SGML Application Development: Tradeoffs and Choices*†

Joshua Lubell and Lisa Phillips
Manufacturing Systems Integration Division
National Institute of Standards and Technology
Building 220, Room A127
Gaithersburg MD 20899 USA
Email: `apde@cme.nist.gov`

This paper appeared in the proceedings of SGML'96, Boston MA

November 27, 1996

Abstract

Developing SGML applications involves making choices driven by end user requirements and by the availability and functionality of third party SGML parsers, authoring tools, search engines, browsers, and data converters. Capabilities of HTML and the World Wide Web should factor into these decisions as well if users are geographically dispersed or have diverse computing platforms. SGML application developers typically build some or all of the following components: a DTD; legacy data conversion tools; a DTD-tailored authoring environment; a document repository; browsing and searching interfaces; and tools for producing formatted output. For each component, we discuss design and implementation alternatives, the approach we decided to use in building our SGML environment for authoring and accessing STEP product data exchange standards, and our rationale for choosing that approach.

Keywords: SGML; STEP; DTD; data conversion; authoring; search engine; formatting; Tcl; HTML

---

*Trade names and company products are mentioned in the text in order to adequately specify experimental procedures and equipment used. In no case does this identification imply recommendations or endorsements by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

†This work is funded by the National Institute of Standards and Technology's Systems Integration for Manufacturing Applications (SIMA) program, a part of the US government's High Performance Computing and Communications initiative.

1 Introduction

Developing software applications using the Standard Generalized Markup Language (SGML)[GOLD] involves making many design and implementation choices. These choices are driven partly by end user requirements and partly by the availability and functionality of third party SGML parsers, authoring tools, search engines, browsers, and data converters. Capabilities of Hypertext Markup Language (HTML) and the World Wide Web should factor into these decisions as well if the application's users are geographically distributed or have diverse computing platforms. Since choices made early in the development effort often have implications throughout the life of the application, it is important for application developers to be well-informed about their users' requirements and also about available software tools and technologies.

The National Institute of Standards and Technology is building an SGML environment for creating, storing, accessing, and publishing documents related to international standards[PHIL94]. This environment is part of an integrated software tool suite for accelerating the development and deployment of international product data exchange standards known collectively as "STEP" (the *St*andard for the *E*xchange of *P*roduct Model Data)[STEP1]. When complete, our environment will include document type definitions (DTDs) developed for STEP, tools to convert existing standards into SGML using our DTDs, a STEP-tailored authoring environment, and a tool for converting SGML-tagged documents into PostScript conforming to STEP's documentation guidelines[WELL]. The environment also contains a repository of SGML and non-SGML STEP documents indexed for efficient retrieval and accessible through the World Wide Web. Some portions of the SGML environment are already in place and are being used by developers of STEP standards.

The life cycle of a STEP document begins with the document's collaboratory development[1] and culminates in its being submitted to the International Organization for Standardization (ISO) for publication and added to the indexed repository for access by other standards developers. The SGML environment for STEP supports this entire life cycle and, therefore, makes a good case study for SGML application development. STEP (as well as standards development in general) is a good place to apply SGML for the following reasons:

- Standards function as "legal" documents and therefore must have a precise structure, including "boilerplate" text. SGML is useful in a standards development environment for rigorously defining and enforcing document structure.
- International standards documentation tends to have a long lifetime, and the

[1] *Collaboratory development* is a team effort, accomplished using a "virtual" research center where developers can communicate with colleagues, share data and resources, and obtain digital information from repositories, without regard to geographical location[KOUZ].

standardization process discourages frequent revisions to documents. Therefore, the initial effort required to represent standards in SGML has a high payoff.

- STEP defines an extensible architecture for sharing product model data. SGML provides a way to capture the semantics of this architecture.
- STEP's creators and implementors are geographically dispersed and work on a wide variety of computer systems. SGML provides a platform-neutral way to represent STEP documents such that their contents are accessible to anyone in the STEP community.

The rest of this paper discusses the tradeoffs and choices inherent in building an SGML environment, using our SGML environment for STEP as an example implementation. We begin by describing some general SGML application development concepts. Then we discuss the various design and implementation alternatives available for each component in an SGML environment, the approach we choose for our SGML environment, and our rationale for choosing that approach.

2 General Concepts

An SGML environment in its most general form consists of a collection of application components, each of which interacts with other application components and/or with an SGML document repository. The SGML environment may serve users who are producers or consumers of data, or both. Data producers and data consumers have different, and sometimes contradictory, requirements. Thus, whether an SGML environment's requirements are producer-driven or consumer-driven has a large effect on the environment's design, as we will see in Section 3.

Figure 1 shows the high level architecture of a generalized SGML environment. An SGML environment may contain up to five modules, depending on how much of the document life cycle is supported. These five modules are:

- An *authoring environment* for humans to create SGML documents.
- A *formatter* for converting SGML data into a human-readable form for hard copy or electronic delivery. Formatted output is useful both as an end product and as a means for providing feedback during the authoring process.
- A *document repository* for storing SGML data.
- A *data access services* module for handling requests for data from the repository. If the request is for a portion of an SGML document, the result is returned to the user by way of the formatter.

