Java Bridge Module and Java API for SID Simulator
Hasrul Ma’ruf, Febiansyah Hidayat, Jin Baek Kwon
Department of Computer Science and Engineering
Sun Moon University
Asan-si, Chuncheongnam-do 336-708
SOUTH KOREA
hasrulwho@gmail.com, havban@gmail.com, jbkwon@sunmoon.ac.kr

Abstract: - This is a sample 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 create GUI-based components. However, we have observed that Tcl/Tk components causing slow simulation response. Developing GUI using Tcl/Tk is also inflexible. Thus it is not proper for developing the cutting-edge products with rich graphics. In this work, we introduced the idea of Java as an alternative platform for developing components in SID. We enabled this possibility by developing a bridge module for SID using socket and JNI. SID API in Java is also proposed to ensure the compatibility and simplicity of SID components in Java.

Key-Words: - Embedded system, Simulator, sid, Bridge component, Java, Software

1 Introduction
Embedded systems are getting more rich in features nowadays. Both hardware and software in embedded system has to keep the pace of development before they’re entering the market. Time-to-market remains the single most challenging issue that sophisticated electronic equipment manufacturers need to address and ultimately control in order to remain competitive. Simulation system help developers to improve time-to-market, as system complexity and ever more stringent demands upon overall system quality rapidly increase.

The SID simulator consists of an engine that loads and connects embedded system’s simulated components, based on a configuration file, and runs simulation sessions [1]. Virtual Development Environment for Embedded Software (VDEES) is virtual environment that provides a configuration tool, a code editor for writing simulated components, building tools for binary images, a debugger, and a system monitor for the investigation of the virtual target [2]. VDEES is based on SID and employ Eclipse plugins as its basis.

SID provides a built-in system monitor written in Tcl/Tk to monitor a running simulation. The system monitor lists the components in the active virtual platform, showing specification component attributes such as pins, registers, etc. Since VDEES 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.

Started by the possibility of integrating the system monitor to VDEES, we proposed the idea of using Java as an alternative platform for developing components in SID. In addition to the first motivation, current SID components can only be written in C/C++ and Tcl/Tk. Our tests showed that SID components that use Tcl/Tk, are running very slow. For each touch input require few seconds for evaluation and 1 second for display refresh[3]. Moreover, GUI in Tcl/Tk is considered not feasible to develop nor user-friendly, compared to the other language such as Java or .NET.

In our previous work, we briefly presented the main idea of Java bridge and its prototype system using socket [4]. In this paper, we present the idea of Java bridge in 2 alternatives, using socket and JNI. Here, we also propose that every newly written component in Java has to follow a certain API, to ensure the new component compatibility with SID. As Java supports Object Oriented design pattern, the API will also be able to help SID component’s development in Java becoming faster and easier.

2 Related Works
Our work is based on SID Simulator and VDEES. Our goal is to create an alternative to Tcl/Tk bridge, which is Java. This chapter will give a brief explanation about some basis of this work.

2.1 SID Simulator
SID is a framework for building computer system simulations and SID is made for debugging, testing, and verifying embedded software. 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. SID models a component’s communication through buses and pins. SID also support the higher level communication by using attributes and relationships. 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.

2.1 VDEES

VDEES is an integrated development environment that we built on top of Eclipse and SID, see Fig. 1. It uses Eclipse and employed its plugin as the engine for easier SID development. So far, there are four plugins that exist in VDEES. There are VDEES Binary Image Builder, VDEES Configuration Builder, and VDEES Custom Component Wizard, and VDEES monitor [5]. We created the Java bridge component for SID using VDEES.

2.2 C++ to Java Communication

SID engine and native SID components are written in C++. Therefore we need a protocol that can enable communication from C++ to Java, and vice-versa, if we want to create a working SID component in Java.

A number of alternative approaches also allow Java applications to interoperate with code written in other language such as C++. Java Native Interface (JNI) is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++, and assembly.

Other than JNI, a Java application may connect to a legacy database through the JDBC API. 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 socket connection or through other inter-process communication (IPC) mechanisms.

3 Java Bridge Module

To get supportability with other parts of SID architecture, the design of Java Bridge has to be based on the current architecture of SID. Through this basis, we conclude that Java Bridge has to comply with these requirements:

- Java Bridge Module is a native (C++) SID component. The bridge module will be located in components pool as a shared library and loaded by SID engine in the runtime, see Fig. 2.
- It enables C++ to Java two-way communication
- Java Bridge module has to be transparent to SID calls. Thus, Java Component in SID can be treated like other SID component.
- Able to maintain the Java component state, and reuse it for the later.

The bridge module is composed of C++ SID component and the foreign language. First, SID engine loads bridge module, and then lists for foreign components available in that bridge, i.e. in Tcl/Tk or Java bridge. After getting the list of available components from the bridge, the
Java Bridge module can do various call to/from the foreign component through interpretation and communication by any means. Below, Fig. 3 is an example of how SID, bridge, foreign component, and user could interact to each others.

