Contemporary Engineering Sciences, Vol. 7, 2014, no. 22, 1171 - 1179
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2014.49146

Open Source-based G2S (Game to System) Engine Design and Implementation

Sangmin Kim and Heejune Ahn*

Dept. of Electrical and Information Engineering, SeoulTech
232, Gongneung-Ro, Nowon-Gu, Seoul 139-743, Korea
*Corresponding author

Abstract

The casino gaming industry has introduced G2S (game to system) standards for the future management protocol for slot machines. G2S does not simply replace the 20 year old SAS protocol with Internet technology but also provides various application and business opportunities to the casino industry. In this paper, we present our design experience of C/C++ G2S engine for slot machines, based on publically available open-sources, gSOAP and SQLite libraries. We discuss design issues by comparing ours with openG2S, another open-source implementation mainly for G2S hosts. Our engine shows excellence in many aspects, including smaller footprint, no-JVM interpreter license requirement, and better command response time.

Keywords: G2S (game to system) standards, slot machine accounting, game IT, protocol design, open source, embedded system

1 G2S Protocol Standards

The casino industry keeps growing its size every year steadily, reaching to 159 billion dollars in 2014 world market at the increase rate of 9.3%. Among the casino games, the slot machine plays a key role in recruiting casual non-serious customers [1], because of its easiness to play and no dealer supports requirement. However, the slot machines still need system maintenance, and accounting recording for imposition of tax, jurisdictional purpose and business enhancement.

Since SAS (Slot Accounting System) [2] is introduced by IGT in early 1990's, it has been the de facto standards designed to automate slot machine meter reporting and event logging, player tracking, bonusing, ticketing and cashless gaming. In spite of 25 years successful operation of SAS systems, they has many limitations, the necessity of SMIB (slot machine interface board) for connecting to hosts, the proprietary protocols between SMIB and host systems, slow communication speed of serial protocol – 19,200 bps, no way to download content to the EGM, and no room for server-based games. GSA released G2S protocol [3] in 2008, a new protocol standards armed with Internet and web-based protocols. G2S has various benefits, making every EGM be able to talk directly to multiple hosts, leveraging off-the-shelf Internet technologies, host accessibility to incredible data in each EGM, and allowing new applications to flourish.

Despite these flexibility and chance of business, technical details are almost hidden in the closed casino IT industry. The paper tries to open up the technical details for the interested IT industry and engineers so that they can contributes the industry and suggest better direction of evolutions. The authors believe that the open source based implementation of G2S protocol engine of EGM side suits for this purpose, because of the easy accessibility of open source software. The technical key features are discussed in Section 2, the architecture and key components of our design and implementation are described in Section 3, and finally in Section 4 we evaluate the systems in performance and functionality together with the comparison with OpenG2S [4], another open source implementation by IGT.

2 Technology in G2S protocol

Fig. 1 and 2 illustrate the topology of G2S systems and layered architecture, respectively. The G2S standards use Internet-based 3 layer protocol. G2S host and EGM communicates in peer-to-peer model, i.e. they work both a server and client simultaneously.

Transport layer functionality and service interface are fully defined by WSDL (Web-Service Definition Language) documents. A typical transport mechanism is HTTP and for security purpose HTTPS. Optional multicast mechanism should be supported for the system providing 'progressive' bonus functionality.

G2S message layer provides addressing and delivery mechanism. Remarkably, the delivery is asynchronous. When a G2S-Body message arrives at host or EGM, its delivery is acknowledged with a G2S-ACK message, but the acknowledgement does not guaranteed that the requests or response commands are processed.

