The Exploitation of SGML

A collection of published papers authored by staff
of the SGML Technologies Group

Second Edition
We should like to thank the Graphic Communications Association (GCA) for permission to publish papers that first appeared in the Proceedings of the GCA SGML/XML conferences both in the United States and in Europe. It is a pleasure for us to continue to support these conferences which are widely acknowledged as the best means of learning (more) about SGML/XML. The GCA, and the International SGML Users’ Group (ISUG), have sponsored the standard since its inception, the latter being another organization we are pleased to support. Our thanks are also due to the ISUG for permission to publish articles that first appeared in its Newsletter, more recently known as interCHANGE.

But it was the authors of the papers who really made this publication possible, and here we wish to pay tribute to Stéphane Bidoul, Tom Catteau, Pierre Colot, Jean-Paul Daisomont, Mladen Damjanov, Philippe Fontaine, Jacques Kasprzak, Jorge Leal Portela, and Philippe Vijghen, and to thank them, not just for taking the time and effort to write the papers but especially for the ideas behind them. Last but by no means least we gratefully acknowledge the painstaking task carried out by Sandra Marionex who, once again, was responsible for overseeing the publication process, with her usual attention to detail. They are all part of a team of which we are justly proud.

JMS and J-PG
Foreword

Although the SGML Technologies Group was only founded in July 1996, its subsidiaries ACSE in Belgium and ISEA in Luxembourg had been working with SGML for several years. In fact ACSE (Associated Consultants and Software Engineers) was founded in December 1989, and ISEA (International Software Engineers Associated) a few years later in November 1993, joining the Group at the end of 1996. SSEL (SGML Systems Engineering Limited), based in Bristol, joined the Group mid-1998, but had been using SGML since the inception of the company in May 1996.

That is not to say that the customers of ACSE and ISEA, in particular, necessarily knew that an international standard was being used as a means of solving their particular problem, usually a large-scale project involving many man/months of work, and in some cases many man/years. The Standard Generalized Markup Language (SGML) is an international standard for structuring information in a comprehensive way, breaking down a document, however this may be regarded, into elements or objects of information. So it will come as no surprise that the Group uses an object-oriented approach to handling information. In fact we are members of CORBA, the Common Object Request Broker Architecture. SGML, CORBA, and an object-oriented approach are seen as methods to be used in designing turnkey systems and generally providing the necessary software systems for customers, whenever and wherever appropriate.

Of course, if we are able to design an architecture from first principles, we can make full use of our experience, which has been accumulated over many years, even before ACSE, ISEA, and SSEL were founded. Some of us were closely involved in the drafting of ISO 8879 SGML from the early 1980s; others were busy implementing the drafts that came out of the International Organization for Standardization (ISO) to result in the first commercially available software parsing system, using all the features described in the standard. This has meant that we have been building on our cumulative experience not only with the Group’s parsing system itself but also with an application language, resulting in an SGML integrated toolkit. In addition to using the software system ourselves for clients’ large-scale projects, it is marketed as a product called VISUAL SGML (SGML Integrated Toolkit) so others may benefit from our endeavours. And we now also have the benefit of SGMLC (SGML Composer) developed by SSEL.

Over a hundred staff are currently employed by the Group, some of whom have been giving presentations at the SGML/XML series of conferences in the United States and Europe which have been organized by the GCA. The resulting papers were published in the Proceedings of the conferences. And some staff have had articles published in the Newsletter of the International SGML Users’ Group, now called interCHANGE. The Exploitation of SGML is collection of those papers and articles so their message may reach a wider audience.

Two things may become apparent to the reader. Firstly we do not employ programmers: we employ software engineers who are capable of designing architectures and devising clever
means of designing code with fast execution times, using all the tricks of the trade. Secondly we do not use SGML: we exploit SGML, using the optional features of the standard, often in multilingual applications, in ways of which the standards-makers never dreamed.

Of course, HTML (the HyperText Markup Language) as an application of SGML for the Web pages on the Internet did much to popularize structured information, whether or not it was appreciated by those who surf the Web. And we have been putting this to good use in some of the projects with which we are associated. More recently there is XML, the eXtensible Markup Language, where this is a subset of SGML designed to give additional functionality to Web information. We are into XML as well, as you would expect.

Judge for yourself. For the latest information you can always visit our Web site on http://www.sgmltech.com, or contact us directly.

Joan M Smith
Jean-Pierre Gaspart

London, June 1999
Contents

Stéphane Bidoul, Object Orientation and SGML: LINK Revealed..................................................1

Stéphane Bidoul, Reusability in SGML with Focus on Software Engineering...............................17

Stéphane Bidoul, From Prototype to Production System: Managing the Growth............................27

Stéphane Bidoul, A Transactional Approach to SGML Storage: Why You Should
Ask More From Your Repository ..........................................................................................39

Tom Catteau, The European Union’s Budget: SGML Used to its Full Potential.............................51

Tom Catteau, The Addition of a Multilingual Component to an Existing Document
Processing System.............................................................................................................67

Pierre Colot, SGML in a Multilingual Environment ......................................................................77

Jean-Paul Daisomont, Using Management Information Within an Editorial Context.....................89

Mladen Damjanov, SGML as the Lingua Franca in a Multilingual Environment............................97

Philippe Fontaine, SGML Boosts Standard Document Processing for a Major
European Bank ..................................................................................................................101

Jacques Kasprzak, The Use of SGML for a Police Information System......................................107

Jorge Leal Portela, SGML Joins the Battle Against International Art Robbery............................121

Philippe Vijghen, Experience of EDI for Documents: The Role of SGML..................................133


The Exploitation of SGML
Object Orientation and SGML: LINK Revealed

Author

Stéphane Bidoul

Keywords

- SGML
- LINK
- ESIS
- Application
- Process
- Event-Driven Programming
- Object-Orientation
- OO

Abstract

Several studies have tried to address the topic of object orientation around SGML. The question asked was too simple and dichotomic; the answer given was a far too simple 'yes' or 'no'. The SGML application aspect, that is not covered by the standard, was not considered when searching for commonalities.

This paper intends to show that some application architectures coupled with an SGML parser offer an object mechanism with embedded SGML. The relationship between the parsed tokens and the application methods shows that application objects are connected to parsing objects in a simple and efficient paradigm which fully conforms to the LINK feature of the SGML language. Adopting this view of an SGML application makes all the facilities offered by the LINK feature suddenly self-evident and useful.
Biographical Note

Stéphane Bidoul has been working for four years at ACSE sa/nv (a member of the SGML Technologies Group) as a developer and systems architect for object-oriented distributed applications and complex documentary workflow automation systems (automation of the editorial process for the budget of the European Union, automation of the legislative procedures for the Belgian French Community Parliament, etc). All these applications have in common their use of SGML, either as a document storage and exchange medium, or as a formal message specification tool for communications between distributed application processes. He obtained a degree, specializing in electromechanical engineering, from the Free University of Brussels in 1992. He may be contacted at sbi@sgmltech.com.

Introduction

This paper presents an event-driven object-oriented computing model that is capable of processing an SGML data stream in real-time. The nature of the events processed is similar to those defined in ESIS. However, the application program that handles those events can be tied contextually to element instances. This relieves the programmer from managing context information himself. The next section introduces the basic principles of this model.

The Implementation Strategies section discusses an implementation where an application language integrated with the parser is provided as a way to bridge the gap between the parser (where the processing is tied to the parsed DTD) and more generic data processing packages that are not tied to a specific representation of the data structure in a particular DTD. The Applications section gives a few practical examples.

An Event-Driven Object-Oriented Computing Model for SGML

One output of the parsing of an SGML instance can be seen as a succession of events that correspond to the parsed tokens. Such an event-oriented view of the parser output has been formalized in ESIS. Hooking application code onto these events gives a powerful technique for processing SGML documents.

However, ESIS gives no contextual information to the application. It leaves the whole burden of maintaining this important information (ie the parse tree) on the programmer's shoulders, to perform simple tasks such as distinguishing two instances of the same element in different contexts of the model.

The real power of the paradigm presented here resides in the fact that the events are automatically routed to application objects that correspond to the elements of the SGML instance. These objects have inherent information about the SGML context where they are created.

To program such a system, the developer writes a set of classes that all expose a common interface through which they receive notification of the events from the parser. The instances
of such classes correspond on a one-to-one basis to the SGML element instances in the
document.

The connection between the parser and the classes is done in a standard fashion throughout
the three flavours of the LINK feature of SGML. Link rules are used to tell the parser which
class to choose when it creates the objects associated with the SGML elements. This
mechanism unleashes the full power of the LINK feature, by associating classes (with code
and data) to elements in a particular context with #USELINK and #POSTLINK.

In the next subsections, we successively examine the following topics:

- identifying the events;
- defining the classes, the lifetime of objects, etc;
- linking the classes to the DTD in a context-sensitive fashion by using LINK.

**Event-Driven**

The computing model is event-driven, because the application code is triggered by events
resulting from the parsing of the SGML instance. These events are generated by the parser
and fed into the application code in real-time through a mechanism which we shall come back
to later.

We can identify a simple (yet non-exhaustive) set of events that is of interest from an SGML
programmer's point of view.

The list below includes a brief description of each event, as well as the basic information it
carries.

- **onStart**, triggered on element start (start-tag).

  The basic information carried by this event is the element name and the list of
  attributes. It also receives the doctype to which the element belongs. This event is
  fired even when the start-tag is implied by OMITAG.

- **onEnd**, triggered on element end (end-tag).

  This event carries no additional information.

- **onData**, triggered for all the data chunks that make up the data content of the element.

  The basic information carried by this event is the character data. This event is fired
each time a data token is recognized by the parser (eg the data flow of the element is
interrupted by a sub-element). Moreover, if the data content is large, it could be
tokenized arbitrarily by the parser. The point here is that the application which
receives such events should be prepared to deal with the data content in multiple
chunks. Only when receiving the onEnd event can the application be sure that all the
data content of the element has been passed.
• **onEntity**, triggered on general entities.

    This event carries all the information that is necessary to identify the entity, its nature, and its content. We can identify its name, its type (CDATA, SDATA, NDATA, PI, SUBDOC, etc), and also the replacement text, possibly as a system identifier. If there is a notation attached to the entity, the event also carries the notation identifier so that the application can process the data accordingly. The bottom line is that the parser must pass enough information so that the entity manager can be fully or partly replaced by the application.

• **onPI**, triggered on processing instructions.

    The basic information carried by this event is the data content of the PI.

• **onRegExp**, triggered on regular expression matches.

    The regular expressions can take the form of Lex-like [Lesk 75] regular expressions. Those regular expressions can have embedded 'tagged parts' whose matched content will be mapped to variables at run-time. This mechanism is useful to let the application have some lexical capabilities without modifying the DTD. It should be used either when SHORTREF and DATATAG mechanisms are not powerful enough, or when the programmer does not want to change the DTD or SGML declaration to process the document in a fashion not foreseen by the DTD designer.

    This powerful mechanism can even be used to parse non-SGML documents (eg RTF and \TeX), the SGML parser becoming a multi-purpose data processing tool. But that is another story.... The basic information carried by this event is the regular expression name, as well as the data that triggered the match.

Although this simple set of events is enough for developing sophisticated SGML applications, it is possible to define other kinds of events. Some of the events presented are defined in ESIS; others are new.

Our goal here is not to define an exhaustive list of such events, but to highlight the basic principles of this event-driven SGML computing model.

**Object-Oriented**

Event-driven programming lends itself very well to object-oriented programming. The basic principle is to map events to methods of (or messages to) objects.

To achieve this, a common interface is defined for all classes that will handle the events. Objects (instances of such classes) are associated with each of the SGML elements.

**Classes**

The events we defined previously will guide us in defining the interface the objects will need to expose in order to interact with the parser.
Here is such an interface, defined in CORBA IDL.

```c
interface ISGMLEventSink
{
    void onStart( in sgmlname_t LinkSetName,
                 in sgmlname_t DocTypeName,
                 in sgmlname_t ElementName,
                 in attlist_t SourceAttributes,
                 in attlist_t TargetAttributes );
    void onEnd( );
    void onData( in sgmlstring_t Data );
    // inout parameter leave room
    // for a replaceable entity manager.
    void onEntity( in sgmlname_t EntityName,
                   inout entity_t kind );
    void onPI( in sgmlstring_t PIData );
    // regular expression gives the matched data,
    // and the matched expression variables;
    // the matched data can be replaced by the application
    void onRegExp( in sgmlname_t RegExpName,
                   inout sgmlstring_t MatchedData,
                   in attlist_t MatchedVariables );
    // called when the active link set changes
    void onLinkChange( in sgmlname_t FromLinkSet,
                       in sgmlname_t ToLinkSet );
};
```

The methods used in this interface correspond to the events described previously. The LinkSetName and TargetAttributes arguments will be explained in the next section (LINKing the Classes to the DTD).

**Instances**

What are the instances of such classes?

In a nutshell, let us say they correspond to SGML elements. Each time an SGML element is encountered, an instance of a class that exposes the ISGMLEventSink interface is created (see next subsection for details about the lifetime of such objects).

Once the object is created, it will receive all the events for the corresponding element. This means that its ISGMLEventSink methods will be called upon at the element start and end. The data content of the element will be dispatched to this object too (as well as entity replacements, etc).

If a sub-element is encountered, an instance of another or possibly the same class (see LINKing the Classes to the DTD, below) will be created to handle the corresponding events.

**Lifetime of Instances**

Basically the instances of application classes have the same lifetime as the elements to which they are mapped.
This is obviously true when we consider the events that are passed from the parser to the objects: the first event occurs with the start of the element, and the last occurs with the end of the element.

In certain cases, however, the code that executes on the onEnd event may still be active after the element is terminated. This occurs when the application code lets the parsing continue while executing the onEnd() method. One must be cautious of such issues when implementing the object manager part of the parser.

**Aspects of Inheritance and Object Hierarchies**

In object-oriented systems, one often confuses inheritance and object hierarchies (i.e. parent-child relationships).

In the system we describe here, the parser will establish a hierarchy of objects that correspond on a one-to-one basis to the parse tree. This is *not* inheritance. The only inheritance in effect here is the *interface inheritance* mechanism that makes all classes expose the same ISGMLEventSink interface.

What can be extremely useful is to have an *implementation inheritance* mechanism which eases the coding of the classes. However, this is purely a programming language issue, and the parser does not need to be aware of such relationships between classes.

**LINKing the Classes to the DTD**

In this section we shall answer the question 'from which class should we create the objects?'. In other words, we shall define a standard-based mechanism that allows the programmer to link elements of the DTD(s) to the classes he (or some other programmer) has coded according to the ISGMLEventSink interface.

This powerful mechanism is based on the LINK feature of ISO 8879, and provides great flexibility in the level of detail the programmer has to provide to realize the mapping. This flexibility comes from the three kinds of links defined by the standard.

This flexibility ranges from very simple (where all elements are mapped to instances of the same class) to very elaborate (where we tend towards an ideal situation where the application can be written independently of the actual structure of the DTD that models the data being processed).

**Summary of the LINK Feature**

As defined in ISO 8879, the LINK feature is a mechanism which enables the dynamic association of additional information to elements at parse time. This additional information is provided in the form of *link attributes*, which are new attributes (not defined in the DTD, and thus not present in the source document). These are dynamically added to the elements at parse time.
The feature is named LINK because it also provides a way of linking elements between a source and a result document type.

ISO 8879 defines three kinds of links. They differ in the amount of additional information one can provide in the process of linking source and result elements.

- **Explicit link** provides a mechanism to map elements from the source document to elements in the result document. Link attributes can be specified for the source element. Attributes, that are declared in the result DOCTYPE, can be set for the result element.

- **Implicit link** is similar to explicit link, except that the result DOCTYPE cannot be specified. One cannot thus give result elements; implicit link boils down to a way of associating process-specific attributes with the element in the source document.

- **Simple link** provides a way to associate one or more link attributes with #FIXED values to the base element of the document.

The association between a source element and a result element is specified in a *link rule*. Link rules are grouped in *link sets*. Link sets are grouped in *link type declarations* which are the formal definition part of *link process definitions* (LPDs).

Multiple link types can be active at the same moment. However, inside a link type there is only one link set active at a time, and thus there is only one active link rule per link type. Note here that the simultaneous availability of multiple link types allows for simultaneous 'concurrent' processing on the same document instance. For a summary of LINK, see also [Kimber 95].

It is important to note here that an *LPD is merely a vehicle for specifying processing .... It is up to the application to determine what the specification means. In particular, although the nominal result of processing is a document instance that conforms to the result DTD, this is merely a conceptual device to permit specification of the processing in terms of the result document instead of, or in addition to, the source document, if that should be desirable. The actual result could be anything at all. [Goldfarb 90]*

The point here is that ISO 8879 sees LINK as the way of hooking application processing to an SGML parser.

In the next subsections we shall examine how we can use the three kinds of link to express the mapping between SGML elements and the classes defined above in a way that conforms to the standard.

**General Concepts**

We introduce here the notion of *class set* which is simply a group of named classes that have in common the fact that they expose the ISGMLEventSink interface.

Associated with the *class set* we assume there is a mechanism for creating an instance of a class, given its name.
Moreover, we assume there is a mapping (implicit or explicit) mechanism between link types and class sets. An implicit mapping (based on the name) can be sufficient in most cases, but an explicit mapping would be useful to reuse the same class set in different conditions (ie on different DTDs).

Using SIMPLE Links

Since the only information that can be added by a simple LINK is a set of fixed attributes on the base element, we can only deduce an implicit mapping between all the elements of the DTD and one class. For the sake of simplicity, we assume this class has the same name as the class set to which the link type is mapped.

For instance, if we assume a class set named APPLICATION, and a link type named THELINK, both tied together, the elements of the document will all have an associated object of class APPLICATION.

We have a situation where each element of the application will trigger uniform processing. If the application wants to specialize processing based on context or element type, it can do it, but it must keep track of context or element type by itself, with no help from the parser. With simple link, most of the dispatching work is done by the application, and the application is tied to the DTD for which it is initially written. In this configuration all the events are passed to the program which receives no context information from the parser. This is very similar to ESIS.

Using IMPLICIT Links

Implicit link allows for the specification of additional attributes for each element of the source DTD. The introduction of the source DTD suggests that we can have a processing that is specified on a doctype basis.

When a link rule is activated, the parser looks in the class set for a class that has the same name as the element. If one is found, an object is created from that class, otherwise nothing is done.

In this situation the programmer receives more information from the parser since a different class is activated for each element. The #USELINK and #POSTLINK facilities allow for activation of the classes on a contextual basis. However, the application still remains tied to a particular DTD, since the classes are named according to the SGML element names. Moreover, if the same element appears in a different context, the contextual information can be passed to the application as the link set name, but it is up to the programmer to select which code to execute in each context.

Using EXPLICIT Links

With explicit link one can specify a source and a result document, and for each link rule a source and result element, both with additional attributes. The nominal output of the parser is a concurrent document structure, with elements sent from the link rules. The trick here is to define a system result document. Let us name it PROCESS for the purpose of the discussion.
The result element in a link rule will then correspond to the class name of the application object to be created. The additional attributes for the result element will be passed to the application as the TargetAttributes parameters of the onStart() method.

This is the most elaborate mapping scheme. It allows for the design of classes that are used for different element types. But most importantly, the classes can be tailored for a particular context of parsing, since the parser will use them for link rules that are to be activated only in a particular context with #USELINK or #POSTLINK.

Link attributes and target attributes are also important features that let us create a simple mapping between the DTD and the application classes. We can pass a lot of interesting information, including the name of the DTD attributes, so that the application code can be as independent as possible of the original DTD.

**Implementation Strategies**

Following the paradigm expressed in this paper, it is possible to foresee many implementations.

We shall present here a three component structure which we believe to be capable of solving a large number of information management and transformation problems in an elegant manner.

The extended SGML parser program includes two components:

- the SGML parser itself;
- the application language module which handles classes, creates the objects according to the parsing, forwards parser events to the objects, destroys objects, etc.

The third component is built of external information handling packages, which can be called by the parser's application language module.

What we expect is for the user to develop simple SGML processing applications with the embedded language, while complex data processing applications will be developed in the form of specialized packages that are not tied to a specific DTD. In that case, the embedded language will be used as the glue that maps the structure of a particular DTD to the structure expected by the generic external package, reflecting the well-known fact that there are multiple ways to model a non-trivial data structure in SGML.

