aicas news
realtime

Spring 2004

News for developers of
critical software applications

Editorial

Dear Reader,

Here we are again with a new collection of articles with a technical focus on dynamic class loading.

As the days lengthen this spring, so does the list of our offerings for your embedded and realtime programming needs. We have strengthened our cooperation with Software and Hardware partners to better serve your embedded needs. For example, Objective Interface System's ORBexpress can provide high performance network access between legacy applications and your state-of-the-art JamaicaVM application.

We wish you a pleasant read and please do not hesitate to let us know if there is any subject you would like to see addressed in these pages.

Sincerely,

Dr. James J. Hunt

BMW Scientific Award for aicas' Realtime Technology

BMW awards its most prestigious prize for aicas' realtime technology.

More than 200 graduates competed for the BMW group 2003 Scientific Award held under the motto "Passion for Innovation". The award is endowed with 70.000€, making it the highest international prize for graduate and Doctor theses. Among the six laureates, the dissertation by Dr. Fridtjof Siebert, director of Development at aicas GmbH, won second prize in the category doctoral dissertations.

With his work, Dr. Siebert created the technological basis for the application of modern software development tools based on Java in the ever complexer area of embedded systems. Dr. Siebert developed the required key algorithm for memory management, known as the garbage collector. The success of this new technology has been demonstrated by the success of aicas' realtime Java virtual machine JamaicaVM.

Scientific Award

During the award ceremony with representatives from industry, academia, and research, Prof. Dr. Göschel, development board member at BMW AG, said: "One who thinks and acts in networks advances innovation more rapidly and creates the basis for new opportunities as well as preserving existing employment and consequently our collective future. There will be no growth and no employment tomorrow without education and innovation today.". *(aw)*

News

JamaicaVM in Japan

24. January 2004: Microsystems Co. Ltd. Japan and aicas GmbH today joined a sales cooperation for the JamaicaVM realtime Java solution. Microsystems is well known in the Japanese realtime sector. "Japan is one of our strategic markets for our embedded realtime Java technology", opined Andy Walter, Director of Sales at aicas. "This cooperation helps us to improve our local customer service. Both technical and sales requests can now be handled by native Japanese engineers." For further information, please contact Izuru Oki, microsystems Co., Ltd., *www.microsystems-ltd.com.*

Java Hotline

The Forschungszentrum Informatik Karlsruhe (FZI) offers expert consulting via the internet free of charge. For details on the Java Hotline online see *www.fzi.de/javaexperte.*

New version of RCE and VRCE

aicas has released a new beta version of RCE 4.0 and VRCE 4.0. The new API version now contains a streaming interface for better performance. With the new functions, API developers can send files directly over network sockets. This simplifies the use of the API and increases the performance of the resulting application. The graphical user interface of VRCE has also been improved. Among other changes, the new difference dialogue makes it easier and faster to see changes between different versions of a file. The final release will be available in March 2004.

Dynamic Programming with user defined ClassLoaders

A major strength of Java is the ability to load code dynamically. Taking full advantage of this facility requires writing a user defined *ClassLoader.*

Dynamic loading off code, which is enabled in Java through user defined class loaders, brings an enormous flexibility to the development of software. The dynamic loading of classes via a dedicated *ClassLoader* is much more than merely loading classes via the function *Class.forName().* Particularly, the definition of a user *ClassLoader* enables three powerful techniques

- One can load classes from a source that can be chosen freely (e.g., a network address, a flash card, ...).
- A new name space can be defined, i.e., all classes loaded via a *ClassLoader* may use equal names as classes loaded via a different *ClassLoader* without causing conflicts.
- Finally, the memory of classes no longer in use can be automatically reclaimed. These classes will be removed from the system automatically as soon as the *ClassLoader* which loaded these classes is no longer referenced. This process is called class file garbage collection.

With these features, applications can be developed which are able to add arbitrary code at runtime. Furthermore, independent modules can be created, that will be executed separately from one another within the same Java environment.

The class file garbage collection eventually makes sure that the memory required for dynamically loaded code will be reclaimed automatically. Even in a long running application new code can be loaded frequently via a user defined *ClassLoader*, .

