# Formal data models for SGML and HyTime

*Peter Bergström, EuroSTEP AB*
*Eliot Kimber, Isogen*



Peter Bergström
EuroSTEP AB
Vasagatan 38
SE-111 20 Stockholm
Sweden
Phone: +46-708-111 966
Fax: +46-708-111 965
E-mail: peter.bergstrom@eurostep.se
Web: http://www.eurostep.se

W. Eliot Kimber
Senior Consulting SGML Engineer
ISOGEN International Corp.
2200 N. Lamar St., Suite 230
Dallas
TX 75202
U.S.A.
Phone: +1 214-953-0004
Fax: +1 214-953-3152
E-mail: eliot@isogen.com
Web: www.isogen.com

## Abstract

The EXPRESS language of the STEP standard, ISO 10303, provides a powerful and standardized language for describing complex data models. The SGML family of standards define the formal data model of SGML documents (and other data types and data abstractions) using property sets. One goal of the STEP and SGML harmonization is to apply the more complete modeling power of the EXPRESS language to SGML and related standards by restating their existing data models using the EXPRESS language, in particular, the SGML property set and the HyTime property set. This paper presents the results of the initial effort to define these data models, first describing the EXPRESS formalism and then showing how it was used to express the structures defined by the corresponding property sets.

Included in this discussion are:

- The basic components of an express model
- The relationship between the EXPRESS syntax and graphical EXPRESS diagrams
- The design principles used
- The details of the resulting designs

## Introduction

The SGML standard began life simply as a syntax standard: it defined the rules for constructing and parsing character strings that represent structured documents. As originally formulated, the focus of the SGML design was on the string representation of documents: their syntax. The data model that underlies the abstractions represented by those strings was implicit in the definition of the syntax. The basic model was fairly straightforward: a document is a tree of elements and data characters. Elements can have attributes. Of course, the reality of the data model is much more complex than this: there are many details that are well described syntactically but were not always well defined as an abstract data model. Implementors and users could reasonably disagree about the abstract implications of particular syntax features. This was not too much of a problem in the early days when most SGML processing was essentially string processing: see a string, figure out what kind of token it is (start tag, end tag, character data), and do something based on the token. The fact that most SGML tools were properietary and standalone help too: there was little demand for integration and therefore little presure to define an implementation-independent data model for SGML documents.

However, as SGML processors became more sophisticated and as standards for doing SGML processing were defined (DSSSL and HyTime in particular), it became clearer that SGML needed to have a formal

data model definition so that these processors could be rigorously and unambiguously defined and so that they were defined in terms of the same standardized data model. Because the people developing these standards were not data modeling experts and because time was of the essence, they developed their own simple data modeling formalism sufficient to, and optimized for, the task of modeling and processing SGML documents. This formalism, "property sets", is based on a simple data model of nodes with properties organized into graphs. This simple data model, "groves," has some special features to optimize them for representing structured documents. However, the grove model is sufficiently general that groves can be used to represent any kind of structure. While the grove model is useful as an unifying base data model, no claims were made about the power of property sets as a data modeling formalism: we explicitly made them no more powerful than we needed them to be for the task at hand.

At the same time that SGML and its related standards were being developed, standards in the manufacturing area and elsewhere were maturing, in particular the STEP family of standards. As the documentation and manufacturing domains matured, the requirement to be able to have document data and manufacturing ("part") data more closely tied together grew stronger. It had always been a requirement, but the tools and infrastructure necessary to satisfy this requirement simply hadn't existed: documents were fundamentally paper, parts information was fundamentally databases. This requirement was expressed from both directions: manufacturers and engineers wanted to integrate documents into their information worlds and technical writers wanted to integrate parts data into their information worlds.

From the SGML perspective, once it was realized what we actually had by having defined a more formal data model for SGML and what that enabled in terms of processing and information integration, it became clear that the full integration of structured documents with product data would be possible. The reason: there were now comparable levels of abstraction and modeling on both sides. For SGML there were property sets and groves, for STEP there was EXPRESS and STEP-based repositories. We also realized that the simplicity of property sets was a limiting factor in the wider adoption of thet SGML abstractions: it was simply unreasonable to expect people to use property sets in place of more complete modeling languages, languages developed by and for data modeling experts. We realized that if we could simply define the details of the mappings between these two comparable formalisms, it should be possible to view the same data in terms of either formalism and therefore achieve seamless data integration and system interoperation.

One of the first steps in defining the mapping between the formalisms is to use the formalisms of each domain to model the abstractions in the other. This paper describes our experience in using the EXPRESS data modeling language to represent the formal data models of SGML and HyTime, which are normatively defined using property sets. We feel that the results demonstrate both the usefulness of EXPRESS as a modeling tool and provide more-accessable renditions of the SGML and HyTime data models, a keep prerequisite for accurate and fully-functional implementation of supporting systems. These models also make the SGML and HyTime models available to STEP-based tools and processes so that they can, if appropriate, directly manage SGML-based data and take advantage of HyTime-defined hyperlinking and addressing (regardless of whether or not they involve SGML or XML documents).

The end result is that each world gets to take advantage of the best of the other world's facilities while helping to better illuminate weaknesses that can be fortified by applying new techniques and tools.

## *Introduction to the EXPRESS Modeling Language*

The EXPRESS language, defined in ISO 10303 Part 11, is a data modeling language that provides a fairly rich set of facilities for defining complex data types. It is based on the entity-attribute modeling approach. The EXPRESS language is used to define *entity types*. Entity types are organized into type and supertype hierarchies. Each entity type provides a set of attributes. For example, to model a business, you might define the entity type "person", which as attributes such as "name", "employee number", and "manager". The "name" and "employee number" attributes might be string values. The "manager" attribute would be a pointer to another "person" entity. Note that the term "entity" used in the EXPRESS context has nothing to do with SGML and XML entities.

The EXPRESS language includes a rich constraint language that lets you define constraints on the values of attributes and the constitution of data populations that conform to the EXPRESS model. This formal constraint language is one of the things that distinguishes EXPRESS from similar modeling languages, such as UML.

The EXPRESS language is very general, so that it can be used for a wide variety of modeling tasks. However, it is designed primarily for defining abstract data models, as opposed to defining implementation-specific data models, as might be created when designing database implementations. This makes EXPRESS particularly well suited to the modeling needs of abstract standards like SGML, HyTime, and Topic Maps, because it operates at the same level of abstraction and implementation independence.

To date, EXPRESS has been used primarily in large-scale industries to define standard models for "product" data, including CAD data, in various domains, automotive, construction, and processing (oil and gas, chemical). The standard is about the same age as HyTime.

The EXPRESS language has two defined representation forms: a graphical language (EXPRESS-G) and a character-based syntax (EXPRESS).

## Graphical EXPRESS Models (EXPRESS-G)

An entity is drawn as a rectangular box in EXPRESS-G, and the name is written inside. Attributes are drawn as arrows, where the arrow head is a small circle, from the entity to a data type, e.g. a STRING. The name of the attribute is written on the arrow. A data type is represented by a rectangle with an extra line to the right, and with the data type in capital letters inside it.

