USING DIFFERENT TYPES OF DATA IN THE OPEN-SOURCE BUSINESS APPLICATIONS

Danut - Octavian SIMION, PhD.
Economic Informatics Department
Athenaeum University, Bucharest, Romania
danut_so@yahoo.com

Cosmin OLTEANU, PhD.
"Athenaeum" University of Bucharest
contact@olteanucosmin.ro

Abstract:
The paper presents the study cases for using different types of data in the open-source business applications. These types of applications may use various sets of data that come from XML files, DBMS-s, simple files or results from other business applications. A different way of using these data is to pack them through Java methods and unpack them through other Java methods that are persistent in the same package. The data will be encoded and prepared in the Java classes to be used be open-source applications. The methods used for packing and unpacking data is addressed to a client-server communication that is the most common way to transfer data from server side to client side through applications, even if those are Web based or other types. The Java package that perform these tasks is custom and easy to use be every type of open-source business applications and may integrate it into the source code of the business logic. Using Java programming language to develop this package makes possible the integration of data into any type of open-source applications and even to build a separate business service for communications with other business applications.

Keywords*: Java pack/unpack methods, Java objects integration, XML files, open-source Java components, UML diagram, business applications.*

1. Introduction

Different types of applications need to use data that may come from various sources like XML files, DBMS-s, simple files or results from other business applications. A good strategy is to pack and unpack data through a custom Java package that can integrate different types of data and make them available to an open-source application. Also through this package, the

data will be useful through an application service that can be easily integrated in the source code a design of a business application. The data will be encoded and kept persistent through Java methods that are encapsulated in this package, so the applications can use them in an easy way after they are unpacked.

The Java programming language can dissociate the working processes of the interface designers, web designers, java programmers, the business consultants and the integration software architects, and so the business logic of the applications can be separated from the design work and different parts of applications could be integrated more easily and the duration of completing these tasks is shorter. The open-source Java components will be used by clients to have access at different components that are structured in java classes that are customized for each type of Web Applications and contain description for actions like orders, supply, purchasing, delivery, etc. An important role to develop these types of open-source Java components have the frameworks used to integrate Java classes into Web Applications and to extend the existing components that will improve performance and facilities of existing tools used to access data from different DBMS-s or XML files. The data can be transmitted through network very easily because is packed for IP packages and ready to use after it is unpacked in a open-source business application [3], [4].

2. The Java programming language for open-source components

Open source describes practices in production and development that promote access to the end product's source materials. Some consider open source a philosophy, others consider it a pragmatic methodology. Open source software represents software whose source code is published and made available to the public, enabling anyone to copy, modify and redistribute the source code without paying royalties or fees. Open source code evolves through community cooperation. These communities are composed of individual programmers as well as very large companies [1].

Developers can design, generate, and visualize their code with UML, Java and database diagrams. An advanced coding environment and declarative and visual editors facilitate faster code development. Integrated testing, profiling, and code auditing features ensure the quality of the applications. Interfaces with versioning, build, and deployment tools enable packaging and deployment of the applications [2].

Different frameworks that use Java programming language includes a database modeler allowing forward and reverse modeling as well as reporting for relational databases structure. These frameworks include a visual HTML editor for Web page design with simple drag and drop

gestures. A JavaScript editor with code insight and refractory capabilities along with a JavaScript debugger simplify adding dynamic behaviors to Web pages. A CSS editor helps with achieving the right design for Web pages, and an HTTP Monitor tracks the network traffic between the browser and server. For Java Web developers Java frameworks, extends the visual HTML editor to support JSP and JSF visual editing, as well as providing code insight while coding these Java artifacts. These frameworks also include a JSF and Struts diagram tool enabling visual design of page flows [4].

Java programming language permits building business applications, in special Web applications, regardless of system operations and can access different types of data that can be structured in databases or semi-structured in XML files. Java is a programming language object oriented that offers many possibilities in the domain of accessing data, is very adaptive, is open-source, can be used with any type of operating system, it can be expand through the creation of new classes, new packages, new libraries by a large community of programmers, it extends the capacities and possibilities of Web servers and also is the most popular and robust programming language for building business applications. Java in business applications through specific drives can access data from databases that are managed by different RDBMS like MySQL, Microsoft SQL Server, Oracle, Microsoft Access, etc. and so these applications can be built for dedicated Web servers and the data becomes available for a large group of end users that are located in different places.

