SGML Architectures:

A More Abstract Formalism for
Interchanging Structured Information

Steven R. Newcomb

Abstract

By itself, SGML offers a means whereby the syntax and semantic associated with a given information construct (an element) can be expressed: an element definition in a DTD. Whenever such an information construct must be used in a document, the corresponding DTD must allow this construct to appear there. Problems can arise, however, when it becomes necessary for whole sections of various DTDs to resemble each other. While the method of using master DTD fragments that are inserted into actual DTDs is usually workable, this method is often needlessly and cripplingly rigid.

The use of architectures, rather than sets of DTD fragments, for formalizing the structures needed to permit interchange of information common to several document types, permits the architects of DTDs to retain full and optimum control of SGML's validation apparatus, while still guaranteeing information interchangeability. DTDs can be permitted to change in any way that does not violate the constraints imposed by the architecture. Each architecture can be developed in such a way that it imposes no constraints on document structure that are not actually necessary for interchange of the information with which the architecture is concerned. The DTDs themselves can then be used to impose as much further constraint on actual document instances as desired, so none of the constraining power of SGML is lost to any particular document type simply because of a need to allow the interchange of information elements whose general outline must appear in more than one type of document.

About the author:

Steven R. Newcomb, Ph.D. has been the Vice Chairman of the American National Standards Institute's X3V1.8M committee since its inception in 1986, and the co-editor (with Charles F. Goldfarb) of the HyTime Hypermedia/Time-based Structuring Language (now known as ISO/IEC International Standard 10744:1992). Dr. Newcomb was the founding Chairman of the SGML Users' Group's Special Interest Group on Hypertext and Multimedia ("SGML SIGhyper"). He founded and chairs the Conventions for the Application of HyTime (CApH) activity of the Graphic Communications Association (GCA). He serves as founding Chairman of the GCA's annual International HyTime Conference.

Dr. Newcomb is President TechnoTeacher, Inc., a software development and consulting firm emphasizing SGML/HyTime tools and application development environments. TechnoTeacher's "HyMinder" object-oriented C++ class is used in hypermedia research projects and for software product development worldwide. TechnoTeacher has been honored with the Graphic Communications Association's "Tekkie" award "for meritorious contributions to the technical documentation industry."

A music theorist by training, Dr. Newcomb was the founding Associate Director of the Florida State University Center for Music Research in Tallahassee, Florida. He continues to play an editorial role in the development of the ISO Standard Music Description Language (ISO/IEC DIS 10743, "SMDL").

Dr. Newcomb has spoken on HyTime at many SGML-oriented and other conferences, including CD-ROM, Hypertext, ACM SIGGRAPH, and ACM SIGDOC, as well as several of the Graphic Communications Association's TechDoc, SGML, International Markup, and Information Technology Week conferences. He has given many HyTime tutorials in North America, Europe, Japan, Singapore, and Australia. For the last several years, HyTime Workshops are being team-taught by both Dr. Newcomb and Dr. Michel Biezunski, Director of High Text S.A.R.L. (Paris, France).

Steven R. Newcomb, President
TechnoTeacher, Inc.
(courier: 3800 Monroe Avenue, Pittsford, NY 14534-1330 USA)
P.O. Box 23795
Rochester, New York 14692-3795 USA
direct +1 716 389 0964
main +1 716 389 0961
fax +1 716 389 0960
Internet: srn@techno.com
FTP: ftp.techno.com
WWW: http://www.techno.com

Introduction

SGML has always offered the means whereby the syntax and semantics associated with a given information construct (an element type) can be expressed; these are expressed in element definitions in document type definitions (DTD). Differing document types may contain some similar or identical element definitions, and sets of software applications can be made to contain or use similar or identical software modules for processing such similar or identical element types. In this way, anyone who controls some set of DTDs can heighten the application-neutrality of the information contained in documents conforming to those DTDs, save money on software development, and reduce expensive confusion in general, by maximizing the generality of each information construct (element type), and by avoiding, insofar as possible, any duplication of semantics which do not also duplicate syntax. However, until quite recently, with the advent of the HyTime (ISO/IEC 10744) international standard, there was no agreed-upon formalism for the expression of similarity in structure and semantics. Now these things can be expressed formally, and enforced, at least to some extent, automatically, in a new, more abstract kind of document type definition called a "meta-DTD." A meta-DTD describes the structure and semantics of a class of documents which therefore conforms to an "SGML architecture."

