제34회 한국정보처리학회 추계학술대회 논문집 제17권 2호 (2010. 11)

Developing a Bridge Module to Java Component for SID Simulator[1]

Hasrul Ma'ruf*, Jin Baek Kwon*
*Dept. of Computer Science & Engineering, Sun Moon University
e-mail : hasrulwho@gmail.com, jbkwon@sunmoon.ac.kr

SID 시뮬레이터와 자바 컴포넌트 연동 모듈 개발

하스룰*, 권진백*
선문대학교 컴퓨터공학과

Abstract

Simulation tools help creating a low cost and efficient development of embedded system. SID is an open source simulator software that consists library of components for modelling hardware and software components. A component can be written in C/C++ and Tcl/Tk. Currently, the SID simulation toolkit only provides support for C++ and Tcl/Tk. Tcl/Tk is used to write GUI-based components. However, we have observed that Tcl/Tk components cause slow simulation response because Tcl/Tk is a script language. It is not proper for developing the cutting-edge products with rich graphics. Therefore, in this paper, we suggest Java to a new language for GUI components in SID by developing a bridge module for SID to interworking with Java components.

1. Introduction

In recent years, there has been notable growth in the use and the application of embedded systems. However, the improvements to the design and testing tools have not kept pace with the rapid development of customized hardware parts. The simulation of the target environment enables embedded software developers to analyze and test their software, even in the absence of the physical hardware.

The SID simulator consists of an engine that loads and connects simulated components, based on a configuration file, and runs simulation sessions. Currently, SID only provided GUI simulation of a component by providing bridge to Tcl/Tk scripting technology.

SID also provides a built-in system monitor written in Tcl/Tk, to monitor a running simulation. However, the system monitor is still experimental, and it is somewhat limited and not so user-friendly. The system monitor lists the components in the active virtual platform, showing specific component attributes such as pins, registers, etc. Since our system is based on the Eclipse framework, the system monitor should also be made to an Eclipse plug-in, which must be written in Java. However, SID cannot support components written in Java directly without a Java bridge component[1]. Tcl/Tk with the system monitor also did not give a good performance in GUI simulation[2].

In this paper, we provide an alternative by creating a Java Bridge to SID. Hence, a GUI component for SID, e.g. LCD panel, can be created in Java by employing Java Bridge component along with the newly written component in Java.

2. Related Work

2.1 SID

SID is a framework for building computer system simulations and SID is made for debugging, testing, and verifying embedded software[3]. Specifically, a simulation is comprised of a collection of loosely coupled components. Simulated may range from a CPU's instruction to a large multi-processor embedded systems. SID has the following features:

- Open source framework for building computer system simulations/
- A growing library of components for modeling hardware and software parts, instrumentation, control, and external interfaces.
- Support GDB debugger
- Virtual Target Platform
- Embedded System software testing & verification.

SID defines a small component interface which serves to tightly encapsulate them. Components may be written in C++, C, Tcl or any other language to which the API is bound. However, the SID simulation toolkit only provides support for C++ and Tcl/Tk. The SID is based on C++, therefore C++ is the main language, and for additional language a special component, a bridge, is required. Currently only Tcl/Tk bridge is available.

Typically, components are separately compiled and packaged into shared libraries. A standard run-time

[1] "This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2010-C1090-1031-0004)

linking/loading interface is defined for these components. The package includes a growing library of components for modeling hardware and software parts, instrumentation, control, and external interfaces. During simulation start-up, components are instantiated, interconnected, and configured as necessary to represent some specific system. All these configurations are written in one configuration file, and required before SID runtime, see Fig. 1.

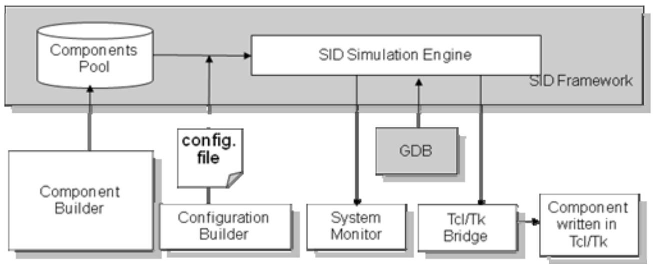


Fig. 1: Current SID Simulator Architecture

2.2 C++ to Java communication

A number of alternative approaches also allow Java applications to interoperate with code written in other language such as C++. Java Native Interface (JNI)[4] is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be calledhttp://en.wikipedia.org/wiki/Java_Native_Interface - cite_note-role-0 by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly. Unfortunately, JNI resides C++ and Java in the same process[4]. Therefore, JNI will create a highly coupled C++ and Java code.