**Thoughts on the Embedded Language**

The ISGMLEventSink interface presented above has the potential to interface the parser with any programming language capable of generating objects that comply with this interface.

However, some additional features on top of the minimal support of that basic interface would be very desirable.
The language module should be available on a wide range of platforms, and follow the SGML philosophy of universal exchangeability.

The language should be very open to external packages. It should be able to access a wide range of libraries (Windows DLLs or OLE objects, CORBA objects, etc). Such openness is a must to let users transform their SGML data in any way they see fit (transforming it, putting it in databases, feeding it to an interactive subsystem, etc), unleashing the true promises of electronic data interchange.

For instance, one could imagine an Invoice Processing Package that is generic in scope and is used to input the incoming invoices into the invoice tracking system of a company. Invoices coming from various sources could all be fed to this package, whatever their format, provided a stub is written to map the input format to the application package.

Moreover, a language that would be embedded in the parser would gain additional benefits.

- Easier access to the parser data structures (SGML declaration, detailed contextual information, etc) for more advanced applications.

- The handling of the attributes could also become more straightforward, really mixing the application and SGML objects.

Indeed, DOCTYPEs and ELEMENT declarations are often considered as classes from which documents and elements in documents are instances. In particular, element attributes are declared in classes (ATTLIST declarations) and instanciated within the objects (the element instances). For a detailed account on the subject of classes and instances in SGML document see also [DuCharme 95].

It is really interesting to mix the SGML element class, element instance paradigm with the application class, application object paradigm described in this paper.

From this viewpoint, we can consider that we have different kinds of classes:

- the element class, with its attribute declaration in the DTD;

- the link source element class, which inherits the attributes from the element class and adds the link attributes;

- the link target element class, which inherits from the link source element class and adds the attributes of the target element;

- on the other hand, there is the application class, which is written by the programmer.

So, when a link rule is activated the parser can mix the link target element class and the application class and create an object that receives the attributes of the SGML part as normal attributes in the object-oriented application world!
Applications

The first application (Changing the Delimiters) shows an example of uniform processing of all the elements of the application, using a SIMPLE link.

The second example (Formatting) shows how #USELINK can be exploited to trigger different processing of the same element, the decision being taken by the parser, not the programmer.

Both examples are written with SGML-conformant LINK specifications, and the classes are written with a C++ like syntax. They are only meant as demonstrations of possible applications of the paradigm presented here, and not as complete working applications!

Changing the Delimiters

In this example we show how to use SIMPLE link to change some delimiters from the reference concrete syntax to another:

- stago (l) will change to { ;
- etago (l/) will change to {/ ;
- tagc (> ) will change to };
- lit (” ) will change to |.

We begin by declaring a SIMPLE linktype to tell the system that a link process has to be activated.

```xml
<!LINKTYPE DELIMCHG #SIMPLE #IMPLIED>
```

Then we write the class set that execute the transformation. Since we use the SIMPLE link there is only one class for all the elements of the application.

```xml
classset DELIMCHG
{
    // global variables
    ofstream file;
    const string target_stago = "[";
    const string target_etago = "]/";
    const string target_tagc  = "]";
    const string target_lit   = "|";

    // code that runs on program start
    onStart()
    {
        file.open("output.sgm");
    }

    // code that runs on program end
    onEnd()
    {
        file.close();
    }
}
```
// Only one class
class DELIMCHG : implements ISGMLEventSink
{
    // class local data
    sgmlname_t LocalDocTypeName, LocalElementName;

    // on element start
    void onStart( sgmlname_t LinkSetName, 
                 sgmlname_t DocTypeName, 
                 sgmlname_t ElementName, 
                 attlist_t SourceAttributes, 
                 attlist_t TargetAttributes )
    {
        // save element and doctype names for use on the end event
        LocalDocTypeName = DocTypeName;
        LocalElementName = ElementName;
        // write the beginning of the element
        file << target_stago << "(" << DocTypeName << ")"
             << ElementName;
        // write the attributes
        for (int i=0; i<SourceAttributes.count(); i++)
        {
            file << " " << SourceAttributes[i].name
                 << target_lit << SourceAttributes[i].value
                 << target_lit ;
        }
        // write the tagc
        file << target_tagc;
    }

    // on data tokens
    void onData( sgmlstring_t Data )
    {
        file << Data;
    }

    // on element end
    void onEnd( )
    {
        // write end tag using saved element and doctype name
        file << target_etago << "(" << LocalDocTypeName << ")"
             << LocalElementName
             << target_tagc;
    }
}

The above class set works according to the following scenario.

- When the program starts (the associated linktype is activated), the onStart method of the class set is called and the output file is opened.

- For each element instance an instance of the DELIMCHG class is created, this object receiving all the events for the corresponding element. For each event the (obvious) processing is done to write the newly delimited tags to the output file.

- When the program finishes (the associated link type is deactivated), the onEnd method of the class set is called and the output file is closed.

**Formatting**

In this example we show how #USELINK can be exploited to trigger different processing on the same element, the decision being taken by the parser, not the programmer.
We shall take a simple application: converting a document marked according to the GCAPAPER DTD to HTML. We shall not present the whole code here; however, we shall examine a typical example: handling the TITLE element.

Here are some excerpts of the GCAPAPER DTD which show the use of the TITLE element.

```xml
<!-- The TITLE tag is used for: headings within the paper, figure captions, and to give the author's job title. Each type of title is formatted differently. -->

<!ELEMENT title     - o    (#PCDATA) +(#emphs;|ftnote|fnref) >
<!ELEMENT front      - o    (title, subt?, author+, keywords?, abstract, biography) >
<!ELEMENT author    - o    (fname, surname, title?, address ) >
<!ELEMENT section    - o  (nbr?, title, para*, subsec1*)
<!ELEMENT subsec1    - o  (nbr?, title, para*, subsec2*)
<!ELEMENT subsec2    - o  (nbr?, title, para*, subsec3*)
<!ELEMENT subsec3    - o  (nbr?, title, para+) >
```

This element has the peculiarity of appearing in different contexts: the main title of the document, headings within the paper, the author's job title, and so on. In each context it needs to be formatted differently.

Note that this is a real world example. While writing this, I periodically ran such a converter to preview and let others review my document in a more enjoyable format.

We shall show how this TITLE element is output in HTML in each different context (except for figure captions). So let us write the LINKTYPE.

```xml
<!-- link type for GCAPAPER source dtd, and PROCESS target (ie the application language module) -->

<!LINKTYPE GCA2HTML GCAPAPER PROCESS
<html>
<body>

<!-- link type for GCAPAPER source dtd, and PROCESS target (ie the application language module) -->

<!LINK #INITIAL

<!LINK AUTHORLK
```

The Exploitation of SGML
The point in this link type is that each title element instance will be handled by a different class, based on its context. All section headings will be handled by the same class, and a level class is set to tell the application that there is a heading level change.

Now let us look at the classes.

classset GCA2HTML
{
    int level = 1; // top level (for main title)

    // The datawriter class is used as an ancestor
    // for all classes that need to output all the
    // data content of their element to the file.
    class DATAWRITER : implements ISGMLEventSink
    {
        void onData( sgmlstring_t Data )
        {
            // write element data on output file
            file << Data;
        }
    }

    // Class to handle the main title.
    class MAINTITLE : implements ISGMLEventSink
    {
        // Local variable to store the title text
        string title = "";
        // The title has to be saved, because it
        // will not be written directly to the file
        // but used twice at the end (see onEnd()).
        void onData( sgmlstring_t Data )
        {
            title += Data;
        }

        void onEnd( )
        {
            // Write the HTML header and
            // the first heading, centered.
            file << "<head>" << title << "<title>l/title>l/head>l/center>";
        }
    }

    class JOBTITLE : inherits DATAWRITER
    {
        // Title data is written by DATAWRITER ancestor class
        void onEnd( )
        {
            // finish with a hard break
            file << "<br>";
        }
    }
}
class LEVEL : implements ISGMLEventSink
{
    void onStart( , , , )
    {
        // increment global heading level
        level = level + 1;
    }
    void onEnd( )
    {
        // decrement global heading level
        level = level - 1;
    }
}

class SECTITLE : inherits DATAWRITER
{
    void onStart( , , , )
    {
        // write HTML heading start tag
        file << "<h" << level << ">";
    }
    // Data is written by DATAWRITER ancestor class
    void onEnd( )
    {
        // write HTML heading end tag
        file << "</h" << level << ">";
    }
}

What makes this application so simple is that the code does not need to take particular actions to keep track of the context: the parser does the hard work and selects the right portion of code (ie class), according to the link rules.

Conclusions

A structure-controlled SGML application operates on the structure that is described by SGML markup. The usual way to communicate such structural information is through the ESIS interface.

ESIS defines a set of events that are of interest to an SGML application. However, most tools only deliver this information in a linear fashion, leaving to the programmer the chore of maintaining contextual information. This contextual information, which is very important in all but the simplest SGML applications, is naturally known by the parser. And SGML defines a standard mechanism to pass this information to the application: the LINK feature. However, even when full link information is available in ESIS, the programmer is often left alone to arrange the dispatching of a linear suite of events.

What we have presented here is an object-oriented computing model that offers the ability to write context-sensitive event-driven SGML applications. This model is based extensively on the standard LINK feature to map the element of the DTD to application objects. We have also described an extended SGML parser that automatically dispatches ESIS events to application objects written by the programmer.

With this programming model one writes code for a particular context only, letting the parser select the right code fragment for the right context. This opens up the whole world of reusability to SGML programming.
References


- [Lesk 75] ME Lesk and E Schmidt; *Lex - A Lexical Analyser Generator*. Bell Laboratories, 1975


Please e-mail your comments to Stéphane Bidoul at sbi@sgmltech.com.

This paper was first published in the *Conference Proceedings of SGML'96 US*, November 1996, pp 485-98.

© The SGML Technologies Group 1996
Reusability in SGML with Focus on Software Engineering

Author
Stéphane Bidoul

Keywords
- SGML
- LINK
- ESIS
- Application
- Process
- Event-Driven Programming
- Object-Orientation
- OO
- Reuse
- Reusability
- Scripting Languages
- Python

Abstract
There are many opportunities for reuse in SGML. Among the reasons considered are:

- because SGML is an ISO standard;
- since the information is structured, it may be reused for output on a variety of media;
Reusability in SGML with Focus on Software Engineering

- the modularity of Document Type Definitions (DTDs), allowing reuse of parts in DTDs for other applications.

Whilst all the above are considered, this paper concentrates on the area least discussed to date, that of reuse of code.

One recently introduced technique enabling SGML application code reuse is the concept of SGML architecture. This technique is discussed as well as one to write reusable code by binding it to the DTD through the LINK feature.

Biographical Note

Stéphane Bidoul is a project manager at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. He is a software engineer and systems architect specializing in object-oriented distributed applications and complex automated document workflow systems; all these systems use SGML either as a document storage and exchange medium or as a formal message specification tool for communications between distributed application processes. He obtained a degree, specializing in electromechanical engineering, from the Free University of Brussels. He may be contacted at sbi@sgmltech.com.

Reuse of a Standard Method

SGML is considered here as an international standard [Goldfarb 90] and as a method of structuring information. It has become widely used internationally, being ISO's fastest selling standard when first published in 1986. People share the same DTDs, for example as information is prepared in accordance with the same structure by international organizations where sharing is of importance.

Consider the telecommunications authorities where information on telecommunications hardware and message handling facilities has to be available to authorities in countries to which the messages are sent, by voice or other medium. The overall authority can mandate that information shall be provided in SGML (an international standard) in accordance with an agreed DTD; it could not have demanded that its members from different countries used specific software that may be popular today but gone tomorrow.

Equally, SGML can be mandated by an oil company for documenting all the equipment that goes to make up an off-shore oil production platform or rig. It includes engines that have to be documented as well as all the drilling equipment and the rig structure itself. This information must be able to be used on-site and be in a common format, easily updatable to comply with safety regulations. SGML as an appropriate international standard may be mandated for compliance by all contractors and subcontractors when submitting information.

SGML lends itself naturally to reuse at all levels. It is indeed what it has been designed for in the first place. In the next sections, we shall discuss reusability of SGML structured information, and reusability of SGML DTDs or parts of DTDs. We shall then describe
techniques that one can use to improve reusability of the application software written to process SGML data.

**Structured Information**

With SGML information is structured, and it is essentially because it is structured that it may be reused in a variety of ways and output on a variety of media. A document may be published on paper, in whatever format is chosen by the publisher or perhaps publishers in cases where the same document is republished at some time, perhaps in another format (SGML does not address format, merely the neutral representation of the content). This same document could be published in large print for those who are partially sighted. It may be published on CD-ROM, on the Internet, or any other electronic medium, transmitted anywhere.

So the same information that goes to make up a document can be reused or republished on different media in different formats.

Recently, the adoption of the DSSSL standard [ISO/IEC 10179] extended the reusability further, allowing a style specification associated with a DTD to be defined, in a system-independent manner. This means that one can describe how instances of a particular DTD must be output, independently of the particular tool used for final rendering. For example, to some extent the same style specification could be used for both paper rendering using a typesetter and online display in a browser.

**Document Type Definition**

There is widespread reuse of DTDs themselves, parts of DTDs, and element names by a diversity of applications, not just the one for which they were originally designed.

A DTD for technical documents in the aerospace industry, for example, may be used as a whole or in part by other industries having a similar complexity of technical documents. When DTDs are constructed in a modular way, with a well-reasoned structure for the content of paragraphs, say, the same definition can be reused with ease for paragraphs of the same construction. This principle applies equally to other modules of DTDs, the well-written DTD being able to be reused in different ways. There can also be reuse of well-chosen element names for similar applications.

Document declarations can be common to many applications, being reused many times, to an even greater extent than DTDs and their component parts.

**SGML Processing Applications**

Most SGML applications rely on the generic identifiers to decide which processing to trigger when parsing SGML data. By this very fact they are tied to a particular DTD. Often, however, these applications implement tasks that are more general. Unfortunately, they cannot be
reused because of their dependency on element and attribute names. Such applications are said to be markup-dependent.

In the previous section we discussed the reuse of DTDs and/or DTD fragments. Recently, however, it has been recognized that it is not always practical to reuse the same element and attribute names to express the same semantic constructs in various DTDs. In other words, it is sometimes desirable to use different syntaces to express the same semantic.

A mechanism must therefore be introduced to specify semantics independently of element and attribute names, allowing easier DTD and code reuse.

The HyTime standard [ISO/IEC 10744] has been first to introduce this concept under the name of architectural form. An architecture or architectural form is a meta-DTD, or fragment of a meta-DTD. Semantics are associated with an architecture in the same way that semantics are associated with element types and attribute lists in a normal DTD. The benefit of the architecture concept is that many different DTDs can be mapped to a given architecture using a simple renaming mechanism, providing they have a compatible structure. Thus, using architectures, one can apply standard semantics to existing DTDs, with little or no change to those DTDs.

In this section, we briefly present the mechanism of architectural forms, and we explain how this technique dramatically improves the reusability of SGML processing applications. We also describe a technique based on the SGML LINK feature which provides a markup-independent mechanism to bind application code to a DTD.

Introduction to SGML Architectures

In its simplest form, an architectural form is an element type declaration, with or without an attribute list. This declaration has a well-known semantic. The SGML architecture concept relies on a general name mapping mechanism, which allows element type declarations from existing DTDs to be related to the declarations in the architecture.

Let us explain this mechanism with a simple example from HyTime. Here is the clink architectural form:

```
<!-- Contextual Link -->
<!element clink    -- Contextual link --
    - O      (%HyBrid;)* >
<!attlist clink    HyTime   NAME     clink
    id       ID       #IMPLIED  -- Default: none --
    linkend  -- Link end --
    -- Constraint: No HyTime reftype constraints,
    -- but application designers can constrain
    -- element types with reftype attribute --
    IDREF    #REQUIRED
>
```

This is a normal element type declaration with an attlist and some comments describing constraints and semantics.
Now suppose you have already a DTD named mydtd, say, with some kind of local cross-referencing mechanism, in the form of an element named myref with a target attribute indicating the referenced element:

```xml
<!DOCTYPE mydtd [
<!-- ... -->
<!ELEMENT myref - O (#PCDATA)>
<!ATTLIST myref
  id ID #IMPLIED
  target IDREF #REQUIRED
>
]>
```

To make it work as a HyTime clink, you must slightly modify your DTD to establish the name mapping between your DTD and the HyTime architecture:

```xml
<!DOCTYPE mydtd [
<!-- ... -->
<!ELEMENT myref - O (#PCDATA)>
<!ATTLIST myref
  id ID #IMPLIED
  target IDREF #REQUIRED
  -- HyTime AFO mapping --
  HyTime NAME #FIXED clink
  HyNames NAMES #FIXED "linkend target"
>
]>
```

The HyTime attribute specifies the mapping for the generic identifier. In this case, it means the myref element is to be treated as a clink HyTime element. The HyNames attribute specifies the name mapping between the attributes in the source DTD and the HyTime specification. In this case, it means the target attribute in mydtd is playing the role of the linkend HyTime attribute.

Alternatively, you can avoid modifying your DTD by adding a Link Process Definition (LPD), which is a somewhat cleaner process whereby you avoid cluttering your DTD with AFO references:

```xml
<!LINKTYPE mylink mydtd #IMPLIED [
<!ATTLIST myref
  HyTime NAME #FIXED clink
  HyNames NAMES #FIXED "linkend target"
>
<!LINK #INITIAL myref
]>
```

This IMPLICIT link declaration defines two fixed LINK attributes. If the mylink LINKTYPE is activated, the application will have access to the two link attributes, HyTime and HyNames playing the same role as if they had been declared in the DTD.

**SGML Architecture and Code Reuse**

Associated with the concept of architecture is the notion of an architecture engine, a software component working closely with the SGML parser:
1. It validates the parsed instance relative to the linked architectural forms by interpreting the architecture renaming attribute (e.g., HyTime, HyNames);

2. It produces its output relative to the architecture meta-DTD.

While (1) is most important for formal specification of semantics, (2) is equally important from the point of view of code reuse. Indeed, if you process the output of the architecture engine, your application becomes markup-independent, as element and attribute names are reported to your application as if they came from an instance of the meta-DTD.

If an application is written to an architecture, the programmer can concentrate on the semantics of the parsed data, being confident that his program will be reusable for compatible DTDs expressing the same semantics.

**The LINK Feature and Code Reuse**

Working with architectures sometimes implies an overhead that is not affordable for small applications. If it is actually necessary to define formally semantic concepts independently of any specific DTD, as in HyTime, the process of defining an architecture may be overkill for many simple to medium-scale applications.

The following describes a simple technique which allows you to write markup-independent application code using an IMPLICIT LPD, effectively retaining the reusability potential of architectural form processing.

In [Bidoul 96], an object-oriented mechanism has been proposed to pass ESIS events to the application in a context-sensitive manner.

In a nutshell, we can summarize the principle by saying that an application object can be associated with an SGML element. Such application objects expose an interface through which they receive events such as `onStart()`, `onData()`, `onEnd()`, and `onPI()`. Using LINK rules, we can specify the type of objects to associate with the different element types, depending on context.

The figure below shows SGML application objects associated with SGML elements. The application objects receive events from the corresponding SGML elements.

In [Bidoul 96], we proposed the use of the generic identifier to choose which class to instantiate. The same mechanism can be used in a fully markup-independent manner by exploiting a reserved attribute instead of the generic identifier to decide which class to instantiate.

We shall not expand here the full details of the implementation of such a mechanism. We shall, however, describe a simple example showing how the various components are tied together.

Let us say we have to write an application where we must handle clink elements and carry out very complex but nevertheless useful work with them. In this example, let us say the application is written in the Python language [Lutz 96].
We write a class to handle events for the clink element: the TCLinkHandler class. Then we tie it to the DTD, using an LPD.

