Implementation of the Session Manager for a Stateful Server

JiYeon Lee[0], ChangHoe Kim, Jong-Pil Yi, Hoon Choi
Dept. of Computer Engineering, Chungnam National University

Chungnam National University, 220 Gung-dong, Yuseong-gu, Daejeon 305-764
Republic of Korea

E-mail : {eunbi, kchoe, jpyi, hchoi}@ce.cnu.ac.kr

Abstract - Advances of mobile communication technologies and device technologies brought the mobile computing era where people use multiple information devices such as a PDA, a cellular phone and a handheld PC to connect with various Internet services. Very often, the client device needs to exchange multiple packages or messages with a server in the course of service access which is called a session. There are several ways for a server to keep certain information that is required to be maintained during a session. This study proposes and implements three of these mechanisms for a stateful server. We carry out the performance measurement from the implementations and compare each mechanism.

1. Introduction

Advances of mobile communication technologies and device technologies brought the mobile computing era where personal information devices such as PDA(Personal Data Assistant)s, handheld PC(Personal Computer)s and cellular phones become important part of our life. People access Internet service more often than before and from anywhere, anytime.

Client devices access various types of Internet servers by sending a service request. Sometimes it is enough to send a single message to a server and get a response. But it is common that the interaction between a client and a server needs exchange of multiple messages in sequence. A stateful server maintains the information about the client and the on-going service request during this interaction which is called a session. A stateless server does not store the session information [1].

An easy way to keep this temporary, session-related information for a stateful server is to fork a process for each session so that the process saves the information in its local variables. But the server may suffer from performance degradation due to the overhead of creating processes and heavy memory usage. An alternative mechanism is to use a DLL(Dynamic Link Library) function. Because DLL shares memory spaces, it can relieve from a server's memory overhead. However, this mechanism has a problem to keep the session information after DLL is unloaded. So the session information must be stored in a persistent storage of the server such as a database or a file system.

This paper investigates mechanisms to maintain the session information for stateful servers. We implement three mechanisms and evaluate their performance for comparison. The application that the server is used for is data synchronization between client devices such as PDAs, cellular phones and a network server [2][3]. Three versions of this server implement above-mentioned mechanisms to store session information.

After a brief introduction of the SyncML Server architecture in Section 2, we describe various mechanisms we implemented for a stateful server in Section 3. Then, we compare the performance of these mechanisms in Section 4.

2. SyncML Server Architecture

Proliferation of personal information devices results in a problem of maintaining user data that are spread over multiple devices. A user may have an address book in a cellular phone and has almost identical data in his PDA as well. The user must update the data from all of these devices whenever a change is made. This situation is painful and often leaves some data to be inconsistent with the same item in other devices. Therefore, the SyncML Initiative has produced a standard protocol suite in order for multiple information devices to synchronize their data with a server [4][5][6][7].

We implemented three versions of data synchronization servers, i.e. CNU SyncML Server 2.0, the Server 3.0 and the Server 3.1, based on the standard protocol suite from the SyncML Initiative [8]. The CNU SyncML Server consists of 8 frames as shown in Figure 1.

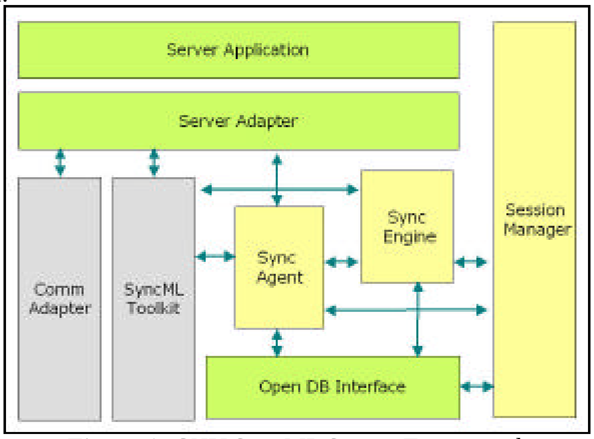


Figure 1. CNU SyncML Server Framework

The Server Application provides an interface for a user or the application administrator to modify contents data at the server. The Server Adapter passes messages from client devices to the SyncML Toolkit via the Communication Adapter. The Server Adapter also passes the parsed data from the Toolkit to the Sync Agent. The SyncML Toolkit encodes and decodes SyncML messages. We used the reference implementation code from the SyncML Initiative for the SyncML Toolkit. The Sync Agent implements the SyncML protocols [4][5]. Therefore, its function is application independent and can be commonly used for all SyncML applications. On the other hand, the Sync Engine implements application dependent part such as a service policy, resolution rules in case of data conflict, etc. The Session Manager manages a temporary, session related information that needs to be maintained in the server during a session. Lastly, the Open DB Interface implements interfaces to access data in database.