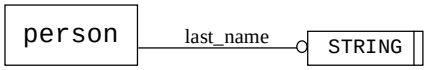


Figure 1. An entity "person" with an attribute "last_name" of type string.

Attributes can be optional, which is indicated by a dashed arrow line. And the data type can be defined as more complex, e.g. by aggregating simple data types. A defined data type is represented by a dashed rectangle. Common aggregates are list, set, and array. They are always resolved into simple data types.

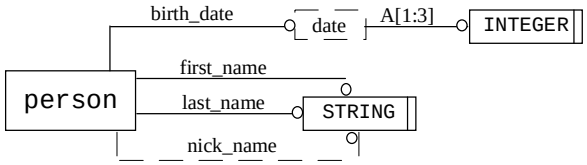


Figure 2. The attribute "nick_name" is optional, and "birth_date" is a defined data type called "date", which resolves to an array of three integers, presumably year, month and day.

Another data type is the enumeration, where the attribute value can assume one of a set of fixed values. Enumerations are drawn as dashed rectangles with a dashed line to the right. The values of the enumeration is not shown in EXPRESS-G, only in the lexical form of EXPRESS.

Attributes can also be derived, i.e. they are calculated by some rules from other data. In EXPRESS-G, we can only see that an attribute is derived by the indication (DER) before the attribute name. To see how it is calculated one must consult the EXPRESS code.

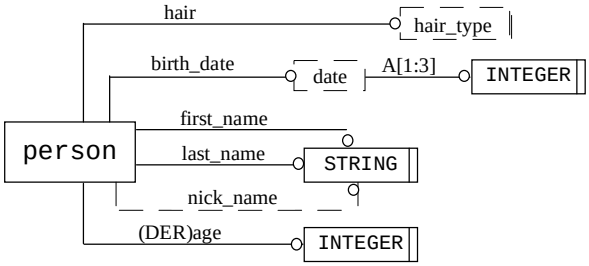


Figure 3. The attribute "hair" is of type "hair_type", which is an enumeration. "age" is a derived attribute of type integer, presumably calculated by subtracting "birth_date" from the date of today.

Class hierarchies are expressed by a thick multi-branching tree-arrow, with the supertype pointing at its subtypes. The most general case of subtyping permits instantiation of multiple subtypes for one object, but if the subtypes are all exclusive (only one of) a number "1" is written on the subtype arrow. Supertypes can be specified as being abstract if the only allowed instantiation is through one of its subtypes. The subtypes are specialised entities that inherit all attributes and relations from the supertype. Multiple inheritance is allowed.

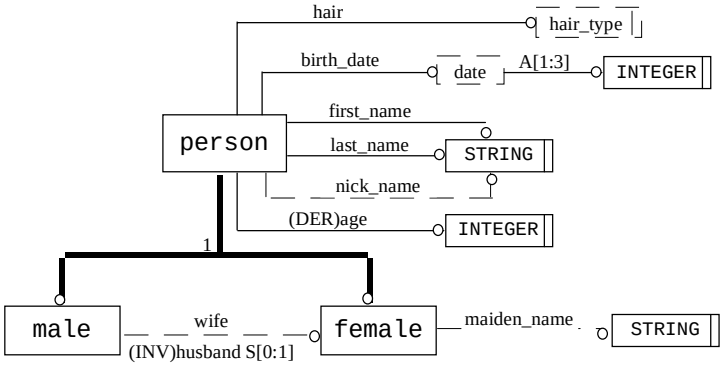


Figure 4. Entity "person" is subtyped into "male" and "female". The number "1" indicates that it is an exclusive-or subtyping. It is still possible to instantiate "person" without indicating whether its a "male" or "female".

In figure 4, a relation between two entities is shown: a "male" can have a "female" as "wife". Relations are often given names that correspond to the role they play. They are really a specialised attribute, and have the same type of notation as attributes; a dashed arrow indicates optionality and aggregates (e.g. array, list, set) can be defined. One thing is however specific for a relation: the possibility to define an inverse relation.

All relations are by definition bi-directional in EXPRESS, but the implementation forms are not bound to implement all inverse attributes (that would be a terrible overhead). Therefore, if you are dependant on the inverse for any reason, it should be explicitly written out. The inverse attribute is represented by the letters INV within parenthesis before the inverse attribute name.

Cardinality can be expressed explicitly in EXPRESS. The normal case, with a single relation arrow pointing from e.g. "male" to "female", indicates that one "male" can only have one "female". But the inverse in the normal case is always "many". Therefore, if the inverse relationship should be equally constrained, it must be written explicitly, as in figure 4: A "female" has a set of zero to one "husbands".

Finally, there is the select type in EXPRESS, which makes it possible to say that a relation is to "either one of these or one of those". The select data type is drawn as a dashed rectangle with a line to the left (not to be confused with the enumeration data type!). A specific feature with the select type is the possibility to mix entities and data types in the select.

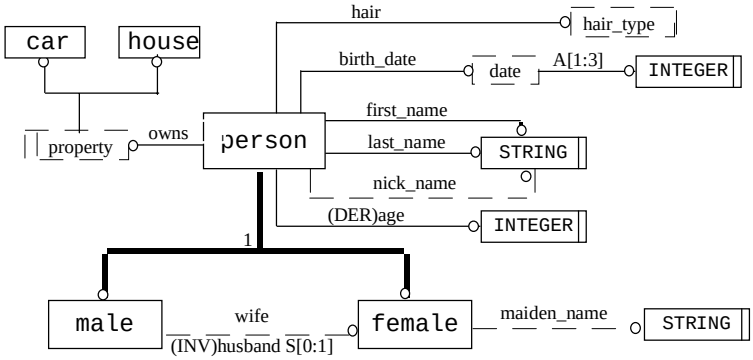


Figure 5. A "person" owns either a "car" or a "house", through a select data type called "property", according to this model.

## EXPRESS Language Syntax

EXPRESS-G is the graphical representation of EXPRESS. It is very powerful to use in modelling sessions, and for communicating concepts, etc, but it is not complete, as we have already seen. EXPRESS is the lexical representation, and the formal definition for models.

This document is not aimed at giving a deep understanding of the EXPRESS lexical form, but a few samples will be given as an illustration. Figure 1 above looks like this in EXPRESS:

```
TYPE STRING;
END_TYPE;

ENTITY person;
    last_name  : STRING;
END_ENTITY;
```

In the above, one type and one entity is defined. The entity has one attribute.

## Relationship to Other Modeling Languages

**UML** UML is a collection of graphical modeling languages for different types of things: data and object models, use cases, data flow, etc. Most of these bear no relation to EXPRESS at all. UML's graphical object modeling language similar to EXPRESS-G. Does not yet have a corresponding character-based syntax (but XMI being developed).