Here is the LPD we write to bind the application code to the DTD:

```xml
<!LINKTYPE mylink mydtd #IMPLIED [ 
  <!ATTLIST clink 
    py CDATA #IMPLIED -- PY is the reserved attribute used to map application objects to SGML element -- >

  <![INITIAL]
  -- this rule maps clink elements to instances of the TCLinkHandler class, passing the LINKEND attribute to the constructor of the class. -- 

  clink [ py="TCLinkHandler(a['LINKEND'])" ]

  > ]>

Now let us look at the TCLinkHandler class:
This class has a constructor which receives the value of the end of the hyperlink, and method to deal with start-tag, data, and end-tag events, for the corresponding clink SGML element.

From the point of view of reusability, two things must be noted:

- the class makes no reference to the clink generic identifier;
- the class makes no reference to the linkend attribute name, but receives its value as a formal parameter of the constructor.

Now let us examine how we can reuse this code with another DTD. Here is mydtd again:

```xml
<!DOCTYPE mydtd [
<!ELEMENT myref O (#PCDATA)>
<!ATTLIST myref
  id ID    #IMPLIED
  target IDREF #REQUIRED
]
>
Let us say we want to make the same processing on myref elements. Since the TCLinkHandler class is markup-independent, it is very easy to reuse it for the myref element which is architecturally compatible with clink:

```xml
<!LINKTYPE mylink mydtd #IMPLIED [
<!ATTLIST myref
  py CDATA #IMPLIED -- PY is the reserved attribute used to map application objects to SGML element
  --
  ]>

-- this rule maps myref elements to instances of the TCLinkHandler class, passing the TARGET attribute to the constructor of the class. --
myref [ py="TCLinkHandler(a["TARGET"])*"]
```

Note that the value of the TARGET attribute is given as the formal parameter linkend of the TCLinkHandler constructor. This effectively implements attribute renaming.
Other Aspects of Code Reusability

In addition to these SGML aspects of code reusability, traditional software engineering techniques of modularity should be kept in mind while designing SGML systems. Designing markup-independent applications is one step in the right direction, however.

Summary

It has been shown that SGML lends itself to reuse in many ways, through which use of an SGML system can lead to great efficiency. Careful initial design, where modularity is a keyword, can affect reuse of many ingredients which go to make up an SGML application.

These include the reuse of the standard method, reuse of DTDs and parts thereof as well as element names, and reuse by the design of generic applications as opposed to specific ones.

SGML architectures have been shown to be an enabling technology for writing reusable applications. The reusability benefits of markup-independent applications, however, can be achieved with alternative techniques. Such a technique was presented, using the SGML LINK feature to bind an application language to a DTD.

References


Please e-mail your comments to Stéphane Bidoul at sbi@sgmltech.com.

This paper was first published in the International SGML Users’ Group Newsletter, volume 3 issue 3, July 1997, pp 16-19.

© The SGML Technologies Group 1997
From Prototype to Production System: Managing the Growth

Author

Stéphane Bidoul

Keywords

- Scalability
- Production Systems
- Dissemination Systems
- Workflow
- Quality Assurance
- Repositories
- Collaborative Work
- Conversions

Abstract

SGML information systems usually come into being in the form of small-scale prototype systems supporting a few users and a relatively small set of representative documents. After a successful proof-of-concept phase comes the time of production on a larger scale where the problems encountered are of a totally different nature from those uncovered during the prototyping phase.

This paper addresses scalability of SGML authoring and dissemination systems. An area highlighted is the need to have a set of detailed production procedures taking into account human as well as automated operations.

Biographical Note

Stéphane Bidoul has been working for four years at ACSE sa/nv (a member of the SGML Technologies Group) as a developer and systems architect for object-oriented distributed applications and complex documentary workflow automation systems (automation of the
editorial process for the budget of the European Union, automation of the legislative procedures for the Belgian French Community Parliament, etc). All these applications have in common their use of SGML, either as a document storage and exchange medium, or as a formal message specification tool for communications between distributed application processes. He obtained a degree, specializing in electromechanical engineering, from the Free University of Brussels in 1992. He may be contacted at sbi@sgmltech.com.

Introduction

SGML information systems usually come into being in the form of small-scale prototype systems supporting a few users and a relatively small set of representative documents. After a successful proof-of-concept phase comes the time of production on a larger scale where the problems encountered while growing to a full-scale production system are of a totally different nature from those uncovered during the prototyping phase. For example there are the different and sometimes contradictory constraints of the authoring and dissemination systems, which often show up only in high-volume/high-update rate conditions.

This paper addresses scalability. Neglecting the more obvious aspects of scalability it highlights some issues which are not always considered when designing complex document management systems.

One aspect highlighted is the need to have a set of detailed production procedures which are adhered to in order to avoid cascading effects of incorrectly entered data, among other potential problems. An analogy could be an industrial production system where here there is:

- software to take over work previously performed by humans;
- closely defined workflow procedures;
- the need for an information base;
- the need for quality assurance.

The handling of objects in a well-populated information base is discussed, where software systems can deal with everything from relatively small amounts of data to huge amounts of information of many kinds, its manipulation, and management.

Overview of a Typical Non-trivial SGML System

This section gives an overview, based on real-world examples, of typical SGML systems. Essential components are described, as well as components which, though not directly SGML-related, are essential parts of a successful solution. Different requirements for production systems and dissemination systems are highlighted. This description will serve as a reference model which will be used throughout this paper to support the discussion.

By non-trivial is meant a system with the following characteristics:
multi-user,

multilingual,

distributed, where authors and translators are possibly spread across a WAN.

While a single-user monolingual authoring environment can easily be created using an off-the-shelf SGML editor, storing the work in progress in operating system files, the situation gets more complex as soon as one of the above factors is added.

**Production System**

The components usually found in a production system include:

- the SGML repository;
- the authoring and translation environments;
- the workflow subsystem;
- the communication subsystems;
- the quality assurance environment.

**The SGML Repository**

The repository ensures the robust, permanent storage of the SGML document being produced. As a minimum it provides the necessary features for:

- collaborative work including document locking and simultaneous work on different parts of the same document;
- version control;
- validation.

For a production environment, sophisticated search capabilities are somewhat less important than in a dissemination environment. Indeed, authors usually know what they want to do and which document they need to change.

In a multilingual environment the repository (or some associated component) should provide support for translators, such as replicating document structure changes automatically.

**The Authoring and Translation Environments**

These document creation and modification environments usually take the form of an SGML editor, associated with content manipulation tools, such as a thesaurus, spelling and grammar checkers, and translation memories.
When user training costs are an issue, or when then turnover of users is high, it can sometimes be economically justified to develop an authoring environment by customization of already deployed tools (for example word processors) which are familiar to users. Thus training costs are reduced while paying a one-time customization effort.

In the latter case, conversion filters will be necessary between SGML and the user's authoring tool format.

It must be noted that a full SGML solution does not necessarily avoid the use of some sort of conversion filters.

- It could be that the target DTD is not easy to handle for all authors; some constructs may be too complex to handle with an SGML editor (for example complex relationships between figures in a budget document).

  In such case a conversion step could transform the data to an intermediate format (for example tables) easier to visualize and edit, while the inverse filter would perform a final validation and conversion back to the internal format.

- In some cases not all validation rules can be expressed in a DTD (for example for content-dependent validations).

  In such cases some additional validation filters are necessary.

The Workflow Subsystem

The workflow subsystem is an important part of a multi-user production environment. Its purpose is to track the work performed by each person working on the system, to allow precise monitoring of the production process, and to help the decision at each step of the procedure.

In some cases the workflow system can be used to dispatch the work automatically to users according to the result of the previous step.

The Communication Subsystem

The communication subsystem is often overlooked in a document management environment. A few alternatives are often present:

- a file sharing system;

- a file transfer system;

- an e-mail system.

The file sharing solution is very efficient on a LAN, while the file transfer or e-mail approach has to be set up when users are spread over a potentially slow WAN.
Dissemination System

Components covered include:

- the documents’ input environment;
- the quality assurance environment;
- the SGML repository;
- the conversion filters.

The Documents’ Input Environment

In a dissemination system there is usually an environment where documents are validated and prepared for insertion in the database. This is typically a place where authors put documents ready to be delivered. After a quality assurance phase, these documents are periodically taken in charge by a batch system which loads them in the diffusion database and indexes them for faster retrieval.

The SGML Repository

In a dissemination environment the repository should provide the following basic features:

- support for a large number of concurrent read-only accesses;
- support for a sophisticated search engine.

Here the search capability must be emphasized, because the users requesting the documents usually do not know the structure of the information base; thus they need support from the system to find their way through the potentially vast amount of information.

Collaborative work support is less important, however, since most users only need read-only access, and the input environment only needs to lock documents for the time of the update of the modified documents. This is somewhat different from the production environment where users need to lock (parts of) documents while they work on them.

The Conversion Filters

Conversion filters for dissemination have different requirements from those used in a production environment. For instance, an SGML to RTF conversion filter needs only to focus on presentation for dissemination, while in a production system the SGML to RTF conversion should focus on keeping the structure for later reconstruction of the SGML source.
Scalability

This section discusses the scalability of SGML systems. The more obvious aspects are not discussed. These necessarily include:

- ensuring the repository can handle large documents and/or a large number of them;
- ensuring the repository can handle the required number of users;
- ensuring the editors can comfortably handle large and/or complex documents.

While certainly not exhaustive, this paper focuses on a few specific scalability-related problems, which may not be self-evident at first glance.

Allow Users to Work on Parts of Documents

In cases where the size of individual documents is potentially large, care must be taken to provide users with chunks of documents they can handle easily.

A first approach is to split the documents according to a predefined granularity. However, this is not always convenient because at different steps of the process a different granularity is necessary. For instance:

- an author who knows precisely what to do may want to work on small parts of documents to gain speed;
- later, translators may wish to receive larger chunks containing all the text to be translated;
- during a proof-reading phase it is convenient to have access to the full document to fix typos, which are spread all over the text.

The simple examples above show that the granularity of documents must not be fixed once and for all.

When users are locking documents for long periods, it can quickly become a scalability problem because a coarse granularity can lead to users working on different parts waiting for each other, simply because the system needs to lock documents for one user at a time. It is thus important for the system to let users select precisely which part of the document they want to work on.

This feature, however, has impact on the repository, which needs to let them access but also lock individual parts of the document, with a variable granularity over time. The authoring systems must also be ready to receive fragments of documents.

Variable granularity also has an important impact on the workflow system.
Indeed, most workflow systems are tailored to work on individual objects that keep the same scope throughout the process. However, it has been seen that at different steps of a procedure it is desirable to have different granularity levels. It is very important that the workflow system is flexible enough to allow for those scope variations across time. For example authors could work on several parts of a document, each part taking a whole acceptance procedure. Then, when the time comes to translate the work of the authors, the workflow system must be able to acknowledge the fact that all the authored parts are translated by a single user operating on a larger document fragment.

Taking care of variable granularity allows a system to manage documents as large as several megabytes modified by tens of users. And there is no potential limit on document size.

**Implement an Asynchronous System**

As soon as conversion filters take a part in the production process, there are potentially long operations. By long we mean here more than a few tens of seconds.

This is a perfect place for a bottleneck. The worst case prototype architecture would be a linear system, processing each request one at a time, and potentially requesting the user to wait until his request is complete.

An individual request may take a few minutes, which is acceptable for individual users requesting a few conversions. However, if requests are queued, when several users issue commands at the same time, the first will be happy while the last one may well be angry after waiting for half an hour or more.

Several improvements are necessary to support more than a few users.

- Since a large-scale production system is likely to have a lot of server-side processing, care must be taken that the server is able to handle the work of multiple concurrent users, while not producing bottlenecks. To reduce significantly the elapsed time seen by one user, the only way is to improve the performance of the individual processes. It is comparatively easier to improve the total throughput of the system. System architects used to traditional transactional systems often overlook this problem.

  Document management processing often involves long procedures, which must be taken into account in early system design. This means careful control of the system load, to let multiple users get their job done as soon as possible while avoiding overloading the server. This also means that each long process must be designed to be re-entrant, since a single non-re-entrant process may reduce the total throughput of the system.

  The point is that by the very fact that individual document processing operations take longer that traditional applications, we need to implement system monitoring modules even when the number of concurrent users is relatively small.
• The user interface can also be improved by implementing a mailbox-like system, where the user posts requests and gets notified when the results are ready. This will free the user interface for other tasks, such as preparing other requests.

**Watch the Communication Subsystems**

While not an SGML scalability topic *per se*, the following may take some implementors by surprise.

When delocalized users want to be connected to a document management system, we often think of e-mail to let them receive documents and submit their input.

While this is a perfectly valid and elegant solution, it must be taken into account that an existing e-mail system, which works very well for day-to-day exchange of small messages between workers, might not support the load of an automated system exchanging potentially large documents.

**Dissociate the Production from the Dissemination System**

As has been seen above, the requirements of a dissemination system are totally different from those of a production system. Separating the dissemination repository from the production repository not only avoids security problems (for example when external users cannot have access to work in progress), but it also allows you to choose which product is best for each task.

**The Importance of the Workflow Procedures**

In this section we highlight the fact that a badly designed workflow procedure, perhaps copied from existing manual procedures, combined with new system constraints, can lead to delays and extra work.

Imagine the following simple manual procedure:

- authors change a document in some master language;
- translators apply changes to other linguistic versions;
- proof-readers ensure that all languages are coherent, and fix typos;
- the document is published.

This procedure is relatively straightforward and works very well.

Now it is decided to automate this procedure with an SGML system. Among other improvements the system brings automated validation of the translators' work, ensuring they produce documents which have the same structure as the master linguistic version. This eases
the work of the proof-readers, who can rely on the fact that the linguistic instances are consistent at the structural level, and can concentrate on the content.

If this system is put into operation while keeping the same work procedure, it will induce a subtle problem. Indeed, if an author introduces a small structural error (say a list item is badly tagged), the translators will reproduce it in all the target languages. In the manual system this error may remain unnoticed, since it may not be visible using a presentation-oriented system. Translators would produce a correct list and the master language version would be fixed by translators just before publishing.

In the new system, however, the translators will produce the document in all languages with the error, since their only option is to produce a document which is consistent with the master version. If the proof-readers then detect the error, they will need to fix it in all languages!

This is typically a problem which is easily overlooked during a prototyping phase where a few documents are tested without time constraints. However, operating in high-volume conditions where deadlines are critical and the number of languages is important, it can cause grey hairs to the proof-reading and quality assurance teams.

So here is an adapted procedure, taking into account the new features of the system:

- authors change a document in some master language;
- proof-readers validate and correct the master language version;
- translators apply changes to other linguistic versions;
- proof-readers ensure that all languages are consistent at the content level, and fix typos in the other linguistic versions only;
- the document is published.

Quality Assurance

Quality assurance is a crucial part of a scalable system. Only good quality content will allow for smooth operation and future extensions of the system. While this may seem obvious at first glance, a few simple rules must be kept in mind while implementing the quality procedures, in order to avoid bottlenecks.

Quality assurance in document management systems can take two forms:

- human validation by sampling;
- automated validation, which produces a quality report;
- automated validation, which enforces some predefined quality rules.

Both techniques have advantages and inconvenience.
While manual sampling is the most accurate it is time-consuming, costly, and cannot take place at any time in the procedure. For instance, manual sampling is not feasible on work in progress. Automated validation is usually cheaper and can be exhaustive (that is it may be applied to all documents). It is not feasible, however, to perform some kinds of content validation mechanically.

Both techniques are thus necessary to achieve good results.

To avoid bottlenecks, and avoid some financial surprises, it is necessary to evaluate precisely the time necessary for each task.

As described in the previous section, it is also important to place the quality assurance phase(s) at the right moment in the production process. This is especially true for automated validation tools which enforce some predefined rules. Indeed, some rules which must be valid for the finished document are not necessarily valid during the document production phase (for example linguistic coherency rules). In such cases, it is advisable that such validation tools only produce warnings, but do not enforce the rules which may not be applicable.

**Conclusions**

We have described the typical components of an SGML production system, and of an SGML-based document dissemination system.

Based on these typical system models, a few potential scalability problems have been described, not related to individual systems but to their integration in a complex production process. For each topic, hints have been given for technical solutions.

- A system with fixed granularity (where users can only work on whole documents) can give rise scalability issues when the document size grows and multiple users need to work simultaneously to meet deadlines. This is solved by using an SGML repository, which gives access to variable-granularity document fragments but needs an adapted workflow system which is able to track parts of documents.

- If one fails to take into account the long nature of document processing operations, the system can experience bottlenecks as soon as a few users work simultaneously. We have highlighted the need for a system-monitoring module, such as those found in high-load transactional systems.

- Also highlighted has been the fact that a badly designed workflow procedure, perhaps copied from existing manual procedures combined with new system constraints, can lead to delays.

- The manual and automatic quality assurance procedures are critical parts of a production system. They need to be well evaluated, to define both their objective and cost. It is also necessary to integrate them at the right moment in the production process.
Those real-world examples confirm that a successful large to medium-scale document production system must be designed and planned using techniques similar to those used for industrial production plants.

Please e-mail your comments to Stéphane Bidoul at sbi@sgmltech.com.

This paper was first published in the Conference Proceedings of SGML'97 US, December 1997, pp 533-37.

© The SGML Technologies Group 1997
A Transactional Approach to SGML Storage: Why You Should Ask More From Your Repository

Author

Stéphane Bidoul

Keywords

- SGML
- XML
- Repository
- Database
- Schema
- Transaction

Abstract

Most SGML (Standard Generalized Markup Language) repositories are heavily oriented towards document storage. Because of this, there is a tendency to have an interface that is based on a check-out/check-in mechanism of documents or parts of documents. Such an interface is very well adapted to the way in which humans work when interacting with the repository. However, when SGML is considered as a data modelling language, and the stored data gets more complex, the document-oriented check-out/check-in approach becomes inappropriate as a data manipulation language.

In this paper the benefits of a transaction-based interface to an SGML database are presented, along the lines of the update capabilities of traditional databases. Several real-world applications of this mechanism are described. An interface of this type is then presented, and it is shown why this is a very flexible way to access any SGML database, including document-oriented information bases.

Biographical Note

Stéphane Bidoul is a project manager and has been working at ACSE sa/nv (a member of the SGML Technologies Group) since 1993 as a developer and systems architect for object-oriented distributed applications and complex documentary workflow automation systems.
(automation of the editorial process for the budget of the European Union, automation of the legislative procedures for the Belgian French Community Parliament, etc). All these applications have in common their use of SGML, either as a document storage and exchange medium, or as a formal message specification tool for communications between distributed application processes. He obtained a degree, specializing in electromechanical engineering, from the Free University of Brussels in 1992; he may be contacted at sbi@sgmltech.com.

**Introduction**

Information repositories are an important component of most information systems. These repositories take the form of a database (relational, object-oriented, or other) in traditional applications. When the information to be manipulated by the system is perceived as being 'documents', the choice goes to document repositories, which have very different characteristics. Most notably, document repositories have support for versioning, but, compared to traditional databases, they lack fine-grained access facilities to parts of the information (the granularity is usually the document or some kind of document fragment).

For today's businesses, which are increasingly information driven, the information stored in documents becomes as important as information stored in traditional corporate databases. However, this information is often inaccessible to the knowledge worker, because it is not available online, and when available, it is usually in a mostly unstructured format, unsuitable for precise automated queries and processing. Indeed, where in corporate databases it is possible to formulate queries and manipulate virtually any single item of data, the documents are often handled as an indivisible unit of information, except for simple metadata (title, author, ...).

When SGML is applied to document management systems, unfortunately it is often seen only as a standard 'data format'. This view only addresses the problem of protection of data against tool changes; it does not add much in the way of semantics to the information.

This paper emphasizes the need to design document-oriented information systems much like traditional database-oriented applications, with thorough process and data analysis and modelling. To support this, it is held that the functionalities of SGML repositories must match more closely the capabilities of database management systems. Techniques to achieve this goal are presented and real-world applications developed by our Group using such a system illustrate the benefits.

**The SGML/DBMS (Database Management System) Analogy**

In many respects SGML concepts can be compared with database concepts. The most interesting analogy is the one whereby the SGML DTD (Document Type Definition) is considered to be a data modelling language. In database parlance, the DTD is equivalent to the combination of the schema and integrity constraints.

Consider a trivial example of a contact database, where each record holds a name, an e-mail address, and (optionally) a telephone number. In a database, a DDL (Data Definition
Language) is used to define the schema of the database. In SQL (Structured Query Language), for instance, the DDL statement for the example could be the following:

```sql
create table CONTACTS
{
    ID    number(10)   not null unique,
    NAME  char(256)    not null,
    EMAIL char(64),
    PHONE char(24)
};
```

The structure of the data, together with a few constraints, are expressed in the DDL: the identifier must be present unique, and the name has to be present.