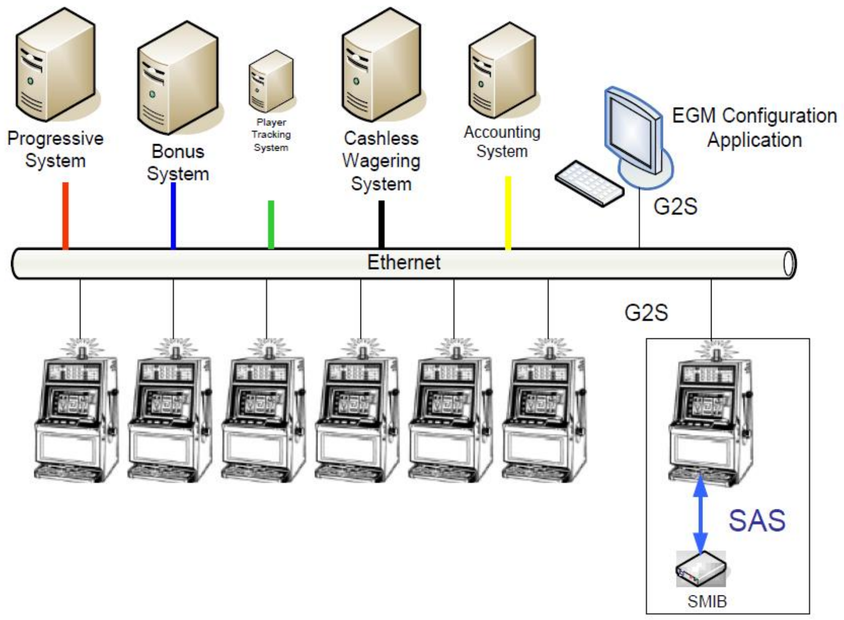

Fig. 1 G2S system topology

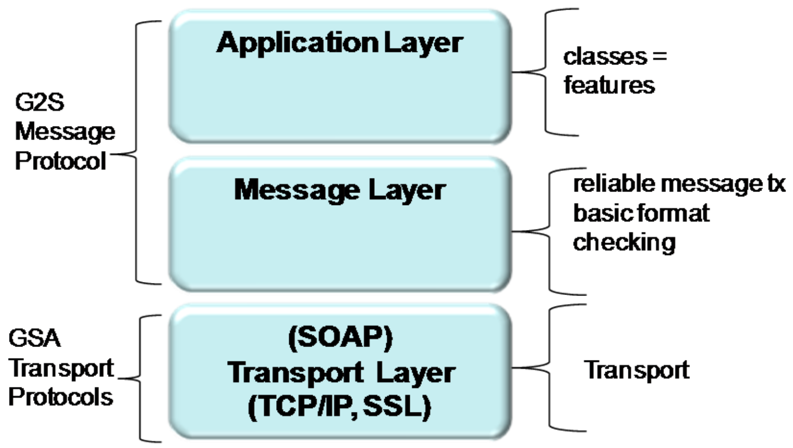

Fig. 2. G2S protocol layers

Key functionality of G2S application functions are classified into G2S-Classes. Some classes provide communications services (communications, eventHandlers), some classes provides game and bonusing service (gameplay, player, bonus, GAT), some for accounting and performance measure (meters, handpay, WAT, Vouchers), and some for systems management (cabinet, idReader, coin and bill accepters) and some for advanced configuration (commsConfig, deviceConfig, optConfig).

3 Open Source Based G2S Engine

Programming Environment: Our G2S protocol is designed for EGM engine, but it can be applied to host development with minor modification due to the symmetric structure of host and EGM functionality. The system is implemented in C++ (99) and operable for both Linux and Microsoft Windows environments. Specially, VS 2012 and eclipseCDT are used as development tools for Microsoft Windows and Linux environments, respectively. Note that some APIs (e.g. select call) work differently between Linux and Windows, so that we implemented a wrapper for full portability.

Architecture: The architecture follows the original 3 layers protocol structure and utilizes open source software libraries as much as possible according to the moto "Never Re-invent Wheel."