**XML Schema** Scope of XML Schema effort is much narrower than EXPRESS and explicitly focused on document structure modeling. Probably the case that any XML Schema model will be representable using EXPRESS but not the reverse.

**RDF** Some similarity of application, but fundamentally a different approach and focus to data modeling. RDF intended to model and associate metadata with resources. Probably the case that any RDF model could be represented using EXPRESS but not the reverse (because RDF does not provide a constraint language).

**IDL** IDL is a programming interface definition language, not a data modeling language. The STEP standard defines a binding from EXPRESS models to IDL for the purpose of creating programming APIs for accessing EXPRESS-driven data in terms of its schema.

## *EXPRESS Model of the SGML Document Grove*

In the HyTime standard (ISO/IEC 10744:1997), a simple data modeling formalism called "property sets" is defined in Annex A.4. It is used in other annexes to define the "data models" of SGML and HyTime, for the purpose of making SGML and HyTime documents available to HyTime and DSSSL applications. Property sets can be developed for any notations, in order to make them HyTime-processable.

The EXPRESS model of the SGML document grove is created by a semi-automatic mapping from the "abstract document" part of the "SGML property set" defined in Annex A.7 of the HyTime standard. These are the classes and properties needed to represent a document abstractly. The abstract properties capture all of the original data content and structure, but do not capture the details of the original markup strings (such as omitted markup, non-data whitespace, etc.).

The model consists of three main entities (in the EXPRESS sense of the word), *sgml_document*, *element*, and *sgml_entity*.

An SGML document has a prolog of one or more components. It is governed by a document type, and has a document element (the "root element"). It may contain SGML entities, and zero or more elements. Finally, the SGML document may have an epilog, too. This is described by the EXPRESS-G model in figure 6.

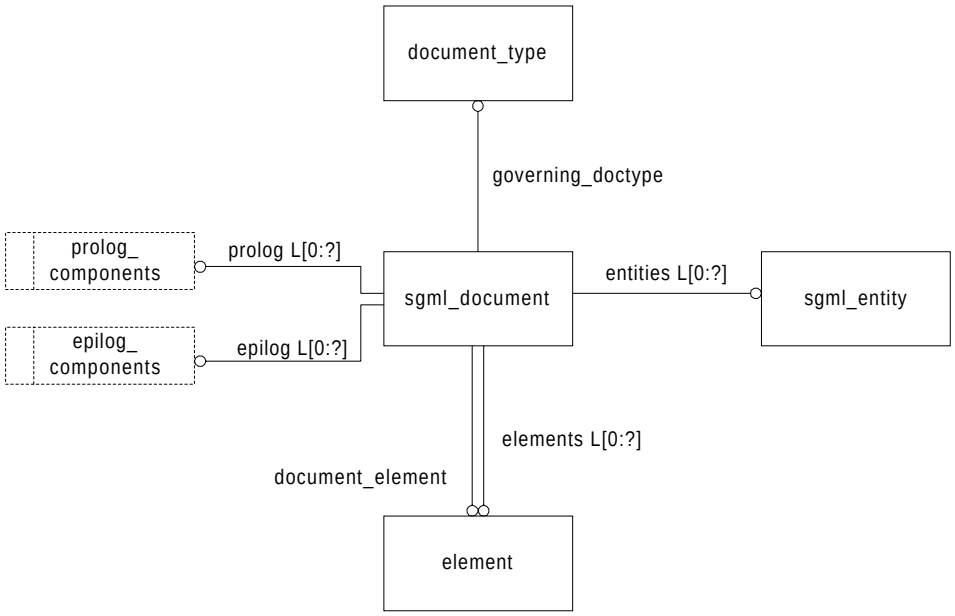


Figure 6. Simplified EXPRESS-G model of SGML document.

The model of elements and attributes, see figure 7, says that an element can have an attribute called "id", and must have a "gi" attribute. It may have any number of SGML-attributes assigned to it, and it has a content property, which may be any number of a number of different object types: sdata, data_char, subdoc_ref, pseudo_element, processing_instruction, external_data_ref, or element. Any number of these can occur in any order within the content of an element.

Furthermore, this model shows that attributes have a name, and a value (which is the "content" of the attribute). An attribute may have a token separator (if the value is tokenized), and it has a boolean attribute indicating if the attribute is implied or not.

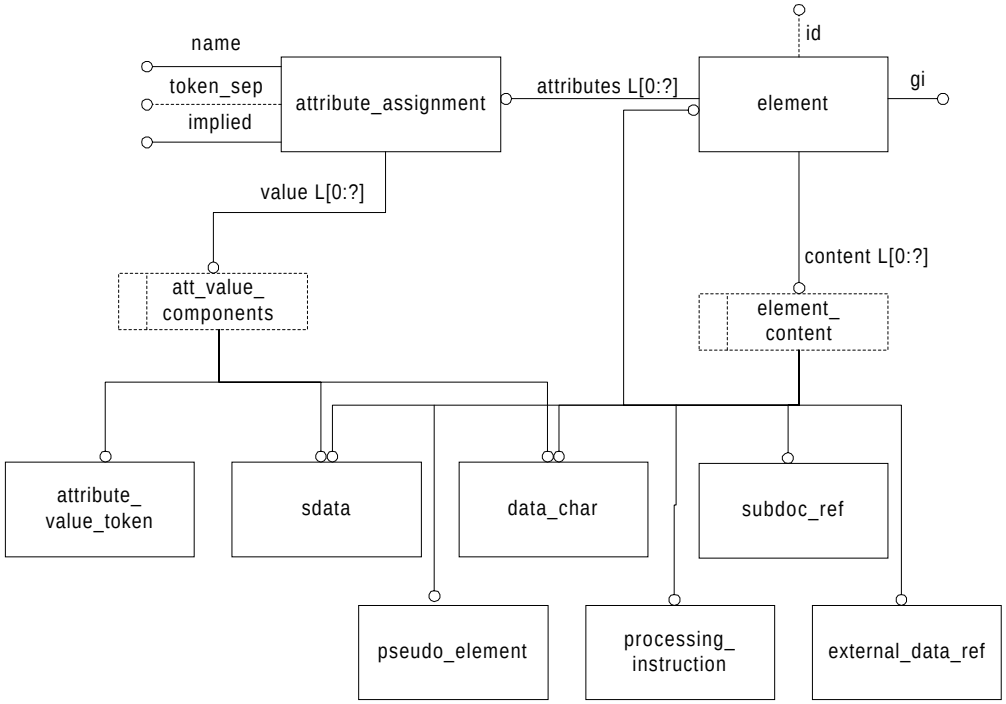


Figure 7. Element and attribute model (simplified).

The model describing an SGML entity (figure 8) contains a class hierarchy, or subtyping. The sgml_entity is an abstract supertype of external_entity and internal_entity, i.e. an entity must be either internal or external. The external_entity is subtyped further into external_data_entity, subdoc, and external_text_entity.