Other than JNI, a Java application may connect to a legacy database through the JDBC^{TM} API[4]. Java application also may take advantage of distributed object technologies such as the Java IDL API. The other alternative is a communication via a TCP/IP connection or through other inter-process communication (IPC) mechanisms.

3. Java Bridge

To get supportability with other parts of SID architecture, the design of Java Bridge has to be based on the current architecture of SID. In addition, the bridge also has to support the flexibility of various components that will be created on Java, see Fig. 2. Through these basis and experiments, we conclude that Java Bridge has to comply with these requirements:

- Enabling C++ to Java communication
- The SID bridge code in C++ part has to be able to run as a C++ shared library. Since the bridge will be called from SID simulation engine as a SID component.
- SID Bridge code has to be able to run Java, and to do procedure call to Java (also get return from the call).
- Java Bridge has to keep a state of objects so that SID can reuse them in the next calls.

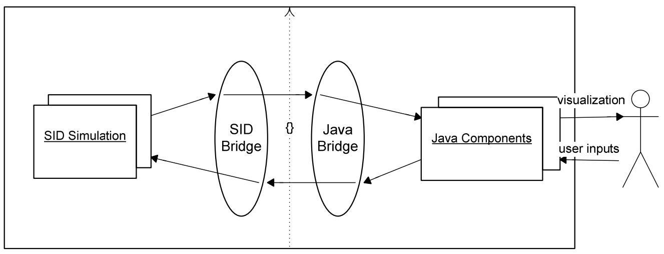


Fig. 2: Overview of SID – Java Bridge Architecture

In our experiment, JNI is unable to be used in C++ shared library, only a single executable C++ code can run JNI. There is no need of database or complex object structure in SID. Therefore, for simplicity, we chose a TCP/IP connection as method of communication between C++ and Java in SID-Java Bridge.

Asynchronous communications, which is a form of input/output processing that permits other call to continue before one call has finished, was also considered as a requirement in the bridge. There is a chance that another call happen before the previous call finished or giving return in SID simulation scenario.

We put a client-server architecture on both bridges to support with this asynchronous requirement. Specifically, we created a single TCP server that can handle multiple requests (procedure calls) and created multiple handlers for each request. As shown in Fig. 3, considering a call as a client in the server, the server will create a single handler for each call, and therefore let the handler (thread) to take care of individual call afterwards.

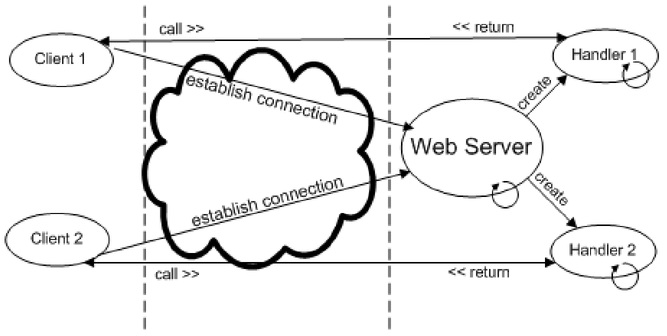


Fig. 3: Client-Server Architecture

Though XML might create some trade off of performance, for the reason of readability and also portability, XML is used as the message format in this bridge. The other reason is XML processors are already available for Java and C++. Therefore, the usage will reduce the error possibility in message processing.

Below, see Fig. 4, is a complete class diagram of Java Brige. To keep the objects, that previously created or modified in Java, we used a hash table that maintains pointers to objects in Java. Hash table uses "key" to enable the remote program to access these objects again another time.

The main code is the JavaBridge class. Basically, it has a thread pool and a receiver thread. A thread pool contains a number of hThread (HandlerThread instances), and always preserve a minimum number of them to prevent a big thread spinoff, which can lead to inefficient memory and CPU usage. HandlerThread itself is a template to create a thread that can serve a call (CALLEE) as well as doing a call (CALLER). The Caller and XMLUtil are classes that used when Java is going to do a remote call to SID. SIDJComponent is an abstract class that represents every SID component in Java, therefore it will be extended as a super class for every SID Component in Java. Respectively, the C++ SID Bridge looks almost exactly the same as the Java, with the restriction of different set of libraries to build the bridge in C++.