A corresponding SGML DTD could look like this:

```xml
<!DOCTYPE CONTACTS [ 
<!ELEMENT CONTACTS - - (CONTACT)*> 
<!ELEMENT CONTACT - - (NAME,EMAIL?,PHONE?)> 
<!ATTLIST CONTACT 
    ID    ID    #REQUIRED > 
</ELEMENT> (NAME|EMAIL|PHONE) - - (#PCDATA)> ]>
```

Here, the DTD expresses roughly the same structure and constraints as the SQL table creation statement above.

This analogy of DTD versus schema plus constraints is now accepted by many people, and more and more applications are using SGML and/or XML (eXtensible Markup Language) as a general data modelling and representation tool, in addition to more traditional document structuring.

**Extending the Analogy**

At the repository level, most SGML database systems are heavily oriented towards document management. The interface they present to users and programmers is based on a check-out/check-in paradigm. To change the content of the database, a document or a fragment of a document must be extracted, changed, and put back into the system. This process is well adapted to the way in which humans work when interacting with the system. However, application programs could benefit from a more flexible interface.
Taking the contact database example, it is evident that the SQL database provides many features to query, insert, delete, and update individual contacts and/or contact data in a precise way. For instance the following SQL statement:

```
update CONTACTS set EMAIL='sbi@sgmltech.com' where ID=1;
```

would update the EMAIL field of the contact record with ID 1. The following SQL statement:

```
update CONTACTS set NAME=null where ID=1;
```

would fail, however, because it would break the constraint ensuring that the NAME field always has a non-null value.

If the SGML equivalent of the contacts is stored in an SGML repository, most systems provide only a less flexible check-out/check-in approach which is not very well suited to the creation of a contact database management application.

Thus continuing the analogy, it could be said that current SGML databases have a reasonably good DDL (Data Definition Language), that is the DTDs, but a very poor DML (Data Manipulation Language).

While this example is trivial and the data probably not suitable for storing in an SGML system, the last section of this paper (Real-World Applications) shows cases where complex SGML production systems benefit greatly from a true SGML database providing both a sophisticated DML and a check-out/check-in interface.

In the next section, approaches are presented that allow a real DML to be created for SGML databases.

**Proposed Features**

Four basic concepts of an SGML database are discussed in this section:

- addressing techniques needed to identify the content to be manipulated;
- elementary data manipulation operations;
- validation services;
- versioning services.

By way of summarizing the interfaces of the system are described.

**Addressing**

A DML needs ways to address the data that is to be manipulated. In SQL this functionality is provided by the 'where clauses'. Two broad categories of addressing are needed:

- content addressing, used to identify content to be read, deleted, or updated;
position addressing, used to identify positions in the tree where new content must be inserted.

There are many possible location addressing techniques that can be used, the TEI (Text Encoding Initiative) extended pointers being an example. The HyTime location module also proposes very general addressing techniques.

Addressing techniques are based on the parse tree. At a minimum there is the need to address the tree nodes, for instance, through a combination of their ID attributes and a relative address (à la treeloc). It must also be possible to address data chunks (between element nodes). Position addresses can be expressed relative to tree nodes.

Data Manipulation Operations

Data manipulation operations include:

- inserting content at a given position address;
- deleting content (data content or whole nodes);
- updating content (data content or whole nodes).

The type of API (Application Programming Interface) that can be provided to execute these operations depends in part on the validation services requested from the repository. This is the subject of the next section.

Validation

It goes without saying that it is important that the data be kept valid against the corresponding DTD, as the DTD is considered to be the schema of the database. Two approaches are possible to achieve this:

- validate at all times, during each elementary operation;
- validate on transaction boundaries.

Each method has its strengths and weaknesses. The first allows the data to be parsed when inserted into the repository, the content in the repository remaining valid at all times. It is therefore possible to restore the context in the parser and build the parse tree as the data is being inserted in the repository. Because parsing is allowed, it is possible to have a full SGML repository. It is also similar to the way in which relational databases work, ensuring that the integrity constraints are valid at all times.

The second method does not permit the parsing of SGML input since this operation generally requires a valid context, which is not necessarily always available. Parsing XML is permitted, however, provided the well-formedness is preserved. It is thus possible to provide an API to manipulate the tree and validate it against the DTD at the request of the client application, and
on transaction boundaries. One such API could be the DOM (Document Object Model), currently under development in W3C (World Wide Web Consortium).

Both techniques have advantages and disadvantages, and are thus useful in different applications. Without going into too much detail, in general it could be said that the first is well suited to machine processing, while the second is more adapted to interactive manipulation of the repository content (with an SGML editor, for instance).

**Versioning**

Document-oriented applications often have a need for version control. Version control covers many different needs including the tracking of changes made to the documents, the retrieval of past versions, and so on. Database-oriented applications generally do not provide this functionality.

The 'best of both worlds' approach presented in this paper is a system which aims to provide equal support to both document-oriented and database-oriented applications. As such, it provides basic support for versioning, powerful enough to build sophisticated versioning systems, while keeping the fine-grained operations of the data manipulation language.

Keeping this approach in mind, here is a minimum set of features to support versioning in an SGML database:

- each transaction is a logical unit of work, but also increases the version number of the instance;
- an 'undo' feature can be used to restore an instance to a previous version;
- it is also possible to clean-up historical data, when it becomes unnecessary to keep it.

**Sample Transactions**

A system working along these lines has been built by our Group. The following samples illustrate the kind of elementary operation which can be executed by the system. Of course, very complex transactions can be built by combining the basic primitives.

Consider this sample SGML fragment:

```
<SECTION ID="SEC1">
  <TITLE>The section title</TITLE>
  <FIGURES>
    <VALUE ID="V1">1000</VALUE>
    <VALUE ID="V2">2000</VALUE>
    <VALUE ID="V3">3000</VALUE>
  </FIGURES>
  <COMMENTS ID="SEC1-C">
    <p>Some text</p>
    <p>Some more text</p>
  </COMMENTS>
</SECTION>
```

The following transaction
would remove the third value.

The following transaction

would replace the content of the second paragraph.

This would lead to the following result:

Needless to say, such transactions are not intended for end-users. It is very important, however, that such a precise level of control be available to applications:

- when the check-out/check-in approach is used, the modified fragment goes through a difference analyser which generates the transaction, only updating the modified content, limiting history space consumption, and increasing the performance of the check-in operation;

- any program wanting to manipulate the stored data can also generate transactions directly, much like an SQL application updating a relational database.

**Interfaces**

As shown in the diagram below, several interfaces are available to access the SGML database.

- The most important one is the transaction interface (the DML for the SGML database), used to change the instances stored in the database. This is the lowest-level interface, providing fine-grained write access to the stored content.

- Another very important interface is the browsing and navigation API. This API is very similar to the DOM, providing read-only access to the stored objects (elements, attributes, text content, and so on).
- The check-out interface is built using the primitives of the navigation API. This is at a higher level and used by document-oriented applications which are capable of parsing SGML or XML.

- The traditional check-in operation is built using the transaction interface, with the help of an integrated delta analyser. This 'SGML diff' application compares the stored version of the check-in fragment with the new version submitted by the user and generates a transaction which is submitted to the database in the normal way. From the user's point of view, the check-in works at the fragment level; however, the actual changes to the stored content are limited to the parts effectively modified by the user.

**Implementation Considerations**

**Storage Model**

When storing SGML in databases, one common approach is to work at the entity management level. This approach consists in the creation of an entity manager which fetches the entities from a database instead of operating system files. The entities are then stored as chunks in the database and version control acts at the entity level.

This approach is relatively easy to implement and does not require a high level of SGML awareness from the repository. It allows for the storage of the SGML fragments 'as-is', keeping the SGML source intact.

To support the requirement to have fine-grained write access to the stored content, a radically different approach was chosen.
The main stored objects are SGML instances, elements, attributes, text chunks, and processing instructions. Internal entities are resolved, except for SDATA entities. External SGML text entities are also resolved. SUBDOC entities are stored as separate instances in the database, while data entities are stored in the database as separate chunks.

Another point is that this approach allows for the creation of structure-controlled SGML applications, as defined in [Goldfarb 90], pages 588-93. Applications working with the content stored in the SGML database do not need the help of a parser, since the ESIS (Element Structure Information Set) is immediately available through the browsing and navigation API.

When updating, the application needs to provide a transaction, which contains the fragments to be inserted in the database in the form of SGML data, which must be valid at the place where it is inserted in the database. Inserted fragments can be as small as needed (a new paragraph or a new attribute value, for instance). These are parsed by the database and converted to the corresponding storage objects. They are then immediately available for processing through the browsing and navigation API.

Alternatively a check-in operation can be emulated by providing a transaction saying 'update that element with this new content', or by using a diffing process to generate the transaction corresponding to the smallest set of modifications needed to reproduce the changes required by the user.

**Storage Back-End**

As the schema of the database is provided by the DTDs, schema facilities of the back-end database are not used. Thus there is no direct mapping between SGML concepts (elements, attributes, and entities) and relational database concepts (records).

Should an object-oriented database be used, SGML objects could be mapped to stored objects. However, we chose to have an architecture which is independent of the storage back-end. Thus the basic requirement for a storage back-end is the capability to store and retrieve binary chunks. The content of those chunks is managed by the SGML database layer. Additional services of the storage back-end are of course exploited (robustness - commit/rollback, concurrency, security). Additionally, this allows the SGML database to run on top of flat files, as well as with an RDBMS (Relational Database Management System) such as Oracle.

**Real-World Applications**

Descriptions are given in this section of real-world applications of such a transactional interface to SGML repositories.
Manipulating Embedded Structured Data

Consider a document that contains text and highly structured numerical data. The classic check-out/check-in paradigm works well for a user who wants to change the text or the numerical data in a stand-alone authoring environment. This authoring environment could even be a specialized tool should the numbers have a structure that is too complex for display using an SGML editor.

However, if an application program had to manipulate these numbers, it would probably benefit from having them stored in a traditional structured database with flexible and precise access techniques (to perform computations and advanced validations, for instance).

Another situation where the check-out/check-in approach is not practical is when the changes to the content are specified under the form of 'change requests' which are not immediately applied. Typical change requests are 'update that number to this new value', or 'delete this section'. Using the transactional approach, the change request can be defined without the need to do a check-out first. Once defined, change requests can be applied later, in any order.

The approach presented in this paper provides the best of both worlds: a check-out/check-in mechanism with versioning, well suited to an editorial approach, with the flexibility that is expected for structured data manipulation.

Consider a customs tariff regulation, a legal document defining the rates applicable for importation of various goods. It includes textual parts (comments) and highly structured numerical values associated with short text labels (the rates). A traditional approach would lead the systems architect to store the structured data in a relational database and textual parts in a document storage system, with the need to have a complex synchronization mechanism between the two separate databases.

Based on the SGML database, the system can store the rates and textual comments in a single SGML instance, where synchronization between the two kinds of data is ensured at all times. Features of the system include the following.

- A 'traditional' document editor to manipulate the textual parts and perform proof-reading operations on the rates. A check-out/check-in approach is used for those parts, where the end-user selects the parts on which he wants to work by browsing the table of contents of the document.

- A specialized tariff editor is provided to manipulate the rates, providing highly specialized features for manipulating the structure (split, merge, transpose, ...). This editor directly generates transactions submitted to the repository, in the same way as a traditional database application would do with an SQL database.

- The rates data is immediately available to other applications which can access the SGML database to obtain the value of single rates.
Benefits

In short, it can be said that the repository provides a unique storage and access medium for the **all** the data, with capabilities of both:

- a document management system for document-oriented work on textual content by end-users;
- a database management system for the handling of numerical data by automated subsystems and end-user applications with a specialized user interface.

Having a common storage system for both numerical and textual data ensures the consistency of the data at each step of the production process.

Replicating Changes in a Multilingual Environment

In a multilingual environment, where documents are updated frequently, translators spend most of their time finding the changes authors made to the master language version. Once they have found the relevant changes, it appears that most changes are language-independent: numbers have been modified, parts suppressed, and so on. In language-dependent modifications, an important part is structure (eg chapters, tables). Finally, the actual text must be translated.

In practice, many changes authors make to the master language version of the document can be applied automatically to other linguistic versions. Here are some examples of such changes:

- deleting a chapter, graphic, or table;
- modifying numbers in a table;
- when inserting a new chapter or table, a skeleton can be generated in other languages.

The basic principle is simple, using transactions on the SGML database. A specialized difference analyser compares the modified document submitted by the user to the original version in the repository. The resulting transaction is split into a language-independent and a language-dependent part. Both are applied on the master language version while only the language-independent part is applied to the other linguistic versions. The translators only have to complete or update truly language-dependent content.

Benefits

The possibility of defining updates to the repository as transactions (as opposed to check-in of fragments) gives rise to a very efficient solution. Once a 'master' transaction has been computed and split into its language-dependent and language-independent parts, the language-independent part can be applied on any number of 'slave' languages. Moreover, this split enables the translators to be shown a 'content only' view of the changes made by the authors, where all the language-independent changes have been filtered out.
Conclusions

Considering the DTD as the Data Definition Language (DDL) of an SGML repository, an approach to provide an equally sophisticated Data Manipulation Language (DML) has been shown. An SGML/XML repository working along the principles highlighted in this paper can be considered more like a true database management system than a document storage and retrieval system. This mechanism can be used to build complex applications that manipulate structured information stored in the repository, as well as document-oriented systems based on a check-out/check-in interface.

A repository providing such fine-grained data manipulation primitives is a key towards the creation of sophisticated corporate information systems where the data stored in documents is treated on a par with data stored in traditional databases.

Reference


Please e-mail your comments to Stéphane Bidoul at sbi@sgmltech.com.

This paper was first published in the Conference Proceedings of Markup Technologies '98 US, November 1998, pp 101-107.

© The SGML Technologies Group 1998
The European Union's Budget: SGML Used to its Full Potential

Author

Tom Catteau

Abstract

The editorial process of the budget of the European Union provides a good example of a production environment that is entirely SGML-based, and meets severe constraints in terms of production time, quality, and costs.

As such, it illustrates the fact that SGML realizes its full potential when used as a means of manipulating structured documents. It also highlights certain aspects of SGML, usually considered as advanced, making their significance apparent through a concrete example of their use.

Biographical Note

Tom Catteau is a software engineer at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. He specializes in advanced uses of SGML and the provision of object-oriented solutions. He graduated in electronic engineering at the Katolieke Universiteit Leuven, Belgium. He may be contacted at tct@sgmltech.com.

Introduction

For a newcomer, SGML is often considered to be yet another way of formatting documents. Part of the strength of SGML indeed resides in its capability to structure complex documents. However, the full potential of SGML is realized when it is used as a means to manipulate structured documents, where manipulation implies processing through a program or application.

Some aspects of SGML, although fundamental to the standard, are frequently regarded as advanced and not pertaining to an introduction to the subject. But their significance becomes apparent as soon as their use is illustrated in a concrete example.

The editorial process of the budget of the European Union is an annual, on-going process in which different players such as authors, translators, and editors all operate in a common environment to enter, translate, and correct data needed to produce the budget. The system, designed to fulfil requirements in terms of the timely delivery of high-quality documents,
together with short production times, and hence minimized costs, is entirely SGML-based. It has evolved to a complete and mature production environment.

This paper gives an overview of the architecture of the system and describes the rationale behind the key technical choices that were made. It highlights certain aspects of SGML, such as concurrency and links, which are clearly explained by illustrating their use in the budget. It also outlines the principles on which the project's SGML modules are based, where these include conversions to and from popular document formats.

The need for reliability and stability is shown to have led to a client/server system in which SGML acts as the backbone of the modules which govern the production workflow. These modules communicate with each other through SGML-formatted messages.

The implementation has been made possible through the use of a fully-featured SGML parser and an associated application language that combine to make a powerful SGML engine.

**Situation**

The creation of the European Union's budget is a process which takes place in three phases and to which all the institutions which make up the EU's administration contribute. The three phases are:

- the preliminary draft;
- the draft;
- the budget.

The institutions are:

- the Commission;
- Parliament;
- the Council;
- the Court of Justice;
- the Court of Auditors;
- smaller institutions.

For the preliminary draft, each institution makes up its own budget. This phase takes place from February to June. That version is then revised by the Council to produce the draft budget. This takes place from July to August. Finally, that draft is revised by Parliament. This takes place from September to December.
Each version of the budget is published in the eleven official languages of the European Union and consists of seven volumes. Each of these volumes can be considered a large SGML instance, on which many people work simultaneously in different ways, progressing from one version to another.

**System Architecture**

The system used to produce the budget is called SEI-BUD, which stands for *Système d'édition informatisé du Budget*. The system has a client/server architecture. The server is physically located in Luxembourg. Clients are distributed over several places: Luxembourg, Brussels, and Nancy (France).

**Server Architecture**

The server consists of two modules which run continuously in the background: a scheduler and a process-server.

The scheduler is used to process requests originating from clients as messages which are put in a queue that is the input to the scheduler. Each message pertains to a certain class of messages, and for each class, a workflow has to be gone through in order to process the request successfully. The workflow is managed by the scheduler. Processing a message results in the posting of a command onto the queue of the process-server.

The background process-server (BPSRV) manages the execution of all the commands which are put in its queue by the scheduler. The scheduler has attributed a subclass and a class to the command. According to a configuration file, BPSRV limits the number of processes of the same subclass that can be executed at a time. It does the same at the level of the classes.

With this scheme, BPSRV:

- manages the overall load of the machine;
- ensures that processes can only get blocked by the resources they need;
- optimizes the response time by allowing a reduced number of processes to be executed rapidly rather than a large number executing slowly;
- prohibits the concurrent execution of some processes, which might harm the integrity of the system in the multitasking environment;
- implements technical constraints, for example when only one occurrence of a particular type of process can be executed at a time.

When a command to be executed is added to BPSRV's queue, it is accompanied by two callback messages. After the execution of the command, depending on the status of the result (success or failure), one of the two messages is sent back to the queue of the scheduler, together with contextual information given by the scheduler.
Since the number of processes being executed is limited, the input queue might not be empty, and in fact, under peaks of load, might be well full.

The distinction between scheduler and process-server allows for stability and reliability. The system is stable because the failure of a process (which might be caused by wrong inputs, bugs, or external causes) is caught by the system. The system is reliable because of its modular concept and because the principle on which it is based (asynchronous communication between scheduler and process-server through the use of input queues) is simple and consistently used throughout the system.

**Clients**

A client consists of two parts: a control station and an editing station.

The control station is the place where requests are sent to the server, and the server's answer analysed. The different requests are summarized below.

There are two kinds of editing stations: stations specialized in one type of editing, related to the nature of the project, and customized versions of popular text-editors.

**Communication between Client and Server**

The communication protocols used are FTP and X.400, but except for the lowest level the implementation is protocol-independent.

**The Typical Workflow in a Phase of the Budget**

For the creation of a new version of the budget, the previous version is taken as a starting point. Several steps have to be performed to come up with a new version. These steps are:

- authoring, which is done in the master language, French;
- translating into all the other languages;
- reviewing of each document in each language;
- publishing.

In all of these steps the same operations can be performed:

- request for nomenclature;
- consultation of an editorial object;
- reservation of an editorial object;
- editing of an editorial object;
• update of an editorial object;
• release of an editorial object;
• the printing of the differences of two versions of the same editorial object.

There are two levels of workflow. The first one is the system-wide workflow, which involves the sequence of authoring, translation, review, and publication. The second level is at the server where each user request triggers the start of an appropriate workflow. Both levels are managed at the scheduler, but differently.

The system-wide workflow is an *enabling workflow*. From status to status (authoring, translating, editing, and publishing), it allows certain clients to work, but it does not do their work. They have to consult, reserve, and so on. Between the statuses of authoring, translating, editing, publishing, and again authoring, there are other statuses, which are END_AUTHOR, END_TRANSLATOR, END_CORRECTOR, and END_PRINTER. To move to such a status, certain integrity checks are performed (for example the synoptism or alignment of the languages, that is structure comparison among all languages).

**Implementation**

In this section we shall discuss some of the implementation of the system described above. We shall first describe the SGML engine, which serves as the basis for most of the modules developed for SEI-BUD. Then we take a look at DTDs where the emphasis is put on process-orientation. Finally, we discuss some SGML modules which are used in SEI-BUD.