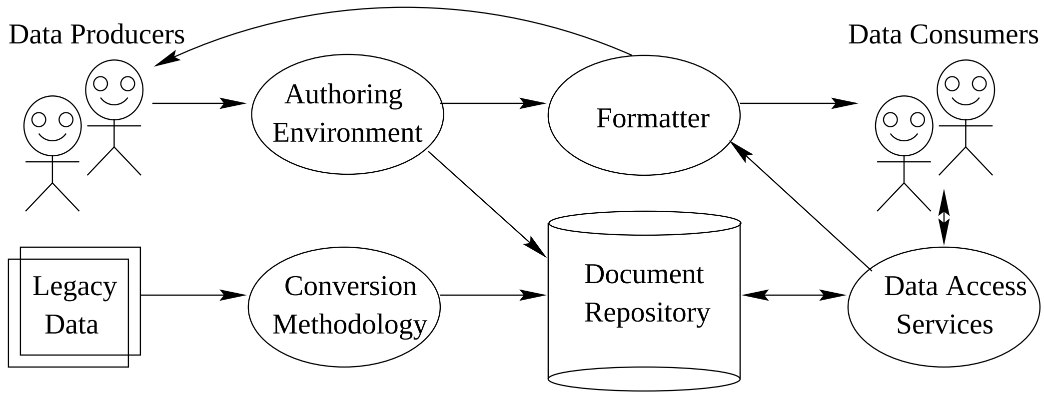

Figure 1: Generalized architecture of an SGML environment.

- A *conversion methodology* module to automate as much as is feasible the conversion of legacy data into SGML.

One very important component of an SGML environment not shown in Figure 1 is the DTD. This is because DTDs are involved in nearly all the modules of an SGML environment. As Figure 2 shows, SGML application components often act on a document instance by combining it with a DTD and an SGML declaration[2] to form a valid SGML document, feeding the document through an SGML parser, and performing some application-specific tasks on the parser output. The SGML parser output is a representation of the document's Element Structure Information Set (ESIS)[3], which describes the data as it is structured according to the DTD, but without any description of the document instance's actual markup. The ESIS is useful for SGML processing tasks that can be performed without any knowledge of the original markup. SGML application components performing these kinds of tasks are known as *structure-controlled*. SGML application components that need to know the actual document markup are known as *markup-sensitive*. All of the SGML environment modules shown in Figure 1 are structure-controlled, except for the authoring environment which is markup-sensitive.

An acknowledged weakness of SGML is that there is no analog to ESIS for markup-sensitive application components. As a result, markup-sensitive application components' behavior is less standardized than that of structure-controlled application components. However, the HyTime Technical Corrigendum[4][TC] introduces *property sets*

[2]The SGML declaration provides information about the dialect of SGML being used such as the character set, delimiter characters, identifier lengths, etc.[TEI]

[3]See Attachment 1 of Appendix B of [GOLD].

[4]Although the HyTime Technical Corrigendum's primary purpose is to correct the HyTime[HYTIME] standard, it also extends the facilities of SGML.

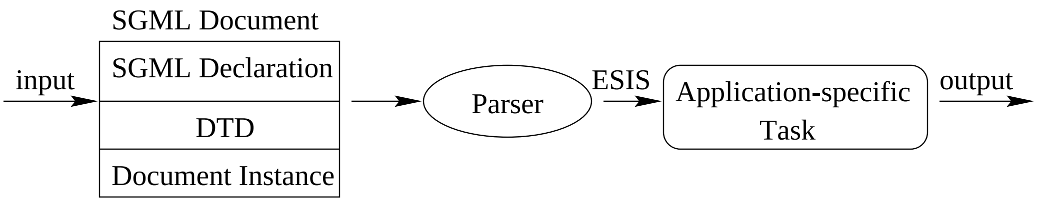

Figure 2: Typical SGML processing model.

and *groves* as a way for application developers to control the behavior of SGML parsers to a greater degree than was previously possible[5], which will result in more standardized behavior for markup-sensitive SGML application components.

3 Application Components in an SGML Environment

SGML implementors must typically select a DTD and/or build some or all of the application components discussed in Section 2. This section discusses the design issues involved for each of these tasks and describes the approach used in the SGML environment for STEP.

3.1 DTD

The DTD is arguably the most important piece of any SGML implementation. DTD design choices affect nearly every other application component. Two central issues are whether to use an existing DTD or write one's own and how complex a DTD to use. Using an existing DTD has several advantages. The most obvious advantage is that it saves the effort required to write and maintain a DTD. Another advantage of using an existing DTD is that an application developer can use whatever tools have already been developed for that DTD. A potential disadvantage of using an existing DTD is that it may not be application-specific enough. An existing DTD may meet some but not all of an application's requirements, resulting in a partial mismatch between the DTD and the application.

In order to compensate for such a mismatch, the application developer must either modify the DTD or hard-code the missing DTD structure into the application. Either of these

[5] An in-depth discussion of property sets and groves is beyond the scope of this document. The reader is encouraged to consult the HyTime Technical Corrigendum[TC] for additional information.

alternatives make the application harder to maintain. Modifying an existing DTD creates possible configuration management problems down the road. If the existing DTD's maintainer makes changes, then the application developer's modifications must be kept consistent. Hard-coding information about the document structure into the application makes the application less portable across computing platforms, negating some of the intended benefits of using SGML.

