Mister Model, Beans, and Seriailization - A Rapid Web Services Deployment FAQ

Chad S. Matsalla

Agriculture & Agri-Food Canada
Saskatoon, Saskatchewan, Canada
matsallac@agr.gc.ca

August 8, 2004 Document Version: 1.2 by: csmsf

1 What is Mister Model?

Mister Model is a code generation system. It takes a UML model expressed as XMI and generates source code via a plug-in architecture. At this time, Mister Model has plugins to generate Perl, Java, and XML Schema.

A plesant side effect to use this approach is that models that documenting the classes used in a system remain as design artifacts. These artifacts are important in understanding the relationships between the classes in the system.

2 What type of project would require a system like Mister Model?

Mister Model was created to enable a system to:

1. **Deploy existing code as web services as fast and painless as possible.**
2. Rapidly deploy plain old Java objects as enterprise java beans so that they can be used as web services
3. Rapidly deploy plain old Perl modules as web services

3 Why would you want to do that?

- Enable distributed computing regardless of the language of the service or client.
- Enable and encourage UML models as living artifacts that document the system as accurately as possible without compelling coders to update them.

4 Why web services?

Web services allow the system use pass complex (non-primitive) objects:

- as arguments in procedure calls
- as return types
- as references on which procedures are called (foo.getSomething())

Web services and its components (SOAP, XML) preserve the concept of objects while still enabling communication over HTTP.

5 What stood in the way of using web services?

- The overhead of writing and understanding EJBs for web services in Java.
- The overhead of understanding the way SOAP::Lite delegates web service calls to perl modules and serializes complex objects.
- Serialization issues

6 How do I turn POJOs into EJBs as fast as possible?

XDoclet/Ant/JBoss

1. Create java source code.
2. Mark up java source with javadoc-like comments indicating that you want them exposed as services.
3. Process java source with Ant and XDoclet- this creates all structures necessary for deployment.
4. Drop the archive into JBoss.

7 How do I turn POPM's into web services as fast as possible?

SOAP::Lite

1. Create a Perl module.
2. Test it in the normal way. (ignore web services)
3. Create a five line .wsr that delegates all calls to that module.

8 Under what constraints do non-Mister Model enabled web-services work?

1. The language of the service and the client are the same
2. The service and the client both know about the objects that are being moved about - This means that the same objects are installed in @INC (for perl) or are in the CLASSPATH (for JAVA)
3. Many deep objects[1] have trouble. Shallow objects[2]

9 Why do these limitations suck?

The languages have to be the same. What's the point?

10 Serialization Failure

10.1 Why can't Java clients talk to Perl web services?

Serialization and deserialization failure. Apache Axis has no idea how to serialize objects to form the arguments of a procedure so that Perl can decode them. Apache Axis cannot decode (deserialize) the objects returned from the Perl web service.

10.2 Why can't Perl clients talk to Java web services?

Serialization and deserialization failure. SOAP::Lite has no idea how to serialize objects to form the arguments of a procedure so that JBoss can decode them. SOAP::Lite cannot decode the objects returned from the JBoss web service

11 How are serialization issues resolved?

1. Create UML models for all non-primitive datatypes that will be moved in the system
2. Generate code modeling those models. Create a jar for java or a gzip for perl.
3. Place the archives with those classes in the paths for both the client and the server.
4. Register the classes with the SOAP engines (SOAP::Lite, JBoss, Apache Axis)
5. Use the web services.

12 What tool do you use to draw UML models?

Poseidon, from Gentleware.com

13 Why use Poseidon?

1. A 'Community Edition' is downloadable for free.
2. Poseidon generates OMG-compliant XMI via the NetBeans XMI writer.
3. Poseidon is implemented in Java. This allows users to run Poseidon on a number of different operating systems.
4. Poseidon generates excellent vector-image representations (Postscript) of the UML models. This allows end-users to manipulate images of UML models with a vector image manipulation application like Adobe Illustrator. This is important for preparing quality documentation and presentations.

14 Will other types of XMI work?

The following applications generate XMI that is known to be parsable by Mister Model:

1.

The following applications generate XMI that is known to be *un*parsable by Mister Model:

1.

15 Why UML? Why not define XML?

Although it is possible, XML Schema is not intended to be an object modeling language. From the "XML Schema Part 0: Primer" [World Wide Web Consortium, 2002]: "The purpose of a schema is to define a class of XML documents".

UML was designed to be an object modeling language. From the UML specification[Object Management Group, 2003a]: The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems.

16 Are there other approaches?

Sure.

- Create a DTD, Use XML-Object binding to generate classes. (This is the approach used by Sun's JAXB)
- Create a Schema and use one of the several small schema-object binding frameworks out there.
- Manually generate classes to try to match the model. Write custom serialization routines.

Although these approaches can work, they are difficult to implement consistently. Mister Model works reliably in a consistent way.

17 Language Specific Issues

17.1 Perl

17.1.1 Coderefs

Coderefs cannot be serialized and it is difficult to put such references in UML models. I recommend that coderefs be removed from objects prior to serialization to maintain reliability.

17.1.2 Hashes

Arbitrary hashes are difficult to map to other languages. It is poor practice to have such objects anyway - create associative relationships instead.

17.1.3 Collections

The Collection type most amenable to serialization is the array type.

17.1.4 Iterators

Iterators are not serializable. If it is the case that associative objects are reachable via Iterators only, you should use the iterator to populate an array prior to serialization.

For example, a Bioperl SeqIO-compliant object contains an iterator method 'nextSeq()' that returns some sort of object. In the case that you want to return all of the results in a SeqIO stream, you should create an array and call nextSeq() repetitively to fill the array and then return that array. Incorrect would be to return the SeqIO object itself because, well, it has no data.

17.2 Java

17.2.1 Abstract Classes, Interface Classes