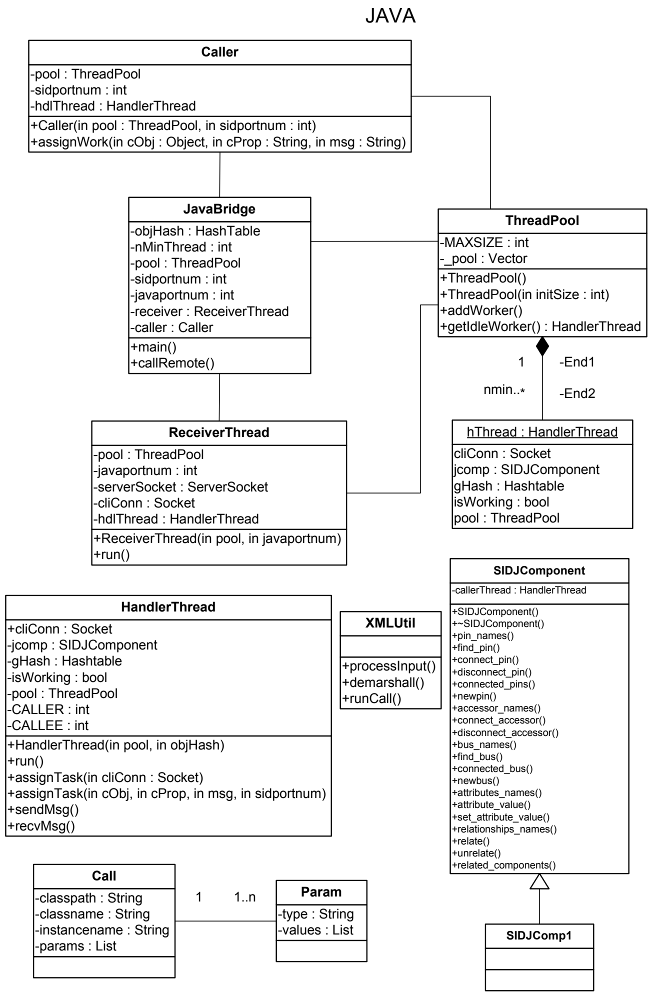
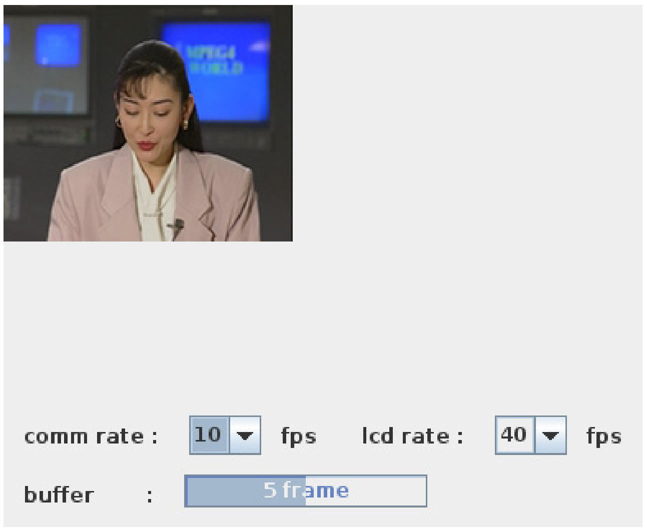


Fig. 4: Java Bridge Class Diagram

4. Experiments

SID is known to run on Linux, Solaris, and Cygwin hosts. Currently, we just tested SID – Java Bridge in Ubuntu to run a LCD Panel. The scenario is to send continuous pixel data (motion images) from SID to Java LCD component. In Ubuntu 8 (Hardy) with Intel Quad CPU @2.33 GHz and RAM 2 GB, the maximum communication rate for QCIF raw bitmap format is 20 fps.


comm rate : 10 fps lcd rate : 40 fps

buffer : 5 frame

Fig. 5: SID – Java Bridge Example Application

5. Conclusion

In this paper we propose a Java Bridge as an alternative to the current Tcl/Tk Bridge. It is designated to be more flexible GUI components with also a better GUI performance than Tcl/Tk components.

References

[1] Hadipurnawan Satria, Baatarbileg Altangerel, Jin Baek Kwon, Jeongbae Lee, "Configurable Virtual Platform Environment using SID Simulator and Eclipse," In *Proc. of Software Technologies for Embedded and Ubiquitous Systems*, SEUS 2007, pp.394-398.

[2] Febiansyah Hidayat, Hadipurnawan Satria, Jin B. Kwon. "Verifying a Virtual Development Environment for Embedded Software". Korea Information Processing Society (KIPS) Fall Conference 2010. Seoul, South Korea

[3] SID. http://sourceware.org/sid/

[4] The Java Native Interface Programmer's Guide and Specification, Sun Microsystems, chap. 1.

"This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) Support program supervised by the NIPA(National IT industry Promotion Agency)" (NIPA-2009-C1090-0902-0020)