All entities have a name and are of a specific entity_type. Internal entities have replacement_text, while external entities have an external_id. And finally, the external_data_entity class has a notation and may have attributes.

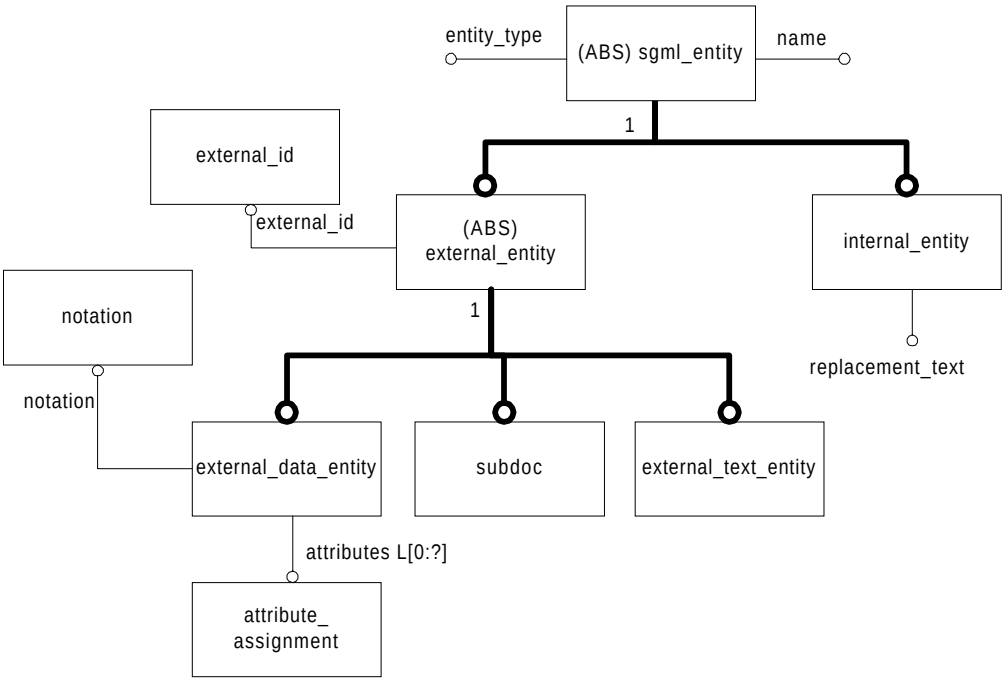


Figure 8. SGML Entity model (simplified).

As you can see, the EXPRESS models are describing the basic rules that govern all SGML documents.

When creating an "SGML object model", we found that there are three possible approaches:

- Make a simple transliteration from the SGML property set
- Make a transliteration and add some class hierarchies and constraints
- Create a new, more abstract and semantic model

The first is what you get with the automatic mapping rules, and does not add any level of understanding or control to the SGML property set. The second is still compformant to the SGML property set, but enforces most of the implicit rules, and also provides a better understanding of the object model. The third is what we would have preferred, since it would give us a simpler and more coherent model, which would better describe the underlying rules of SGML. It is probably what should have been done before the standard was written in order to make it more consistent. It is also something that must be done by SC34, if at all. Of course, when the SGML standard was written in the early 80's, formal data modeling was not as well developed or understood. In hindsight, we can better appreciate the value of it by seeing the results of not having done it.

The simple transliteration gave us an EXPRESS model with very little value (see figure 9). The SGML property set is, compared to EXPRESS, a much less powerful object modelling language, and for a good reason: It was developed to describe SGML to the level necessary for HyTime and DSSSL grove purposes, and it does not enforcing any rules of SGML, that's the job of the DTD, parsers, and SGML grove constructors. Therefore, this model allowed anything that SGML itself allows, **plus** a lot more that SGML does not allow. The property set implies the use of a DTD to ensure compliance with SGML, but EXPRESS does not, and therefore the resulting EXPRESS model was for all practical reasons useless in that it did not define any of the constraints of SGML (beyond those inherent in the base data model itself).

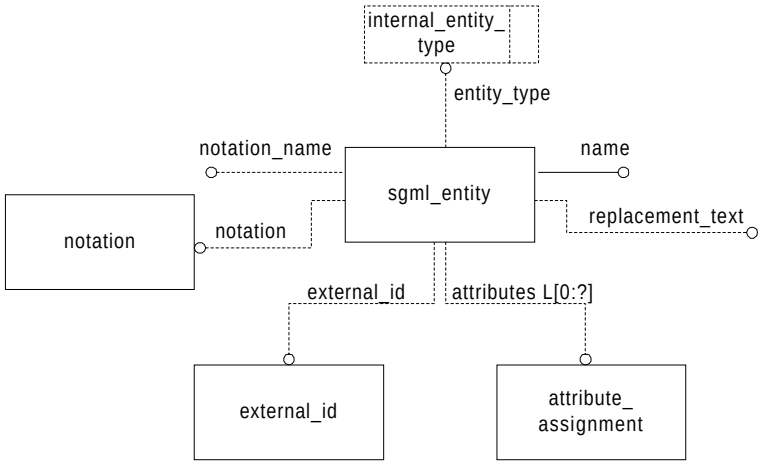


Figure 9. SGML entity with simple transliteration.

The second approach, adding class hierarchies and modeling the implicit rules as formal constraints, gave us a model which could be used as an object model of SGML. An example of this can be seen by comparing figure 8 with figure 9. In the latter case, the use of different attributes becomes much clearer, e.g. "external_id" can only be used for external entities.

It turned out, however, that class hierarchies could be built in several different ways, none of them "perfect" from a modelling standpoint. And the constraints were sometimes so unnecessarily complex because of how the model was built, that we seriously wanted to change the model for "something better". This would have taken us halfway to the third approach, which was not within our scope. Therefore, approach two could not be automated, but involved a human interpretation of the property set (and the SGML standard).

## *EXPRESS Model of the HyTime Semantic Grove*

The HyTime standard, ISO/IEC 10744:1997 (HyTime 2nd Edition) is a large and wide-ranging document. It consists of two main parts: the HyTime Architecture and the SGML Extended Facilities. Within the HyTime Architecture, there are several modules relating to different aspects of modeling hypermedia. Of these modules, the base, location address, and hyperlinking modules provide the facilities needed to do hyperlinking. The scheduling and rendition modules relate to the problems of presentation of information synchronized in time and space. Because the hyperlinking parts of HyTime are both implemented and of immediate benefit, we chose to only model those parts of the HyTime data model to start with. Modeling of the scheduling and rendition objects will follow on from this effort as time allows and implemention efforts require. Thus, the data model presented here does not reflect the entire HyTime data model. It only reflects those components of it needed to model and manage location addresses, hyperlinks, and use-by-reference relationships (value references).