The need for modularity, consistency, and reusability is addressed in similar ways in the context of good object-oriented system design, and the similarity between SGML element types, on the one hand, and object classes, on the other, is not coincidental. The notion of "inheritance" of the structure and functionality of one class by another is at the heart of the innovation of SGML architectures; SGML architectures make it possible for element types to "inherit" the characteristics of the meta element types defined in SGML architectures.

SGML Architectures vs. Current Practices

In the absence of SGML architectures, syntactical similarities between constructs used in some set of DTDs could be represented and enforced by SGML's parameter entity definition/reference feature. This permits any arbitrary portion of a DTD, such as an attribute definition list, to appear in multiple places by a text substitution mechanism. Another tool for enforcing similarity in some set of DTDs is simply to create a single DTD entity which contains all of the elements used in all of the DTDs in the set. This single entity can define all of the element types common to all of the DTDs, and all of the other element types as well, so long as each document type is uniquely associated with a single element type as its document element. All documents conforming to all of the DTDs in the set refer to the same DTD entity, but each document's `<!DOCTYPE...` declaration, by specifying the root element type, specifies the particular DTD that the parser will automatically derive from the DTD entity. If both mechanisms are used (i.e., the parameter entity definition/reference feature and the `<!DOCTYPE...` declaration to specify the root element type), certain DTD maintenance advantages can be gained.

While the method of using common DTD fragments that are inserted into actual DTDs is usually workable, this method can bring with it some undesirable rigidity. Wherever a DTD fragment is inserted into a DTD, it is inserted verbatim. Even if parameter entities are not used, or if they are used in complex ways (such as having the inserted text of a parameter entity contain a reference to a previously-defined parameter entity), the insertion of DTD fragments can result in the propagation of unnecessary and unnatural constraints on the structure of documents, or, alternatively, less structural constraint than is desired by the architect, and than can be usefully validated by an SGML parser or SGML database engine. Moreover, the impact of a change in a parameter entity on any given DTD can be surprising and confusing to everyone but a computer.

The use of SGML architectures, rather than sets of DTD fragments, for formalizing the structures needed to permit interchange of information common to several document types, permits the architects of DTDs to retain full and optimal control of SGML's validation apparatus, while still guaranteeing information interchangeability. DTDs can be permitted to change in any way that does not violate the constraints imposed by the SGML architecture. Each SGML architecture can be developed in such a way that it imposes no constraints on document structure that are not actually necessary for interchange of the information with which the SGML architecture is concerned. The DTDs themselves can then be used to impose as much further constraint on actual document instances as desired, so none of the constraining power of SGML is lost to any particular document type simply because of a need to allow the interchange of information elements whose general outline must appear in more than one type of document.

HyTime is the Pioneer SGML Architecture

Like all SGML architectures, HyTime is formally described by a meta-DTD consisting of a set of meta element types, called *architectural forms.* In a document instance conforming to HyTime, any element that inherits the characteristics of a HyTime architectural form is recognized by a HyTime application by means of the value of that element's `HyTime` attribute, which is always the name of the architectural form. The name of the attribute, `HyTime`, corresponds to the HyTime architecture, and its value (e.g., `ilink`) corresponds to the ilink architectural form (the independent link meta element type) in the HyTime meta-DTD. (It has been aptly said that the value of an architecture attribute, such as `ilink` used as the value of a `HyTime` attribute, can be considered a "meta generic identifier" or "meta-GI.") Any element whose `HyTime` attribute's value is `ilink` is universally known to HyTime applications as an element that expresses a relationship of some kind, and that has certain HyTime-defined syntactic characteristics.

HyTime is a very important architecture for several reasons:

- It is an international standard: ISO 10744:1992. It is the only SGML architecture that has (so far) become an international standard.
- HyTime pioneers the whole idea of SGML architectures; it is the first such architecture, and the concept of SGML architecture was devised in order to allow HyTime itself, and the applications that HyTime informs, to be expressed in a formal manner. In its Annex C, the HyTime standard explicitly standardizes the meta-DTD formalism used to express not only HyTime itself, but also all other SGML architectures, regardless of whether they are derived from the HyTime architecture.
- HyTime establishes the syntax and semantics of several kinds of information that are essential for expressing information in general. It is far more general in scope than any particular application of it. (Other SGML architectures can be as application-specific or application-neutral as their creators desire.)