Fig. 3 SID, Bridge, Java Component, and user interactions

3.1 Java Bridge Component using Socket

In our first experiment, JNI was unable to be used in C++ shared library. Therefore, it cannot be used as the bridge communication mechanism. So we chose a TCP/IP socket connection as method of communication between C++ and Java in SID- Java Bridge. Using socket enables us to assume the Java component to be located on any host.

At first, we put single-server-multiple-client architecture on both sides, C++ and Java, to support asynchronous communication. Specifically, we implemented single TCP server on Java and C++ that can handle multiple requests (procedure calls) and created multiple handlers for each request. Later it turned out that using thread is resulting slow return to SID communication, even slower than Tcl/Tk. Even so, it was certainly fast on no return, only sending communication. Unfortunately, the no return scenario on SID is only limited to pin communication. Currently, our proposed socket architecture are single threaded message sending and message handling on both sides.

We proposed XML as a message format in calls and returns. Though XML might create some performance trade off, for the reason of readability and also portability, we chose this syntax standard. The other reason is that XML processors are available for Java and C++.

SID Java bridge is not only be able to create a Java component, but also has to be able to communicate with the created component afterwards. Thus it is important to make the bridge keeping a pointer to the Java object (component). In socket based solution, we used Java object ID, kept the id and the object in the hashtable, and then sent the id to the C++ side, see Fig. 4. Later, if SID (C++) needs to communicate with the component, it will use the object id as the key to interact with the right object.

In other direction, we used the name of pin, bus, etc. as a key for calling from Java component to other components. We used bijection lookup table that can create, if has not exist, from string a pointer, or retrieve a pointer, if exists, of pin or bus in SID. The lookup table can also be used in the reverse direction, getting the name from pointer.

3.2 Java Bridge Component using JNI

In contrast to socket-based communication, JNI enables both C++ and Java to keep pointer and to do call to foreign object. JNI supports remote object creation. For example C++ can create Java object, along with its object pointer. Later, C++ can call the remote method of object in Java and vice versa.

For C++ to Java communication, we used GIWS. Giws is a free software which greatly simplifies the call of Java method/object from C/C++ [7]. It has a straight-forward code generator that let us use a simple XML to create C++ class declaration and implementation that manages JNI calls, conversion of the data/objects. Having C++ classes for every Java classes that we used in C++, we could create or call the Java object in natural way as we call it in Java.

For Java to C++ communication, we used registerNatives function. If GIWS uses XML to create the interpretation between different languages, registerNatives function takes method names in C++ and Java along with its method signatures as its parameters. The function also allows us to register a batch of native functions that can be called from Java.

4 SID Java API

An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. This section explains about the API used in Java in order to create proper SID component. Thus the component is able to be called as or calling SID service.

To make the API, we created abstract classes and interfaces, see Fig. 5. The goal of these classes and interface are to ensure that every new component created in Java is following strict SID rules and will always be compatible with SID call, either it is in a pin, bus, attribute, or relationship call.
SIDJComponent is an interface that ensures all SID communication functions are implemented in the Java Component. It contains function template for pins, buses, attributes, and relationship. Pin and Bus are abstract classes that resemble the Pin and Bus in SID. Pin has an abstract driven method that will drive integer value from a pin to another. Bus has abstract write and read methods that enable a bus to write and read data from certain memory address. OutputPin and OutputBus are used to drive, write, or read data to/from Tcl/Tk or C++ component. SIDAttribute is to let the component has attributes with a name, value, and category.

SimpleComponent is an abstract class that has default implementations for Pin, Bus, and Attribute communication. It is intended to make creating SID component easier and faster. Meanwhile, RelatedComponent is an abstract class that gives an abstraction for related component that is used when we use relate and unrelated in SID communication. The same SID Java API is used on both socket-based and JNI-based SID bridge communication.

A concrete Java component would extend SimpleComponent class. Once extending, the component is not only ensured that it has all SID component’s functions but also has some of them already-implemented. Some of already implemented are to list, to find, to connect and disconnect pins, buses, and attributes on the fly. The concrete Java component might be required to implement the concrete Bus or Pin if it needs one. The concrete Bus or Pin implemented in concrete SID Java component will have the instructions of what to do when a pin or a bus sends or retrieves data. Below is an example class diagram of concrete Java Component along with its pins, see Fig. 6

5 Conclusion
In this paper we proposed a Java Bridge as an alternative to the current Tcl/Tk Bridge in SID. The bridge between SID and Java component would be built by using JNI and socket. We also proposed Java API to ensure compatibility of every SID Component in Java. The API is also intended to make SID component’s development in Java easier and faster. Both of them are intended to give more flexible GUI components with also a better GUI performance than Tcl/Tk components.

References: