XSIL: Extensible Scientific Interchange Language

Kent Blackburn[1], Albert Lazzarini[1]
Tom Prince[1,2], Roy Williams[2]

[1]LIGO Laboratory, California Institute of Technology 18-34, Pasadena, CA 91125, USA
{kent,lazz}@ligo.caltech.edu

[2]Center for Advanced Computing Research, Caltech 158-79, Pasadena, CA 91125, USA
{prince, roy}@cacr.caltech.edu

Abstract. We motivate and define the XSIL language as a flexible, hierarchical, extensible transport language for scientific data objects. The entire object may be represented in the file, or there may be metadata in the XSIL file, with a powerful, fault-tolerant linking mechanism to external data. The language is based on XML, and is designed not only for parsing and processing by machines, but also for presentation to humans through web browsers and web-database technology. There is a natural mapping between the elements of the XSIL language and the object model into which they are translated by the parser. As well as common objects (Parameter, Array, Time, Table), we have extended XSIL to include the IGWDFrame, used by gravitational-wave observatories.

1 Introduction

The Extensible Scientific Interchange Language (XSIL) is designed to represent collections of common scientific data objects. There are constructors for objects such as Parameters, Arrays, Tables, and other types, as well as support for binary objects, both MIME-typed and untyped. There is also a container object that may contain these other types as well as other containers, so that an XSIL file can be hierarchical. For each object, the entire object with all its data may be represented in the XSIL file. Alternatively the XSIL file may only contain metadata, with the bulk binary data external: there is a flexible Stream object that may contain URL-type links to external data.

XSIL is based on the XML language[1], an industry standard for which a large amount of software is available in the form of editors and parsers, as well as the familiar Microsoft and Netscape browsers. The aim has been to keep the language simple, intuitive, and easy to create and use, using a text editor, an XML editor, or from a program. The XSIL file can be used by either a human or a computer: it can be displayed, edited, summarized, sorted, or printed for human consumption; or it can be parsed as machine input with a number of methods and tools, as outlined below.

XSIL is a way to represent collections of scientific data objects—small objects with data explicitly contained in the file, and large objects represented by the salient metadata and references to binary files elsewhere. We intend the XSIL format to be

used:

- As a flexible and general transport format between disparate applications in a distributed archiving and computing system; a text-based object serialization that can be handled by common tools, or
- As documentation mechanism for collections of data resulting from experiments or simulations; with all the parameters, structure, filenames and other information needed to keep a complete scientific record.
- As an "ultra-light" data format: a user can, if he wants, simply read the markup, then delete all except the actual data, or all except for the filenames where the data may be found.

We should note that an analogous format[2], WDDX, is under development as part of Allaire's Cold Fusion product line.

1.1 Applications

The LIGO project[3] (Laser Interferometric Gravitational wave Observatory) is a large, federally funded physics experiment that will produce several megabytes per second, 24 hours per day, 7 days per week. This data will be processed, looking for matches with astrophysically significant events, for example coalescence of neutron stars and black holes. The data will also be processed and distributed in other ways, and will be supplemented by instrument status data, candidate events from the pattern matching, and other data. While a format has been fixed for the raw data, in collaboration with the French/Italian VIRGO observatory[4], there is a need for a more flexible, more generic format for many of the other datasets, which motivated the design of XSIL.

Other projects at Caltech and elsewhere which may benefit from XSIL include

- Digital Puglia Synthetic Aperture Radar Atlas[5], an archiving and processing facility for knowledge-discovery in remote-sensing databases,
- Digital Sky, a prototype confederation of astronomical surveys,
- Center for Simulation of the Dynamic Response of Materials[6], a multidisciplinary consortium at Caltech for simulations at multiple scales.
- Interferometric SAR Library, a facility to improve the usability of this promising technology.

2 Presentation and Content

The syntax of XSIL is based on XML[1] (eXtensible Markup Language), now an industry standard for representing structured textual documents. XML combines the popularity of HTML in the wide Internet community with the battle-hardened power of SGML in the library community. Every XSIL file is an XML file; some references to XML are at the end of this document. Here is a small example of an XSIL file:

```
<?xml version="1.0"?>
<!DOCTYPE XSIL SYSTEM "XSIL.dtd">
<XSIL>
    <Comment>Five Measurements of voltage</Comment>
    <Param Name="Gain" Unit="milliVolt">1.453</Param>
```

```
    <Array>
        <Dim>5</Dim>
        <Stream><Metalink Format="Text" Delimiter=" \n"/>
            1.28374 1.23453 1.94847 2.148474 2.39484
        </Stream>
    </Array>
</XSIL>
```