**SGML Engine**

The SGML engine used in this project, and in all applications at the SGML Technologies Group, is the SGML Integrated Toolkit. It consists of a fully-featured SGML parser combined with an application language.

Programs created with this software are stand-alone applications, that can have one type of SGML instance as input, depending on the SGML declaration, and the DTD or DTDs used at the compilation of the program. A application module consists of several parts:

• an optional SGML declaration;
• one or more DTDs;
• one or more link process definitions, which contain the application code.

The execution flow of an application is tied to the DTD using LINKs. Even without knowledge of LINK, the following example will make its use self-evident. Let us take the following example of a DTD which can contain paragraphs and lists.
If we want to associate with that DTD an application that processes instances of that DTD, we need the following skeleton:

```xml
<!LINKTYPE app DOC #IMPLIED [  
  <!LINK #INITIAL DOC  
    -- code for processing DOC starttag and endtag --  
  P  
    -- code for processing P starttag, endtag, and data --  
  LIST  
    -- code for processing LIST starttag and endtag --  
>  ]>
```

This is called a Link Process Definition (LPD) and it is the way provided by the standard to add processing capabilities to a parser.

Apart from a containing envelope, we see that there is one LINK. This link contains the code to be executed when certain tags are encountered in the input. #INITIAL informs the parser that this link should be used when starting to parse an input document. An #INITIAL link is required. Suppose now we want to process P within a list differently from the other Ps (for example to insert bullets or to indent a P within a list). To do so, we create a new link, containing the execution code for a P within a LIST, and we inform the parser that within the context of a LIST, Ps are to be processed differently, by telling it to use the newly-defined LINK when in a list.

```xml
<!LINKTYPE app DOC #IMPLIED [  
  <!LINK #INITIAL DOC  
    -- code for processing DOC starttag and endtag --  
  P  
    -- code for processing P starttag, endtag, and data --  
  LIST  
    #USELINK ListLink  
      -- code for processing LIST starttag and endtag --  
>  ]>
```

This example makes it clear that extensive use of LINKs lifts the burden of the implementation of the execution flow off the developer's shoulders. The developer can concentrate on the management of the execution flow through USELINK.

So much for the coupling of a document instance to the application code. Up to now, everything follows the standard. The application language itself, which implements the application code, is not standardized. The application language used in the SGML Integrated Toolkit was developed in-house, with facilities to deal with common tasks in the processing of SGML instances.
Experience shows that this approach, where applications are inherently tied to the input documents they can handle, proves to be an excellent way of processing SGML instances in terms of flexibility and rapid application development.

Applications created with our software are used in a wide range of areas. Certain applications will test whether some semantic rules, not expressed in the DTD, are fulfilled. Others will extract a table of contents, still others might convert a document to another format, while yet more (as is the case in SEI-BUD and as we shall see shortly), can be used to drive a complete client/server system.

**A Processing-Oriented DTD**

A DTD can be designed from several points of view: from the point of view of presentation, from the point of view of document processing, or from no view at all (which is probably often the case). In an SGML-based system, the quality of a DTD corresponds to the ease with which its instances can be handled. We talk of processing-oriented DTDs. Processing-oriented means two things:

- the DTD ideally should reflect the possibilities of the system;
- the DTD should be designed to facilitate the processing of its instances.

**Capabilities of the System**

As stated above, the DTD ideally reflects the possibilities of the system. For example it makes no sense to allow tables within tables in a DTD if neither of the conversion filters can process them, nor the printer print them. Of course, a program can be developed to manage more complex DTDs, but as the modules work in a chain, the DTD should reflect the possibility of the overall system, hence the capability of the weakest link of the chain. This is a simple principle, but when it is strictly applied it prevents unnecessary development, and it confines and defines the area in which those applications are tested.

**Processing of Instances**

A DTD should also be designed to facilitate processing. To illustrate that, two simple examples will be given.

First consider the following definition of a cell which can contain either plain #PCDATA or #PCDATA within a P:

```xml
<!ELEMENT CELL - - (#PCDATA|P)>
<!ELEMENT P - - (#PCDATA)>
```

In this case, all conversion filters must duplicate the code for processing the content of a cell. The following alternative is hardly better:
The European Union's Budget: SGML Used to its Full Potential

<!ELEMENT CELL - - (P?)>
<!ELEMENT P - - (#PCDATA)>

Here, the aim is to avoid empty Ps within a cell. But in this case each conversion to that DTD should check whether the cell content is empty in order to know whether or not to include tags for P. To avoid code duplication and unnecessary checks, the following definition is better:

<!ELEMENT CELL - - (P)>
<!ELEMENT P - - (#PCDATA)>

As another example, let us compare the two following constructions:

<P>Introduction to the list</P><LIST><ITEM></ITEM></LIST><P>

and

<P><LIST><INT.LI>Introduction to the list</INT.LI><ITEM></ITEM></LIST></P>

In the first construction we do not know we are in an introduction to a list until we encounter a LIST starttag, whereas the second solution leaves no doubt from the beginning. In other words, the second solution is more processing-oriented than the first.

These two simple examples make clear the fact that a DTD, whose design was made from a processing-oriented point of view, will significantly alleviate and streamline the work to be done in associated applications.

Conversion Filters

Conversion filters are used to convert an editorial object between different DTDs and to and from popular document formats. Document formats used in SEI-BUD are Word RTF, WordPerfect, and Interleaf ASCII. Whereas conversion filters from DTDs to presentation formats are not that hard to imagine, the converse is less obvious. Nevertheless, documents formatted in any format can be considered to be instances of a particular DTD. Let us explain this with Interleaf documents.

An Interleaf document contains a declaration part and a content part. The content consists of components and tables. The DTD for an Interleaf document thus looks like

```
#<!ELEMENT DOC O O (DECL,CONTENT)>
#<!ELEMENT DECL O O #PCDATA>
#<!ELEMENT CONTENT O O ((COMP|TABLE)*)>
```

Of course, the level of detail used in the DTD depends on the information that one expects to be present in the document and that one wants to extract. In Interleaf, for example, and in the case of SEI-BUD, it is unnecessary to take graphics into account since they do not occur in budget documents.
Shortrefs

Once a DTD is defined, the markup in the document has to be coupled to start-tags and end-tags of that DTD. To do so, short references are used. Several steps have to be performed. First, the markup that is to be recognized is added as a short reference to the SGML declaration. Here we want to recognize the end of the declarations, the start of a component, and the start of a table. So we add to the SGML declaration in the SHORTREF section the strings

<End Declaration>
'<"'
'<!Table'

Then we link them to start-tags and end-tags in two steps: entity definitions and a shortref map.

```xml
#<!ENTITY edecl ENDTAG DECL>
#<!ENTITY scomp STARTTAG COMP>
#<!ENTITY stable STARTTAG TABLE>

#<!SHORTREF docmap
  '<End Declaration>' edecl
  '<"'    scomp
  '<!Table' stable
>
```

This means that when `docmap` will be active, the string

```xml
<End Declaration>
```

will be interpreted by the parser, via the entity `edecl`, as the end-tag of the element `DECL`. To make `docmap` active for the complete instance, we map `docmap` to the element `DOC` using the following markup:

```xml
#<!USEMAP docmap DOC>
```

This ensures that within the element `DOC`, the parser will recognize and interpret the strings as declared in the SHORTREF map `docmap`.

Character transformations are implemented using the same mechanism of SHORTREF and USEMAP.

As an extension to the standard, the SGML Integrated Toolkit allows short references to be regular expressions. A broad range of markup can thus be expressed in a more concise way than it would be without regular expressions.

If, in theory, it is simple to consider any document format as a particular DTD, in practice many details make that process a non-trivial task. The hardest part in converting a text-editing document to an SGML instance resides in the design of an appropriate DTD. The application itself is straightforward, and usually consists of rearranging elements and their content.
Tables

In addition to text and figures, the budget also contains tables. Tables are divided into table classes, each table class being implemented as a subdocument. This means that each table class has its own DTD and its own associated application code. In our software system, using the SUBDOC feature, a seamless integration of subdocuments and their applications with the base document and its application code is possible.

SGML Declaration

The only changes made to the SGML declaration for conversion filters are those which are meant to avoid some characters, such as

```
<
```

being wrongly interpreted by the parser. Therefore the start-tag opener (STAGO) is redefined to

```
#<
```

the end-tag opener (ETAGO) to

```
#</
```

and the markup declaration opener (MDO) is redefined to

```
#<!
```

as indicated.

This explains the

```
#<!
```

markup found above.

Printing of the Differences

Printing the differences occurs in two steps. First there is a non-SGML module that compares the two versions of the same document. The result of this module is the input to an SGML module which converts its input to an Interleaf ASCII document that highlights the differences. Let us have a look at the intermediate result. The intermediate result is a document that combines the old and the new version of the document. If for example the first version shows

```
P>First paragraph.tP><P>Second paragraph.tP>
```

and the second
The European Union's Budget: SGML Used to its Full Potential

First paragraph. Second paragraph. Third paragraph.

then the intermediate result might be

First paragraph. Second paragraph. Third paragraph.

To understand this output, we should first notice that this document contains two concurrent grammars, old and new. The old grammar is the original one, to which the element

```xml
<!ELEMENT ins - - (#PCDATA)>
```

has been added. The new grammar is the original one, to which the element

```xml
<!ELEMENT sup - - (#PCDATA)>
```

has been added.

For each tag we say to which grammar it belongs by stating this between parentheses. When we suppress all the tags belonging to the new grammar, as well as the element ins (insertion) and its #PCDATA, we get

```xml
<old>First paragraph. Second paragraph. Third paragraph.</old>
```

that is, the old document. Conversely, when we suppress all the tags pertaining to the old grammar, as well as the element sup (suppression), we get the new version of the document:

```xml
<new>First paragraph. Second paragraph. Third highlighted paragraph.</new>
```

The meaning of the ins and sup elements is now straightforward: ins indicates to the old grammar that what lies within its tags has been inserted, whereas sup indicates to the new grammar that what lies within its tags pertains to the deleted part of the old instance. This document is processed as follows. In a concurrent grammar document tags pertain to one grammar, but the #PCDATA is common to all grammars. So for the common elements one LPD (for example associated to new) processes the document, the other LPD skipping all processing. When the old grammar encounters an ins, it sets a flag that tells the new LPD that what follows is inserted. The flag is unset at the end-tag of the ins. In between the two ins tags there are only new tags. So ins contains only #PCDATA. The processing of that #PCDATA is skipped in the old LPD. The opposite happens when a sup element is encountered. Then a flag is set to inform the old LPD that it should process what follows.

This application shows how natural the use of concurrent DTDs can be.

Scheduler

The scheduler is the server module that processes users' requests. It is completely implemented as an SGML module.
A first DTD is used to retrieve messages from the input queue.

```xml
<!DOCTYPE QUEUE[
  <!ELEMENT QUEUE (READMSG*)>
  <!ELEMENT READMSG (#PCDATA)>]
```

The application of the start-tag of the queue passes in a loop READMSG start-tags and end-tags to the parser. Each time the parser gets a READMSG start-tag, it executes the associated application, which retrieves the first message in the scheduler's input queue and passes it to the parser, where the associated application is executed, before the READMSG end-tag is parsed.

The workflow associated with a request consists of states and events. A message is an event in the context of some state. A state represents either the initial state (in that case the event will be a request), or the execution of a process, the event being the result of the process, which is failure or success. The event E_PROC_OK is used to announce the success of the process; PROC associated with the state S_PROC; E_PROC_NOK is used to announce failure of PROC.

The user sends his request in the form of an SGML-formatted message:

```
<(REQTYPE)S_REQ> <(REQTYPE)E_REQ ATT1="VAL1" ATTN="VALN">t<(REQTYPE)S_REQ>
```

That message, being the input to the scheduler, will be parsed by the SGML parser and the appropriate code will be executed according to the LPD of the DTD REQTYPE. Typically, a process will be scheduled and sent to the process-server, together with the messages that have to be sent back to the queue of the scheduler in case of success and in case of failure. These messages will look like:

```
<(REQTYPE)S_PROC1><(REQTYPE)E_PROC1_OK ATT1="VAL1" ATTN="VALN">t<(REQTYPE)S_PROC1>
```

(where att1 and att2 are whatever attributes, and val1 and val2 their values) for success and

```
<(REQTYPE)S_PROC1> <(REQTYPE)E_PROC1_NOK ATT1="VAL1" ATTN="VALN">t<(REQTYPE)S_PROC1>
```

for failure.

Typically the attributes contain information needed by the scheduler (username, language, etc). While the process PROC1 is being executed, other messages from any type of request can be handled by the scheduler. Then when BPSRV puts one of the messages onto the scheduler's queue, the scheduler will process that message, first by restoring its context, and then taking the appropriate action.

By way of example, let us sketch the outline of the implementation of the graph for consultation of an editorial object. The tasks that have to be performed are:

- extraction of the data from the repository;
- conversion of the data into the appropriate format.
The execution of this graph will be triggered by a message, posted by a user. The format of the message might be:

```
&lt;(CONSULT)S_REQ;&lt;(CONSULT)E_REQ USER="VAL1" VOL="VAL2" ID="VAL3" LG="VAL4" FORMAT="VAL5">&lt;(CONSULT)S_REQ&gt;
```

indicating that the user VAL1 makes a request for consultation of the document in volume VAL2 with id VAL3 in the language VAL4 and in the format VAL5. The relevant portion of the grammar CONSULT is:

```
&lt;!ELEMENT E_REQ - O EMPTY&gt;
&lt;!ATTLIST E_REQ      USER   CDATA #REQUIRED
LG CDATA #REQUIRED
ID CDATA #REQUIRED
VOL CDATA #REQUIRED
FORMAT CDATA #REQUIRED&gt;

&lt;!ELEMENT S_REQ      O O 55E_REQ, S_EXTRACT)?&gt;
```

The SGML engine will redirect this message to the application associated with the elements S_REQ and E_REQ in CONSULT.

```
&lt;!LINKTYPE lconsult CONSULT #IMPLIED [
&lt;!LINK #INITIAL
S_REQ #USELINK S_REQ_LINK &gt;
&lt;!LINK S_REQ_LINK
E_REQ --application code for event E_REQ -- &gt;]
```

The application for E_REQ will post a command to the background process-server for extracting the appropriate editorial object. Together with that command will be associated a message for the case when the command fails and a message for the case when the command succeeds. These messages will look like:

```
&lt;(CONSULT)S_EXTRACT&gt;&lt;(CONSULT)E_EXTRACT_OK
USER="VAL1" VOL="VAL2" ID="VAL3" LG="VAL4" FORMAT="VAL5">&lt;(CONSULT)S_EXTRACT&gt;
```

for success and

```
&lt;(CONSULT)S_EXTRACT&gt;&lt;(CONSULT)E_EXTRACT_NOK
USER="VAL1" VOL="VAL2" ID="VAL3" LG="VAL4" FORMAT="VAL5">&lt;(CONSULT)S_EXTRACT&gt;
```

for failure.

Actually, the S_EXTRACT element following E_REQ in the definition of S_REQ indicates that the state following E_REQ will be S_EXTRACT. This knowledge can be used to generate automatically the outer part of the above messages.

In case of failure, the next state will be S_ERR. Otherwise, the conversion still has to be done. This gives as definition for the base element:
The European Union's Budget: SGML Used to its Full Potential

The exploitation of SGML

```xml
<!ELEMENT CONSULT O O (S_REQ|S_EXTRACT|S_CONVERT|S_ERR|S_END) * />
```

The reason for this structure is to decouple the different steps of the graphs and thus to allow several consultations to be performed at the same time.

The definitions for the states are:

```xml
<!ELEMENT S_REQ O O ((E_REQ,S_EXTRACT)?)>
<!ELEMENT S_EXTRACT O O ((E_EXTRACT_OK,S_CONVERT)| (E_EXTRACT_NOK,S_ERR)) ?>
<!ELEMENT S_CONVERT O O ((E_CONVERT_OK,S_END) | (E_CONVERT_NOK, S_ERR)) ?>
<!ELEMENT (S_ERR, S_END) O O (E_SEND)>
```

Finally, the definitions for the events:

```xml
<!ELEMENT E_REQ - O EMPTY>
<!ELEMENT E_EXTRACT_OK - O EMPTY>
<!ELEMENT E_EXTRACT_NOK - O EMPTY>
<!ELEMENT E_CONVERT_OK - O EMPTY>
<!ELEMENT E_CONVERT_NOK - O EMPTY>
<!ATTLIST (E_REQ,E_EXTRACT_OK, E_EXTRACT_NOK, E_CONVERT_OK, E_CONVERT_NOK) USER CDATA #REQUIRED LG CDATA #REQUIRED ID CDATA #REQUIRED FORMAT CDATA #REQUIRED>
```

The skeleton for the LPD for this graph will be:

```xml
<!LINKTYPE lconsult CONSULT #IMPLIED [  
<!LINK #INITIAL  
S_DEM #USELINK S_DEM_LINK  
S_EXTRACT #USELINK S_EXTRACT_LINK  
S_CONVERT #USELINK S_CONVERT_LINK  
S_ERR #USELINK S_ERR_LINK  
S_END #USELINK S_END_LINK  ]>
<!LINK S_DEM_LINK  
E_REQ  
-- application code for event E_REQ --
>
<!LINK S_EXTRACT_LINK  
E_EXTRACT_OK  
-- application code for event E_EXTRACT_OK --  
E_EXTRACT_NOK  
-- application code for event E_EXTRACT_NOK --
>
<!LINK S_CONVERT_LINK  
E_CONVERT_OK  
-- application code for event E_CONVERT_OK --  
E_CONVERT_NOK  
-- application code for event E_CONVERT_NOK --
>
<!LINK S_END_LINK  
E_SEND  
-- application for event E_SEND --
>
<!LINK S_ERR_LINK  
E_SEND  
-- application for event E_SEND --
>
]>
```

Summarizing, each type of request will be implemented using one DTD and an associated application. There will be as many DTDs as there are types of requests. These are
implemented as concurrent DTDs each with its associated LPD. Since messages do not contain #PCDATA, the different messages will not interfere with each other.

The use of concurrent DTDs for the different types of requests ensures a strict separation of each request (each DTD has its own application code), while permitting different requests to be put in any order, since the parser will direct every message (which is an element) to one LPD according to the DTD to which the message belongs.

In practice, the generation of the DTD, the skeleton of the associated LPD, and even the messages associated with the commands, can be generated automatically from a formal high-level description of the graph. That is beyond the scope of this paper.

As a conclusion, we might say that the scheduler represents an SGML module where the input document is a sequence of messages created in real time, whose purpose is to drive the server. Using the mechanism of concurrent grammars for each type of request, the scheduler and BPSRV implement a truly multitasking environment, that is entirely SGML-based. Benefits of using SGML as the backbone of the system include:

- the richness that messages can have, which boils down to the richness SGML instances can have;
- the inherent multitasking capabilities of concurrent grammars;
- all that being directed by a proven and fully-featured SGML engine.

Repository - Document Storage Model

The document is stored in an Oracle database. As previously stated, the budget contains eleven languages and seven volumes. One table is used per language and per volume.

In the budget's DTD, every element of the nomenclature is assigned a required ID attribute. Each editorial object has its root ID. To store the structure associated with an ID, at the level of that node, each of its subnodes is stored with its ID in a recursive way. Recursion ends at the level of the granularity of the DTD. As soon as there is no ID in the substructure of an element, that element is considered to be a leaf in the tree, and its content is stored as a whole, as one string in the database.

This scheme allows us to implement efficiently incremental updates and retrievals of parts of a document. It also allows for a locking mechanism up to the level of granularity of the DTD, and all features of relational databases (indexes and recovery mechanisms) can be used to enhance the system's performance.

Versioning

When an update is sent to the repository, only the updates are actually stored, and the version number of the document is incremented. The history of modification and creation is kept on disk and any version can be consulted online at any time.
Conclusions

We have described a complex SGML-based client/server system that is used for the creation and maintenance of the European Union's budget, a huge eleven-language document, revised three times a year.

We have shown that an SGML system can be much more than just having SGML instances as input and output. We have described and illustrated how SGML is used in every aspect of the system, ranging from the server modules, via SGML-based processing modules, to an SGML-formatted messaging scheme between clients and server.

Finally, we have outlined how at the very heart of the system presented here is the SGML Integrated Toolkit, the SGML Technologies Group's fully-featured SGML parser and integrated application language.