Example

It is fairly straightforward to create a user defined *ClassLoader.* This will be illustrated here with a small example. Figures 1 and 2 show the source of the classes *Receiver* and *Sender*, a small network application. *Receiver* waits for a network connection, while *Sender* creates this connection.

```
import java.io.*;
import java.net.*;
import java.util.Hashtable;
public class Receiver {
  static final int port = Integer.parseInt(System.getProperty("PORT","1024"));
  public static void main(String[] args) {
    try {
      ServerSocket server = new ServerSocket(port);
      System.out.println("Waiting on server socket port "+port+" ...");
      boolean done;
      do {
        Socket s = server.accept();            /* wait for next connection */
        DataInputStream in =     DataInputStream(s.getInputStream());
        String msg = in.readUTF();             /* read message type        */
        done = msg.equals("STOP");             /* handle message "STOP"    */
        if (msg.equals("UPLOAD")) {            /* handle message "UPLOAD"  */
          String    main    = in.readUTF();    /* main class name          */
          Hashtable classes = new Hashtable();
          for(int i=in.readInt(); i>0; i--) {  /* receive classes          */
            String name = in.readUTF();
            byte[] data = new byte[in.readInt()]; in.readFully(data);
            classes.put(name, data);           /* add name/data to hashtab */
          }
          ClassLoader l = new DataClassLoader(classes);
          Class       c = l.loadClass(main);   /* load main class          */
          Object      o = c.newInstance();     /* call constructor         */
        }
        in.close(); s.close();
      } while (!done);
      server.close();
    } catch (Exception e) { e.printStackTrace(); }
  }

  static class DataClassLoader extends ClassLoader {  /* user class loader */
    Hashtable classes;              /* hashtable of names/data for classes */
    DataClassLoader(Hashtable classes) { this.classes = classes; }
    /* overwrite findClass() to search for class in hashtable */
    public Class findClass(String name) throws ClassNotFoundException {
      byte[] data = (byte[]) classes.get(name);
      if (data==null) throw new ClassNotFoundException(name);
      return defineClass(name, data, 0, data.length);
    }
  }
}
```

Figure 1: Class Receiver load classes via a user defined ClassLoader.

Via static fields *host* and *port*, the name and port of the target systems that runs *Receiver* is set. Port 1024 will be used by default and the default host is "localhost". Thus, *Sender* and *Receiver* can run on the same system for test purposes. Other values for host and port can be set via properties *HOST* and *PORT*.

Sender and *Receiver* know two commands: *STOP* to terminate the application and *UPLOAD* to load new classes from *Sender* to *Receiver*. These classes will be loaded via a user defined *ClassLoader* and then executed.

As an example, load the class *Demo.class* via the network and execute it by *Receiver*. For this, one needs to first start *Receiver*, e.g., with this command:

```
> jamaicavm Receiver
Waiting on server socket port 1024 ...
```

In another terminal window, one can now transfer class *Demo.class* with the *Sender* to *Receiver*. Figure 3 illustrates the Java source code of *Demo*.

```
> jamaicavm Sender Demo
```

The transferred class *Demo.class* will now be loaded and executed on the *Receiver* side:

```
Hello dynamically loaded World!
```

How does it work

To develop such a network application, one first has to define a network protocol to be used by Sender and *Receiver*. In this example, the protocol consists of two commands *STOP* and *UPLOAD as* shown in Figure 4.