Creating a specific DTD for an application can be advantageous in that the DTD can be tailored to fit the application's requirements. However, building one's own DTD increases the application development and maintenance costs. Also, training and consulting costs are likely to be increased because, unlike industry-standard DTDs, there are no third party software tools tailored to custom-built DTDs, and there is no existing group of experts.

3.1.1 The STEP DTDs

STEP documents are developed according to a rigorous methodology[PALMER] dictated by the STEP community. The SGML environment for STEP must support STEP standards developers in using this methodology while at the same time ensuring conformance to STEP's documentation guidelines. However, the complexity of STEP documents makes it a challenge to meet both of these requirements. STEP's documentation guidelines dictate the order of presentation and the content allowed so that the documents properly reflect the STEP methodology. STEP documents contain front matter (title page, list of contents, foreword, introduction, scope, references to other standards, and a glossary of terms) followed by information requirements written in computer-interpretable modeling languages and annotated with descriptive text. They also include tables, figures, notes, and examples. A set of appendices contain additional items such as documentation for STEP implementors and graphical descriptions of information requirements.

In order to avoid the pitfalls associated with developing one's own DTD, we initially tried to find an existing DTD to use for representing STEP documents. However, there was no existing DTD that could adequately represent the STEP architecture so that an SGML-aware search engine could perform structure-based searches for STEP components. Also, no existing DTD contained the element structures needed to verify that a document tagged with that DTD conforms to STEP's documentation guidelines. Therefore, we decided to develop our own DTDs for STEP. We have created DTDs[PHIL96] for two document types in STEP: Application Protocols (APs) and Integrated Resources (IRs). APs are the self-contained, implementable parts of the standard. IRs are the reusable building blocks of information structures that are shared by the APs. STEP's documentation guidelines specify the documentation requirements for both of these document types at fine levels of granularity.

STEP standards have traditionally been produced using very little software support besides generic tools such as word processors and electronic mail. Therefore, conforming to the STEP documentation guidelines is typically a time consuming and error-prone effort. As a result, development of STEP standards is too expensive, and errors tend to slip into initial releases. The goal of the STEP DTDs is to reduce development costs and increase quality by capturing as many of the documentation requirements as possible in a standard, reusable, computer-interpretable form. In keeping with common SGML DTD design principles, explicit formatting requirements are not captured in the STEP DTDs. However, the DTDs provide an unambiguous representation of all required and optional content in STEP APs and IRs, the order and frequency with which the elements corresponding to that document content can occur, and all of the necessary relationships between the elements. Because of the specificity of the documentation guidelines in all of these areas, the DTDs are also highly granular and complex.

The high level of granularity of the STEP DTDs enables us to specify a lot of application-specific behavior in an implementation-independent manner, and also maximizes our ability to leverage third party SGML software tools. However, it makes the DTDs hard to maintain and also potentially makes them harder for a STEP part editor to use with SGML authoring software. Our challenge, therefore, is to create an environment in which all content and structural requirements for STEP documents is enforced without making the DTDs prohibitively difficult to use.

In an initial response to this challenge, we developed two AP DTDs: one for editing STEP APs from scratch, and the other for publishing existing STEP APs. A filter can then be implemented to translate documents written using the AP-editing DTD into documents conforming to the AP-publishing DTD using an approach similar to the one discussed in Section 3.6[6]. We were able to use this "two DTD" approach because much of the content in an AP can be derived from other content in the document or is common to all APs. Examples include such items as boilerplate text and required cross references. When editing an AP, the author should only be concerned with providing the minimum amount of information needed for an application to derive the remaining document content. The AP-editing DTD, therefore, only represents this minimal subset of the AP. The rest of the content will then be generated by the filter according to the AP-publishing DTD, which represents all of the document content including redundancies. By using separate DTDs for editing and publishing, we are able to ease the burden on document authors while not sacrificing the power of SGML in codifying and enforcing the STEP documentation guidelines. We also prevent inconsistencies by eliminating the opportunity for authors to specify redundant information.

The following example illustrates the relationship between the AP-editing and

[6]We have not yet implemented the filter, although we plan to do so in the near future. The STEP AP developers currently using our SGML environment have existing documentation for most of their AP in a word processor format and, therefore, chose to work directly with the AP-publishing DTD.

AP-publishing DTDs. Consider the DTD fragment from the publishing DTD modeling the "Units of Functionality" sub-clause of an AP[7]:

```
<!ELEMENT UoFs.SubC - - (UoF.SubC.Intro.Text,
 UoFs.List,
 UoF.CL3+)>

<!ELEMENT UoFs.List - - ( UoF.List.Item+ ) >
<!ATTLIST UoFs.List Ordered.List (Y|N)  "N" >

<!ELEMENT UoF.List.Item - o EMPTY >
<!ATTLIST UoF.List.Item UoF.Name.Linkend CDATA #IMPLIED
-- this value should be equal to a Name attribute on a UoF.CL3 element -->

<!ELEMENT UoF.CL3 - - ( #PCDATA) >
<!ATTLIST UoF.CL3 Name CDATA #REQUIRED >
```