Looking at that part of the HyTime standard that is about hyperlinking and addressing, the HyTime standard defines two separate but related things: an abstract data model for representing hyperlinks and location addresses as objects and a syntax for representing hyperlinks and addresses in SGML and XML documents. The HyTime standard is fundamentally a processing standard in that it defines an abstract data model and the processing semantics associated with that data model. The representation syntax it defines exists simply to enable the reliable interchange of data to which the processing semantics (linking and addressing) can be applied. However, there is nothing in the HyTime abstract data model that ties it to SGML or XML: HyTime is defined completely in terms of operations on groves, which are generic data objects to which any type of data can be mapped. This means, in part, that HyTime hyperlinking and addressing semantics can be applied to data in any format. There is no requirement in the HyTime standard that links and location addresses be created exclusively through the processing of SGML or XML documents that conform to the HyTime architecture.

The HyTime standard treats hyperlinks as first-class objects. It also models the generic properties of location addresses and the results of resolving those location addresses into lists of addressed objects. In the HyTime standard, the HyTime data model is formally defined using a property set, a simple data modeling formalism defined in Annex A.4 of ISO/IEC 10744:1997. This property set defines the "HyTime semantic grove". The objects in the HyTime semantic grove are the objects a HyTime engine

needs to use to represent the result of applying HyTime defined semantics to data (e.g., SGML or XML documents linked using HyTime hyperlinks).

In this discussion, only the abstract data model is relevant. The syntax of the data representation is irrelevant because we are entirely in the abstract domain, that is, in the realm of processing programs that operate on the results of having parsed documents (or any other data) into their abstract "in-memory" representation. At this level, it is just as meaningful to talk about linking STEP entities as it is to talk about linking XML elements: they are both abstract data objects. All that is required is that both sets of objects be mapped to HyTime's fundamental model for data: groves. Given that mapping, as far as HyTime is concerned, all data is the same and therefore linking and addressing of any kind of data is meaningful and rigorous.

The HyTime semantic grove is nothing more than the formal definition of the information a "HyTime engine" has to manage in order to provide the services of managing and resolving hyperlinks and location addresses. The job of a HyTime engine is to examine a set of data objects, recognize hyperlinks, location addresses, and use-by-reference relationships in those data objects, resolve the location addresses, and contruct objects representing the hyperlinks it found. In terms of the HyTime standard, the result of this processing is a "HyTime semantic grove". Once constructed, this grove becomes the storehouse of information about the links in the system (essentially, a "link database"). Other applications can use this information by using normal grove access and navigation mechanisms to look up what they need to know. For example, an application that is processing a grove for a documement can use the HyTime semantic grove to determine if any of the nodes in that grove are linked to anything else. Thus, the HyTime engine and HyTime semantic grove provide a general hyperlink and location address management service intended to be used by other applications. Both the GroveMinder product and PHyLIS tool reflect this design by providing general grove management and access layers and HyTime semantic groves. (Of course, there is no requirement that HyTime applications be implemented literally using groves.

The purpose of the EXPRESS model of the HyTime semantic grove is to define the same data model as defined by the HyTime property set but using the EXPRESS language. This provides two benefits. First, it makes the data model clearer by expressing it using a more familiar syntax. Second, it allows for a more rigorous definition of the data model through the use of EXPRESS facilities that property sets do not provide. One benefit is that this should help enable the implementation of HyTime-conforming semantics in non-grove-based systems by making the data model easier to understand for programmers familiar with common data modeling and object design techniques. The data model can also be used as a guide to developers of ad-hoc or self-contained systems who want to be consitent with the HyTime model. For example, the model provided below can be easily translated into relational tables using normal database design techniques. While the property set based model can be as well, the translation would probably not be as obvious to database practitioners and software engineers as the EXPRESS model would be.

At its core, the HyTime data model for hyperlinks is quite simple. A hyperlink object represents a typed relation instance among two or more "anchors". An anchor is a typed collection of zero or more data objects of any type. The type for an anchor is its "role", meaning its role within the relationship represented by the hyperlink as a whole. The data objects collected by the anchor are said to be the "members" of the anchor. Hyperlink and anchor types are defined by separate objects to which each hyperlink and anchor object points. The simplest form of this model is shown in Figure 10.