Please e-mail your comments to Tom Catteau at tct@sgmltech.com.

This paper was first published in the Conference Proceedings of SGML'97 US, December 1997, pp 645-53.

© The SGML Technologies Group 1997
The Addition of a Multilingual Component to an Existing Document Processing System

Author

Tom Catteau

Abstract

This paper discusses the addition of a multilingual component to an existing document processing system, where a trade-off has to be chosen between innovation in terms of new functionality for multilingual processing and the stability of the system.

Biographical Note

Tom Catteau is a software engineer at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. He specializes in advanced uses of SGML, and the provision of object-oriented solutions. He graduated in electronic engineering at the Katolieke Universiteit Leuven, Belgium. He may be contacted at tct@sgmltech.com.

Introduction

Many organizations, both public and private, deal with multilingual documents. A major issue when dealing with such documents is the concern for equivalence between different linguistic versions of one document: the concern for synoptism or alignment. The check for synoptism takes place at two levels: at the structural level of the document, as well as at the content level, where, although not explicitly expressed in a DTD, certain types of information might be present which are language-independent.

This paper is concerned with the addition of a multilingual system on top of an existing document processing system. Two points are of interest when discussing the addition of a multilingual component. First, it is mandatory to avoid modifications to the existing system as much as possible in order to guarantee that the system keeps its current level of stability. Secondly, since new information will be extracted at the content level for the check of synoptism, this information will be given permanency by enriching the DTD at the level of the repository. Since it is preferable not to change the modules which process the repository's content and use the original DTD, two new modules will have to be created. The first of these two modules will be inserted at the point where the repository is updated. It will extract relevant information and replace that information with structure in the repository. The second module, to be inserted at the point where fragments of the repository are retrieved, consists of converting the newly-defined elements in the enriched DTD into formatted content, or in
removing some attribute values. The added value of the enriched DTD will be exploited by
the synoptism check module, which will verify the equivalence among different languages, as
well as providing a way to indicate points of inconsistency to the reviewers.

In the first section, the context of the discussion will be set. To do so, a general architecture of
a document processing system and of a multilingual repository is described. Then, typical
language-independent components are described, as well as a way to formalize them. Then
follows a general scheme for updating and extracting fragments into and out of the repository.
Subsequently, the synoptism check is discussed. Thereafter, the incremental implementation
and the conversion of legacy documents is discussed. Finally, the versioning of multilingual
documents is described briefly.

This paper is the result of experience gained with projects carried out, among which a project
for the editorial system for the budget of the European Union at OPOCE, the Office for
Official Publications of the European Communities in Luxembourg.

**General Architecture**

When documents are published in several languages, the various instances represent the same
version of one document, in different languages. We thus can speak of different views of a
single document. As a consequence, special care has to be taken to ensure that what is written
in the different views of each language instance reflects the same content. For this purpose, a
multilingual repository is introduced, in order to distinguish language-independent from
language-specific content.

**Multilingual Repository**

In order for the system to be able to manage the consistency of the document, the repository
will be split into two parts: the language-specific (LS) repository and the language-
independent (LI) repository. The LI-repository will hold language-independent features of the
document, whereas the LS-repository will contain, on a per language basis, data which is not
controlled by the system. Three types of modules will directly operate upon these repositories.

These modules include functionality for:

- creating a fragment of a document in a language, based on the LI-repository's content
  and the language-specific repository for that language;

- splitting a fragment of a document in language-independent and language-specific
  parts;

- verifying the synoptism rules.

The LI-repository and the LS-repository, together with these modules, give rise to a
multilingual repository.
Language-Independent Versus Language-Specific Content

Language-independent content is a substructure of a fragment whose presence, structure, and/or location can be matched in several (or all) languages present in the repository. This typically includes structure and floating elements within the data content. What remains is what is not explicitly handled by the system and is language-specific. This typically includes most of the data content.

Language-independent features are only of interest insofar as they serve at least one of the two following purposes:

- they are of use in the construction of a fragment of the document (these features decide on the structure of the document fragment);
- they serve during the checking of the synoptism (here this feature is detected but its presence cannot be enforced automatically).

DTDs for the LI and the LS-Repositories

A document will be stored partly in the LI-repository, and partly in the LS-repository. Naturally, in each repository DTDs will be used which will be derived from the document's DTD.

For the sake of clarity, the example in this section uses the scheme for concurrent DTDs, even if in the implementation another scheme might be used.

The LI-DTD

The LI-DTD must reflect both the language-independent structures and the floating elements. These can in turn contain language-independent structures, and so on. First, the case where the document contains one section and two chapters is considered. This tree will be reflected in the LI-DTD's instance as follows:

\[
\begin{align*}
\text{\langle LI \text{SECTION ID} = \text{AAFGH}\rangle} \\
\text{\langle LI \text{CHAPTER ID} = \text{AAFGI LEAF} = \text{Y}\rangle} \\
\langle / \text{LI \text{CHAPTER}} \rangle \\
\text{\langle LI \text{CHAPTER ID} = \text{AAFHA LEAF} = \text{Y}\rangle} \\
\langle / \text{LI \text{CHAPTER}} \rangle \\
\langle / \text{LI \text{SECTION}} \rangle
\end{align*}
\]

First note that since the LI-DTD might be used in conjunction with the LS-DTD, in order to avoid interference between both instances the LS-DTD will not allow any PCDATA to occur in its instances.

Depending on the nature of the document, it might be decided that paragraphs too are part of the language-independent structure and that this level should be included in the language-independent part. The corresponding instance might be:
Note that every element that is part of the language-independent structure must be uniquely identified. This is necessary for the extraction module to be able to locate these structures in the LI-repository.

Also note that the LI-DTD will be a copy of the document's DTD up to the level that is the same in all languages.

The difference between the two examples also makes clear another point. Typically, tables are elements with a language-independent structure. In the first example, the TBL element itself is floating. Its presence can be detected but its location cannot be derived from the LI-repository. The TBL could be included as floating element as follows:

Here TBL has been inserted as an element of the first chapter. However, in this example TBL, being a floating element, will be an inclusion in the CHAPTER element. Again, the TBL element has an ID which will be used for the extraction.

Now consider a reference, which will, for example, be caught using regular expression recognition. Its occurrence in one language describes it completely; there is no need for identification of the reference. With a reference in the first chapter, the example might be:

This example tells us that in the first chapter, a table and a reference should be present in every language, but without any order of occurrence being prescribed.
As a general rule, the LS-DTD contains the same structure as the document's DTD, up to a certain level. Floating elements are inclusions to this DTD. All structure elements have an ID, as do the floating elements that have an internal language-independent content. Floating elements that do not have any language-independent content need not have an ID.

The LS-DTD

The LS-DTD will be the document's DTD itself. The document will be stored as it comes from the user. During the extraction, only relevant elements will be kept, and new elements will be added where needed. The instance in the LS-repository could be:

```xml
<LS:SECTION ID=AAFGH>
  <LS:CHAPTER ID=AAFGI>
    <LS:TBL ID=AAFGK>
      ...
    </LS:TBL>
    <LS:P ID=AAFGL>
      </LS:P>
  </LS:CHAPTER>
</LS:SECTION>
```

In this case, during the extraction the first paragraph in both the first chapter and the second chapter will be added at extraction time.

A General Scheme for Extraction and Update

Extraction

To start the extraction of a fragment (identified by an ID), first there is the extraction of the appropriate fragment out of the LI-repository and the LS-repository. The extraction is then driven by the LI fragment. For the sake of extraction, language-independent data can be subdivided into three categories.

- Data which can be used to construct a view in another language.
- Data which can only be used to check synoptism: floating nodes which are not uniquely identifiable within the encompassing element. In this case they cannot be used for the extraction.
- Floating nodes which are uniquely identifiable within the containing element (inclusion elements of the LS-DTD). Either they are already present in the LS-instance, in which case their location is known and their content can be extracted from the LI-instance, or they are not yet present in the LS-instance. They still can be added to the document, for example at the end of the element. This minimizes the user's work, since he will only have to move the element to the appropriate place.

The resulting fragment consists of the structure which is retrieved from the LI-instance, to which the LS-instance's content is added. When an identified floating element is encountered, it will only be added to the output if it also occurs in the LI-instance.
The extraction process itself can be done the usual way, either after having put both structures in memory, or as parsing progresses. In the latter case, both LI- and LS-instances are parsed concurrently. This approach is better suited to larger documents.

Storage

When storing a fragment of the document in a particular language in the repository, it is necessary to distinguish between two cases: in the first there is a master language; in the second there is not.

Master Language

When there is a master language, only that language has the right to update the LI-repository. From an organizational point of view, this is often the simplest way to ensure proliferation of language-independent data across all instances. An update of the document in the master language results in an update of the LI-repository. Updates in other languages will not affect the language-independent data repository. The master language choice is often a functional one: a document is written in the master language; only then is it translated into the other languages. The notion of a master language is equivalent to that of a user with more access and lock priority than the others.

No Master Language: Locking Mechanism and Differential Updates

When there is no master language, all languages are equivalent, technically speaking. When extracting a portion of a document, that fragment consists of both language-specific and language-independent content. This means that the update will also consist of an update of both the language-specific and the language-independent part. There can be a conflict if two languages want a lock on overlapping parts of the document. This conflict can be resolved through the use of differential updates. With this construction, the updated version is compared with the version at lock-time. Only modified parts will be added to the repository.

Using this principle, non-exclusive locks become possible; they offer a significant improvement to system flexibility. Conflicts are of course still possible, but this is an organizational problem.

The Checking of Synoptism

The goal of synoptism checks is to verify the consistency of the different language views of a document. Synoptism checks are always performed on the richest form of the DTD. Again it is necessary to distinguish among structure elements, identifiable floating elements, and other floating elements. Since structure elements are used for the construction of a document, by definition there is synoptism at that level. It is thus possible to confine the discussion to floating elements.

For floating elements, only their presence can be verified. Two approaches are possible with regards to the check of synoptism: a global and a language-based approach.
Global Check

In a global check, a particular feature is tested across all language views. A convenient way of implementing synoptism checks is to add a language attribute to all floating elements in the LI-DTD. During the update of a fragment in a particular language, this attribute is then updated accordingly. To check the synoptism then means checking the presence of the languages in the language attribute.

Language-Based Approach

As an extension to the global checking, language-based checking can be used. For several reasons it could be appropriate to perform more checks on one pair of languages than on another. Such reasons could be that two languages are more similar to each other than other languages are; or simply that there is a better knowledge of some languages than of others.

There may also be other external reasons.

This can be easily implemented by a configuration that states which languages are aware of a certain feature. In that case, the language attribute of elements will only be checked for the presence of those languages which know of that feature. Special cases must be implemented by ad hoc customizing of the synoptism check.

Non-coercive Versus Coercive Implementation

In case of a coercive approach, only updates which do conform to the synoptism will effectively be performed. In this case, there must be a master language against which the other languages can be matched. With a non-coercive approach, even inconsistent documents are updated. The reasoning behind this is that an incomplete new version is better than a complete but 'outdated' version.

An Incremental Approach to Implementation

Owing to the brittle nature of language synoptism, it is important to perform an incremental implementation. Incremental implementation can be done at two levels concurrently: feature by feature, and language by language.

Feature by Feature Implementation

Typically, there is a need for conversion from the DTD in the repository to some user format. Even when an SGML editor is used, tables, for example, still need to be converted to one of the formats supported by the editor. Other operations also have to be performed. Thus between the extraction from the repository and the delivery to the user, various modules have to be gone through.
As experience shows, every change in a DTD, even a minor one, ripples through the complete system, and has an impact on most modules. To ensure the stability of the existing system, the DTD should remain unchanged. This means that between the repository and the first module a downwards conversion has to be performed. On the way back, an upwards conversion is needed in order to enrich the document.

The disadvantage of this approach is that there is no possibility at the user level to benefit from the new functionality when inserting new data. Only after an addition proved to be useful, can the adaptation of the modules be considered. This will allow the users to benefit from the proven functionality.

In a feature by feature implementation, one language-independent feature at a time is added to the system. Only after the upwards conversion has succeeded is it possible to go to the next feature.

**Language by Language Implementation**

Here, the new feature is first tested for a set of languages. After that, it can be extended to the others.

**The Conversion of Legacy Documents**

As long as a particular feature is not enforced by a process, that is as long as the checking of a particular feature is left to the user, there will be no error-free existing instances with respect to that feature. This means almost by definition that after the conversion from the current DTD to the one which includes the new features (this process is usually done by pattern matching), the resulting document will not be synoptic with regards to that new feature. The quality of the automatic conversion will of course depend on the quality of the original documents. Even with inconsistencies in the repository, the situation can only be better than what it was before the introduction of the new feature.

**Versioning of Multilingual Documents**

**Versioning**

The goal of versioning is to be able to retrieve different versions in the course of the life cycle of a document.

The notion of versioning of documents in itself is clear. It means that it must be possible to retrace every version of a document during the life cycle of the document, where there is a new version at each update of the repository. When only one language is concerned, updated fragments will be consistent, and it is not difficult to keep the complete document consistent.
The Addition of a Multilingual Component to an Existing Document Processing System

Versioning for Multilingual Documents

When speaking of multilingual documents, the notion of versioning becomes less self-evident, and the question of consistency is more difficult to answer.

Versioning

Consider the following situation. In one language, several changes are made which have an impact on the language-independent content. Then, even without any explicit changes to the other language views, they too will have changed. The situation is still more difficult to handle where during a lock on a fragment in one language, the corresponding constructive part is changed. In this context, there is a wish to answer the following question: how is it possible to determine or extract a version of a document from the repository, and what is the value of that extracted version?

For the versioning, the following system is used. A version number consists of two numbers. The first is the version number for the language-independent part. The second is incremented each time a language part is updated without modification to the structure part.

The extraction itself is rather simple. On the basis of a given version number, the structure fragment corresponding to the first number is taken, and for each languages, the language fragment corresponding to the second number; in the case when a version number does not exist for the given language, the previous version is taken.

Consistency

Consistency can only be guaranteed to the extent that is delivered by the language-independent data which is explicitly present in the data repository. No other consistency can be guaranteed, unless all language versions of a document are updated at the same time. This is a constraint that usually cannot be enforced. Indeed, different people working on different languages work at a different pace, and synchronization comes down to the lowest common denominator. This means that most of the time the repository might not be aligned.

It could be part of the workflow to prevent entering a particular state (eg dissemination state) unless the rules of synoptism are obeyed. However, care must be taken. Indeed, when synoptism is enforced, there are situations where it is possible to see that users circumvent the synoptism check by inserting spaces or dots to fill structure element they do not need, but without whose presence there is no synoptism. This means that the benefit of enforcing rules has to be carefully weighed against the risk of having users fiddling so as pass the document through a state.

Differences Among Versions

It is frustrating when there is a need to skim many versions of small fragments in order to find two consecutive versions with differences. Since only the modified parts are physically updated, finding the previous different version of a fragment comes down to going through the previously stored parts of the fragment and retaining the highest version number.
Conclusions

In this paper, how to add a multilingual component to an existing document processing system has been described. As soon as there are enough similarities among the different languages, synoptism starts to be an amenable way to improve the quality of a document; the ultimate goal of a multilingual component is the enhancement of the overall quality of the document from the point of view of language-independent aspects of that document.

Quality of a document may be hard to quantify, but it has been shown that as a tool to improve the quality of that document, synoptism certainly has an added value.

Please e-mail your comments to Tom Catteau at tct@sgmltech.com.

This paper was first published in the Conference Proceedings of SGML'98 Europe, May 1998, pp 305-11.

© The SGML Technologies Group 1998
SGML in a Multilingual Environment

Author

Pierre Colot

Keywords

- Multilingual
- Documents
- Validation

Abstract

The problems relating to multilingual documents are introduced in the context of the official publications of the European Union. Currently dealing with eleven languages, the new system has to be capable of dealing with yet more. The solution is the development of an SGML validation workshop, able to take account of all the problems which arise in an efficient and effective manner.

Biographical Note

Pierre Colot is a senior software engineer employed by ISEA in Luxembourg, a member of the SGML Technologies Group. He specializes in SGML workshop development. Prior to this he built information systems to manage WAN hardware cables in Belgium and participated in the European Ariane 5 space programme. He graduated from the Free University of Brussels in 1988. He may be contacted at pco@isea.lu.

Introduction

The introduction of every new country to the European Union implies an increase in the number of national languages used in official publications. This increase means that the validation cycle of the publications has to be reviewed.

In managing multilingual publications, the following have to be addressed:

- how to represent multiple character sets within a publication;
- how to represent multiple languages within a publication;
- how to consolidate the different language versions of the same publication;
- how to modify the structure of all language versions simultaneously;
- how to publish the different language versions of an extract of a publication;
- how to browse through a multilingual publication;
- etc.

New strategies have to be developed to solve these multilingual problems. It is held that SGML can assist in several ways. The separation between structure and content, character set and coding, etc, offers possible solutions.

This paper comprises four main parts:

- a global presentation of the context of the problems;
- the general architecture of a workshop that is faced with the problems;
- the advantages of using SGML in their solution;
- a conceptual description of how to manage multilingual publications within the newly-developed SGML validation workshop.

**Global Presentation**

The reasons that led to the adoption of a multilingual approach when building the workshop are outlined. Firstly, the context in which the multilingual publications are used will be discussed. Secondly, the workload created by multiple language versions will be described. Thirdly, the economic factors of managing costs of these multiple versions will be highlighted. Fourthly, managing the complexity of numerous DTDs will be presented.

**Context**

In the early 1980s the Office for Official Publications of the European Communities (OPOCE) was looking for a standard format for exchanges of its electronic publications. Given the wide variety of documents involved and the heterogeneous nature of existing computing systems, the search turned towards the use of an international standard. The SGML standard emerged as the best solution, as its flexibility made it possible to describe and monitor the various structures used in the publications. To do this, a European character set was specified, the SGML declaration was formalized, and the DTDs were defined.

The first version of the Formalized Exchange of Electronic Publications (Formex) standard appeared in 1985.
In the context of the contracts which the SGML Technologies Group has undertaken for OPOCE, publication certification workshops have been established. These workshops produce the certified versions of the Official Journal C, L, and S series.

Within these workshops, the tasks of several dozen translators are distributed and assisted by the documentary systems. Some eight thousand pages are produced every week. The function of these workshops is to monitor SGML conformity and to check for consistency between the paper version of the publications and the corresponding electronic version.

Workload

The end of the 1990s brought the prospect of the enlargement of the European Union, the diversification of publications, and the widespread use of distribution techniques on the Web. In order to respond to these new challenges, plans were made for a complete revision of Formex and the reorganization of the production cycle. The forthcoming accession of central European countries entailed the need to support Slav languages. By extending the character set to include ISO character entities, it became possible to write treaties between the European Union and various international partners. Once each type of publication had been classified, some forty DTDs were identified. Following a complex revision process, twenty-three Formex V3 DTDs came into production in April 1999.

To ensure consistency between the paper versions and the electronic versions, the new production cycle requires that the photocomposition generation carried out by the printers be based on the original SGML instances.

Economic Considerations

The constant increase in the number of languages to be checked and the diversification of the documents to be processed have resulted in a substantial rise in the workload and the cost of validation. A new approach to validation was sought in an effort to curb costs. On the one hand the reorganization of the production cycle ensures consistency between the paper and electronic versions of the publication. On the other hand the use by the validation mechanism of the symmetry that exists among the various language versions of the same publication makes it possible to highlight the independent transformations in all language versions.

Managing the Complexity

OPOCE publications are SGML instances which comply with the Formex standard. Since 1985 the DTDs which make up this format have undergone three major revisions and eight minor revisions. Following the specialization of the Formex V3 DTDs, they have increased in number from six to thirty-five. The major Formex V3 revision has been entering into production since the beginning of 1999. The last DTDs making up this version are currently (spring 1999) in the final consolidation phase. The final version, on the basis of which the complete life cycle of the publication will be undertaken in Formex V3, is Formex V3.0.1 or Formex V3.0.2, depending on the DTD. Production started in April 1999.
After the definition of the new version of Formex, the publication certification process evolved. The certification process which merely validated SGML conformance of a publication has now been supplemented by a series of rules for monitoring the way in which Formex V3 is used when each publication is written.