A Units of Functionality sub-clause contains introductory text, followed by a list of the names of the UoFs, followed by descriptions of each UoF. Since the introductory text is fixed, the element representing it, `UoF.SubC.Intro.Text`[8], can be omitted from the AP-editing DTD. Since the list of UoF names can be generated by an application, the `UoFs.List` and `UoF.List.Item` elements and their attributes can also be excluded from the AP-editing DTD. The portion of the editing DTD corresponding to the AP-publishing DTD fragment above is the following:

```
<!ELEMENT UoFs.SubC - - (UoF.CL3+) >

<!ELEMENT UoF.CL3 - - ( #PCDATA) >
<!ATTLIST UoF.CL3 Name CDATA #REQUIRED >
```

3.2 Conversion of Legacy Data

Conversion of legacy data can be one of the most difficult and challenging components in an SGML environment. This is because most word processing and desktop publishing systems are based on visual formatting rather than logical markup, and the few tools that do support logical markup tend not to require authors to conform to a particular document structure. It is desirable to automate conversion as much as possible, particularly for large volumes of legacy data, but two problems often arise:

- Legacy documents in the same format with the same look and feel may have different internal representations. For example, two documents written using the

[7] It is not necessary for the reader to know what Units of Functionality are in order to follow this example. Also, the actual declarations in the DTDs were simplified for this example in order make it easier to follow.

[8] Its element declaration is not shown in the example because of space limitations.

same word processor might look the same even though one was created using styles and the other was not.

- The legacy data may exist in multiple formats, requiring that a different converter be implemented for each format.

If all legacy documents were created using uniform styles, conversion could be fully automated by writing programs to handle each document format. On the other hand, if styles are not applied uniformly, automation becomes more difficult.

When inconsistent use of styles in the legacy documents makes full automation impossible, some combination of automation and manual tagging is required. Individuals performing manual tagging should ideally have familiarity both with SGML and with the subject matter in the legacy documents. If those tasked to do the manual tagging lack expertise in either area, then the appropriate training needs to be provided.

3.2.1 Converting STEP Legacy Data

Converting legacy STEP AP and IR documents into SGML is a challenging endeavor. Legacy AP and IR documents exist in LaTeX[LAMP] as well as in more than one version of WordPerfect. There are also AP documents nearing completion that have been written in Word. Furthermore, although style templates are available to STEP part editors for some of these formats, some document authors modify the templates or lack training needed to use them properly. As a result, automating legacy data conversion has been difficult and we have had to rely heavily on manual tagging. We have decided to out-source the conversion of the remaining WordPerfect and Word STEP documents. We hope to be able to use an auto-tagging tool developed in-house to partially automate the LaTeX document conversion[WIL96A]. What we cannot automate, we will do in-house manually or out-source.

3.3 Authoring Environment

Because an SGML authoring environment is markup-sensitive and not structure-controlled (see Section 2), its functionality is only loosely governed by the SGML standard. All that the standard requires is that the authoring environment somehow enforce the document structure as specified in the DTD. As a result, any of the following alternatives can comprise an SGML authoring environment:

- A dedicated SGML authoring tool (designed for creating SGML documents and nothing else);

- An SGML add-on, i.e., a text editor or word processor with added functionality to support SGML authoring;
- A generic text editor or word processor and a stand-alone SGML parser.

All three options provide a satisfactory means of validating an SGML document. However, they differ in their editing features, ease of configuration, and ease of use.

Dedicated SGML authoring tools, as well as some SGML add-ons, provide such useful editing features as context-sensitive search and replace, traversal based on a document's tree structure, convenient commands for inserting markup, and the ability to selectively expand or collapse elements and entity references in the text editing window. However, these features tend to steepen the learning curve for non-SGML experts, and most first-time SGML document editors find these tools difficult to use if they are not customized for a particular DTD. Also, dedicated SGML authoring tools provide a lot of overhead if one is making only minor changes to an existing SGML document.

An SGML add-on for a particular word processor or text editor is a particularly attractive option for document authors who already use that word processor or text editor for other tasks. While it might not have all of the SGML-specific features of a dedicated SGML authoring tool, it is easier to learn how to use. Also, SGML add-ons for WYSIWYG ("What You See Is What You Get") word processors permit users to see their document with formatting while they are editing it, although this requires that the DTD author supply a mapping from the DTD elements to word processor styles. A disadvantage of SGML add-ons is they have only recent begun to appear in the marketplace, so their track record is less proven than that of dedicated SGML authoring tools.

A generic text editor or word processor is probably easiest to use when cutting and pasting from an existing SGML-tagged document because the SGML markup is already provided in the existing document. This alternative is also ideal when the document consists of mostly text and few tags, and little SGML understanding is required. The main disadvantage, however, is that the user is not at all guided or constrained by DTD-specified document requirements while editing, and errors can easily and frequently be introduced. Also, there is no support for facilitating SGML-specific tasks such as inserting markup, setting attribute values, etc.

Depending on the granularity and complexity of the DTD, manually tagging an SGML document corresponding to that DTD can be a very laborious and time-consuming task using any type of editor. An SGML editor customized for a specific DTD, however, can be used to alleviate some of the tedium in tagging SGML documents. A customized editing application can be used to provide such capabilities as prompting the user for required content or providing "pick-lists" from which the user can select otherwise long or hard-to-type attribute values.

Generic text editors and word processors do not have any understanding of the SGML encoding in the documents. Therefore, they cannot provide such DTD-specific behavior. However, both dedicated SGML authoring tools and SGML add-on software are SGML-aware (at varying levels) and therefore can be customized to enable DTD-specific behavior by means of a scripting language or an application programmer interface (API). Customizations are especially beneficial when they are successful in hiding some of the SGML details from the user or reducing the user's tagging effort.