The first line is like a "magic number" which must be the first line of any XML file. The second line says that this XML file is of a particular kind, an XSIL file. The `<Comment>` element is supposed to contain text that appears in presentation of XSIL, but does not play a part when the computer parses the file. This is followed by a `<Param>` object and and `<Array>` object. The `<Param>` object is an entry in a list of keyword-value pairs — there may also be units and comments associated with the `<Param>`. The `<Array>` has one dimension, with five elements, as indicated by the `<Dim>` element, and the array is then expressed explicitly in ASCII text through the `<Stream>` tag.

2.1 Presentation

One of the advantages of basing the XSIL format on a standard language such as XML, is that standard desktop tools can be used to view and edit the file. Figure 1 shows how

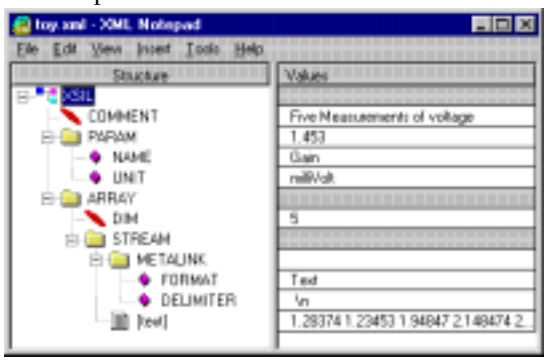

Fig. 1: The file above rendered by an XML-editor.

this file looks when viewed with an XML editor, in this case XML Notepad from Microsoft. There can also be customized presentation of XML and XSIL files in the browser through style sheets. Large numbers of XML tools are coming to market, for browsing, editing, sorting, and converting XML files. Some examples are shown for a more sophisticated XSIL example later in this document.

2.2 Parsing XSIL

In addition to providing a document that can be browsed by a human, an XSIL file is designed to allow computers to “understand” its content. By this, we mean that the structure defined by the XSIL file is mapped into objects on the program side—that is, an API. The current collection of XML parsing implementations are based either on:

- Event based parsing: The user associates handlers with tag names, then hands

control to the parser.

- Document Object Model[7,8]: The parser returns a document object which the user can interrogate, a fully-formed tree with the tags and text from the XML attached.

In both cases, access to the attributes of an element is generally through an associative lookup: given an attribute name, there may be a corresponding value, but it may be that the list of attribute names cannot be retreived.

3 The XSIL Language

3.1 Objects and Streams

In the XSIL language, there are ***objects*** and ***streams***. An object defined in the XSIL API implements the `XSILObject` interface, which has the method `readStream()`, as well as other methods to get the name and type of an object. While XSIL itself is designed for flexibility rather than speed, it is expected that streams may be implemented with an opposite sensibility: as powerful, high-bandwidth, possibly parallel, data streams. A stream is defined in the XSIL file with a `<Stream>` element, which may contain data explicitly, or may be a link to data stored elsewhere through a URL-like syntax.

Streams provide a separation between control and data; XSIL is the control and its Stream carries the data. Just as the anchor chain of a ship is pulled to shore with a small rope, so the flexible, low-bandwidth XSIL can be used to set up high-performance data channels.

When an object is read in to a program, it may have an input stream attached to it, from which it may read data. For example, in the implementation of the Array class, dimensions of the Array are used to allocate an appropriate amount of memory, then the data can then be read in through the `readStream` method of the underlying XSILObject.

3.2 XML

XML is much more than “glorified HTML”, but rather it is a “language for creating languages”. It is a hierarchical structure of elements that may contain other elements. An ***element*** generally consists of a ***start tag***, a ***body***, and an ***end tag***, for example: `<Fruit>Banana</Fruit>`, where start and end tags are distinguished by the presence of a slash. An element may be ***empty***, meaning that there is only a single tag, with no body, for example `<EmptyElement/>`; note the position of the slash. Elements may contain ***attributes***, for example: `<Fruit color=”yellow”>`. We should point out that XML is case-sensitive, so that `<Apple>`, `<apple>` and `<APPLE>` are all different tags.

Finally we distinguish ***presentation*** and ***parsing:*** When an XML document is presented, the element structure is used to generate formatting information so that a human can visualize its content—conversion to HTML, TeX, VRML, etc. When it is parsed, a computer reads the file and the elements are converted to objects that are returned from the API.