In order to include all the character sets likely to be used in a publication, the following mechanisms have been specified and rules for use have been drawn up:

- the Formex V3 character set is based on:
  - ISO 2022:1986 set substitution
  - ISO/IEC 6429:1992 control functions
  - ISO/IEC 6437:1994 Latin and Latin supplementary sets
  - ISO 8859-5:1988 Cyrillic set
  - ISO 8857-5:1987 Greek set

- Formex V3 character entity
  - euro character
  - unmapped ISO character

- ISO character entity
  - ISO 8879:1986
    - Added Latin 1
    - Added Latin 2
    - Box and Line Drawing
    - Diacritical Marks
    - Greek Letters
    - Monotoniko Greek
    - Non-Russian Cyrillic
    - Numeric and Special Graphic
    - Publishing
    - Russian Cyrillic
There are three techniques which may be used to manage the multilingual aspect of the publications:

- the reference language is associated with monolingual publications;
- several reference languages are attached to each multilingual document;
- in addition, any language may be overwritten as regards the text or a table.

The various language versions are consolidated by highlighting the multilingual structures and content within the publications. This multilingual version of the publication can be used to manipulate the various language versions of a publication simultaneously.
General Architecture

The quest for an integrated multilingual solution leads to the re-examination of the certification and the storage process within the workshop.

In this section the building of a general architecture is described, and the conceptual approaches that are required to support the multilingual aspect. The problem is considered from different angles. Certification, storage, consolidation, and modification of multilingual publications are discussed, with particular reference to the provision of an efficient and integrated multilingual system.

The general architecture of a multilingual workshop comprises four modules.

Reception receives a publication, inserts it into synoptic (aligned) storage, and concatenates certification.

The certification process has been subdivided into an automatic validation phase and a manual verification phase. In the automatic validation phase all the objective checks are carried out which do not require any human intervention. The manual verification phase identifies situations which are likely to contain an error as regards compliance with the rules for use, and directs the manual validation process, justifying the issuing of a warning.

During the automatic validation process the following basic checks are carried out:

- compliance with the physical specification;
- SGML conformance;
- SGML consistency and completeness.

Manual verification comprises the following basic checks:

- use of the physical specification;
- that the DTD rules are obeyed;
- for the synoptic nature of structure and content.
Manual editing is carried out by means of an SGML editor, the publication being edited in accordance with its DTD. The results of the automatic validation and the manual verification are presented in the form of a specialized interface. Support functions are integrated to locate errors, bear out diagnoses, and facilitate repetitive corrections.

The relevant synoptic status is attached to each element in the instance. When the element exists in all the languages, it is multilingual. Any modification of a multilingual element is automatically repeated in all the language versions. Any modification in a monolingual element affects only its native language. An element may be defined as being monolingual in order to limit the dissemination of the modification.

As soon as the certification process is complete, the document is sent for final archiving and an electronic publication is produced.

**Advantages of Using SGML**

In making the choice of a standard for document representation, there are many considerations. What possibilities does SGML have to offer in the context of the provision of an efficient and integrated multilingual system? How can they be turned to advantage? How can they be applied in this new approach? Among the many advantages that can be discussed, language-independent tagging will be considered as a leverage in the proposed solution of this particular multilingual problem.

Other criteria include:

- independent development of market tools (storage/archiving/dissemination);
- continuity of information;
- continuity of production systems;
• independence of external applications;
• independence of sources;
• independence of formats;
• homogeneous nature of methods;
• standardization of data access methods;
• flexibility;
• extensibility;
• modularity;
• strict specification of interfaces.

Conceptual Description

The case study is of the new SGML validation workshop developed for the publication of laws in the Official Journal of the European Union. There are currently eleven official languages within the European Union. Publications considered by the workshop are therefore multilingual.

Flows within the workshop are shown in the diagram below.
During automatic validation, the following controls are chained:

- **Automatic validation**
  - Physical specification check
  - SGML conformity check
  - Completeness and consistency check
  - Validation report and status

The semi-automatic validation is subdivided into processes, the sequence in which they are carried out being illustrated below.
The exploitation of SGML

AIP = Arcel In the Pocket (HTML interface to browse and obtain SGML publication from Storage 2 and 3)

CREJO = Bibliographic database

The semi-automatic validation process consists of immediate processing, which must be carried out within a twenty-four hour time frame, and delayed processing that depends on the priorities at that time. Each process follows the same diagram. The end-user edits a publication with an SGML editor. The consolidation of the linguistic versions is carried out automatically, based on the configuration and the range that is associated with the modification at the time of editing. This consolidation is done in a transparent and asynchronous manner at the time of synoptic storing. Moreover, immediate export is carried out at the end of the immediate and semi-automatic validation process. When the semi-automatic validation process is complete, the final archiving is carried out.
Conclusions

The addition of several more language versions of official documents to be published by OPOCE led to reappraisal of the method then employed. SGML was always the key technology used so the solution was to devise the SGML validation workshop, thus capitalizing on early work and experience gained.

Already there are plans for enhancement with respect to possible integration with full-text search engines. That is in the future. Today the signs are that a more efficient and effective way has been developed to address the problem, one which has the potential to cope with many more languages in the years to come.

From the end-user's point of view, how will this new approach affect his work? What modifications will be required in his organization? These questions are still to be addressed. The answers will come only after the SGML workshop has been operational for several months.

Please e-mail your comments to Pierre Colot at pco@sgmltech.com.

This paper was first published in the Conference Proceedings of XML Europe‘99, April 1999, pp 437-44.

© The SGML Technologies Group 1999
Using Management Information Within an Editorial Context

Author

Jean-Paul Daisomont

Biographical Note

Jean-Paul Daisomont has been working with SGML for over nine years. He gained experience in applications for management information, notably for the Courts of Appeal, and also for Canal+ (pay TV). More recently he has been closely involved with the large-scale project to produce the budget of the European Union, as well as another EU project to check printers' output prior to its being archived. A software engineer, he has a master's degree in informatics from the Faculté Notre Dame de la Paix in Namur, and also a degree in economics. He may be contacted at jpdaisomont@sgmltech.com.

Introduction

The majority of the information handled in everyday life is textual. Therefore in the future management tools developed and used will probably put the emphasis on editorial aspects.

It is our conviction that management information and text information will be more integrated in future tools and that the quality of this will directly define the quality of the tools. Let us refer to this integration as cohabitation.

One of the problems is to know which shape this cohabitation will take. This problem may occur on two occasions:

- during information storage;
- during information handling.

One of the features of an implementation is to match the format used for storing information with that used for handling it. Unfortunately, this is not usually possible. Nevertheless it constitutes a target to be reached. In the context of this article, we shall tackle the problem only from the modelling viewpoint.

To analyse cohabitation, we shall describe the characteristics of both management information and text information. We shall then define the circumstances in which the two kinds of information optimally cohabit and the targets that we hope to meet. Finally, we shall present some examples of cohabitation.
Management Information and Text Information

We are in the habit of classifying applications according to the kind of information that they handle. Consequently, we find management applications to manipulate management information and editorial applications to handle text information. Of course, there are other types of applications, but we shall restrict comment only to the interactions between management applications and editorial applications.

Within a management system, the essential characteristic of the information handled is that it is very structured and very rigid. To define its structure:

- carefully list the objects that the information system has to cover;
- for each object, specify the attributes that may be appropriate to associate with the object;
- in order to identify one occurrence of an object, use a reference that has no significance for most of the time;
- finally, preserve the relevant relations between the objects.

Concurrently, we define access methods to the data. Control over these information systems was acquired a long time ago. There are few domains of human activity that these kinds of information systems have not covered.

In the case of editorial applications, the nature of the information is quite different. Where the concept of sequence plays a subordinate role in the case of management information, for text information it acquires immense importance. For management information the links between the objects are identified at the conception of the system, whereas for the text information different forms of relations can be established at any given moment.

The tools available to treat the problems linked to information handling are listed in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Management Information</th>
<th>Text Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Research mechanisms</strong></td>
<td>SQL</td>
<td>No standard available</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>Relational model</td>
<td>SGML</td>
</tr>
<tr>
<td><strong>Access tools</strong></td>
<td>ODBC</td>
<td>No standard available</td>
</tr>
</tbody>
</table>
Generally speaking, management information is information that talks about something. In this context a marker showing a modification to a text is management information, even if it appears inside the text. In fact this information is a new form of information. It does not correspond to traditional management information, neither is it text information because it does not belong to the text of the document. This information will be called text management information.

**Context**

The analysis we present is particularly relevant to situations in which the following elements are present.

1. The target of the system is to create a document or a set of documents. In the latter case, relations between these documents are of importance. The creation of this document (or of these documents) requires the participation of different operators who work in a stand-alone manner. Few means are available that allow them to communicate. This situation can occur when the operators do not know each other.

2. Even if the work is done in a stand-alone manner, the information contained in the document presents a high degree of homogeneity. As a result of this homogeneity, the operators at the end of the workflow have a real need to know which specific choices were made upstream in the production cycle.

Behind these organizational aspects, we are assuming that the information handled has an SGML representation. The handling can be the result of tools or of operators.

**Integration Requirements**

- The text management information has to be located within the text information, where it can be handled efficiently. Therefore, it must be present in the text in the place where management information is most significant.

- To access the text management information and to understand it, it is unnecessary to have resources other than the handled document. This means that all the information needed to understand the message of the management information should be available in the document.

- In order to analyse the text management information, we can use object-oriented methods. This means that we can associate information attributes, behaviour, and methods with text management. Among these methods, those relating to access methods are of prime importance.
Examples of Text Management Information

We shall present two kinds of text management information that can be found in a document. In these two examples, the location of information is important. In one case we shall try to record the work of a proof-reader, and in the other we shall analyse the presence of information produced automatically by a tool.

A Proof-Reader's Work

Validation of electronic (SGML) documents can sometimes happen in a context in which it is necessary, from an organizational point of view, to keep track of all the interventions by the proof-reader.

For different reasons, it could be necessary to store:

- time (1) and place (2) of handling;
- the proof-reader's identity (3);
- erroneous text (4);
- corrected text (5);
- seriousness of the problem (6);
- justification and error status (7).

In this case, the error and its correction make up one management object. This object has two textual contents. We shall analyse how to keep track of this information in an SGML context.

One possible implementation is to use marked sections, ie IGNORE or INCLUDE. According to the value of a parameter entity, we can include or ignore either the erroneous text part or the corrected text part. This solution allows the cohabitation of information (4) and (5). It also allows us to keep track of embedded mistakes and corrections. However, this solution does not allow us to keep management attributes such as (1), (2), (3), (6), and (7), no more than it made both (4) and (5) available at the same time.

Another implementation consists in using a concurrent DTD, specifically used to indicate the start and the end of the correction, ie the scope of the correction. In this case the erroneous text (4) can be stored inside an ENTITY attribute. The other management information, (1), (2), (3), (6), and (7), can be kept as attributes, whereas the correction (5) is placed between the error's start and end-tags. These elements are defined as EMPTY, and ID and IDREF attributes are used to establish the connection.

A third implementation consists of storing the error as a separate SGML instance and keeping the link to the error within the corrected SGML instance. This separate document should conform to a specific DTD and use another concrete syntax to redefine the STAGO, TAGC, and ETAGO delimiters.
As can be seen, different implementations are available in SGML. Others can be envisaged.

The choice among these implementations depends upon:

- limits imposed by the tools (parsers or editors) available or used;
- the way the information is to be handled;
- the willingness to impose one solution over another.

As far as we are concerned, we think that a concurrent DTD is very beneficial insofar as it offers:

- a more supple treatment;
- more autonomy for information representation.

Insertion of Tool Inferred Information

We shall tackle two aspects: the one of anomaly and the other of document property.

Anomaly

In most editorial systems the use of SGML is not limited to editorial or repository tools alone. In between these two, we find a whole range of tools that either validate or transform SGML instances. In general these tools are able to detect and locate anomalies. To correct them we need to edit the document again. In this case, the availability of a clear description of the anomaly within the document is of great help.

Property

When starting up an editorial system, we have the possibility of defining properties that will facilitate management and follow-up of production. The stage in which document types are defined is the right time to introduce these properties. Establishing rules for use also allows the introduction of new property types. There are also properties that a tool is able to compute.

Among these, it is desirable for users to know whether a part of the document is new, has been modified, or deleted. The reader can easily imagine other properties that an automatic tool can compute.

Representation of This Information

In essence, there is no fundamental difference between a correction, an anomaly, and a property; therefore, the correction mechanisms described above remain valid.
Document's Life Cycle

Another example of management information that describes a document and not its contents is a document's life cycle. The life cycle of a document encompasses a set of events that make the document evolve. In order to describe a step in the life of the document, we generally gather the following information:

- the related event - either a request or an action;
- the author of the event - either a tool or a user;
- the result associated with the execution of the event - in general the result is *initialized, in progress, successful, failed*, etc;
- the explanation of the obtained result - this explanation can be produced by a tool if it is a tool that handles the event, or by a user if it is an operator who transmits the event.

The goal is to display all the transformations that a document has undergone. The sole availability of this information can be helpful to the user in his work. The appearance of information resembles database information very closely. Chronology and changes to state are the most pertinent. The availability of this information is of real value to stand-alone users. It allows operators to be more autonomous and better informed in their decision making.

Its localization, within the document or outside, is of little importance in the end. In this case, the information can be stored within a concurrent document, within a subdocument, or within another document.

Conclusions

The intention of this paper was to point out that much information related to the document can be stored efficiently within the document itself. In this context, SGML offers all the mechanisms needed for its representation.

The four fundamental questions are as follows:

1. Does the information to be represented have a scope within the document, ie does it begin at a given position in the document and does it end at another?
2. Does the information have a recursive structure?
3. Is the knowledge of the position of the information important?
4. Does the information have a related text content?

The most complex situation to represent is text information with a recursive structure, which has a scope, a related text content, and whose location is of prime importance. This situation occurs when an operator who corrects a document must keep track of his work.
To represent this situation we propose the use of a concurrent DTD that uses two EMPTY tags, one to mark the start of the area, another to mark its end; these two tags are related with ID/IDREF attributes. The scope is the document text between the two tags. The related text information can be stored with ENTITY attributes.

Please e-mail your comments to Jean-Paul Daisomont at jpdaisomont@sgmltech.com.

This paper was first published in *interCHANGE*, the Newsletter of the International SGML Users' Group, volume 4 issue 1, January 1998, pp 24-26.

© The SGML Technologies Group 1998
SGML as the *Lingua Franca* in a Multilingual Environment

Author

Mladen Damjanov

Biographical Note

Mladen Damjanov studied industrial engineering, specializing in informatics, at l'Ecole Centrale des Arts et Métiers in Brussels. He started his professional career with Bull, later joining the Sema Group, Belgium, where he worked on SGML projects. He is now with the SGML Technologies Group, being deputy manager at ISEA in Luxembourg. He may be contacted at mda@sgmltech.com.

SGML in a Multilingual Environment

The need for multilingualism is a direct consequence of market opportunities in world trade. Indeed, when information of any type is provided in a country, there is usually a legal requirement that it be in the language of that country.

This is especially true for consumer goods provided for the mass market, for which the documentation and associated information form a key component of the product itself. As in the case of any component, there has to be a clear approach both for the production and maintenance of the multilingual information. When the product itself is information, as in the case of the Internet, multilingualism will become essential.

Historically, multilingualism has been considered an ancillary problem, and has never been solved in an effective way. Rather, it has been considered as a problem to be circumvented, one rarely, if ever, totally surmountable.

Basically, two sets of costs can be applied in achieving multilingualism in addition to the cost of the original version (in what could be described as the master language):

- supplementary costs taking into account the different linguistic versions to be prepared and distributed;
- further costs for the maintenance of the various linguistic versions.

Different approaches to achieve the desired results depend on whether the information is for translation purposes or those of document management. With the Internet, there is an ever-
increasing number of users from many different countries; there is a need here for users to navigate by use of the browser in their own language. This must be considered from two points of view:

- the representation and exchange of information in the linguistic version in which it has been generated;

- the management of information in a given linguistic version or through multiple linguistic versions.

For the representation and actual exchange of the information increasing use is made of the ISO 10646 character sets and Unicode, where the latter can be considered as a subset of the former for the more popular languages. The Unicode character coding system is designed to support the interchange, processing, and display of the written texts in many diverse languages in the world.

Unlike the ASCII character set, the Unicode standard is based upon a character set that includes over 39,000 combinations. In order to distinguish particular characters it is no longer necessary to use escape sequences or control codes.

It should be noted, however, that the ASCII value of a given character will be the same in the Unicode standard.

Example:

<table>
<thead>
<tr>
<th>ASCII/8859-1</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>: A equals 0100 0001</td>
<td>: A equals 0000 0000 0100 0001</td>
</tr>
</tbody>
</table>

Nevertheless, in addition to the ISO 10646-UCS-2 and ISO 10646-UCS-4 encoding schemes, others, such as UTF-8, introduce the problem of escapes sequences in a different manner.

However, where the management of the information is concerned, Unicode (or ISO 10464) does not provide the solution. What is needed are operations for the management of the information, with focus on the fine processing of the structure of the information (documents), and the exploitation of the semantics. With manual procedures, even perfectly aligned, it is impossible to manage the documents in several linguistic versions in a consistent way. For example, there can be no guarantee of synoptism or alignment (common page numbering, etc) when using manual methods, especially when most documents are of high-volume.

Although some tools exist for the management of documents (in machine-readable form), they do not offer the flexibility to identify parts of information or the location of a part of a document independently of the linguistic version considered. This is where the use of SGML becomes so attractive.

SGML allows for the validation of the structure of the document in accordance with its DTD, and by the principle of tagging and the use of SHORTREF, precise positioning within the document is possible, and even extraction of some of the information if desired, independently of the linguistic version considered. It is also possible to allow a user to consult
part of a document in a given linguistic version and obtain the same part of another linguistic version by managing links between the different linguistic versions of the same document.

When this problem is considered in the context of the Internet, the limitations are apparent. But extend HTML (an application of SGML) with functionality provided in XML (a subset of SGML), its usefulness is apparent. And XML can be considered as the next logical step towards full SGML.

Indeed, the sole purpose of HTML is to allow for easy representation of the information on screen. The structure of the information is deprived of all contextual notion which renders it particularly poor. Whatever the richness of the initial tagging of the information, its conversion to an HTML representation results in a loss of capacity to exploit it correctly afterwards.

The need to distribute the information available on the Web in various languages is an obvious requirement. The emergence of tools for automatic translation of information available on the Web (for example SYSTRAN) bears testimony to this fact. Translation of the information is a complex process; it does not merely suffice to translate the words, but is necessary to interpret them according to the context in which they are placed, thus giving them their full significance and weight in a translation to obtain a consistent result. Anyone who has performed translation tests using these tools will be able to testify to the doubtful results.

The level of SGML tagging of a document can express the granularity required, depending on the needs of the user. The semantics need to be added in the names of the tags to increase the possibilities of exploitation of the information. So now there can be added benefits to managing the information provided in multilingual form.

At the application level, the implementation of the SGML standard with all its features, in a multilingual environment, has given rise to the development of software tools that permit the desired functionality/flexibility:

- to handle the validation and correction of documents;
- to apply any modifications automatically to the corresponding parts of other linguistic versions of the same document.

Indeed, it is not merely the use of SGML, but also the exploitation of the standard, plus the precise specification of the grammar of documents, that have made it possible to achieve such functionality readily; hence processing multilingual documents no longer poses a problem - rather this may be regarded as an extension to the original information, resulting in added value.

Tools are becoming available for processing multilingual documents, integrating some of the functionality made available through SGML (for example XCORPUS offers a software environment for the manipulation of a textual corpus represented in accordance with SGML). And more are coming into the marketplace.
These comments and observations are not pure theory. The implementation of these principles and of the SGML standard today allows us to redevelop multilingual applications within production environments. For example, they have been applied to the production of the budget of the European Communities comprising a volume of some 2,000 pages per linguistic version. It is produced and maintained in the eleven official languages of the European Union.

Through the exploitation of the SGML standard together with appropriate application software, it is possible to remove the barriers created by the needs for multilingualism in a cost-effective way. It could even be said that SGML is the vehicle for this; that SGML is a vehicular language.

Please e-mail your comments to Mladen Damjanov at mda@sgmltech.com.

This paper was first published in *interCHANGE*, the Newsletter of the International SGML Users' Group, volume 4 issue 2, April 1998, pp 13-14.