3.3.1 SGML Authoring for STEP

The STEP SGML authoring environment requires tools which are robust enough to handle large, complex DTDs while hiding as much of the DTD complexity as possible from the user. Also, because most STEP document authors are novice SGML users, a friendly, easy-to-use user interface which does not burden the user with excessive SGML tags is preferable[9]. Finally, we want to give our users the freedom to use the software tools of their choosing to produce STEP documents, as long as those tools produce valid SGML with respect to the STEP DTDs.

Therefore, our approach is to give users complete freedom in choosing how they want to produce their documents while, at the same time, making STEP-customized options available to them. We encouraged selected vendors of dedicated SGML authoring tools to distribute versions of their products hard-wired to our DTDs to the STEP community at a low cost. One such vendor is willing to distribute a version of their SGML authoring tool for STEP with user interface customizations we will supply. Also, since many STEP authors already use word processors to edit their documents, we plan to provide mappings from the STEP DTDs onto style templates for popular word processors. This way, users can use word processor SGML add-ons without having to develop the required mappings themselves.

3.4 Document Repository

The document repository is the place where SGML data is stored. Depending on an application's requirements, the repository may be a computing platform's file system, or it may be a relational or object-oriented database. If the repository contains a large amount of data, and if fast access is required, the repository should be indexed using the element structure of the application's DTD. An SGML application's repository requirements depend on whether it is a producer of SGML documents, a consumer of SGML documents, or both. SGML document producers need to easily be able to store the

[9] Another reason for providing a user interface that is as straightforward as possible is that STEP document authors are a geographically distributed group, making them expensive to train in person.

documents they create in the repository. SGML document consumers, on the other hand, are more concerned with fast and efficient access to information in the documents[10].

Another issue worth considering when designing a repository for an SGML environment is whether the repository will contain any non-SGML data. If so, then the repository's search engine needs to be versatile enough to process SGML-structured as well as unstructured, full text queries.

Yet another issue to consider in designing a repository is the programmer-level interface. There should be a set of functions specific to the SGML environment for accessing the repository, and these functions should be easy to invoke from the source code of any SGML application components that use the repository. These functions form a necessary layer between a repository's generic API and the applications.

3.4.1 The SGML Repository for STEP

Since the SGML environment for STEP supports the entire life cycle for STEP documents, it is both an SGML document producer and an SGML document consumer. The SGML environment's authoring and legacy data conversion application components produce new SGML documentation. Its browsers and search interfaces facilitate the "consumption" and reuse of this documentation. In order to satisfy the SGML environment's producer-driven requirements, the repository should store data in its native format, i.e. as ASCII files containing SGML documents. In order to satisfy consumer-driven requirements, the data repository needs to maintain indices for these files reflecting the element structure defined in the DTDs. Also, the repository needs a search engine whose performance scales up well enough so that access remains fast when there are tens of thousands of pages worth of documentation indexed.

The SGML repository for STEP is contained in the Application Protocol Information Base (APIB), an on-line repository under development at NIST for users and developers of STEP. The APIB currently contains a subset of STEP IRs and APs. When it is complete, the APIB will also contain all IRs, additional APs, other STEP documents describing representation and implementation methods, and additional information useful to the STEP community such as schedules, issue logs, and issue resolutions. The APIB's contents are indexed for efficient access using *Pat*[OTC], a commercial text retrieval engine. Each document in the APIB is stored in its native format. Translation for viewing or publishing purposes is done on demand, as will be discussed in Section 3.5. Other documents will be represented as plain ASCII text, or in SGML using less complex DTDs[11]. Pat uses the SGML markup in the APs and IRs to index their data. Thus users

[10] If SGML document producers need to maintain revision histories, then access is important for them too.

[11] We are considering using ISO's exchange DTD to represent the STEP documents describing representation and implementation methods.

can issue queries to the APIB referring to the SGML structures specified for STEP.

Figure 3 illustrates the architecture of the SGML portion of the APIB. SGML databases of STEP IRs and APs are indexed for use with the APIB's search engine. SGML application components communicate with these databases through a Tcl[OUST] binding that insulates the application components from Pat's internals. We chose Tcl as a programming language because it is flexible, portable, easy to use, and several Tcl extensions useful to the APIB had already been implemented. The Tcl binding has both generic and STEP-specific components. The generic component contains a Tcl extension[LUB95] for tokenizing and scanning output from Pat as well as Tcl commands for opening and closing databases and issuing queries. The STEP-specific component contains Tcl commands for performing queries specific to the STEP DTDs.

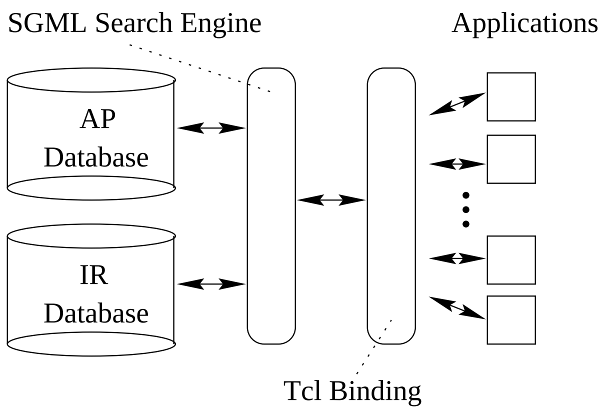

Figure 3: Architecture for SGML portions of APIB.

