AN XML REPOSITORY ARCHITECTURE FOR STEP MODULES

JOSHUA LUBELL
National Institute of Standards and Technology
lubell@nist.gov

ABSTRACT

NIST is developing a web-based repository to serve as the core of a modular environment for developers of STEP, a family of product data exchange standards. Modules are represented in the repository as XML, enabling them to be treated both as documentation and as software components. A prototype implementation of the module repository uses XML and other emerging technologies to dynamically render a module's content in a web browser in response to user input. No server-side processing is required.

KEYWORDS: STEP, XML, module, repository, product data exchange

INTRODUCTION

STEP (the Standard for the Exchange of Product model data, officially ISO 10303) [1] [2] is a family of standards defining a vocabulary and semantics for the exchange of product data throughout a product's life cycle. An application protocol (AP) in STEP describes a particular data model of an engineering or technical application. STEP developers are adopting a new modular development strategy for APs [3] [4], driven by requirements to:

- Reduce AP development costs.
- Allow implementations to contain multiple AP subsets or extensions.
- Improve AP interoperability.

In this new strategy reusable components called application modules, or simply modules, are combined to form modular APs. Modules have a dual nature in that they function not only as standards documents for reference, but also as software components to be used by yet-to-be-developed software tools for building APs. Although guidelines exist for developing and using modules and for evaluating them with respect to APs [5] [6], documentation alone is insufficient for effective development of modules and APs. To move STEP modularization from vision to reality, a "plug-and-play" modular software environment must exist to aid STEP developers in choosing modules and adding constraints to meet an AP's requirements.

The National Institute of Standards and Technology (NIST) is developing a web-based repository of modules to serve as the core of this modular environment for developers of STEP standards. This repository is intended to facilitate the management of and access to module-related documentation. The repository's users will be developers of modular APs as well as creators of new modules. Modules will be represented in the repository as XML (Extensible Markup Language) [7] documents. XML is a particularly attractive choice of representation language for modules because:

- XML parsers are inexpensive and easy to obtain.
- Internet browsers will soon be able to parse XML and will include standardized application programmer interfaces for manipulating XML data.
- XML is well suited both for data exchange over the Web and for producing human-readable output in numerous formats. Thus, XML satisfies the dual nature of modules.

Figure 1 shows a high-level view of the repository. The repository contains *library* and *catalog* data, both represented in XML. Library data is used to produce the actual documentation for a module. Catalog data associates classification properties with modules to facilitate searches. Repository data will be represented in a manner rich enough to enable the generation both of documentation in the traditional sense (page-oriented output) and of interactive, dynamic hypertext taking advantage of the full capabilities of the Web. Repository data can also serve as input to one of the software applications comprising a plug-and-play modular software environment for STEP. For example, repository data might be fed to an interactive software tool that guides AP developers in constraining modules to meet an AP's information modeling requirements.

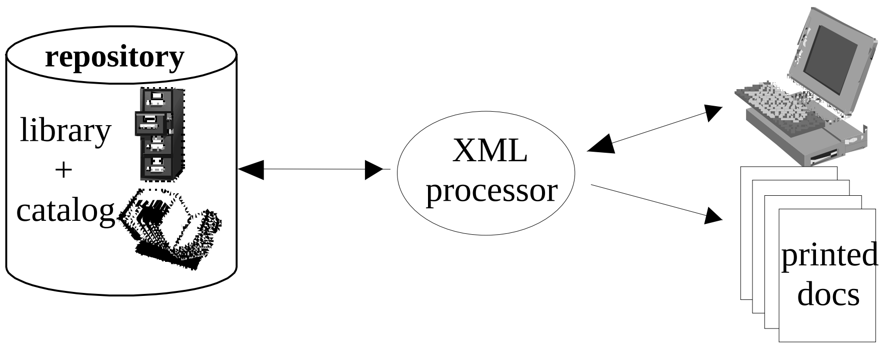


Figure 1. Repository architecture.