3.3 The Container Object

In the XSIL language, there is a generic <XSIL> element which can contain other elements, including other <XSIL> elements, thus inducing a hierarchy. Each of these may have a Name attribute, to provide hierarchical naming that is visible from the API. Furthermore, the XSIL file must be enclosed in <XSIL>...</XSIL> tags, so that when the file is parsed, it is always a single XSIL object that is passed back. Here is an XSIL fragment that consists of a container hierarchy with an array at the second level and a parameter at the third level:

```
<XSIL Name="Fruit">
    <XSIL Name="YellowFruit">
        <Array><Dim>7</Dim></Array>
        <XSIL Name="Banana">
            <Param Name="Inductance">1.34</Param>
        </XSIL>
    </XSIL>
</XSIL>
```

3.4 Object Representation in XSIL

In general, we would like to attach data and metadata to the branches and leaves of the tree of containers, presumably in the form of objects. We may point out here that the terms "object" and "element" have similar meanings here: except that "object" implies a program perspective (the API), whereas "element" implies the document perspective (XML). Thus in this paper these points of view may be somewhat loosely interchanged.

The basic data element of XSIL is simply <Object>, corresponding to the base-class XSILObject in the API. All XSIL elements may contain the following elements:

- <Comment>: this text is not parsed, it is presumably natural language.
- <Param>: to define associations between names and values that are specific to the containing object, and that are not accessible to other objects.
- <Stream>: a definition of the input data stream that the object may draw upon from the API.

Also, most XSIL objects can have certain attributes:

- Name: A string representing the name of the object; defaults to the null string.
- Type: The type of the object or of the relevant primitive type: defaults to "double"
- Unit: A string representing the physical units associated with a number or parameter, for example "Hz" or "km"; defaults to the null string.

In addition to these common properties, perhaps the most important aspect of the generic object is that it may have access to a Stream object, if one has been defined. A generic object might be used to reference a binary file, with the assumption that the meaning is in the metadata or in natural-language text (such behavior is condoned, not encouraged). The generic object may refer to a MIME-typed object, in which case it is expected that the information about interpreting the stream is contained in the MIME-type, which is available at the beginning of the stream.

For more specific objects, we expect the tags of the XSIL definition to provide enough metadata to the implementation that it knows how to read the associated stream.

3.5 Some XSIL Objects

XSIL is an extensible language, meaning that users can implement their own objects or subclass the existing objects. There are mechanisms in XSIL for this through addition of extra handlers in the parsing API, through addition of extra sections in the DTD for syntax checking (see below), and though addition of extra parts to the style language for presentation. In this list are some common scientific objects that many applications can use, such as Parameter, Time, Array, Table. We will extend XSIL with a format that may be less popular: expressing the metadata from the IGWD-Frame file, which is a standard file format for recording data from Gravitational-Wave detectors.

3.5.1 Parameter

A parameter in XSIL is an association between a name and a value, perhaps with additional attributes such as Unit and Type. For example:

```
<Param Name="Fruit_Mass" Unit="kg">0.387</Param>
```

As may be obvious, the meaning here is "Fruit_mass = 0.387 kg", which is the kind of thing usually found in "parameter files" or "header files" in scientific computing. At the API level, there is a dictionary of these parameters available, perhaps with unit conversion, allowing an easy lookup of critical parameters.

All XSIL objects may contain Parameters, as well as Comments. Thus the Parameter object is different from other objects: while a container may contain a Table element, which in turn may contain Parameters, but the Table may not contain other Tables.

3.5.2 Time

In the LIGO experiment, as with many other experiments in physical science, it is critical that timing information be not only accurate, but also easy to understand. The `<Time>` element in XSIL can represent either "natural" time (ISO-8601 standard, YYYY-MM-DD HH:MM:SS.mmmuuunnn), or GPS time, or "Unix time" (seconds since 1/1/1970). The different formats are differentiated by the `Type` attribute in the tag:

```
<Time Type="ISO-8601">1998-11-08 17:40:00.032</Time>
```

3.5.3 Array

An array is a collection of numbers (or other primitive type) referenced by subscripts, which is a list of integers whose maximum values are given by the list of Dimensions of the Array. This definition is very close conceptually to a Fortran or C array, with the `Type` attribute of the `<Array>` tag specifying which primitive type is contained in the array (float, int, etc.). The XSIL element specifies the dimensions of the array, but it does not specify the subscript ranges. For a dimension of 5, a Fortran binding of the API would label subscripts from 1 to 5, but a C binding would have subscripts from 0 to 4.