3.5 User Interfaces for Browsing and Searching

An SGML environment with a large data repository should include a user interface for accessing the repository's contents. The user interface may be a separate client application. However, unless the users are a small group who are all using the same computer system on a local area network, it is often more desirable to make the repository accessible through the World Wide Web for the following reasons:

- A web interface eliminates the need for users to obtain and install client software (other than a web browser).
- Using the web as a delivery medium eliminates the need for application developers to build and maintain separate user interfaces for multiple computer platforms.

- Technologies such as the Common Gateway Interface (CGI) standard for interfacing external applications with web servers[BLEE] and the Java[NIEM] programming language provide the means to build sophisticated Internet applications that can be accessed through a user's web browser.

Assuming that the web will provide a repository's user interface, the next issue is how to display the repository's contents to the user. One possibility is to display the data as HTML. All web browsers are capable of displaying HTML, and HTML meets the presentation requirements for many applications. However, since the data's original representation is SGML and not HTML, it must first be converted to HTML prior to being displayed. Although it is pretty straightforward to translate from SGML to HTML (see Section 3.6), HTML does a poor job displaying mathematical formulas, some foreign languages, and data with very precise presentation requirements. Also, HTML supports hyper-linking far less rich than the kinds of links that can be represented using the Hypermedia Time-Based Structuring Language (HyTime)[HYTIME] standard. Therefore, HTML is not always the best solution.

An alternative is to display the data as formatted SGML. This requires that the user have a web browser that is capable of displaying SGML, or a separate SGML viewer client set up as a helper application for the web browser. However, formatted SGML is a lot more versatile than HTML and is not limited in what it can represent. If the SGML viewer supports HyTime hyper-linking, then a user interface for browsing a repository could capture data relationships that would be difficult to represent using HTML alone.

3.5.1 Browsing and Searching STEP Documents

We have implemented a World Wide Web gateway using CGI for browsing STEP documents in the APIB. We use HTML rather than SGML for displaying the APIB's contents because HTML is adequate for most of our presentation requirements, and because third party SGML viewers are not yet readily available for all computer systems we need to support. CGI scripts generate HTML dynamically in response to requests for web pages. The APIB gateway uses HTML forms to obtain input data from the user. The input is then processed by the APIB gateway's CGI scripts[12]. The APIB gateway provides APIB access services to STEP AP developers through their web browser software, eliminating the need for AP developers to install APIB client software. It also eliminates the need for NIST to build and maintain separate APIB user interfaces for multiple computer platforms.

The APIB gateway provides an interface between a user's web browser and the APIB's

[12] World Wide Web gateways often use HTML forms in this way as a means of obtaining input for CGI programs.

data access services. This interface consists of a collection of CGI scripts written in Tcl implementing various data access services and an SGML-to-HTML translator for STEP as shown in Figure 4. These scripts use a Tcl CGI library[LIBES] for reading input from HTML forms and dynamically creating HTML. Each CGI script answers a particular type of user request specific to STEP. The requests are issued by means of entering information in a form and pressing a button to submit the request. The CGI script triggered by the form issues a query to the APIB and, using the query result, generates a new HTML page to display to the user. The new HTML page typically contains another form soliciting user input. [LUB96] discusses the design and operation of the APIB gateway in detail.

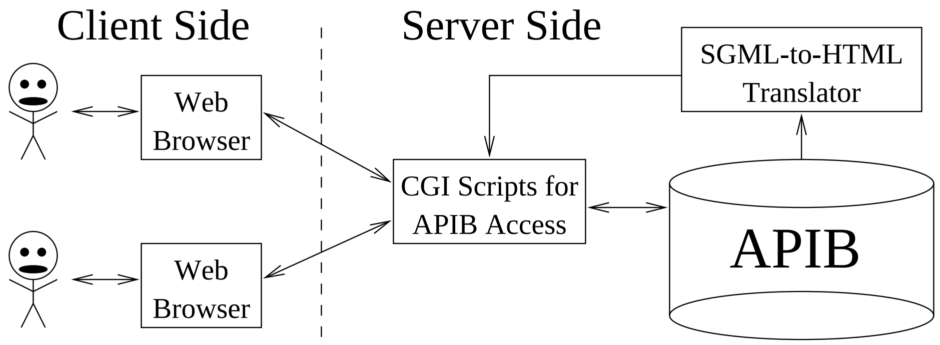

Figure 4: Architecture of the APIB gateway.

The SGML-to-HTML translator is used to convert SGML-tagged data from the source documents in the APIB into HTML so they can be displayed in a web browser. The translation is only necessary for query results containing actual text from the APIB documents. If the query result does not contain an actual piece of the document (for example, the result could be a list of object names satisfying the query), then the CGI scripts generate HTML output without use of the translator. The bidirectional arrow between the CGI scripts and the APIB and the unidirectional arrows from the APIB to the translator and from the translator to the CGI scripts in Figure 4 indicate this conditional use of the translator. The implementation of the translator is discussed in Section 3.6 and also in [LUB96].