All the frames of the CNU SyncML Servers except the Session Manager were implemented into DLLs. We used HTTP(Hyper Text Transfer Protocol) [6] to communicate with a SyncML client device and JNI(Java Native Interface) to load DLLs.

3. Server Session Manager

When the first message arrives from a client device for the request of data synchronization, the server processes it and sends back the status message. The server may also request the client device to synchronize with data that have been modified by other devices or by the server itself. This synchronization procedure is accomplished by exchanging several messages in order [4]. We call this synchronization period a session (Figure 2).

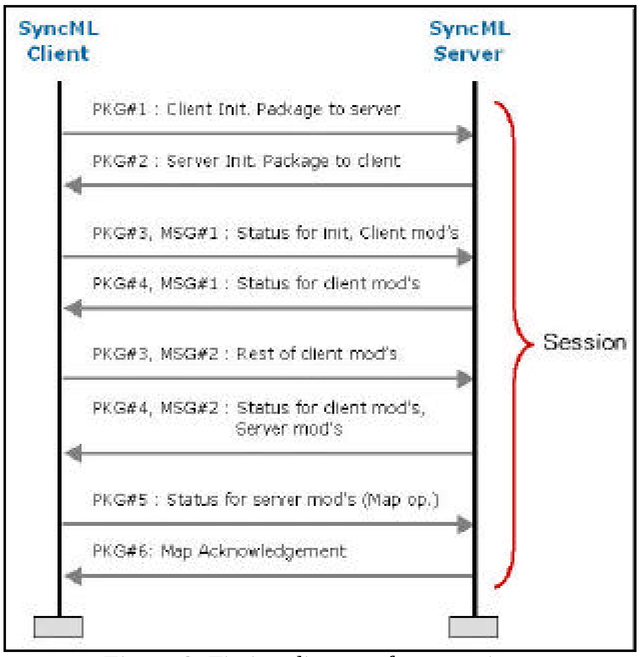

SyncML Client
SyncML Server
PKG#1 : Client Init. Package to server
PKG#2 : Server Init. Package to client
PKG#3, MSG#1 : Status for init, Client mod's
PKG#4, MSG#1 : Status for client mod's
PKG#3, MSG#2 : Rest of client mod's
PKG#4, MSG#2 : Status for client mod's, Server mod's
PKG#5 : Status for server mod's (Map op.)
PKG#6: Map Acknowledgement
Session

Figure 2. Timing diagram for a session

When a session is begun, the server needs to maintain the session information such as a session identifier, the synchronization commands and the status of commands processed by that time. This information is used to build response messages to the client. The session information is temporary and is released once the session is over. The Session Manager takes charge of maintaining the session information in this system.

For the CNU SyncML Server 2.0, we implemented the Session Manager in a form of a DLL as shown in Figure 3. Therefore the Session Manager is loaded into memory just when it is used. This allows us to save the server's resource. But on the other hand, we encounter with a problem in keeping the session information after the DLL is unloaded. To resolve it, the Session Manager stores the session information into a persistent storage, SYNCSM database. Then it is necessary for the Session Manager to connect the SYNCSM database every time it receives a message fromthe client. That may cause the performance degradation as we investigate in Section 4.