As with other XSIL objects, the Array tag may have `Name` and `Type` attributes, and the

Array element may contain Comment, Parameter, and Stream elements. The only element specific to this class is <Dim>, for example:

```
<Array Type="int">
    <Dim>5</Dim>
    <Dim>3</Dim>
</Array>
```

which specifies a 5x3 array of integers, with the last dimension changing fastest. The presumption is that 15 integers may be read from the Stream associated with this Array.

3.5.4 Table

A table is an unordered set of *records*, each of the same format, where a record is an ordered list of values. The contents of a record are defined by column headings, each of which may have a unit and a type. This definition of a table should be thought of as similar to the table object that is found in a relational database; we should point out that this is *not* the complex and exotic typographical beast of TeX or HTML.

The only tag specific to the Table object is <Column>, which specifies the name, type, and possibly units associated with one of the columns of the table. It can be thought of as the heading of a column in a table. In Figure 2 is shown a small table definition, together with a presentation of the table in HTML.

```
<Table>
    <Column Name="Fruit"/>
    <Column Name="Color"/>
    <Column Name="Mass" Unit="kg"/>
    <Stream>
        <Metalink Format="Text" Delimiter=",\n"/>
        Banana,Yellow,0.43
        Cherry,Red,0.01
    </Stream>
</Table>
```

Fig. 2: An XSIL fragment expressing a Table of Fruit. To the right is a presentation of the table using HTML viewed in a browser: because XSIL is an XML dialect, such translations can be done with great facility.

3.5.5 IGWD Frame

We are implementing an XML definition of the metadata found in a IGWD Frame[9], the principle data object of the LIGO[3] (USA) and VIRGO[4] (France/Italy) gravitational-wave observatories. The idea is that an XSIL file can contain the metadata for these objects, such as cataloguing information, perhaps with embedded content summaries. The Frame object in XSIL will also include a Stream that provides access to the actual binary file. The metadata includes timing information, the observatory at which the data was taken, natural language "history" records, together with the list of data channels that are recorded in the file. Each named data channel represents a stream of data from a particular instrument at the observatory, and has a status, data rate, gain, offset, etc.

When the parser sees a `<IGWDFrame>` element, it should pass control to a Frame reader module, which assumes the existence of an XSIL Stream object from which it can read the data. For more detail, see section 7.

4 Streams

When an XSIL element is parsed, the implementation of the resulting object may use the `readStream()` method of the object, for example an Array object would first read the type and dimensionality of the Array, then proceed to read in the data from its Stream. Thus a Stream may be thought of as a data socket, together with metadata about the link: data encoding, recommended timeout values, permission information, delimiter characters, and so on. The data may be contained in the XSIL file directly, either as readable text or as binary that is encoded to text by uuencode or base64.

4.1 A Stack of Streams

As the file is parsed, a stack of stream objects is created, with a new Stream pushed on the stack whenever such an element is encountered, and the stack is popped from the stcak and closed when certain end-tags are encountered. When an element is parsed into an object, it is given access to the Stream which is at the top of the stack. Many objects may then be read serially from the same Stream. For example:

```
<Stream Name="Yangtze">...</Stream>
<XSIL>
    <Array Name="Panda">....</Array>
    <Stream Name="Thames">.....</Stream>
    <Array Name="Quince">....</Array>
    <Array Name="Pumpkin">....
        <Stream Name="Hudson">....</Stream>
    </Array>
</XSIL>
<Array Name="Bamboo">....</Array>
```

Here the Array named “Panda” is read from the only open Stream, “Yangtze”. A new stream, “Thames” is then pushed on to the stack, from which the Array “Quince” is read. The Array “Pumpkin” comes with its own Stream, called “Hudson”, which is closed and popped as soon as the Array has been read. When the final Array in the example is reached, “Bamboo”, only one Stream is still on the stack, which is “Yangtze”, from which it is read.

4.2 Stream Element

A Stream element may contain actual data or a link to the data.

4.2.1 <Data>: Explicit Data

If the data is present explicitly, it is assumed to be unparsed character data; either as delimited text (as specified in the `Delimiter` attribute of the `Metalink` tag); alternatively it may binary encoded as text in one of various ways.

4.2.2 <Link>: External Data

Another way to specify a data stream is by an external reference, which is done in XSIL with a `<Link>` element. The content of the Link element has a URL-like syntax:

```
protocol://hostname:port/filename
```

where the data may be on a local file (`file`), on a web- or ftp server (`http`, `ftp`), or other idiosyncratic words like `tape`. From the API perspective, the protocol list can be extended by suitable handlers.