DESIGN ISSUES

XML Representation of STEP Modules

Modules contain a mix of formal specifications written in EXPRESS (a modeling language used for describing information requirements in STEP) [8] [9], prose describing the EXPRESS, illustrations, and boilerplate text specific to standards documents. Since boilerplate text can be generated automatically, an XML document type definition (DTD) for modules is essentially a DTD for annotated EXPRESS. Therefore, a principal challenge in developing a module DTD for a repository is defining the mapping from EXPRESS concepts to XML. Several issues come into play when serializing an information model into an XML vocabulary. Various EXPRESS-to-XML mapping methods have been proposed and analyzed with respect to these issues [10].

One design issue of particular relevance to the module repository is whether to employ an early binding or a late binding. In an early binding, the named components of the XML vocabulary directly correspond to EXPRESS data types. For example, consider the following EXPRESS definition of a point on a plane with x and y axes:

```
ENTITY point;
  x : REAL;
  y : REAL;
END_ENTITY;
```

This EXPRESS definition specifies a point as having two attributes whose values are real numbers corresponding to the point's x and y coordinates. An early-bound XML serialization of a point might look something like this:

```
<point id=”e1”>
  <point.x id=”a1”><real /></point.x>
  <point.y id=”a2”><real /></point.y>
</point>
```

In a late binding, the named components of the XML vocabulary do not directly correspond to EXPRESS data types. Instead they correspond to EXPRESS meta-objects. For example, a late-bound XML serialization of a point could look like this:

```
<entity id=”e1”>
  <name>point</name>
  <attribute id=”a1”><name>x</name><type><real /></type></attribute>
  <attribute id=”a2”><name>y</name><type><real /></type></attribute>
</entity>
```

Although the late binding is more verbose than the early binding, a late-bound EXPRESS-to-XML mapping is better suited to the module repository than an early-bound mapping. If an early-bound strategy were to be used, there would have to be a distinct DTD for each EXPRESS information model in the repository. This would complicate implementation. A late binding, on the other hand, allows for a single DTD to be used for any EXPRESS specification since the XML vocabulary defined by the DTD corresponds to EXPRESS meta-objects rather than data types specific to the model.

Another issue relevant to the design of the repository DTD is the level of granularity of the markup with respect to the EXPRESS language. At one extreme, all of a module's EXPRESS code could be represented in XML as character data inside a single pair of tags. This would not be very useful because there would be no easy way for an XML application to reference an individual EXPRESS construct or associate descriptive text with it. The other extreme would be to have a DTD specifying markup for every single token in the EXPRESS language. Such a DTD would be overly verbose and cumbersome to work with because the EXPRESS language is so complex.

The module repository takes a middle-of-the-road approach, specifying high-level EXPRESS structures such as data types, their attributes, and inheritance relationships between data types, but not specifying markup for low-level expressions. As an example, consider a point equidistant from the x and y axes. The following EXPRESS definition specifies this as a subtype of the point data type defined previously:

```
ENTITY point_on_diagonal SUBTYPE OF (point);
WHERE WR1 : ABS(x) = ABS(y);
END_ENTITY;
```

The construct following the WHERE keyword is a rule with the label WR1. The rule body consists of a constraint expression specifying that the inherited x and y attributes must have the same absolute value.

This definition represented in XML using a late-bound, middle-of-the-road markup scheme with hooks for English descriptions of some EXPRESS constructs might look as follows:

```
<entity id=”e2”>
  <name>point_on_diagonal</name>
  <description>A <entityref linkend=”e2” /> is a <entityref
linkend=”e1” /> equidistant from the <attributeref linkend=”a1” /> and
<attributeref linkend=”a2” /> axes.</description>
  <subtype><entityref linkend=”e1” /></subtype>
  <whererule>
    <label>WR1</label>
    <description><attributeref linkend=”a1” /> and <attributeref
linkend=”a2” /> shall have the same absolute value.</description>
    <expression>ABS(x) = ABS(y)</expression>
  </whererule>
</entity>
```

Note that the XML captures point_on_diagonal’s subtype relationship with point and specifies that point_on_diagonal is constrained by a rule with label WR1. The constraint expression ABS(x) = ABS(y), however, is not marked up. As far as an XML parser is concerned, the constraint expression is indistinguishable from any other character data.

