Open-DIS: an open source implementation of the DIS Protocol for C++ and Java

McGregor, Don

http://hdl.handle.net/10945/47799
Open-DIS

Open Source Distributed Interactive Simulation

Don McGregor (mcgredo at nps dot edu)
Don Brutzman (brutzman at nps dot edu)
John Grant (johnkonradgrant at yahoo dot com)
Very few open source DIS implementations to date. We’ve done a couple for Java, but there are none that I know of for C++. This results in duplicated effort as people re-implement the wheel. You can buy a commercial license, but this tends to not work well in an academic environment, or for projects with extremely long product life cycles. It takes time and effort to sort out licensing and stay within the license requirements, and avoiding this is valuable. What happens if the vendor goes away? Free is an easy to licensing concept understand and work with.
Licensing

Open-DIS uses the BSD open source license. Anyone can use or modify the code, and it is non-viral; using Open-DIS in your project does not make your project open source.

You can modify the code if you like.
You do not have to return changes to the authors (though this is appreciated and encouraged).
You can use it in commercial products.
No license fees.

*Just use it!*
Implementation Features

Java and C++ code implementation with a similar API
Java and C++ PDU objects can marshal themselves to DIS format
Java objects can also marshal themselves to XML and Java Object Serialization format
An XML schema is provided
Some supporting networking code and example implementations
Availability

http://sourceforge.net/projects/open-dis/

Code is available as both tar.gz and subversion source code control downloads.

Sourceforge forums and mailing lists for tech support.

Full source code, not just binary releases.
Implementation

The Java and C++ code was generated from an XML template

From this:

```xml
<attribute name="x" comment="velocity about the x axis">
  <primitive type="float" defaultValue="0"/>
</attribute>
```
Implementation

To this in Java:

```java
public class AngularVelocityVector
    extends Object implements Serializable
{
    /** velocity about the x axis */
    protected float x = 0;
    ...
    public void setX(float pX)
    {
        x = pX;
    }

    @XmlAttribute
    public float getX()
    {
        return x;
    }
```
Implementation

And this in C++

AngularVelocityVector::AngularVelocityVector():
  _x(0), _y(0), _z(0)
{}

AngularVelocityVector::~AngularVelocityVector(){}

float AngularVelocityVector::getX() const
{return _x;}

void AngularVelocityVector::setX(float pX)
{x = pX;}

Implementation

Getters, setters, constructors, and destructors for each PDU and sub-PDU object

Each PDU also has code to marshal itself to DIS format, and unmarshal itself

The “@XmlAttribute” annotations specify how the java objects should be marshalled to XML; in this case the X field will be marshalled as an attribute to the AngularVelocityVector element

<AngularVelocityVector ... x="17.0"/>
Implementation

XML Template → C++ Code Generator → C++ .h, .cpp files

XML Template → Java Code Generator → Java Source Files
Implementation

If there are fundamental changes, we can modify the template code and re-generate the C++ and Java code

This code that generates C++ and Java is also provided if you want to do something different, or you can do a C# or other language generator if you like

About 4K Lines of template XML and 1K Lines of code generation to create ~30K Lines of C++ and ~20K Lines of Java

The generated Java and C++ is checked into source code control, so we can modify it directly as well
Implementation

Drawbacks to the approach: the classic problem of modifying code that is autogenerated. Changes are lost if you regenerate code after modifying it.

Can work around this via patch files, diff, etc (I think.)
Why Not Schema

Originally we attempted to use XML Schema as the basis for generating Java and C++ source code

• C++ schema-to-code tools generated lousy code
• The generated code didn’t handle variable length lists well
• Parsing schema is complex, and there is no semantic link between variable length lists and the list length fields
• Different C++ and Java APIs

The template XML file and code generator gives us complete control over how the Java and C++ source code looks, and turned out to be not that difficult to write.
XML

JDK 1.6 gives us the ability to marshal and unmarshal XML to and from Java objects via JAXB, which is built into the JDK release.
Can generate a schema for DIS from the Java source code (provided).
As a result we get XML interoperability for free.
Class Hierarchy

- **Pdu**
  - **Entity Interaction Family**
    - Entity State PDU
  - **Warfare Family**
    - Fire PDU
    - Detonation PDU
  - **Collision PDU**
Object Hierarchy

PDUs also contain objects for major records, such as position, EntityType, orientation, etc.
Object Hierarchy

The major records defined in the DIS standard are represented as objects, such as location, an object that contains three double precision floating point numbers.

The object knows how to marshal and unmarshal itself.
Garbage Collection

This can create a problem when receiving a lot of PDUs. Every time a new PDU is created it may contain several objects within it. In Java this stresses the garbage collector in realtime operation. Observation: the vast majority of PDUs are ESPDUs. If we can optimize that we will solve most of the problem. The FastEntityStatePdu class “flattens” the object structure of entity state PDUs and consists only of primitive type fields. Eliminates 11 objects per ESPDU.
Supporting Java Classes

PduFactory creates new Open-DIS PDU objects from binary data
Logger example saves PDU traffic to XML files
X3D example shows DIS being used to drive a 3D scene
XMPP example shows DIS in XML format being sent across chat channels
C++ Classes

Essentially identical to the Java classes, but lacking XML support.
Uses HawkNL library for networking support.
(You can also use plain old Berkeley sockets if you like)
Future Work

Not all PDUs correctly implemented--radio communications in particular needs work
Enumerations support
HLA bridge
Support for DIS-200x
Dead Reckoning algorithms
Finite state machine support
Programming help appreciated
(So is funding...)