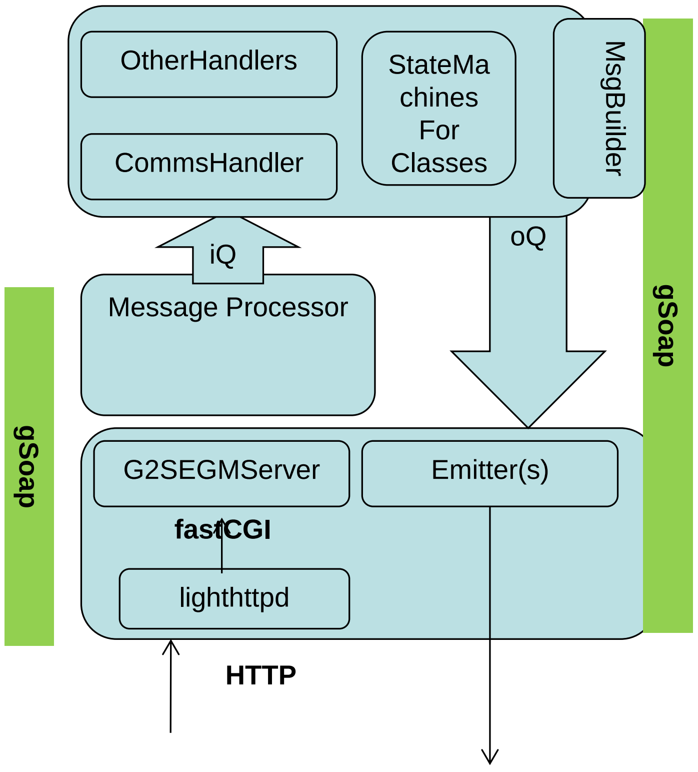

Fig. 3. G2S system topology

Transport Layer HTTP/HTTPS is accomplished by 'lighthttpd' and gSOAP WSDL transport service function. 'Lighthttpd' is used for multi-host concurrency support and easy support of HTTPS's SSL function.

Thread Design: One RX thread and one TX (called an emitter) are used for transport layer and message layer, and single thread is used for applications layer for not using multi-thread locks. Therefore, the application thread waits for 2 event queues, one for commands from host, the other is for systems and user events using multiplexing POSIX select API.

Message Handling: Most application processing is based on event and handler driven way. Fig 4 illustrates the typical message processing flows on a request command arrival. SOAP message decoding is done generated stub of WSDL document by gSOAP. When no message level errors are found in the message, the ‘G2SEGMServer’, playing a role of transport and message layer, responds with g2sAck message. The message on wake-up takes first xml G2S message string into a C++ instance, checks the classes inside g2sBody elements (multiple class elements possible in one message), then dispatches each class or commands to corresponding class handlers. The handlers interpret the parameters in incoming commands and then apply appropriate actions, and build one or multiple response commands for response. The handler’s processing time should be short, for example less than couple of seconds. This kind of handler requirement is very common ones, for example Google Android’s ANR error. The emitter thread is signaled (POSIX select API) for transmitting a message to the corresponding host.

