiving buffer pool.

First, some memory space is opened up for receiving buffer pool, and a receiving-operation-related list, Receive File Manager List, is set here, whose elements are the message receiving threads. All the threads are defined as the objects of the receiving class (Receive File Manage). The class relationship diagram of receiving buffer pool is shown in **Figure 3**.

The Receive File Unit is defined as a message record. The Receive File Manager is defined as a message receiving container class, which contains a dynamic array object. The Receive File Manager List is defined as a buffer pool class, that is, the message receiving buffer pool. As is shown, the buffer pool class, Receive File Manager List, contains 10 Receive File Manager's objects, whose dynamic array object is the Receive File Unit record.

Buffer pool is a kind of FIFO data structure. When a new message arrives, the list will automatically check whether the message exists in the buffer pool. If not, the message receiving thread will be created and added into the list; if it existed, new thread will be no longer created. Starting receiving a message means adding a Receive File Manager object into the buffer pool Receive File Manager List. Only when the thread object is added, the whole message receiving process begins, including the file identification, unpacking, checking and resume broken transfer. The message can be completely received by the middleware only when all of the operations mentioned-above finished correctly.

According to the time order, all the monitored message thread objects will join the receiving buffer pool, which completes the receiving operation one by one. And then the threads will be released, so as not to take up too much system resource. If the thread cannot be access to the buffer pool timely and normally, the list will automatically detect whether the buffer pool reaches its maximum capacity. If so, it gives a blocking prompt to the message senders. Using the buffer pool to manage the reception of messages, no matter how large the volume is, can make sure them received by the middleware platform orderly and timely.

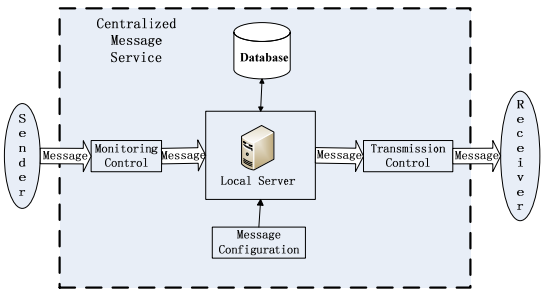

Figure 2. The key achitcture of the middleware.

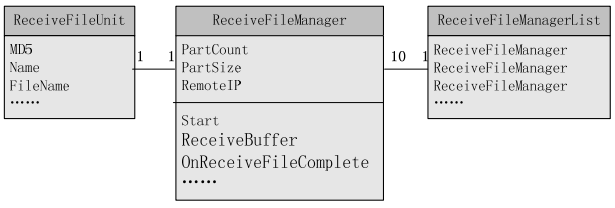

Figure 3. The class relationship of the receiving buffer pool.

4.2. Message Configuration

This module configures the corresponding relationship between the messages and destinations based on the background database of the middleware. As the middleware is connected with all communication devices in the network, its background database has saved all their connection information, such as IP address and port number. When a message is received, this module will configure its transmission routing according to its tag and information of the receiver. When the receiver establishes a connection with the middleware, this message will be transmitted according to the configured routing.

4.3. Transmission Control

This module completes the message transmission function. Similar to the monitoring module, it uses connecting buffer pool to manage the terminals. That is, a Udp-Send List buffer pool is defined to manage all the Udp-Send objects. The class relationship diagram of sending buffer pool is shown in **Figure 4**.

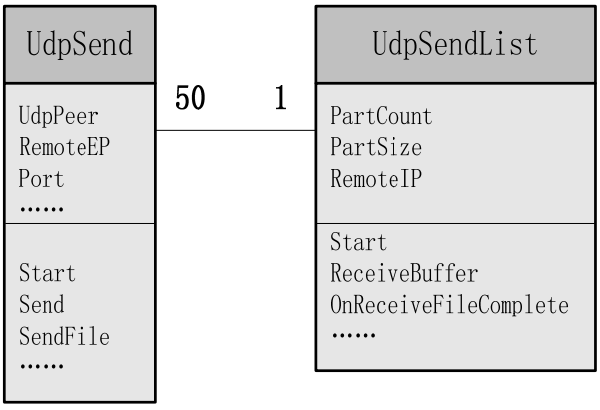

Figure 4. The class relationship of the sending buffer pool.