Accommodating Off-line Repository Use and Collaborative Development

STEP APs are international standards and, as such, are often developed by teams spanning multiple organizations and multiple countries. Meetings are frequent, and much of the work gets done while on travel where Internet access is not always available. Therefore, the module repository needs to be downloadable so that it can function off-line with Internet access being needed only intermittently to update the repository contents. This requirement dictates that the repository favor client-side processing over server-side processing. Also there must be a robust means for users to update their local repository installations and make changes to modules in a master repository, presumably residing on an Internet-accessible server somewhere.

XML and other emerging web technologies such as Java [11], client-side scripting [12], and the Document Object Model (DOM) interface [13] make possible highly interactive web applications with dynamic content and with all processing taking place within the browser. By eliminating the need to interact with a web server, these technologies can be used to create a user interface for off-line access to the module repository. The next section discusses a prototype implemented to demonstrate this.

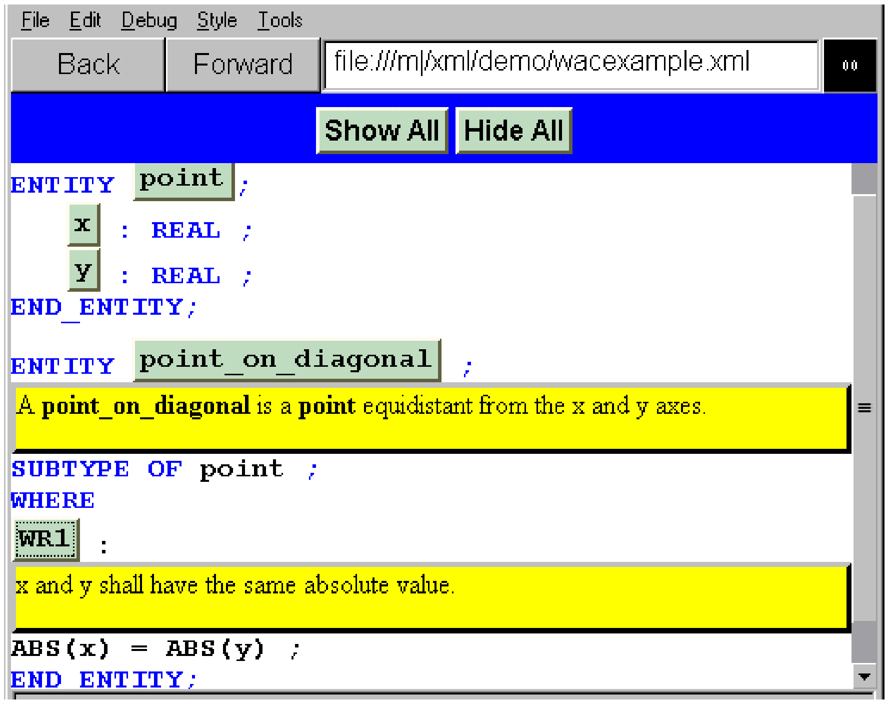
An environment for modifying modules, tracking their versions, tracking errata, and updating local repositories can be built using the same tools that large, distributed software projects use to manage collaborative development [14] [15]. Implementation of this environment using these tools should be straightforward because STEP modules function in a manner similar to software components. Also, like source code, modules are stored as text files, making it easy to maintain revision histories in terms of line-by-line changes.

REPOSITORY PROTOTYPE

A repository prototype has been implemented to demonstrate some of the benefits of representing modules using XML. In particular, it demonstrates how a module can be rendered dynamically in a browser environment in response to user input and independent of a web server. EXPRESS specifications are represented in XML using a late-bound vocabulary with hooks for text descriptions of EXPRESS constructs. The prototype uses client-side scripting in conjunction with the DOM interface to provide dynamic hypertext without server-side processing.