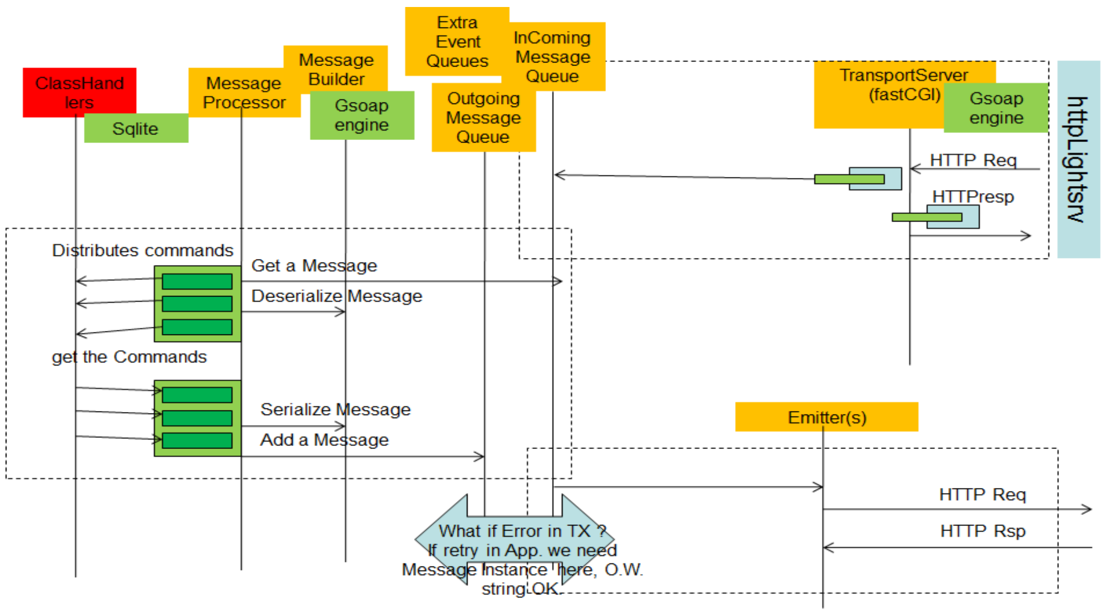

Fig. 4. A typical Message processing flow

XML Engine: Because G2S standards are based on SOAP-protocol, XML processing mechanism is the key component. After technical comparison [8, 9, 10], gSOAP library [5], open source SOAP engine, was adopted due to its well acceptance in commercial product. Despite of its wide acceptance and stability, gSOAP has big room for improvements: unusual class types and members' name especially for XML schema's 'choice', middle of C++ and C style, not modular, single output classes. Even though these limitations, gSOAP fulfill the requirement for G2S message schema binding and run-time operation. Also we provide wrapper functions making it easy for handler developers to write code smoothly and not fully understand the unusual XML binding rules of GSOAP.

Persistent and Device Layer: Most of status of physical and logical devices in EGM is reflected into database tables, especially when it should be persistent on system restart due to operator's commands or failures. Fig. 5 shows the key tables of G2S EGM persistent information and their relationship. Class devices' configuration and some status info, host's subscription and transaction logs, metering data sets are the key records.

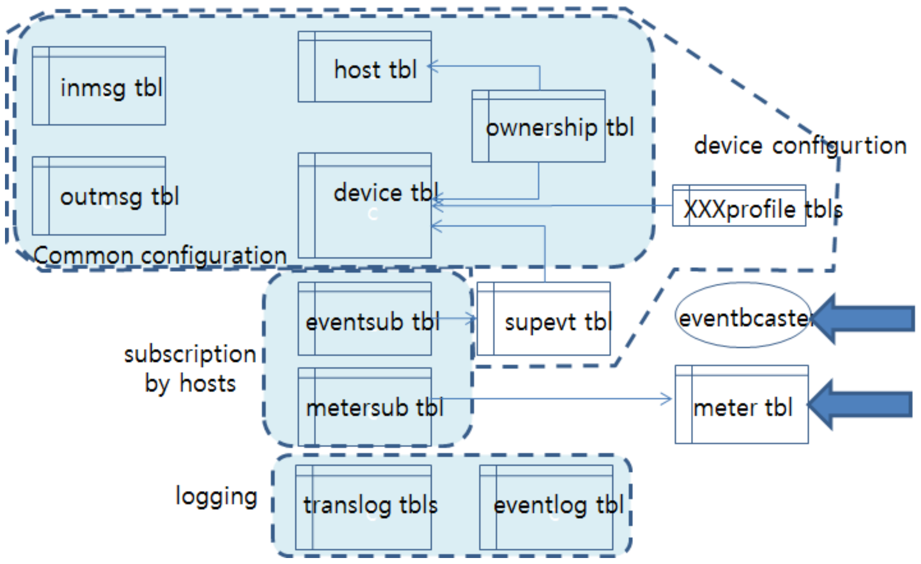

Fig. 5. Database table for G2S EGM side

For embedded database engine, SQLite3.0 [6] is used. SQLite is also open-source for small embedded environment whereas SQLite has much limitation in comparison with other commercial or server-based DBA. However, For independence from specific DB engine and easiness for application programmer, an DAO (data access object) pattern [7] was applied (Fig. 6). One more benefit using DAO is that user does not need to consider if a status or parameter of a device is persistent or not. DAO decide to store or not according to the persistence property of each parameter.

4. Evaluation and Discussion

To test the implemented G2S engine, we integrated the engine into a simple functional EGM system based on GTK library, yet another open source graphic library.