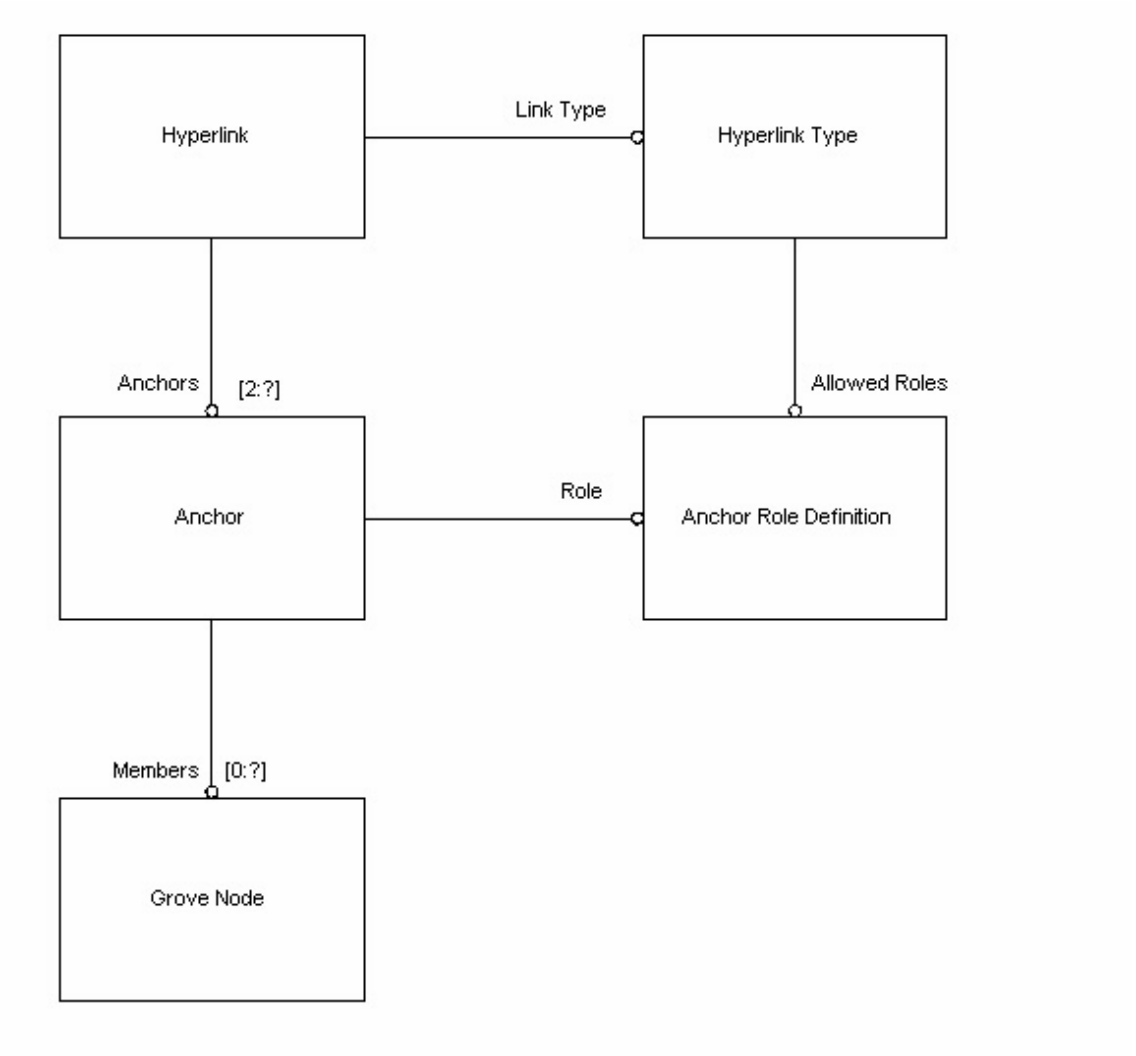


Figure 10. Simple Form of HyTime Data Model

One key aspect of this model is the definition of anchors as abstract objects with their own types. This allows a link to associate a list of multiple objects with a single role within a relationship. Note the parallel between the hyperlink object and its roles and the hyperlink type definition and the anchor role definitions. The hyperlink type defines the basic schema that governs instances of the hyperlink type.

Note also that there is nothing in this model that makes any reference to any particular syntax. This model can represent hyperlinks created using any syntax or programmatic mechanism for creating them. The model shown omits a number of properties from the full model that serve to further refine the rules for links and anchor roles, such as whether or not multiple members are allowed for an anchor, whether or not a given anchor can be omitted from a link instance, rules for navigation (traversal) among the members and between the anchors of a hyperlink.

This model for hyperlink objects forms the core of the HyTime data model. The overall data model starts with the "hyperdocument" object, which is an organizing container for all the other HyTime-specific objects. The objects in the HyTime semantic grove represent the result of processing a "bounded object set," which is simply a finite set of entities (in the SGML sense), at least one of which is a HyTime document (an SGML or XML document conforming to the HyTime architecture). The hyperdocument object has the following key properties:

**hytime documents** — A list of "hytime document" objects. A "hytime document" object is a distillation of an SGML or XML document from its original syntax, through its mapping to the HyTime architecture, to an abstraction of the hyperlinks and location addresses in the document. At this level of abstraction, all syntax distinctions in the original source have been normalized into a single representation form.

**bos (bounded object set)** A list of "bos member" objects. At the syntax level, a bounded object set is defined as a collection of entities. At the grove level, the bos is defined as a set of groves constructed from those entities. For each entity in the BOS there will be exactly one grove in the bos list. For example, an SGML document entity is represented by an SGML document grove. Each bos member object points to the root node of the grove for the corresponding entity. The bos member also indicates whether or not the bos member is in the "HyTime BOS" (the set of entities included in the BOS by the strict application of HyTime-defined BOS construction rules; a BOS can contain entities that were added through application-specific means in addition to those added (or excluded) by HyTime's BOS construction rules). Every grove node that is a member of a hyperlink anchor or listed as a reference target must exist in a bos member grove. In other words, the set of bos members defines the pool of grove nodes that are candidates for linking and addressing (there may be other groves in the system that are not part of the bos, meaning that the HyTime engine has no knowledge of them and therefore cannot manage links to them).

**anchored objects** This is a list of all the grove nodes that are members of any hyperlink anchors. This list provides an index by which an application can quickly determine whether or not a given grove node is linked to anything else. The value of the property is a list of "anchored object" nodes. The list is also a name space where the name of each anchored object is the grove node that is anchored. In other words, the list forms a hash or associative array in which the keys are grove nodes. This simply allows quick and convenient lookup of nodes in the list.

**hyperlink types** A list of all the hyperlink type objects.

**hub document** The hytime document that was the starting point for defining the bounded object set. The hub document is nominally the starting point for navigation within the bounded object set (although the HyTime standard does not require that navigation must start with the hub document--it simply recognizes that processing has to start somewhere).

**value references** A list of "valueref" objects. Value reference objects represent use-by-reference relationships between starting grove nodes and the other nodes that provide the effective values of the starting grove nodes. They are essentially redirections. The value reference facility is part of the base module of HyTime is a class of relationship distinct from hyperlinking. Both hyperlinking and value reference depend on location addressing.