3. Using packing/unpacking data in open-source business applications

Java programming language offers many possibilities to connect at different types of data through a built in driver called JDBC (Java Database Connectivity) which offers access to databases from different vendors and can manipulate the data from the tables through SQL statements, which are arguments to methods in Java interfaces. Java provides database programmer's features such as easy object to relational mapping, database independence, distributed computing, etc. The results from the database are returned as Java objects, and access problems are thrown as exceptions, so these objects can be used in many types of applications [3].

The most efficient way to customize applications is to build the own specific Java package that is able to pack/unpack data and use them in a persistent way through specific methods that are encapsulated in this package. My own solution is to build a custom Java class that pack different types of data from the server side and unpack them to the client side. The

packaging process includes encoding for 32 bits or 64 bits and so the data is prepared for IP packages and ready to for transition over a network.

The source code for the custom package is:

```
package pack_custom_data;

public class pack_data1 {

	static long packf(float f)
	{
	long p;
	int sign;
	if (f < 0) { sign = 1; f = -f; }
	else { sign = 0; }
	p = (long) (((((long)f)&0x7fff)<<16) | (sign<<31)); //
integer part and sign
	p |= (long)(((f - (int)f) * 65536.0f))&0xffff; // fraction
	return p;
	}

	static float npackf(long p)
	{
	float f = ((p>>16)&0x7fff); // integer part
	f += (p&0xffff) / 65536.0f; // fraction
	if (((p>>31)&0x1) == 0x1) { f = -f; }
	return f;
	}
```

The main method:

```
	public static void main(String[] args) {

		float f = (float) 47893.9981415, f2;
		long nf;
		nf = packf(f);
		f2 = npackf(nf);
		System.out.println(f); // 47893.9981415
		System.out.println (nf); // 0x0007682F
		System.out.println (f2); // 47893.9981415

	  }

   }
```

The source code for the server side packing and encoding data is:

```
long pack11(long f, int bits, int expbt)
```

```
        {
        long fn;
        int st;
        long sign, exp, sgnf;
        int sgnfbits = bits - expbt - 1; // -1 for sign bit
        if (f == 0.0) return 0; // get this special case out of the
way
        // check sign and begin normalization
        if (f < 0) { sign = 1; fn = -f; }
        else { sign = 0; fn = f; }
        // get the normalized form of f and track the exponent
        st = 0;
        while(fn >= 2.0) { fn /= 2.0; st++; }
        while(fn < 1.0) { fn *= 2.0; st--; }
        fn = (long) (fn - 1.0);

        sgnf = (long) (fn * ((1L<<sgnfbits) + 0.5f));

        exp = st + ((1<<(expbt-1)) - 1); // sh + bs
        // return the final answer
        return (sign<<(bits-1)) | (exp<<(bits-expbt-1)) | sgnf;
        }
        long unpack11(long i, int bits, int expbt)
        {
        long result;
        long sh;
        int bs;
        int sgnfbits = bits - expbt - 1; // -1 for sign bit
        if (i == 0) return (long) 0.0;

        result /= (1L<<sgnfbits); // convert back to float
        result += 1.0f; // add the one back on
        // deal with the exponent
        bs = (1<<(expbt-1)) - 1;
        sh = ((i>>sgnfbits)&((1L<<expbt)-1)) - bs;
        while(sh > 0) { result *= 2.0; sh--; }
        while(sh < 0) { result /= 2.0; sh++; }
        if ((i>>(bits-1))&1)
               result *= -1.0;
        else
               result *= 1.0;
        return result;
        }
```