Figure 2 shows a screen shot of the user interface displaying an EXPRESS specification containing the point and point_on_diagonal type definitions from the earlier example. Prose descriptions of point_on_diagonal and its constraint on inherited attributes x and y are visible in the screen shot. An AP developer may click on any of the button labels embedded in the EXPRESS to toggle a text description of the labeled item. At any time, the developer may click on the buttons at the top of the page to show or hide all text descriptions. All rendering is done using a single XML document and without any interaction with a web server. In fact, the prototype has been demonstrated successfully on a laptop computer without any Internet connection or server software.

Further information about this prototype is available at http://www.nist.gov/stepmod/.


File Edit Debug Style Tools
Back
Forward
file:///m|/xml/demo/wacexample.xml
Show All
Hide All

```
ENTITY point ;
    x : REAL ;
    y : REAL ;
END_ENTITY;

ENTITY point_on_diagonal ;
```
A **point_on_diagonal** is a **point** equidistant from the x and y axes.
```
SUBTYPE OF point ;
WHERE
WR1 :
```
x and y shall have the same absolute value.
```
ABS(x) = ABS(y) ;
END_ENTITY;
```

Figure 2. Screen shot from repository prototype showing annotated EXPRESS.

FUTURE DIRECTIONS

The proposed module repository addresses some of the requirements resulting from the new modular approach to STEP AP development. In particular, the repository addresses the need for modules to function both as documentation and as software components, and the need for a configuration-controlled electronic catalog of specifications that can be accessed with or without an Internet connection. To realize more fully the plug-and-play modular software environment envisioned for AP developers, we have set out goals for the near future to develop the following:

- A full DTD for modules. This DTD will represent both catalog and library data and will be a superset of the DTD used in the prototype.
- Tools for tracking modifications to modules and managing updates to local repositories.
- A tool to enable module authors to continue using their favorite software for creating and modifying EXPRESS specifications and not require them to use an XML editor to make changes to EXPRESS code. Such a tool would need to create an XML document template from an initial EXPRESS specification and also facilitate the merging of modifications to an EXPRESS specification of a module with the existing XML document for the module.

REFERENCES

1. Kemmerer, S.J. (ed.) *STEP: The Grand Experience*, Special Publication 939, National Institute of Standards and Technology (July 1999).
2. ISO 10303-1:1994 *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles.*
3. ISO TC184/SC4/WG10 *STEP/SC4 Industrial Data Framework* (August 7, 1999). Available at http://wg10step.aticorp.org/.
4. Allison Barnard Feeney and David M. Price *A Modular Architecture for STEP*, World Automation Congress, Seventh International Symposium on Manufacturing with Applications (June 2000).
5. ISO TC184/SC4/WG10 N221, *Guidelines for the content of application modules* (1998-12-23). Available from http://wg10step.aticorp.org/.
6. ISO TC184/SC4/WG10 N222, *Guidelines for the content of application protocols using application modules* (1998-12-21). Available from http://wg10step.aticorp.org/.
7. World Wide Web Consortium *Extensible Markup Language (XML) 1.0*, W3C Recommendation, http://www.w3.org (10-February-1998).
8. D. Schenck and P. Wilson. *Information Modeling the EXPRESS Way*. Oxford University Press (1994).
9. ISO 10303-11:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual.*
10. Kimber, W.E. *XML Representation Methods for EXPRESS-Driven Data*, Grant/Contractor Report 99-781, National Institute of Standards and Technology (November 1999). Available at http://www.nist.gov/sc4/wg_qc/wg11/n095/.
11. Flanagan, David. *Java in a Nutshell, 2nd Edition*. O'Reilly, Sebastopol California (May 1997).
12. ECMA-262, *ECMAScript Language Specification, 2nd Edition*. European Computer Manufacturers Association (August 1998).
13. World Wide Web Consortium *Document Object Model (DOM) Level 1 Specification, Version 1.0*, W3C Recommendation, http://www.w3.org (August 18, 1998).
14. Per Cederqvist, et al. *Version Management with CVS for CVS 1.9*. Signum Support AB. Available at http://www.gnu.org/manual/cvs-1.9/.
15. *Bugzilla* bug-tracking system. http://bugzilla.mozilla.org/.