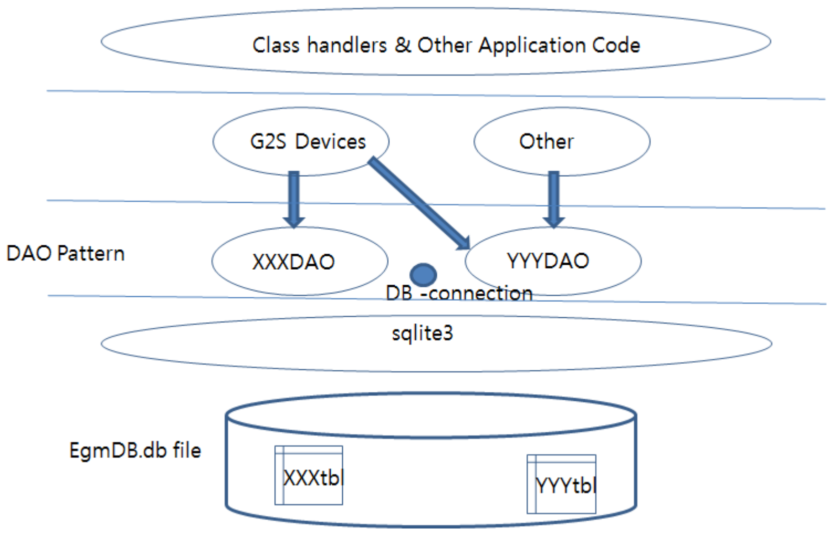

Fig. 6. Device abstraction and DAO pattern

The interoperability test is done with OpenG2S host implementation by IGT. The feature comparison between OpenG2S and our G2S implementation is listed in Table 1. OpenG2S is written in Java, based on the J2EE servlet technology with embedded tomcat server and JAXB for XML processing. The architecture is quite similar to ours. Due to the limited functionality in OpenG2S, furthermore no longer maintained by IGT, only subset of commands and related functions are verified for interoperability: enabling and disabling communication and obtaining device list in Comms class, and enabling, disabling, locking and door-open in cabinet class, a few event subscription and reports eventHandler class, some basic meters and manipulation in gameplay class. However these 5 classes plus handpay class is the core/minimum functionality and considered enough for verifying the architectural issue using the adopted technology and open source library.

Table 1. SAS Protocol Application Functions

Feature	Our G2S in this paper	OpenG2S
Prog. Language	C++	Java
Devel. Tools	Eclipse CDT-gcc (Linux) / VS2012 (Windows)	Eclipse Java- JDK 1.5+
DataBase	SQLite3	JDBC (Oracle, Derby, MySQl)
HTTP	Lighthttpd/gSOAP	Tomcat embedded
XML processing	gSOAP	JAXB
SOAP processing	gSOAP	JAXB
Persistence	DAO patterns	Not fully integrated
Memory requirement	6.8MB for codes (incl. db engines)	Over 30 MB (excluding DB engine)
Memory management	gSOAP and custom	JVM GC
Comments	Good for EGM engine	Good for Host Engine

Table. 1. Comparison of OpenG2S and Our G2S implementation

Acknowledgements. This work is partly supported by the SMA project technology development support project, SMBA, the Republic of Korea (S2100260).

References

[1] 1. J. Kilby, J. Fox, and A. F. Lucas, *Casino Operations Management, 2nd Edition*, John Wiley & Sons, Inc., 2005

[2] *IGT Slot Accounting System*, version 6.01 June 2003

[3] *GSA G2S Message Protocol v2.0.3*, Game to System (2009)

[4] OpenG2S, G2S Open source implementation, http://www.openg2s.org

[5] V. Engelen, A Robert, and K. A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks, *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on. IEEE,* (2002)

[6] Owens, Michael. Embedding an SQL database with SQLite. *Linux Journal* 2003

[7] Crawford, William, and Jonathan Kaplan. *J2EE design patterns*. O'Reilly Media, Inc., 2003

[8] Il-Sun Park and S.-J. Shin, WS Security of XBRL Financial Documents Encoded by SOAP, *International Journal of Security and Its Applications*, vol. 7, no. 4 (2013)

[9] G.- C. Park, S. Kim, N. Yoo, K.-S. Lee and W. S. Jang, An Automatic Conversion HTML/XML to WSDL for Ubiquitous Mobile Services, *International Journal of Multimedia and Ubiquitous Engineering*, vol. 1, no. 1 (2006)

[10] K.-H. Park, An XML Based Communication System for a Ubiquitous Game Simulator, *International Journal of Smart Home,* vol. 7, no. 6 (2013)

Received: August 24, 2014