The main method:

```
public static void main(String[] args) {


               float f = (float) 47893.9981415, f2;
               double d = 47893.998141558979323, d2;
               pack11 fi;
               upack11 di;
               fi = pack11(f);
               f2 = upack11(fi);
               di = (pack_one1.upack11) pack11((float) d);
```

```
            d2 = upack11((pack_one1.pack11) di);
            System.out.println(f);
            System.out.printf("float encoded: 0x%08",  "\n", fi);
            System.out.printf("float after : %.7f\n\n", f2);
            System.out.printf("double before : %.20lf\n", d);
            System.out.printf("double encoded: 0x%016" , "\n",
di);
            System.out.printf("double after : %.20lf\n", d2);

      }
```

The source code for the custom package is:

```
package pack_unpack_custom_data;

public class pack_data2 {

      float f32;
      double f64;
      /*
      ** pck16() -- store a 16-bit int into a char buffer
      */
      void pck16(char buf, int i)
      {
      buf = (char) (buf+(char) (i>>8)); buf = (char) (buf+(char)
(i+1));
      }
      /*
      ** pck32() -- store a 32-bit int into a char buffer  */

      void pck32(char buf, long i)
      {
      buf = (char) (buf+i>>24); buf = (char) (buf+i>>16);
      buf = (char) (buf+i>>8); buf = (char) (buf+i);
      }
      /*
      ** unpck16() -- unpack a 16-bit int from a char buffer
      */

      int unpck16(char[] buf)
      {
      return (buf[0]<<8) | buf[1];
      }
      /*
      ** unpck32() -- unpack a 32-bit int from a char buffer
      */

      long unpck32(char[] buf)
      {
      return (buf[0]<<24) | (buf[1]<<16) | (buf[2]<<8) | buf[3];
      }
       /*
      ** pack() -- store data dictated by the format string in the
buffer
      **
      ** h - 16-bit l - 32-bit
      ** c - 8-bit char f - float, 32-bit
```

```
** s - string (16-bit length is automatically prepended)
*/
int pack(char buf, char format)
{
va_list ap;
int16 h;
int32 l;
int8 c;
f32 f;
int val;
double val1;
char s;
int32 size = 0, len;
va_start(ap, format);
for(; format != '\0'; format++) {
switch(format) {
case 'h': // 16-bit
size += 2;
h = (int16)va_arg(ap, val);
pck16(buf, h);
buf += 2;
break;
case 'l': // 32-bit
size += 4;
l = va_arg(ap, int32);
pck32(buf, l);
buf += 4;
break;
case 'c': // 8-bit
size += 1;
c = (int8)va_arg(ap, val);
buf++ = (c>>0)&0xff;
break;
case 'f': // float
size += 4;
f = (f32)va_arg(ap, val1);
l = pck32(f); // convert to 32 bit
pck32(buf, l);
buf += 4;
break;
case 's': // string
s = va_arg(ap, valc);
len = strlen(s);
size += len + 2;
pck16(buf, len);
buf += 2;
memcpy(buf, s, len);
buf += len;
break;
}
}
va_end(ap);
return size;
}
/*
** unpack() -- unpack data dictated by the format string */
```

```
        void unpack(char buf, char format)
        {
        va_list ap;
        int16 h;
        int32 l;
        int32 pf;
        int8 c;
        f32 f;
        char valc;
        char s;
        int32 len, count, maxstrlen=0;
        va_start(ap, format);
        for(; format != '\0'; format++) {
        switch(format) {
        case 'h': // 16-bit
        h = va_arg(ap, int16);
        h = unpck16(buf);
        buf += 2;
        break;
        case 'l': // 32-bit
        l = va_arg(ap, int32);
        l = unpck32(buf);
        buf += 4;
        break;
        case 'c': // 8-bit
        c = va_arg(ap, int8);
        c = buf++;
        break;
        case 'f': // float
        f = va_arg(ap, f32);
        pf = unpck32(buf);
        buf += 4;
        f = unpack754_32(pf);
        break;
        case 's': // string
        s = va_arg(ap, valc);
        len = unpck16(buf);
        buf += 2;
        if (maxstrlen > 0 && len > maxstrlen) count = maxstrlen - 1;
        else count = len;
        memcpy(s, buf, count);
        s[count] = '\0';
        buf += len;
        break;
        default:
        if (isdigit(format)) {
        maxstrlen = maxstrlen * 10 + (format-'0');
        }
        }
        if (!isdigit(format)) maxstrlen = 0;
        }
        va_end(ap);
        }
}
```