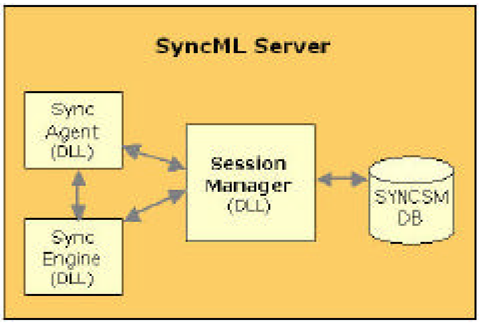


Figure 3. CNU SyncML Server 2.0

The CNU SyncML Server 3.0 has a similar framework to the CNU SyncML Server 2.0. One of the differences is that the Session Manager is implemented into an independent process that communicates with other frames in the server by RPC(Remote Procedure Call) as shown in Figure 4. In this system, the SYNCSM database stores the session information as the same way as the Server 2.0. But the connection to the SYNCSM database is maintained during a session. Therefore, the number of connection set up to the database reduces to once rather than the number of messages from the client.

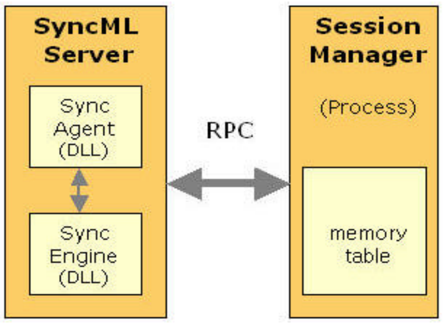


Figure 4. Communication between CNU SyncML Server and Session Manager

In the version 3.1, the Session Manager is an independent process as before, but it stores the session information into tables in the main memory. After completing the synchronization, the Session Manager removes the session information from memory and closes the session. Therefore we avoid accessing a database to store temporary information. Using a persistent storage such as a database or a file system results in increased access time and bottleneck when servicing multiple sessions. By keeping session information in memory, we can reduce the processing time of a message.

Weakness of this approach might be reliability. Despite of the fast processing time, the Session Manager may loose the information if the Session Manager process fails. Therefore the server needs fault tolerant, replicated Session Manager processes. Anyway, if the Session Manager process fails while a session is on the way, the client will retry the synchronization procedure again from the beginning. Loosing the session information does not mean loosing user data.

The Sync Agent accesses the session information through the Session Manager APIs. Figure 4 shows a communication between the server frames and the Session Manager in the CNU SyncML Server 3.1. The difference between the Server 3.0 and the Server 3.1 is the place where the session information is stored. The former stores it in the persistent database while the latter does it in the main memory.

The timing diagram between the Sync Agent and the Session Manager during a session is illustrated in Figure 5. Before starting a synchronization procedure, the server performs authentication of the client and the validation of the last anchor value. After that, the command handler of the Sync Agent begins to process the received message. Then Session Manager creates a new session information for the synchronization. Next, the Session Manager returns the session identifier to the Sync Agent. The Sync Agent performs the synchronization procedure through use of the session identifier and the interface to the Session Manager. When the synchronization is completed, the Session Manager removes the session information in memory and closes the session.

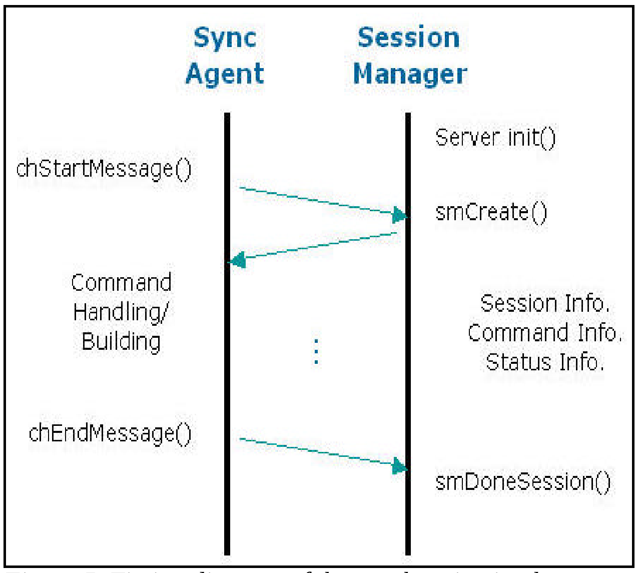
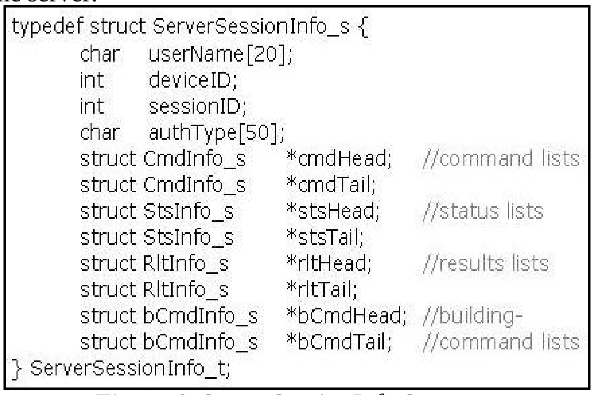


Figure 5. Timing diagram of the synchronization between the server and Session Manager

Figure 6 shows the data structure used in the Session Manager. The ServerSessionInfo structure consists of the following data: user name, device identifier, session identifier, authentication type, list of commands from the client, status list, results list and list of commands from the server.

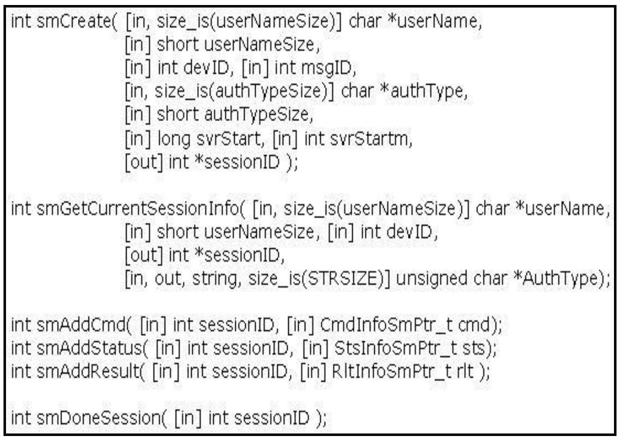


```
typedef struct ServerSessionInfo_s {
        char    userName[20];
        int     deviceID;
        int     sessionID;
        char    authType[50];
        struct CmdInfo_s     *cmdHead;    //command lists
        struct CmdInfo_s     *cmdTail;
        struct StsInfo_s     *stsHead;    //status lists
        struct StsInfo_s     *stsTail;
        struct RltInfo_s     *rltHead;    //results lists
        struct RltInfo_s     *rltTail;
        struct bCmdInfo_s    *bCmdHead;   //building-
        struct bCmdInfo_s    *bCmdTail;   //command lists
} ServerSessionInfo_t;
```

Figure 6. ServerSessionInfo Structure

The communication between the SyncML server and the Session Manager is performed by the RPC with the predefined interface. Figure 7 shows a part of the predefined interface to the Session Manager.

- smCreate() creates a new session.
- smGetCurrentSessionInfo() get the current session information since the creation of the session.
- smAddCmd() attaches the command information to the list.
- smDoneSession() removes the session information and deallocates the memory spaces after completing the synchronization with a client.



```
int smCreate( [in, size_is(userNameSize)] char *userName,
              [in] short userNameSize,
              [in] int devID, [in] int msgID,
              [in, size_is(authTypeSize)] char *authType,
              [in] short authTypeSize,
              [in] long svrStart, [in] int svrStartm,
              [out] int *sessionID );

int smGetCurrentSessionInfo( [in, size_is(userNameSize)] char *userName,
              [in] short userNameSize, [in] int devID,
              [out] int *sessionID,
              [in, out, string, size_is(STRSIZE)] unsigned char *AuthType);

int smAddCmd( [in] int sessionID, [in] CmdInfoSmPtr_t cmd);
int smAddStatus( [in] int sessionID, [in] StsInfoSmPtr_t sts);
int smAddResult( [in] int sessionID, [in] RltInfoSmPtr_t rlt );

int smDoneSession( [in] int sessionID );
```

Figure 7. Session Manager Interface

4. Performance Comparison

For the evaluation of various mechanisms to maintain temporary information for a stateful server, we measured times to process one synchronization session in each version of servers. The implementation and test environment was Microsoft Visual Studio C++ on Windows platforms and measurement was carried out with the parameters shown in Table 1.

We measured the times from CNU SyncML Server 2.0 (session information in database), Server 3.0 (database with reduced connection) and Server 3.1 (session information in memory) under the same conditions. The time we measured is processing delay by the server only, it does not include the message transmission delay on the network. Test messages include various commands: ADD, REPLACE, DELETE, PUT, GET commands and so on.