To upload classes to *Receiver*, Sender generates the command *UPLOAD*. This command contains the name of the main class and the names and class file data of every class that is to be loaded. *Receiver* receives this data corre-

```
import java.io.*;
import java.net.*;

public class Sender {

  static final String host = System.getProperty("HOST","localhost");
  static final int    port = Integer.parseInt(System.getProperty("PORT","1024"));

  public static void main(String[] args) {
    try {
      if (args.length==0) {
        System.out.println("usage: jamaicavm Sender STOP | <class> {<class>}");
      } else {
        Socket           s = new Socket(host, port);
        DataOutputStream o = new DataOutputStream(s.getOutputStream());
        if (args[0].equals("STOP")) {
          o.writeUTF("STOP");                  /* send STOP command         */
        } else {
          o.writeUTF("UPLOAD");                /* send UPLOAD command       */
          o.writeUTF(args[0]);                 /* main class name           */
          o.writeInt(args.length);             /* number of classes to send */
          for(int i=0; i<args.length; i++) {
            byte[] data  = loadFile(args[i]+".class");
            o.writeUTF(args[i]);               /* class name                */
            o.writeInt(data.length);           /* classfile data length     */
            o.write(data,0,data.length);       /* classfile data            */
          }
        }
        o.close();
        s.close();
      }
    } catch (IOException e) { e.printStackTrace(); }
  }

  /* read file name into a byte array */
  public static byte[] loadFile(String name) throws IOException {
    File           f = new File(name);
    DataInputStream i = new DataInputStream(new FileInputStream(f));
    byte[]          d = new byte[(int) f.length()];
    i.readFully(d);
    i.close();
    return d;
  }
}
```

Figure 2: Class Sender sends classes dynamically to Receiver

```
public class Demo  {
 public Demo() {
   System.out.println("Hello dynamically loaded World!");
 }
}
```

Figure 3: Dynamically loaded class Demo

Command	Value	Type
Stop:	"STOP"	String
Upload:	"UPLOAD"	String
	main class	String
	# of classes n	int
	class 1: name	String
	len	int
	data	byte[len]
	class 2: name	String
	len	int
	data	byte[len]
	⋮	
	class n: name	String
	len	int
	data	byte[len]

Figure 4: Protocol used by ClassLoader Example Receiver/Sender

spondingly and saves the loaded classes in a Hashtable, using the class names as keys and the corresponding class file data as values.

This Hashtable can then be used by the user defined *DataClassLoader* in *Receiver* to load this classes into the system. The *ClassLoader* implements *findClass()*, which will be called by the VM to load new classes from the new *ClassLoader*. In the example, *findClass()* searches for the requested class data in the hashtable and generates a class instance from this data via a call to *defineClass()*.

After the creation of an instance of *DataClassLoader* with the data obtained from the *UPLOAD* command, the main class can be loaded via a call to loadClass() in this *ClassLoader*. Via a call to *newInstance()*, an instance of this class is created and its constructor is executed.

Caveats

Generally, it is not sufficient to transfer a single class to the target system. Often, a complex component that consists of several classes will be required in the target system. For this, the *ClassLoader* must load all classes that are referenced by the main class that is transferred. In the example, this is facilitated by giving a list of classes that are to be transferred by *Sender* as an argument, where the first argument is the main class.

The examples in Figures 1 and 2 have no error handling code for the Exceptions that may occur during the network functions or while loading or instantiating classes. A complete implementation requires adequate error handling code and finally clauses to ensure that all resources will be closed and freed cleanly in case of an exception. Error handling and cleanup code has been left out in the examples shown here to focus on the main points.

Outlook

The next issue of the aicas news will present

Realtime Garbage Collection
: The internals of how JamaicaVM's realtime garbage collector works.

(fs)

Trainings & Presentations

Forschungszentrum Informatik
: The FZI Forschungszentrum Informatik in Karlsruhe has extended its offer. The established trainings were extended with new subjects such as realtime programming, automation and integration of legacy systems into Java environments. The trainings currently on offer are listed online at *www.fzi.de/ajc/*.

Java in Wireless and Embedded Environments, Munich
: *12.-16. July 2004.* A 5-day training course is being offered in cooperation with RP-Cube and Microconsult covering Java editions, configurations, and profiles; requirements for embedded platforms; Java environments and Java applications; wireless and embedded scenarios; realtime OS, VM, and tools; Java technology and realtime; performance and code size.
Contact: *info@aicas.com*

Events

RTS Embedded Systems, Paris
30. March - 1. April 2004. The real-time solutions and embedded systems show., Paris Expo - Porte de Versailles - Hall 3. aicas presents its realtime Java technologies JamaicaVM and Aero-VM. Visit us at booth N25.

SSTC, Salt Lake City, USA
19.-22. April 2004. The Software Technology Conference. Visit aicas at this "premier systems and software technology conference sponsored by the U.S. Department of Defense."