4.2.3 Metalink

There may be also be a `<Metalink>` element associated with the `<Stream>` or with `<Link>` elements. This empty element provides metadata about the link itself: how binary has been encoded to text, the delimiter character between numbers in a text stream, the recommended timeout when accessing this data, access protocols, access restriction information, etc. etc.

4.2.4 Fault Tolerance

There may be multiple `<Link>` elements for a given Stream, with the assumption being that any of the links can supply the data. In this way we can provide transparent fault-tolerance for access to the data: one link may be to volatile disk (with a small timeout), and another may link to an archival tape-robot, with a long timeout. The data may also be on a tape offline if both of these fail.

5 An Example XSIL File

To illustrate some of the capabilities of the XSIL format, we have created a fairly comprehensive example, which is page from an “eletronic logbook” for computational scientists:

```
<?xml version="1.0"?>
<!DOCTYPE XSIL SYSTEM "XSIL.dtd">
<XSIL Name="Sample XSIL File">
    <Comment>LIGO power spectrum of 32 magnetometers</Comment>
    <Comment>Created by Jill and Sue</Comment>
    <Param Name="LIGOType">Power Spectrum</Param>
    <Time Name="StartTime" Unit="GPS">609847463.78237325</Time>
    <Param Name="FreqSamp" Unit="Hz">
        <Comment>This is the sampling frequency</Comment>
        1024
    </Param>

    <Stream>
<!-- Open a Stream: this data is at Cacr, Hanford, and on a tape in the
fireproof vault. The implication is to open ONE of them -->
        <Link><Metalink Format="bigend" Timeout="600" Protocol=”DCE”/>
            file://hpss.cacr.caltech.edu/magval_09_25_97.bin
        </Link>
        <Link><Metalink Format="base64"/>
            file://hanford.ligo.caltech.edu/magval_09_25_97.base64
        </Link>
        <Link>tape://347846-6/756473</Link>
```

```
    </Stream>
<!-- Here is an Array object, the data comes from the stream above -->
    <Array Name="Magread" Type="double">
        <Comment>Magnetometer Readings from Sept. 19</Comment>
        <Param Name="Gain">40.76</Param>
        <Dim>1024</Dim>
        <Dim>32</Dim>
    </Array>
<!-- This Array has its own data -->
    <Array Name="Magcal" Type="Float">
        <Comment>Magnetometer Calibration from Sept. 19</Comment>
        <Dim>32</Dim>
        <Stream><Metalink Format="Text" Delimiter=" \n"/>
1.28374 1.23453 1.94847 2.148474 2.39484 2.84746 3.10928 4.92827
5.28374 5.23453 5.94847 6.148474 6.39484 6.84746 7.10928 8.92827
9.28374 9.23453 9.94847 10.18474 10.3984 10.8446 11.1928 12.9827
13.2874 13.2453 13.9847 14.18474 14.3984 14.8446 15.1928 16.9827
        </Stream>
    </Array>
<!-- Nested containers -->
    <XSIL Name="Magnet parameters">
        <Comment>This is the magnetic value and offset</Comment>
        <Param Name="Magname">BerthaSQUID</Param>
        <Array Name="Magvalue" Type="Complex">
            <Dim>32</Dim>
            <Stream><Link><Metalink Format="bigend"/>
                file://hpss.cacr.caltech.edu/magval.bin
            </Link></Stream>
        </Array>
    </XSIL>
<!-- A table expressed in XSIL -->
    <Table>
        <Column Name="ChannelName" Type="String"/>
        <Column Name="Site"        Type="Float"    Unit="meter"/>
        <Column Name="Clock"       Type="Float"    Unit="hour"/>
        <Column Name="Description" Type="String"/>
        <Stream><Metalink Format="Text" Delimiter=",\n"/>
BOX_01_09, 2770, 3, Temperature for the apple
BOX_01_17, 3880, 6, Pressure inside the banana
BOX_01_23, 3990, 8, Pressure in the Banana Cryopump
        </Stream>
    </Table>
<!-- Finally a generic XSIL object. Here Type means MIME-Type -->
    <Object Name="ExcelChannels" Type="application/vnd.ms-excel">
        <Comment>Same table in Excel format</Comment>
        <Stream><Link>
        http://www.ligo.caltech.edu/bertha/channels.xls
        </Link></Stream>
    </Object>
</XSIL>
```

5.1 Attributes and Elements

In many ways attributes and elements are treated equally by presentation and parser mechanisms. The major difference seems to be that whereas attributes are simpler—only a string is returned, but elements can be arbitrarily extended. We shall provide element-based versions of the Name, Type, and Unit attributes.to allow their extendability.