PARAMETER	
Number of user	10 Users
Number of device	More than 2 ea each person
Sync. type	Two-way Sync Slow Sync One-way Sync from Client One-way Sync from Server Refresh Sync from Client only Refresh Sync from Server only
Iteration of the sync.	500 times

Table 1. Parameters for Performance Test

Figure 8. depicts the results from the performance measurement. It turns out that the CNU SyncML Server 3.1 performs best as we expected. The CNU SyncML Server 3.1 is 44% faster than the CNU SyncML Server 2.0 and 31% faster than the CNU SyncML Server 3.0. The Session Manager in the CNU SyncML Server 3.1 manages the session information in memory and do not connect to the SYNCSM database, so it has the fastest processing time than other mechanisms here.

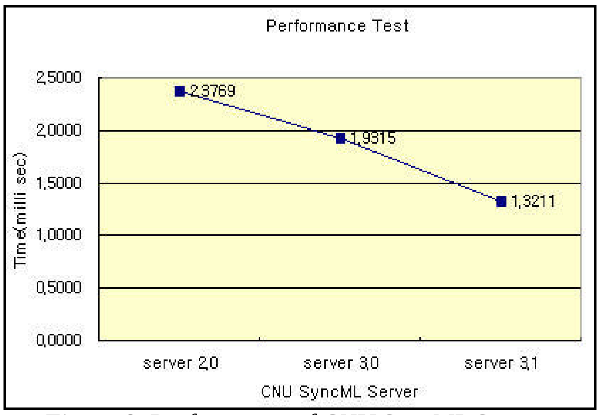


Figure 8. Performance of CNU SyncML Servers

The data exchanged during the synchronization between clients and a synchronization server fall into two categories. One is the application-specific contents data such as the address book data, and the other is the session information which is managed by the Session Manager temporarily during a session. It is obvious to store the contents data in a persistent and stable storage like a database. But, for the information needed to maintain a session, managing it in main memory is more efficient because it is valid during a session only.

The CNU SyncML Server 3.1 has the architecture that the Session Manager manages the session information in the memory. And the experimental results show the improvement of the processing time compared with other server versions that implement other mechanisms to maintain session information.

5. Conclusion

There may be several mechanisms to store temporary, session-related information in a stateful server. In this paper, we implemented three mechanisms and evaluated their performance for comparison.

The application the server is used for is data synchronization between client devices such as PDAs, cellular phones and a network server. We proposed and implemented three versions of data synchronization servers. All three versions are based on the standard synchronization protocol by the SyncML initiative. Three servers implemented above-mentioned three mechanisms.

The CNU SyncML Server 3.1 which stores the session information in main memory showed the best performance: 44% faster than the Server 2.0 and 31% faster than the Server 3.0.

Though the mechanism used for the Server 3.1 performs best, the mechanism used for the Server 3.0 might be an alternative in case the application needs high reliability in the synchronization procedure, i.e., the possible loss of temporary session information due to the failure of the Session Manager process is not acceptable. It depends on an application which property comes first, reliability or the fast processing time.

REFERENCES

[1] G. Coulouris, J. Dollimore & T. Kindberg, Distributed Systems, Concepts and Design(3rd Ed.), Addison-Wesley, 2001

[2] SyncML Initiative, http://www.syncml.org

[3] SyncML Initiative, Building an Industry-Wide Mobile Data Synchronization Protocol, SyncML White paper, Mar. 20, 2000

[4] SyncML Initiative, SyncML Synchronization Protocol version 1.0.1, June 15, 2001

[5] SyncML Initiative, SyncML Representation Protocol version 1.0.1c

[6] SyncML Initiative, SyncML Architecture, version 0.2, May 10, 2001

[7] SyncML Initiative, SyncML HTTP Binding, version 1.0.1, June 15, 2001

[8] Byung-Yun Lee, JinHyun Cho, SooHee Ryoo and Hoon Choi, "Implementation of Data Synchronization Protocol in Mobile Communication Environment," Journal of KISS, submitted

Acknowledgement

JiYeon Lee and ChangHoe Kim were supported by BK21 Human Resource Development Consortium for Information Technology of Chungnam National University.

Hoon Choi was supported by University Research Program of Ministry of Information & Communication in Republic of Korea.