Electronica, Munich
9.-12. November 2004. The bi-annual electronica show will take place in conjunction with the Embedded in Munich. Of course, aicas will again be present.

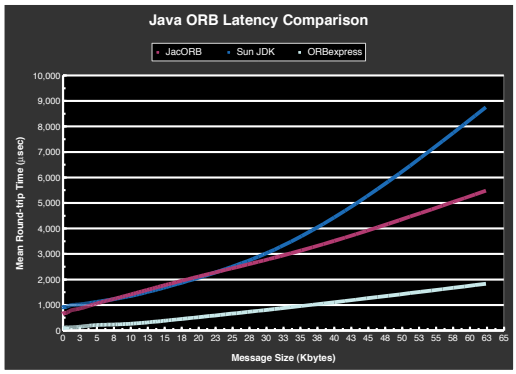
Leading CORBA Technology ORBexpress

aicas announces collaboration with Objective Interface Systems to deliver ORBexpress ST for Java to users of JamaicaVM.

ORBexpress is the leading CORBA technology for embedded, high performance and realtime systems. ORBexpress is the foundation for many mission-critical systems in defence, aerospace, telecom, process control, transportation, medical imaging, and industrial automation. Objective Interface Systems is a pioneer in advanced realtime CORBA technology and is renowned for creating fast, high-quality communications products for embedded systems.

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA facilitates the development of language and location independent interfaces for distributed objects.

In 2003, Objective Interface Systems released ORBexpress ST for Java, a fast, scalable implementation of the CORBA standard for Java. This high-performance ORB lets Java developers reduce risk by building distributed heterogeneous applications with full interoperability and unified support from a single vendor.



ORBexpress performance comparison

Objective Interface's ORBexpress ST for Java actually outperforms many other ORB vendors' C++ ORBs. Lockheed Martin's Advanced Technology Labs benchmarked a number of Java ORBs (see *www.atl.external.lmco.com/projects/QoS/*). As seen in the graph, these independent tests demonstrate that ORBexpress is significantly faster than other Java ORBs. ORBexpress is over 8 times faster than Sun's JDK ORB for small data sizes. Thus, ORBexpress, together with the JamaicaVM, is the ideal solution for your distributed Java requirements. *(jjh)*

JamaicaVM on NetSilicon's NS9750

Jamaica for NetSilicon's latest NET+ARM processor.

Based on the ARM 926EJ-S, ARM's most powerful ARM 9 core, the NS9750 provides the highest level of performance and integration available for embedded device networking. The NS9750 focuses on flexible connectivity solutions by integrating a broad set of industry-standard peripherals, such as 10/100 Ethernet, PCI, USB (host or device), I2C, 1284, serial ports, GPIO, and a cost-effective LCD controller. Featuring DSP instructions, a Java byte code accelerator, and an MMU, the NS9750 is a high-performance 32-bit processor that operates at speeds up to 200MHz. It provides full duplex 10/100Base-T Ethernet, with more than enough additional processing performance and bandwidth to handle even the most sophisticated embedded applications.

JamaicaVM is a perfect addition to NetSilicon's development environment, enabling the static or dynamic loading of realtime object oriented applications code to network attached embedded devices. *(jjh)*

Services via OSGI

Deploying dynamic features on target systems using Java.

Java's ability to load classes facilitates the dynamic deployment of additional services. The Open Service Gateway Initiative (OSGi) defines a standard framework for such services in a networked environment. Members from industries such as home electronics, telecommunication, automotive and software have expanded OSGi's focus to more generally networked environments. The OSGi framework enables software service vendors to concentrate on their services without considering hardware dependencies. Deployable services are packaged as a Java Archive (JAR) with a manifest file containing information about the service.

Please contact aicas for available OSGi solutions for JamaicaVM. *(ip)*

Contact

Editors:
Dr. James J. Hunt (jjh), Ingo Prötel (ip), Dr. Torsten Rupp (tr), Roman Schnider (rs), Dr. Fridtjof Siebert (fs), Andy Walter (aw)

aicas GmbH
Hoepfner Burg
Haid-und-Neu-Str. 18
76131 Karlsruhe
Germany
tel +49.721.663.968-0
fax +49.721.663.968-99
email info@aicas.com
web www.aicas.com