The properties of the hypergrove object are shown in figure 11

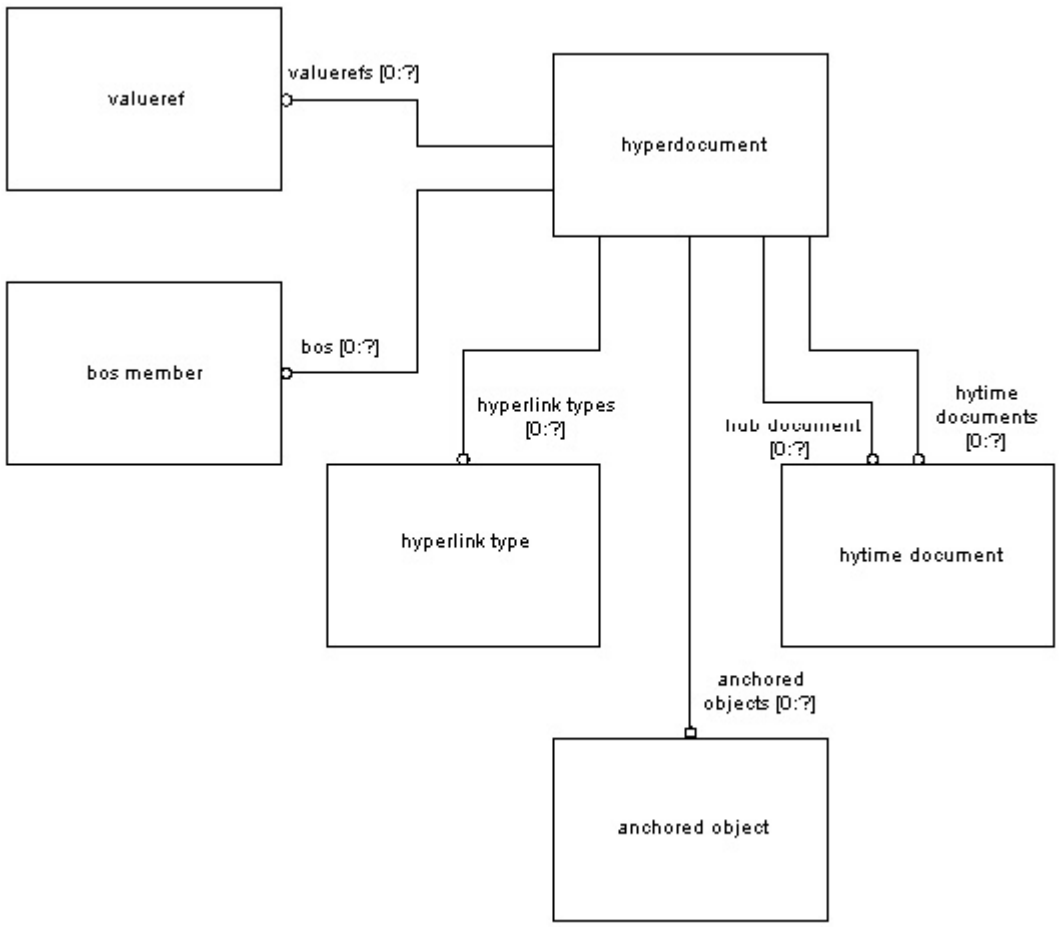


Figure 11. Properties of the Hyperdocument Object

While within the data, hyperlinks and use-by-references are created using some form of location address, such as SGML ID references, XPointers, URL's, HyTime location address elements, etc., for the purpose of modeling hyperlinks, the addresses have all been resolved to pointers to the actual grove nodes addressed. However, for the purposes of other HyTime-related processing, it can be useful to know how the original location addresses were constructed. In addition, in a non-SGML environment (such as a STEP repository), it can be useful to model the location addresses as objects so that they can be modified and managed independent of the things they address. Thus, the HyTime semantic grove includes objects that model the details of location addressing.

The first part of the addressing model is the "reference targets" property of hytime document nodes. This property is a list of "reference target" objects. Each reference target object binds a grove node that is a reference (for example, the node representing an SGML element that makes an ID reference) to the nodes the first node ultimately references.

The reference targets property is bound to the hytime document objects because the references reflect the results of resolving the location addresses in the hytime document.

[Really long footnote: Keep in mind that for a HyTime engine to recognize an ID reference as a location address, the document that contains the element making the reference must be derived from the HyTime architecture. Of course, other applications could synthesize their own hytime document objects in the HyTime semantic grove, but such processing is not defined (or definable) within the scope of the HyTime standard. However, if the resulting hytime document object is consistent with the HyTime data model, any subsequent processing would be in conformance with the HyTime standard. Confused yet?

Just try to remember that there is a functional and normative distinction between the processing that a conforming HyTime engine is required to apply to any conforming HyTime document and the processing that any other application could apply if it chose to. The HyTime semantic data model will

work regardless of whether you got it by applying the standard-defined processes of the HyTime standard or other processes. Because the HyTime standard is only defined in terms of processing its SGML-based interchange syntax, the processing of any other data format to construct all or part of the HyTime semantic grove is beyond the scope of the HyTime standard. However, the result of such processing can be validated against the HyTime semantic grove's formal schema for consistency with the HyTime standard.

For example, while you can't create a HyTime-syntax hyperlink in a Microsoft Word document (because Word documents are not SGML or XML), you could write a processor that interprets what passes for hyperlink and location addresses in word documents and synthesizes them into the appropriate objects in a HyTime semantic grove. That is, the processor could construct an abstract hytime document object that contained hyperlink and location address nodes corresonding to the analogous constructs in the Word document. It could then further construct hyperlink objects reflecting the hyperlinks represented by the Word constructs. The result of that process can then be validated against the semantic (but not syntactic) rules of the HyTime standard and would result in conforming HyTime hyperlinks and addresses.]

Back to the addressing model. The key object for modeling location addresses is the "location address" object. A location address object represents any form of addressing, regardless of syntax. It has the following properties:

**located nodes** The nodes directly addressed by the locatation address

**location source** The list of nodes from which the located nodes were selected. Every location address selects zero or more nodes from a list of nodes.

**location selector** That portion of the location address definition used to select the located nodes from the location source. For example, for an ID reference, the selector would be the ID value specified.

**Definition** The original source nodes that defined the location source (e.g., the SGML element or attribute assignment that defined the location source).

Each reference targets object lists the set of location address objects that resulted in the the list of targets for that reference (the "location path" in HyTime terms). A reference target binds an initial referencing node to the list of nodes ultimately referenced. The location address objects in the location path record the exact process by which the nodes were originally addressed. This information is not necessary for simply navigating links or resolving value references, but it can be very useful in an editing or dynamic data management environment. It is also useful for deferring the resolution of addresses.

One important aspect of the HyTime semantic grove is that it is, in the abstract, static. It reflects the result of applying a single processing operation to a fixed and finite set of objects. It records the state of the HyTime engine's knowledge about the bounded object set at the time processing ended. This is true for all groves: groves are by definition snapshots in time of the state of some set of data. The grove model does not provide any mechanism for dynamic update, because it does not need to (remember that groves, like STEP data, are data models, not object models with object methods). In the abstract, if the data changes, you construct a new grove.

One way to think about this is that the system maintains every version of every data object over time. Rather than imagining one grove that changes over time, you can imagine a sequence of groves, one for each point in time at which the underlying data changed. In this model, you can address any object from any point in time. The notion of "change" goes away. For example, while we think of a document "changing" when it's edited, what you've really got are two completely independent documents, the one from before the edit and the one from after the edit. For example, you could create a new grove that consists of something like "version pair" objects that simply relate pairs of nodes together to define their relative positions in time.

What this means is that any dynamics in a HyTime system (or any other similar system) is a function of how quickly the processor can construct new instances of the HyTime semantic grove. There's no need to build dynamism into the basic model because it simply isn't relevant at that level of abstraction: dynamism is an implementation detail that is transparent to the abstraction and to processes using the abstraction. Of course, in a real HyTime engine that supports the real-time processing of dynamic data (as opposed to only supporting static views of data, such as data delivered on CD-ROM), the engine will be able to update the HyTime semantic grove as necessary to reflect changes in the underlying data. Of course, for some kinds of data this can be a signficant challenge because of the need to re-resolve and recalculate location addresses and the resulting nodes lists. But this problem is inherent in hyperlink and

address processing generally (which means data management generally)--it is not unique to HyTime. HyTime simply makes the problem more obvious. HyTime engines will distinguish themselves in part by how clever they are at updating their semantic groves.

The full HyTime semantic data model is shown in figure 12.