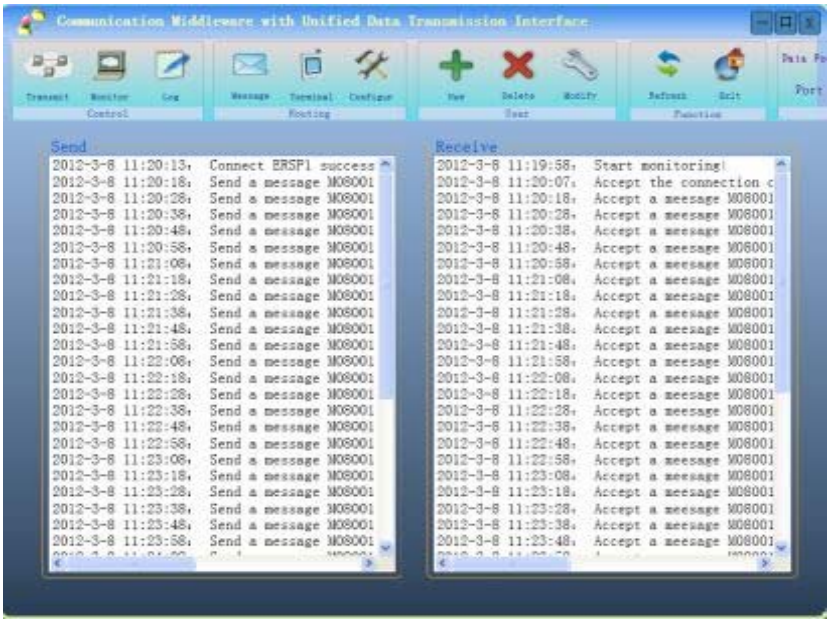

Figure 5. The implementation of the middleware.

When the middleware is connecting with a terminal, it will create a UdpSend object to package IP address, port number and any other properties of UDP connection, and add it to the UdpSendList buffer pool. The buffer pool is conducive for the middleware to judge a terminal is online or not. The terminal's UdpSend object in the list means it online, and the next sending operation can start.

When there are relatively fewer receiving terminals, the UDP point-to-point transmission method can be used, which means the platform operates the UdpSend object of the UdpSendList directly. However, when sending messages to a large number of receivers at the same time, the multicast technology should be used. Before sending a message, according to message routing configuration, the middleware platform adds all the receivers that need it into a multicast group. When the middleware sends the message to this group, all the receivers will get it. This multicast group is dynamic, and the involved receivers are changing depending on the message routing. The middleware has a repeat sending time to ensure every message can be sent to the corresponding receiving terminals, and avoid the information omission because of UDP packet loss.

5. Implementation of the Middleware

The implementation of the middleware is show in **Figure 5**.

The implementation of the middleware contains the interface driver control, routing configuration, traffic monitoring, user authority management and log recording. Therefore, the middleware can connect devices within the network through different interfaces and protocols, and the messages from them can be configured according to the routing and then received and forwarded. The **Figure 5** shows the log of data exchange, in which the entire work process can be recorded. As is shown, a sender keeps sending messages to this middleware, which are processed by the internal buffer pool, and the receiver can get them accurately and timely. The data traffic is monitored by the middleware, in case of the information congestion. Also, users signed in are managed strictly. The middleware realizes the functions above, and ensures all of the data not delayed or lost.

6. Conclusions

A unified-data-interface communication middleware is presented in this paper, which has a kind of centralized message sending and receiving mechanism, and acts as the abstract communication layer in a heterogeneous distributed network. Any two communication devices of the network do not communicate with each other directly, but through messages exchanged by the middleware, then complete the communication functions such as sending request and receiving service. Therefore, the coupling degree of network is greatly reduced. Also, unified data format avoids the consumption of format conversion, accelerates the flow of data, and thereby increases the efficiency of communication process. With the lower communication complexity, each device can focus on their respective functions such as data processing. For a communication network with fixed internal communication requirements, large data traffic and higher node mobility, the middleware platform is widely practical.

REFERENCES

[1] A. Tsutsui, H. Maeomiti, R. Kawamura and K. Yata, "An Adaptive Communication Middleware for Network Service Coordination," *In Proceedings of 1st IEEE Consumer Communications and Network Conference*, New York: IEEE, 2004, pp. 406-411.

[2] J. AI-Jaroodi, N. Mohamed and J. Aziz, "Service Oriented Middleware: Trends and Challenges," *In Proceedings of ITNG2010-7th International Conference on Information Techonogy: New Generations*, IEEE Computer Society, 2010, pp. 974-979.

[3] L. Caviglione, G. Ciaccio and V. Gianuzzi, "Architecture of a Communication Midleware for VANET Applications," *In Proceedings of the 10th IFIP Annual Mediterranean Ad Hoc Network Workshop*, New York: IEEE, 2011, pp. 111-114.
doi:10.1109/Med-Hoc-Net.2011.5970474

[4] V. Schuermann, A. Buda and J. F. Wollert, "XML-based Middleware Approach for Industral Wireless Communication Systems," *In Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation*, NJ, USA: IEEE, 2008, pp. 632-639. doi:10.1109/ETFA.2008.4638463

[5] G. Kitagata, J. Sekiba, T. Suganma, T. Kinoshita and N. Shiratori, "Communication Mechanism of Loose Coupled Agents in Fames," *In Proceedings of Seventh Iternational Comference on Parallel and Distributed System*, Los Alamitos: IEEE Comput. Soc., 2000, pp. 467-472.

[6] G. Gehlen, F. Aijaz and B. Walke, "Mobile Web Service Communication over UDP," *In Proceedings of IEEE 64th Vehicular Technology Conferrence*, VTC-2006 Fall, IEEE, 2006, pp. 2876-2880.

[7] J.-Y. Chwng, D. Ferguson, G. Wang, C. Nikolaou and J. Teng, "Goal-oriented Dynamic Buffer Pool Management for Data Base Systems," *In Proceedings of First IEEE International Conference on Engineering of Commplex Computer Systems*, Los Alamitos: IEEE Comput. Soc. Press, 1995, pp. 191-198.