SGML Architectures Other Than HyTime

A limitless number of SGML architectures can be usefully developed. It now appears that business-context-specific SGML architectures, rather than DTDs, will be the most effective and, at the same time, the least costly method of allowing information interchange within any specific context, such as an entire enterprise, in which there are multiple types of documents. For the context of the Gourd Motor Company (no relation to any existing corporate entity), for example, a company-wide SGML architecture can be developed in which the value of a so-called `Gourd` attribute identifies the Gourd architectural form to which any given element is intended to conform. For example, if `Gourd=requisition`, then the element conforms to the constraints universally specified, throughout the Gourd organization, for requisition documents. This allows each division, department, or other subunit to define its own subclass(es) of Gourd-standard requisitions, each kind meeting all the syntactic and semantic requirements of all who need to use the information contained in it. In the case of a Gourd unit that, unlike any other Gourd unit, purchases radioactive materials, this kind of flexibility can be extremely desirable. In such a case, all the government and environmental paperwork can become part of the requisition document, organized in a fashion determined locally by that unit, but still processable by the purchasing department and all other concerned units at Gourd.

The notion that SGML architectures can and should be used to achieve enterprise integration has a number of interesting and profound implications for the information processing industry. Among these implications are:

- Many business-context-oriented meta applications, supported by reusable software modules, called *engines*, will appear. For example, a company-wide "Gourd engine" would be capable of handling all the ordinary tasks associated with elements conforming to all Gourd architectural forms, and would be used in all applications throughout the Gourd organization. Taken as a whole, such an engine could be described as an implementation of the Gourd meta application. HyTime engines, which may be thought of as implementing a "human civilization oriented" (rather than just "Gourd oriented") meta application, are early examples of such engines.

  Meta applications may be associated with any level of organizational hierarchy, and, in specific contexts, multiple engines may be used in the same application simultaneously. For example, a given element might have both a HyTime attribute and a Gourd attribute, each declaring what the element is in terms of the HyTime and Gourd architectures, respectively. The design of new SGML applications will no longer be a single layer of application specific software built upon supporting general purpose SGML software; instead, new applications will consist of aggregations of engines, of which SGML supporting software and HyTime engines are only two. Of course, there will also be some software in each application which is completely specific to an application, but the amount of such special software will be minimized by the use of engines.

  The mapping of an SGML architecture to an object oriented implementation is completely smooth and natural. A given element type becomes an object class that inherits all of the semantics (and associated processing characteristics) of all of the architectural forms it inherits from all desired SGML architectures. The existence of SGML architectures (sets of architectural forms) and their engine implementations (sets of object classes corresponding to the architectural forms) reinforce each other: software developers enjoy enhanced productivity by re-use of engine software wherever it is appropriate, and the use of SGML architectures in documents explicitly reminds them to do so. Information architects employ pre-existing SGML architectures because they enhance their own productivity, and because of the savings that will be realized when their architectures are implemented.

- Many new kinds of document validation will appear, all in the realms of all the applications and SGML architectures in which documents participate. SGML tools will be regarded as only the basic tools for information interchange, like other communications gear. Validation of a document instance for conformance with its DTD will be seen as a trivial, automatic process in the same league with (and as taken for granted as) the communications protocols that are used to transmit and receive data. Much greater emphasis will be placed on the document instance's conformance to its corresponding SGML architecture(s).

- The primary focus of information architects will shift away from DTD design and toward meta-DTD design. One aspect of their work will be the development of DTDs and meta-DTDs that conform to (are derived from) one or more meta-DTDs. The scope of information architecture projects will usually encompass all of the document types used in an entire context (such as all the documents used and/or produced by a large organization), rather than just a small number of document types associated with some particular subset of organizational functions.
- Document authors will gain control over many (but not all) aspects of the DTDs they use. This will not cause a problem because DTDs will not be used as the primary means of insuring information interchangeability; the meta-DTDs to which document instances will conform will provide that insurance. Document authors will thus overcome the frustration they frequently feel today when forced to use rigid DTDs that were not designed in light of all the requirements that authors actually face, especially when improving or correcting a DTD may require a level of political access that is unavailable to a lowly author. By watching the evolution of the DTDs that are controlled and actually used by document authors, information architects can evolve and tune their SGML architectures in response to widespread trends and good ideas, and thus make more (and more sophisticated) information more widely available throughout an organization. Document authors will use applications that permit them to tweak their DTDs interactively, perhaps even while documents are open for writing; this will greatly alleviate the frustration and distaste many authors now feel about using SGML. However, even though document authors will have more control over DTDs than they typically do today, SGML architects will be able to exercise far more control over authors than is possible for today's architects with SGML alone, because applications that accept data from authors will be able to validate open documents not only for conformance with their DTDs, but also with the SGML architectures (meta applications) to which they must conform. An SGML architecture is only syntactically (and only partially) described by its corresponding meta-DTD; just as HyTime does, an SGML architecture may impose many (and complex) semantic and syntactic constraints which are amenable to validation by architecture-specific engines and applications.
- As reliance on SGML architectures and business-context-specific software engines increases, there will be a concomitant shift of focus away from the comparatively rigid data typing and content modeling found in today's SGML DTDs. The beginnings of this trend can be seen in the HyTime architecture, in which, for example, the declared value of many important attributes is CDATA. A declared value of CDATA causes SGML parsers not to perform any validation on the corresponding attribute values; the validation of those attribute values will be performed by the HyTime engine later, after the attribute value has already been parsed. As reliance on the validation services offered by pure SGML parsers decreases, emphasis will shift to engine-based validation, some of which will be done by HyTime engines, and some of which will be done by engines that support other, additional SGML architectures such as the hypothetical Gourd enterprise-wide architecture.
- In general, SGML architectures impose fewer constraints on DTDs than they impose on instances. DTDs may or may not be used in such a way as to burden the SGML software (i.e., an SGML parser, etc.) with the task of verifying that document instances comply with the syntactic constraints imposed by an SGML architecture. The only purpose in validating a DTD for conformance to an SGML architecture is to determine whether the constraints imposed by