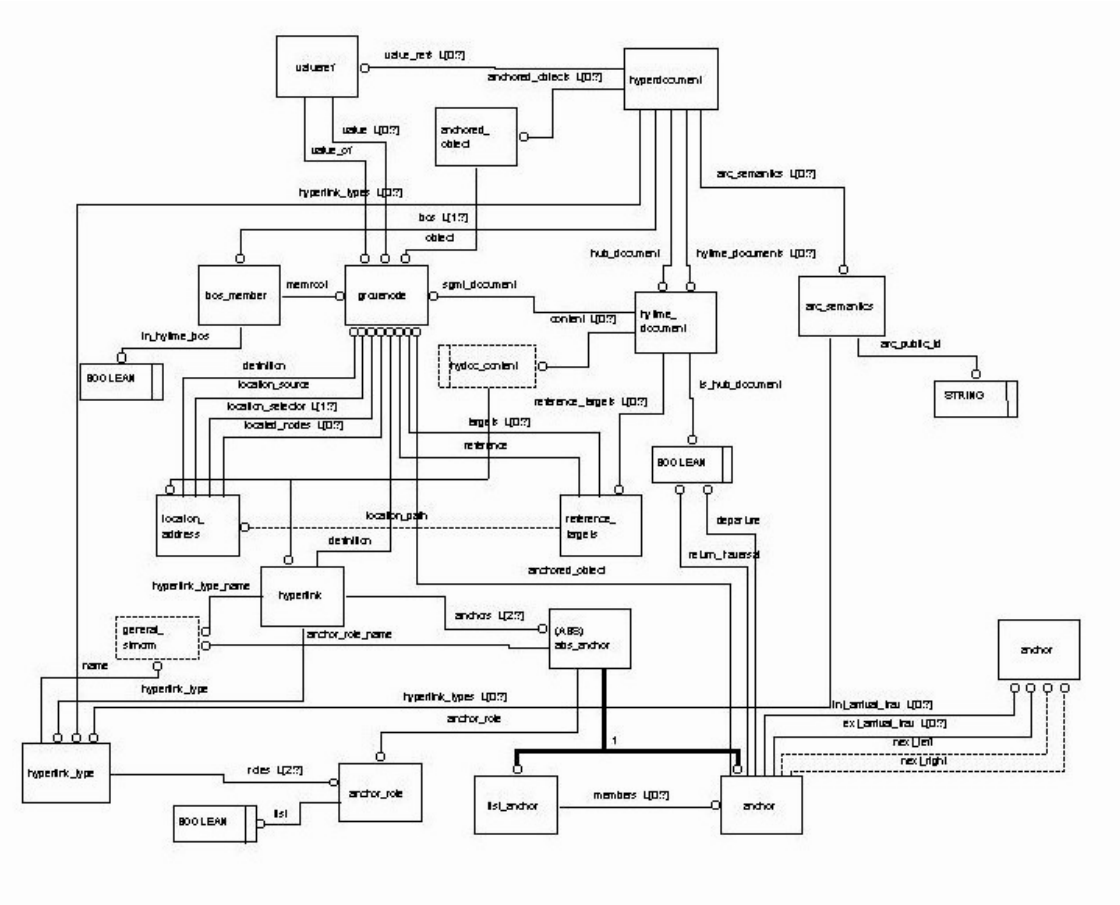


Figure 12. Full HyTime Semantic Data Model

## *Design Principles*

The SGML and HyTime EXPRESS models were developed as literal transcriptions of the original SGML and HyTime property sets. This was done for two reasons. First, we wanted to discover the nature of the mapping of property sets to EXPRESS models in order to determine what the results of an automatic translation would be. Second, we did not want to spend any time refining the existing data models. While these data models may (and do) have design flaws and could benefit from the application of more advanced modeling techniques, we did not feel it was our place to do that work within this project. For example, it is the responsibility of ISO/IEC JTC1/SC34, the manager of the SGML family of standards, to define any new and improved data models for SGML and its related standards. We also felt that we had to do the direct transcription and understand the result completely before we could attempt any refinement of the design.

Classes in the property sets (element classdef) were mapped to EXPRESS entities, and properties (propdef) to attributes or relations to other classes.

For naming, we used the application names of classes and properties, replacing spaces with underscores.

We created superclasses only where such superclasses were clear or explicit from the standard, could be inferred without significant controversy, or were necessary to make the model understandable. We tried to limit constructed supertypes to abstract supertypes wherever possible.

Because the grove mechanism distinguishes node list properties that have empty lists from properties that are not exhibited at all, we used zero-to-many node lists for required node list properties that allow empty lists. Optional attributes were used only for properties that were defined with "when" clauses in the property set.

For the EXPRESS model of the SGML document grove, some "when" clauses have been interpreted as EXPRESS constraints expressions, in order to further clarify the implicit rules (i.e. in the property set definitions, not in the SGML standard) of SGML documents. If the model is used only to hold SGML documents in an EXPRESS environment, such constraints are not necessary, but if the model is also used to create documents in an EXPRESS environment using this model, the model must enforce all details of SGML. Thus far we have not developed the model, as discussed above. The constraints are mostly there to illustrate the possibilities of the EXPRESS language.

## Conclusion

Our experience with using the EXPRESS language to reflect the SGML and HyTime data models has demonstrated first that it is possible and useful to translate property sets into EXPRESS data models. We've shown that there is sufficient correspondence that the result is useful. We've also discovered and refined our understanding of where the simplifications of the property set formalism can be enhanced in an EXPRESS representation. We've demonstrated that the act of applying formal data modeling techniques to the SGML and HyTime data models has helped to make them clearer and helped to bring to light weaknesses in the original model. For both the SGML and HyTime data models, the use of express, and in particular, the graphical forms of the models, helps make the many complex relationships among the components of SGML documents and the compents of HyTime engines clearer and easier to analyze.

One of our assumptions at the beginning of this project was that EXPRESS, as a richer, more complete data modeling language, would enable us to define more complete and rigorous data models. We have demonstrated that this is in fact the case. By doing so, our hope is that formal data modeling and EXPRESS in particular will be used by standards like SGML in the future to define their fundamental data models as part of their normative definition. We think it only makes sense for other ISO standards to take advantage of the EXPRESS standard.

Another goal was to make the SGML data model available to STEP-based environments. We have completed at least the first step toward this goal by providing the SGML data model using STEP's language. The next step will be to prove or work through inspection and implementation.

## Bibliography

1 ISO 10303 Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual

2 ISO/IEC 10744:1997 Hypermedia/Time-based Structuring Language (HyTime), 2nd Edition

## Biographies

**Peter Bergström** has been a mainstay of the Swedish SGML community, most recently as President of the Swedish SGML User's Group and as a participant in SGML and XML standardization activities. Peter has been involved with the STEP standard for the last several years as a member of EuroSTEP AB. Before becoming an SGML expert, Peter taught Swedish as a foreign language. His recent experience includes using STEP and SGML together in IETM projects and developing information and process management systems for suppliers of paper making equipment. When not jetting about the world for his clients, Peter enjoys the quiet life of rural Sweden with his wife and children.

**W. Eliot Kimber** has been involved with generalized markup, SGML, electronic publishing, and hypertext for all of his career, mostly at IBM, more recently for Passage Systems and ISOGEN International. Eliot is co-editor (with Charles F. Goldfarb, Steve Newcomb, and Peter Newcomb) of the HyTime standard and a member of ISO/IEC JTC1/SC34, the ISO committee responsible for SGML and its related standards. Eliot was a founding member of the XML Working Group. Eliot is the author of the soon-to-be published book Practical Hypermedia: An Introduction to HyTime, part of the C.F. Goldfarb Series on Open Information Management. When he is not working on, writing about, or teaching about standards, Eliot works as a systems integrator, helping clients use SGML, HyTime, DSSSL, and related standards to their best effect. In his spare time, Eliot is a devoted husband and dog owner.