3.6 Formatter for Publishing

SGML provides a means of representing the structures and relationships in documents. However, final hard-copy or electronic publication usually requires additional formatting information. Therefore, an application which converts documents tagged according to a given DTD into a format for presentation is required. Two key choices in this area are:

- The translation mechanism to be used;
- The output format of the document.

A translation mechanism for converting SGML documents into a presentation format for viewing or printing consists of:

- A mapping which specifies the relationships between SGML constructs and formatting information;
- The software tool which is used to generate the desired output by implementing those relationships.

The mapping can be specified using the Document Style Semantics and Specification Language (DSSSL)[DSSSL]. However, since at the time of this writing there is very little software available supporting DSSSL, a more practical short term solution is to use an application-specific mechanism for specifying the relationship between DTD constructs and formatted output. While the obvious disadvantage of this mechanism is that a different mapping is required for each application, the advantage of this mechanism is that it does exist now and works very effectively.

The software tool used to implement the mappings between DTD elements and format can either be specific to a public or industry-standard DTD or it can be configurable for use with any DTD. Using a DTD-specific SGML conversion tool is ideal when using the DTD for which the conversion tool is created. In some cases, this can actually eliminate the need to specify a mapping if the application developer uses the tool's default mapping. However, if no conversion tool specific to the desired DTD is available, then a DTD-general SGML conversion tool must be used. DTD-general conversion tools typically follow the SGML processing model shown in Figure 2. The tool requires that an SGML document be fed to a parser in order to obtain the ESIS, which in turn gets fed to the tool. The tool also requires a set of translation rules specifying the mapping between SGML constructs in the document and the output format.

A wide variety of output choices are available provided that the output format supports the SGML environment's presentation requirements. However, some output formats are easier to generate than others. For example, it is much easier to translate an SGML document into HTML than it is to translate it into RTF because HTML's syntax is more SGML-like than RTF. Also, output formats which support cross referencing, automatic section numbering, and other publishing niceties are good choices when paper is the delivery medium. Another consideration in selecting an output format is whether or not application-specific style sheets or templates are available in that particular output format. If such style sheets or templates are available, then much of the formatting work is handled therein, and less work is required to implement the translation mechanism.

3.6.1 SGML Formatting for STEP

In the SGML publishing environment for STEP, both paper and electronic delivery are supported. The translation mechanism for both is SGMLSpm[MEGG], a perl[VROM] library for building structure-controlled SGML applications. SGMLSpm includes a program that converts an SGML document's ESIS into an output format as specified by a collection of rules written in perl. We have written rules to translate STEP documents from SGML to LaTeX for hard copy output and to HTML (as was mention in Section 3.5) for electronic output.

The hard copy output of the SGML conversion tool is intended both for STEP AP developers to generate for review as the document is being developed and for delivery to ISO for final publication. Therefore, it is important to ensure that the publishing environment fully supports the specific formatting requirements for STEP documents as specified by the documentation guidelines. Because the LaTeX styles in accordance with STEP's documentation guidelines already exist[WIL96B], and because tools are readily available to produce high quality output from LaTeX, LaTeX was selected as the output format for SGML-tagged STEP documents. Also, efforts have been made to coordinate STEP LaTeX style sheet development with development of STEP DTDs. This, combined with the LaTeX document preparation system's support for cross referencing, automatic numbering, and generation of tables of contents, simplifies the mapping process as well as ensures that the resulting output is accurate and consistent.

Because electronically-delivered STEP documents are not subject to STEP's documentation guidelines, the requirements for electronic publishing are less stringent than for printing. Therefore, it was fairly straightforward to write the perl rules used in the SGML-to-HTML translator discussed in Section 3.5.

4 Concluding Remarks

The SGML, HyTime, and DSSSL standards are evolving, and third party SGML software tools are becoming more prevalent and powerful. As a result, new capabilities are becoming available for SGML environments. We are particularly interested in the following recent and ongoing developments:

- *Architectural form processing support.* Now that the SP[CLARK] SGML parser can process architectural forms (as specified in the HyTime standard), we are looking at defining document architectures for STEP and redesigning our DTDs so that they use these STEP document architectures. This should result in our DTDs being easier to maintain in the future, and it may also enable us to build more efficient and robust applications using them.

- *DSSSL.* The DSSSL standard is now solidified, and software tools supporting DSSSL are starting to emerge. We therefore are considering rebuilding our SGML-to-LaTeX and SGML-to-HTML translators using DSSSL. We also hope to be able to use DSSSL in implementing the filter to convert APs written using the AP-editing DTD into documents conforming to the AP-publishing DTD. Using DSSSL would enable us to separate our presentation requirements from our transformation specifications, making it easier for us to support multiple output formats.
- *Groves and property sets.* When SGML authoring tools start supporting groves and property sets as defined in the HyTime Technical Corrigendum (see Section 2), we will be able to express some of our STEP-specific user interface customizations in SGML. This will reduce the need for us to work with proprietary scripting languages and APIs.

Although much work remains to be done, our SGML environment for STEP is already having an impact on AP development. Ambiguities and inconsistencies in the STEP documentation guidelines have been exposed through the effort of developing the STEP DTDs. Since the STEP DTDs provide a complete and unambiguous specification of AP and IR document structure, AP and IR documents authored in SGML will be free of the inconsistencies and documentation guidelines violations that plague the APs and IRs in the STEP initial release. The APIB World Wide Web Gateway enables AP developers and STEP implementors to easily access the existing body of information in STEP, without having to laboriously search through individual standards documents. As we continue adding new capabilities to our SGML environment, we will undoubtedly gain additional insights regarding SGML application development.