6 DTD Definition of XSIL

An XML file may be associated with a Document Type Definition (DTD) which defines the allowed tag names in the document, and how these fit together: which elements may contain which other elements, and how many of each element there may be. For example, an <Array> element may have multiple <Dim> element to specify its dimensionality, but it cannot contain <Table> or <Object> elements.

```
<!ELEMENT XSIL((XSIL|Comment|Object|Param|Stream|Array|Table)*)>
<!ATTLIST XSIL Name CDATA "" Type CDATA "">

<!ELEMENT Comment(PCDATA)>

<!ELEMENT Object(Comment | Stream)*>
<!ATTLIST Object Name CDATA "" Type    CDATA        ""    >

<!ELEMENT Param((PCDATA | Comment)*)>
<!ATTLIST Param Name CDATA "" Unit    CDATA "" >

<!ELEMENT Stream((PCDATA | Link | Metalink)*)>
<!ELEMENT Link((PCDATA | Metalink)*)>
<!ELEMENT Metalink(#PCDATA)>
<!ATTLIST Metalink
    Format CDATA "" Timeout CDATA "" Delimiter CDATA "" >

<!ELEMENT Array((Dim | Param | Stream | Comment)*)>
<!ATTLIST Array Name CDATA "" Type CDATA "">
<!ELEMENT Dim(PCDATA)>
<!ATTLIST Dim Name CDATA "">

<!ELEMENT Table((Column | Param | Stream | Comment)*)>
<!ELEMENT Column EMPTY>
<!ATTLIST Column Name CDATA "" Type CDATA "" Unit CDATA "">
```

When a parser sees a document that is supposedly XSIL, it can first use the DTD to check that it meets the specification, that some object does not have two names, that an Array does not contain a Table. This part of the validation can be done with any validating XML parser. After this the parser can check XSIL-specific validation, such as making sure that the dimension of an array is integer.

A DTD may also be thought of as a collaboration mechanism. The members of the collaboration jointly agree on a DTD, then each side can implement against this specification—this is analogous to the collaboration mechanism with object-orented programming, where there is joint agreement on objects and their methods, followed by independent implementation.

7 Extending XSIL

After years of developement, object-oriented languages such as C++ and Java are very good at defining inheritance; unfortunately, XML has not come so far, and the DTD mechanism is not structured for element inheritance and subclassing. Other specifications of document type are under discussion[10,11].

As an illustration, let us consider how XSIL will be extended to include the IGWD

Frame. In the Document Object Model, a document object has been passed back through the API representing the XSIL file, and an element has been found whose name is <IGWDFrame>. Control is then passed to an XSIL-Frame object whose function it is to interface between the Frame and the XSIL. This object reads in the metadata associated with the Frame—provenance, timing, size, channel list, etc.—then returns the XSIL-Frame object to the application. The application may use the XSIL-Frame object as metadata, to present, to choose, to update the catalogue, or it may actually open the Frame file and read in all of its data. From the XSIL Stream object can be extracted a C++ stream or a C file descriptor, which can be passed to the existing Frame library for reading.

8 References

[1] XML resources:
http://www.xml.com

[2] Web Distributed Data Exchange (Allaire Corp.)
http://www.allaire.com/developer/wddx/

[3] LIGO (Laser Interferometric Gravitational wave Observatory)
http://www.ligo/caltech.edu/

[4] VIRGO Gravitational wave Observatory
http://www.pg.infn.it/virgo

[5] G. Aloisio, M. Cafaro, R. Williams, Digital Puglia Synthetic Aperture Radar Atlas, Proceedings of HPCN99, Amsterdam, April 12-16

[6] A Center for the Dynamic Response of Materials,
http://www.cacr.caltech.edu/ASAP/

[7] Document Object Model Specification (W3C)
http://www.w3.org/TR/WD-DOM/level-one-xml

[8] Document Object Model Resources
http://www.xml.com/xml/pub/DOM

[9] IGWD-Frame Class Library
http://www.ligo.caltech.edu/~wmajid/fcl/index.html

[10] Document Content Description for XML
http://w3c.org/TR/NOTE-dcd

[11] XML-Data, a Schema Definition Language for XML written in XML
http://www.w3.org/TR/1998/NOTE-XML-data/