the DTD would prevent document instances from conforming to that architecture. Of course, since the DTDs will be increasingly controlled by authors, authors will be free to enhance the DTDs they use by imposing a variety of validation requirements that can be met by existing SGML and HyTime tools. For example, if an author decides, for consistency's sake, that performing a lexical check on the value of a CDATA attribute would be desirable, that author can add the necessary HyTime-defined "lextype" attribute to the DTD, and thereby cause any HyTime engine to perform that check automatically. (Of course, the SGML architect could make the same decision and impose this validation requirement upon all instances. Still, the idea to do so might originate from an author.)

The Loci of Control in Organizations

The distribution of control among the hierarchical levels of human organizations is a tricky balancing problem. Control which is too centralized reduces the initiative and adaptability of organizational subunits and individuals. Adaptability is survivability; overly centralized organizations court their own demise by reducing the value of their most important assets: the accumulated experiences of their personnel. On the other hand, control which is overly distributed makes organization-wide cooperation difficult or impossible. Opportunities afforded by combining the experiential and other assets of several subunits are can be missed, unless by some lucky accident all of the subunits involved spontaneously cooperate with one another. It is always an open question, in any given organization, whether the distribution of control occurs in an optimal fashion, and constantly changing conditions demand constant re-evaluation of the mechanisms whereby control is distributed.

Control, by definition, is the way in which decisions are taken and implemented, and decisions must always be based on information. The way in which information is structured both reflects and determines the way in which management (at whatever level) perceives the information on which decisions will be based. Control of information architecture is a fundamental aspect of management. For example, while it may not be a manager's job to maintain an organization's records, it is definitely the job of some manager to decide exactly how those records will be organized and maintained. The way in which responsibility for the structure of records is distributed throughout an organization, therefore, may reveal much about how control in general is distributed throughout that organization.

Some general observations follow from all this:

- All managers with responsibility for corporate records, or for the use of corporate records for the support of management decisions, must have some degree of control over the structure of those records. Without such control, they cannot be held responsible for the usefulness, accessibility, or current (or future) relevance of the information. All managers who have such control need at least some of the skills of an information architect.
- Regardless of whether they control the structure of corporate records, all managers need some awareness of the information architecture(s) used by the organization. Otherwise, they would be capable of accessing neither existing organizational policy, nor the case history on which management decisions must be based.
- Any corporate commitment to information technology, regardless of whether it is to traditional filing cabinets, to image archiving systems, or to sophisticated high-speed databases, must be made in such a way as to support the ongoing evolution of the organization's information

architecture, regardless of the organizational level at which evolutionary architectural changes are initiated. The adaptability of the structure of corporate records is a useful indicator of the adaptability and overall health of the organization.

- Responsibility for various aspects of the information architectures used in an organization must be distributable, and, more importantly, redistributable. Control over the structure of various corporate records must be permitted to change hands from time to time, and to undergo overall evolution in response to changing external and internal business conditions.

The ongoing evolutionary development and use of SGML architectures will help all organizations meet the above challenges more efficiently. Using SGML architectures, control over various aspects of the organization's information architectures can be formally distributed, and, when necessary, redistributed. In general, managers already have most of the information architecture skills they need. They have lacked only a formal, standard, and sufficiently flexible way to express and propose evolutionary changes in the way information is structured. The SGML architecture formalism set forth in Annex C of the HyTime standard will increasingly fill this need.