Portions of our SGML environment for STEP are available to the public[13] through the "Application Protocol Development Environment" project page on the NIST Manufacturing Systems Integration Division web server at http://www.nist.gov/msid/.

The authors thank this document's NIST reviewers — Mary Mitchell, Peter Wilson, and Willie Rogers — for their insightful comments.

References

[BLEE] Tim Berners-Lee and Daniel Connolly, *Hypertext Markup Language - 2.0*, Internet Engineering Task Force, HTML Working Group, RFC 1866, November 1995.

[13] The STEP DTDs are available for download. The APIB gateway discussed in Section 3.5 is also available. However, because the STEP documents in the APIB are copyrighted by ISO, access to these documents is restricted. Requests for further information should be directed to the authors.

URL: ftp://ds.internic.net/rfc/rfc1866.txt

[CLARK] James Clark, *SP SGML Parser*, version 1.1.
URL: http://www.jclark.com

[DSSSL] International Organization for Standardization, *Information technology—Processing languages—Document Style Semantics and Specification Language (DSSSL)*, ISO/IEC DIS 10179, 1996.
URL: http://www.ornl.gov/sgml/wg8/wg8home.htm

[GOLD] Charles Goldfarb, *The SGML Handbook*, Oxford University Press, 1992.

[HYTIME] Fujitsu Open Systems Solutions and TechnoTeacher, Inc., *HyTime Application Development Guide*, Version 1.2.4, February 1996.
URL: http://www.techno.com

[KOUZ] Richard T. Kouzes, James D. Meyers and William A. Wulf, *Collaboratories: Doing Science on the Internet*, Computer, Vol. 29, No. 8, August 1996.
URL: http://www.wvu.edu/r̃esearch/

[LAMP] Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, second edition, 1994.

[LIBES] Don Libes, *Writing CGI Scripts in Tcl*, Proceedings of *Tcl/Tk Workshop 96*, Monterey, CA, July 10-13, 1996.
URL: http://www.nist.gov/msid

[LUB95] Joshua Lubell, *Patparse*, National Institute of Standards and Technology, initial release (1995).
URL: http://www.nist.gov/sc4/tools/nist/patparse.tar

[LUB96] Joshua Lubell, *The Application Protocol Information Base World Wide Web Gateway*, National Institute of Standards and Technology, NISTIR 5868, July 1996.
URL: http://www.nist.gov/msid

[MEGG] David Megginson, *SGMLS.pm: A perl5 class library for use with the SGMLS and NSGMLS parsers*, University of Ottawa, version 1.03.
URL: ftp://aix1.uottawa.ca/pub/dmeggins/

[OTC] Open Text Corporation, *System Integration Guide*, Release 5.0, 1994.

[NIEM] Pat Niemeyer and Josh Peck, *Exploring Java*, O'Reilly, May 1996.

[OUST] John Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[PALMER] Mark Palmer and Mitchell Gilbert, *Guidelines for the Development and Approval of STEP Application Protocols, Version 1.2*, ISO TC184/SC4 Technical Report, N433, April 5, 1996.

[PHIL94] Lisa Phillips and Joshua Lubell, *An SGML Environment for STEP*[14], National Institute of Standards and Technology, NISTIR 5515, November 1994.
URL: http://www.nist.gov/msid

[PHIL96] Lisa Phillips, *STEP Document Type Definitions: Design and Architecture*, National Institute of Standards and Technology, to appear as a NISTIR, 1996.
URL: http://www.nist.gov/msid

[STEP1] International Organization for Standardization, *Industrial automation systems and integration—Product data representation and exchange—Part 1: Overview and fundamental principles*, ISO 10303-1, 1994.

[TC] International Organization for Standardization, *Hypermedia/Time-Based Structuring Language "HyTime" Draft Corrigendum*, ISO/IEC 10744 Technical Corrigendum 1, 1996.
URL: http://www.ornl.gov/sgml/wg8/wg8home.htm

[TEI] C.M. Sperberg-McQueen and Lou Burnard, *Guidelines for Electronic Text Encoding and Interchange*, ALLC/ACH/ACL, May 1994.
URL: http://etext.virginia.edu/TEI.html

[WELL] Joan Wellington, *Supplementary directives for the drafting and presentation of ISO 10303*, ISO TC184/SC4 Technical Report, N432, April 5, 1996.
URL: http://www.nist.gov/sc4/howto/methods/supp_dir/

[WIL96A] Peter Wilson, *LTX2X: A LaTeX to X Auto-tagger*, National Institute of Standards and Technology, to appear as a NISTIR, 1996.
URL: http://www.nist.gov/sc4/editing/latex/programs/ltx2x

[WIL96B] Peter Wilson, *The LaTeX Package Files User Manual*, National Institute of Standards and Technology, to appear as a NISTIR, 1996.
URL: http://www.nist.gov/sc4/editing/latex/current

[VROM] Johan Vromans, *Perl 5 Desktop Reference*, O'Reilly and Associates, February 1996.

[14]Also appeared in *SGML '94 Proceedings*, Graphic Communications Association, Vienna VA, November 1994.