The main method:

```
public static void main(String[] args) {

              char[] buf = new char[1024];
              int8 v1;
              int16 v2;
              int32 v3;
              f32 v4;
              String s = "A string value";
              char[] s2 =new char[96];
              int16 packetsize, ps2;
              packetsize = pack(buf, "dgkchlsf", (int)'C', (int)0,
(int)68,
              (int)-17, s, (float)-547490.8679);
              pck16(buf+1, packetsize); // store packet size in
packet for kicks
              System.out.printf ("packet is %" + " bytes\n",
packetsize);
              unpack(buf, " dgkchlsf", v1, ps2, v2, v3, s2,
              v4);
              System.out.printf ("'%c' %" + " %" + " %" +
              " \"%s\" %f\n", v1, ps2, v2,
              v3, s2, v4);

       }
```

In the code above there are methods encapsulated in a Java package and these can pack, encode and unpack data of different types that are coming from other business applications or are gathered for this specific business application [3], [5].

The advantage of using a custom package for manipulating data is that the programmer has the full control over the design and implementation of the applications, with little changes can access every type of data and also this package can be improved and used by a large number of software developers because is a Java class and so is open-source. In the source code of this package could be implemented various methods for managing data in databases (select, insert, update, delete, create, etc.) or from XML files.

Using a custom package for manipulating data (own class) has many advantages like accessing different types of data, create the own objects, manage various structures of data including XML files and because this package is built with Java programming language makes it open-source, easy to customize by other software developers and can be improved by implementing different methods and classes. Using Java for business applications makes possible to split work for designers, software developers by using business objects capsulated in classes and so is easy to manage, to modify, to improve and to create new modules for different type of end users. Java makes possible to upgrade old business applications and to build

new ones that are more stabile and much easier to use by many users and is platform free.

Another programmer or groups of programmers can improve these open source methods used for the integration of data by adding new classes and methods or by integrating in their own business applications the existing objects build on XML files or by integrating the existing data warehouse in more complex star schemas of a bigger data warehouse. Making data available through packing, encoding and unpacking has many advantages because these are persistent through the entire business application and it is possible to build a separate service available to the server side and to various types of frameworks used to develop enterprise business applications.

4. Conclusions

The Java programming language improves the ways of building new enterprise business applications and to reuse old resources from previous applications thus is easy to answer at the specific demands from the clients. Java applications are more adaptive and robust if they are built on a business logic that implements the basic design and implementation according with the UML diagrams. The manipulation of data is an important aspect because the programmer can use data from different types of sources like different DBMS-s, XML files or star schema already built [1], [4]. The open-source code made the implementation of these methods very easy and makes possible the customization afterwards by other groups of programmers and so the application becomes open-source applications. The custom package presented above can be used in different types of enterprise business applications, also may be improved by other programmers through the implementation of their own classes and methods and also may extend the current facilities of the existing methods for working with data [2], [3]. Furthermore the data is available to numerous business applications and even can be accessed an integrated business service based on the custom Java package.

References

[1] David Reilly, *"Getting Started with JDBC"*, David Reilly webmaster, pp. 182-187, 2008;

[2] David Gallardo, *"Java design patterns 101"*, ibm.com/developerWorks, 2009;

[3] Danut-Octavian Simion, *"Using Java in Business Applications"*, WSEAS Conferences in the Universitatea Politehnica, Bucharest, Romania, pp. 218–223, April 20-22, 2010, Conference/Session: ECC_COMPUTING, ISBN: 978-960-474-178-6, ISSN: 1790-5117;

[4] Doug Lea, *"Concurrent programming in Java design principles and patterns"*, Addison Wesley, pp. 247-253, 2009;

[5] James W. Cooper, "Java Design Patterns at a Glance", www.javacamp.org/designPattern/ 2010;

URI: http://javaworld.com/javaworld/
URI: http://www.packtpub.com/service-oriented-java-business-integration/
URI: http://www.sun.com/software/javaenterprisesystem/
URI: http://www.manageability.org/
URI: http://www.javarules.org/