© The SGML Technologies Group 1998
SGML Boosts Standard Document Processing for a Major European Bank

Author

Philippe Fontaine

Keywords

- Case Study
- Professional Knowledge
- Banking Organizations
- SGML
- SGML Filters
- Document Repository
- Authoring

Abstract

Large banking organizations have always been faced with the problem of distributing information that constitutes professional knowledge to the right people, at the right time, and in the right place. With the ever-increasing arrival of new technologies - information highways - this is a hot topic again. Internet and intranet-based solutions can bring new aspects into the communication of information.

Over and above the scope of distributing information, structured documents that contain professional knowledge have to be produced, validated, stored, maintained, and eventually distributed in a traditional environment that is being turned upside down. This paper shows how an SGML-based solution successfully transforms the dream of managing core business information into reality.

Biographical Note

Philippe Fontaine is a project manager at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. For over ten years he has been active as a software engineer and systems architect specializing in object-oriented distributed applications and complex document...
workflow systems. All these systems use SGML either as a document storage and exchange medium or as a formal message specification tool for communications between distributed application processes. He obtained a degree in electromechanical engineering of the Free University of Brussels. He may be contacted at phf@sgmltech.com.

**Introduction**

Banque Bruxelles Lambert (BBL), part of ING one of the largest banks in Europe, has decided to structure, organize, and distribute the standard documents that are its professional knowledge through a new set of document-based applications.

The underlying objectives are:

- to build a centralized structured base of professional knowledge;
- to make appropriate information available to authorized banking employees at the right time, in the right place, and in a non-ambiguous way;
- to ensure better quality and greater efficiency of the customer service by providing the employees with one-off assistance and continual training;
- to guarantee up-to-date delivery and timeliness of information;
- to reduce direct and indirect costs.

The need for timely information (especially text-based information), and of consistency, guaranteed through a well-defined and validated information structure, and the need for separation of data content and data presentation brought the bank to the choice of SGML (Standard Generalized Markup Language) as the pivotal means for building these knowledge management applications.

**Functionality**

Given the size and the complexity of the overall project, several phases were defined on the basis of the different types of documents handled and the distribution channels used.

The documents that are currently handled by the SGML document system are:

- all types of letters of credit addressed to bank clients (these are prepared by the end-user credit managers in accordance with the SGML definitions that specify the structure of letters of credit);
- all documents dealing with the contentious recovery of debts;
- all documents used in the framework of life insurance contracts and claims;
structured documents aimed at specifying and describing 'home-made' operational
data-mining tools (this information is mainly for use by internal application
developers and must be totally and utterly consistent with the current version of these
run-time tools, as well as with the distributed and documented interface
specifications).

The functionality that is currently available for a subset of the professional knowledge is
provided by:

- the central repository which gathers all SGML document instances as well as the
  associated external entities and user-defined application indexes;

- built-in mechanisms in the central repository to implement user-defined versioning at
  a user-defined granularity (a differentiation tool can extract the differences between
  versions of the same document instance);

- the authoring stations which allow for the production of SGML instances in
  accordance with the SGML DTD (Document Type Definition) (these stations are built
  by customizing the bank's standard text processor);

- 'read-only' decentralized SGML repositories which are populated and updated from
  the central repository by means of SGML-based import/export mechanisms;

- different levels of interface, such as document instance retrieval, document
  composition, data entry, document viewing, and printing, are available to application
  developers.

Architecture

This section describes the general architecture that was put into operation at BBL. Given the
overall context of knowledge management, modularity and evolution are the key factors of
this architecture.

From a conceptual and technological point of view, SGML brings a solid base to the entire
current and future application.

In the current architecture, authoring stations enable administrators to create SGML instances
according to specific SGML DTDs and to store them in the SGML central repository. From
there, import/export mechanisms allow for the extraction of document instance updates, and
for the distribution of these updates to the decentralized read-only SGML repositories by
means of SGML messages.

End-user applications are given access to the decentralized repositories for the retrieval and
processing of document instances (e.g., compose a final letter with extracts from different
SGML instances and add application data to the final instance).
SGML filters allow for the conversion of SGML instances into RTF (Rich Text Format) documents for final rendering or authoring (see below) and vice versa for the import of existing RTF documents or newly-authored documents into the SGML repository.

Central Repository

The core of the whole system is the SGML-based central repository. It allows for the structuring and storing of information in a vendor-independent and presentation-independent way.

The DTDs that specify the document structures provide the following functionality:

- insert, delete, and modify (with locking system) a DTD instance;
- retrieve and extract a DTD instance according to the user-defined contextual indexes;
- document instance versioning at a configurable level of element granularity;
- differentiate between two versions of the same document instance from a configurable level of element granularity and extract the differences in an SGML-structured format (this is possible owing to the design of the distribution mechanism).

This central SGML repository is implemented on top of a classic RDBMS (Relational Database Management System) and can be completed with a workflow engine that automates the process of an application request, according to a state diagram.

Authoring Subsystem

Given the choice of text processor by BBL as an overall standard, and given the strategy in terms of workstations allocated to employees, the bank decided to develop authoring stations by customizing its standard text processor.

This choice, motivated by standards-related, strategic, and cost criteria, leads to the need:

- to develop and operate a filter for the conversion of an SGML instance into a text processor processable format (e.g., RTF) to update a document instance;
- to customize the text processor in order to guide the author in his production job so that he may not make errors governed by the underlying DTD (these mechanisms are based on the association of pre-defined styles with DTD elements, the disabling of useless features of the text processor and the addition of functionality such as a contextual style toolbar, and a call to an external DTD parser for validation);
- to develop and operate a filter that converts the RTF document, for example, back into the corresponding SGML instance before a controlled update in the SGML repository.
Although customized text processor authoring stations are much more profitable for large-scale dissemination in such an environment, there is still a need for a native SGML editor for SGML document recovery or for one-shot specific editing.

**Distribution Subsystem**

In the current system, the distribution repositories are (and will remain) installed in the different subsidiaries and departments of the bank as SGML read-only repositories.

These repositories are populated by SGML instances (complete ones or updates) extracted from the central repository as described above. The built-in differentiation tool allows for the formatting of the updates of the DTD instances in an SGML message so that these differences can be replayed in the decentralized repository and thus update the distributed instances.

The distribution process itself is activated either by the central system administrator or through a request message coming from the decentralized site.

These distribution repositories are implemented with dBase files for financial reasons, as well as reasons of performance and conformance with banking standards.

In future, other distribution channels will be developed such as Web-based channels (HTML (HyperText Markup Language), XML (eXtensible Markup Language), intranet and extranet), online help channels (for the data-mining tool specification documents), and printing channels (paper delivery). These developments are already planned.

**Applications**

Apart from the usual application functionality such as search and retrieve, and view and print, the applications made available to the end-users can be operated using two different interfaces.

In both cases, filters to convert SGML documents to the corresponding RTF format are exploited. These allow for the display of the document by the bank’s standard text processing software available on the end-user’s workstation.

**A Menu-Driven Interface**

Menus presented to the user are actually an interactive and guided way of walking through the SGML document structure. Through these menus, the user is able to compose his document by integrating into the final document instance other fragments of documents extracted from the distribution repository. The menus guarantee that the final document is in accordance with the full DTD. The menu interface also enables the user to enter further data to complete variables of the document instance.
A Document-Based Interface

Through this interface, the application addresses a complete document instance directly and is no longer able to compose a document. The document instance is retrieved from the distribution repository according to search criteria and is either displayed on the user's workstation or processed to result in a final format in order to be printed or mailed, for example.

Conclusions

In a banking environment of this kind, full of legacy applications and procedures and with a multitude of users accustomed to their traditional working environment, it is a challenge to bring a completely new approach and new techniques to the end-user's workstation.

Up to now it has been a success, not only from a managerial and a developer's point of view but also - and this is a crucial factor - from the user's point of view.

Through this project, SGML is progressively improving document processes in one of the biggest banking institutions in Belgium. This is being achieved by:

- a phased approach and implementation in accordance with the bank's priorities (this approach enabled the solution to be validated progressively against user's expectations);
- the choice of reusable modules instead of complete but limited products or packages (this enabled smooth integration within the banking environment and enabled planning for further evolution in a step-by-step approach).

SGML proved to be the ideal standard and provided the methodology for structuring and protecting information as important, as diversified, and as complex as the professional knowledge of the bank in a distributed environment.

Please e-mail your comments to Philippe Fontaine at phf@sgmltech.com.

This paper was first published in the Conference Proceedings of Markup Technologies '98 US, November 1998, pp 161-64.

© The SGML Technologies Group 1998
The Use of SGML for a Police Information System

Author

Jacques Kasprzak

Keywords

- Police
- Prototyping
- Encoding Language
- Generic Tool
- Browser-Based Application
- Client/Server

Abstract

In this paper the objectives of the INGEPOL project are explained.

The architecture of the system is then described, this being governed by a number of constraints. Associated problems are highlighted. The role of SGML (Standard Generalized Markup Language) and advantages offered by such an approach are presented within the framework of complex projects.

Biographical Note

Jacques Kasprzak is a software engineer employed by ISEA in Luxembourg, a member of the SGML Technologies Group. He specializes in the design of information systems. A graduate of computer science, he worked in the Institut de l'Information Scientifique et Technique (INIST), part of the Centre National de Recherche Scientifique (France), to administer the information system and databases. He may be contacted at jka@isea.lu.
Introduction

Objectives

INGEPOL is a project being carried out for the Gendarmerie Luxembourgeoise.

The aim of the project is to put into place an information system automating the creation, administration, and exploitation of the centralized database. This is for the Gendarmerie (the state police force) and the local police to serve their needs regarding the prevention of crime, searches, investigations carried out, and the prosecution of offenders.

The principal objectives of the INGEPOL project are the following:

- **to increase the availability of police officers in the field by reducing the amount of administrative work.**

  From information that is input in a standardized manner by the squads and the police stations, the system produces the documents required for case management and ensures their distribution. As far as the local units are concerned, police officers will be relieved of work involving statistics and also combination of data (which is performed automatically at the central level);

- **to increase the effectiveness of controls of people and objects by accessing the desired information in a transparent and fast manner.**

  The central information system permits all police officers of the FO\(^1\) to access police information in accordance with their profile (on the basis of simple requests);

- **to create optimum conditions for investigation work and increase the chances of solving criminal cases.**

  The information system provides the FO with automated methods of research and comparison of information that allow the speeding up and facilitation of a judicial enquiry;

- **to make crime prevention more effective.**

  The availability of statistical data periodically and geographical exploitation of criminal information permit an appreciation of variations in delinquency and, as a result, can draw the attention of the FO to the increase in certain types of crime or criminal activities.

So there is immediate sharing of information within local units, and centralization of that local information, integrating information stemming from other systems at the central site. This improves effectiveness of police work in the field, where there is the possibility of querying a complex system of data compiled from many sources in a manner that is transparent to the end-user.
Functions of the INGEPOL System

The INGEPOL system has several functions, of which the management of police information could be considered the most fundamental. This includes not only the management of cases but also managing descriptions of people and objects be they firearms, jewellery, driving licences, stolen vehicles, banknotes, drugs, or objets d'art, for example.

System exploitation is another important function, one aspect of which is the control of people and objects (operational control). There is also the consultation of people including interviews, and information on objects and of cases within the framework relating to a judicial enquiry.

Statistics are exploited and combinations of data play a part within the framework of crime prevention.

The information system has to be administered, which subsumes validation of the information input. Then there is correctness, with regard to consistency of data and respect of legal dispositions.

Distribution of the information and exchanges with distributed systems are important functions where the security of information must be taken into account, including confidentiality, availability, and integrity.

Finally there is the management of users and access rights to information.

The INGEPOL Project

The INGEPOL project consists of several phases, only one of which is the subject of this paper. The phase in question is concerned with the management of the descriptions of people and objects and their control at an operational level.

Global Architecture

The aim of the INGEPOL system as stated is to communicate with distant partner systems and to centralize and spread information to the different units (squad stations). This implies an architecture that hinges on the three-tier client/server notion allowing, on the one hand, better allocation of tasks that fall to each of the parties involved and, on the other hand, improvement of maintenance and even the integration of new components.

Making up the system are:

- the client workstations;
- a group of servers composed of specialized services whose purpose is to process management rules (security, validation, dispatching, audit, etc);
The Use of SGML for a Police Information System

- heterogeneous information systems which give rise to the various problems of accessing resources.

Communication between the various tiers is achieved through XML (eXtensible Markup Language) messages structured in three layers which are dedicated to:

- security, allowing encoding and deciphering of data transported on the system;
- the logic of the message, by specifying the server addressed, the family of messages, the expected processing method, the objects concerned, etc;
- the information transmitted, which may be reformatted according to the target systems.

An XML parser processes the message by breaking it down into atomic elements that are likely to be interpreted by a specialized system service.

Problems/Constraints

A global study of the project enabled identification of a set of characteristics and allowed the adoption of an SGML-oriented development strategy.

In order to appreciate the SGML approach to the project, technical considerations had to be taken into account in addition to the initial constraints:

- the system must be operational on various target systems
  - PC - Windows NT,
  - VT100 - DPX2,
  - mobile terminals on-board service vehicles (Motorola);
- the number of screens that make up the man/machine interface is important (250);
- the number of messages to be taken into account is important (800);
- the structures of the external databases that are part of the system are different;
- the requests issuing from the end-user can generate additional processing actions which are specific to the target systems.

Under those circumstances it was difficult to move towards target-specific developments and the project required a considerable amount of time to be spent on development, testing, and integration.
Why SGML?

Several considerations led to the adoption of an SGML strategy. These included the need to have an standard interface; standardization (as far as possible) of the process for the generation of services for the updating of a database; the actual generation of services for the updating of a database; and the description of the structure of the information systems in a simple and unique way.

Given the constraints previously listed as a starting point, a first report was produced. This highlighted many points:

- the structures of the different databases are known and conversion tables allow access to other systems from the central INGEPOL system;
- it is possible, owing to the design of the INGEPOL database, to generate a set of primitives that allow them to be checked, added to, and updated;
- messages are numerous, but in general they all contain database objects, the logical layer of the message indicating the nature of the transaction (creation or modification of a description), the processing expected, the object(s) concerned, and the reference base;
- the look-and-feel must be consistent and whatever the target system it is inconceivable to imagine a specific development for each of the targets, therefore the user interface must be described in a unique way;
- the screens are simple and typical for the objects handled;
- management rules are of the referential integrity control type;
- management of screen flows is classical, navigation being determined following a user-triggered event.

Added to this is the necessity for obtaining a uniform and unique interface, of integrating validation and navigation rules within the specification that describes the interface, and of standardizing as far as possible the processes for generation of services that update the INGEPOL database. These requirements led to the design of an SGML development strategy that splits the INGEPOL system into five sub-systems:

- generation of database services;
- specification of data/message groups;
- a server;
- specification of dialogues;
- a client terminal.
After these sub-systems have been presented, the advantages of this strategy within the INGEPOL project will be discussed further.

**The INGEPOL Sub-systems**

**Generation of Database Services**

**Objective**

The objective here is to generate automatically a set of PRO*C and C++ services that allows the server to manipulate the database entities by limiting the knowledge of the structure of the database to the methods and the classes generated. The result of this generation is the creation of all the data groups that occur in the messages.

**Principle**

Modelling tools allow the generation of an SQL database creation file from a physical model. With this file, and other specific files that describe the screens to be used, it is possible to generate a set of services that allow interaction with the database.

The principle is as follows and requires two phases. Firstly, a DTD that allows the validation of the structure of the SQL file and the LKD that integrates an extended application language, generates an instance for each identified table or screen, and describes, in SGML, the object processed. The grammar associated with the instance describes the data group to be taken into account, and indicates whether this group is optional, mandatory, or whether it occurs at all. It also describes for each of the groups the fields that are associated with it, their type and data format as well as a formal description. In short, each table becomes a data group whose name is the name of the table, and each item of the group is one of the columns of the table. The conversion of the different types of data is carried out at that level so as to allow a programming language-oriented interpretation.

Secondly, from each instance previously generated, and based upon skeleton files that describe each of the actions to be taken on the database, a specifically developed parser generates a set of methods (PRO*C, C++) allowing actions such as SELECT, INSERT, UPDATE, and DELETE as well as the C++ classes (*.h and *.cpp) associated with the objects processed.

The generation of data structures and of associated primitives relieves the developer of the tedious task of writing database access routines.
Specification of the Data/Message Groups

Objective

From the data groups used to build the messages that are exchanged between the client terminals and the servers, the objective is to generate automatically C++ and Python structures that can be used directly.

Definition

A message specification is an instance that contains the complete description of all the data groups that can be processed in the messages exchanged between the client and the server. The specification primarily consists of completing the message instances generated previously, by adapting the field descriptions to the client terminal, and by integrating portions of validation codes.

Initial work consisted of defining message instances that reflect the different structures of the databases concerned. This definition allowed the generation of a set of services dedicated to actions such as inserting, selecting, deleting, and updating the databases while respecting the validation rules inserted in the instance. Those same instances are used to define screen layouts by allowing database fields to be mapped to the screen layout.

There are different categories of messages:

- **interrogation of request** - these messages contain sets of search criteria that allow the selection of objects in the various databases that make up the system;

- **occurrence response** - these messages contain elements that allow the indication of the number of occurrences in the database(s) of the objects selected;

- **list response** - these messages contain the characteristic elements of the objects selected, list responses allowing the selection of a precise object and its associated components (for example description, or description documents);

- **response given** - these messages contain all the elements of the object selected, its description, and its components;

- **data** - these messages contain all the elements of the object to be integrated into the database, its description, and its components.

An example of a message containing a complete object and its description could be:

- identification group of the message;

- object group (for example firearm);

- description group
  - 0 or \( n \) insurance groups,
- 0 or \( n \) image groups,

- \( ...; \);

- 0 or \( n \) groups relating to the descriptions;

- for each description group, a group of description documents.

All the messages carried across between a client terminal and the server are structured in XML. The message therefore consists of a chain of previously defined groups, whose elements are the name of the group and whose attributes are the names of the fields of the group. A group of messages can obviously contain other groups. The message is generated either at the client level when sending a request, or at the server level when sending a reply.

The messages completely define the client/server communication protocol.

**Principle**

A DTD allows validation of the structure of the message instance. The LKD that integrates an extended application language generates two things for each data group:

- a Python code file that represents the corresponding data structure - this file will be used by the client terminal for operations related to the messages;

- the set of C++ classes and the header (.h) files to be integrated into the server on the basis of skeleton files that describe the syntax of the classes.

The introduction of a VALID tag at group level allows the integration of Python code that is in charge of the validation rules. These rules check the presence of data and the consistency not only of data in a group but also of data from one group in relation to another. It is possible, for instance, to check whether the fraud group was input by the user when he indicated a national fraud number at object group level. These controls occur on the client terminal when a message is generated. Other controls are described in the dialogue instance; more specifically they check whether the actions set in motion by the user are justified.

**Server**

**Objective**

It is the server's role to handle incoming messages, to process them appropriately by addressing the necessary service(s), and to send back a message containing the answer awaited by the sending client terminal.

**Implementation**

Implementation of the server is based on a transactional monitor, organized in servers. Each server is composed of a set of services. In the INGEPOL project there are several servers that
are dedicated to precise functions, for example a server that manages descriptions and objects, and a server that manages the nomenclatures used.

Because the INGEPOL server can communicate with other distant information systems, it is possible to integrate dedicated servers that consist of specific services in charge of establishing the interface\(^{12}\) between the INGEPOL system and the partner system. Adding another information system then consists of creating a corresponding server.

**Principle**

In short, the server must provide the following functionality:

- receive the message and identify the transmitter;
- decipher the message;
- dispatch the message to the specialized service that is in charge of carrying out the necessary operations according to the process requested (read or create a new object description, etc), and of the information system(s) chosen;
- consolidate the return message with information received from the information system(s);
- encode the message;
- dispatch the message to the sending client terminal.

At the server level, the XML message is regarded as a chain of characters that a specialized parser splits into a data group, thus allowing the supply of the C++ structures derived from the classes previously generated. A set of methods, generated or developed, allows the appropriate server(s) to be activated, depending on the nature of the message and the data groups present.

**Specification of the Dialogue**

**Objective**

The objective of the dialogue specification is the generation of a set of files that represent the user interface and that can be interpreted by the client terminal of the target system (Windows NT stations or VT100 screens), where this is dependent on the different targets identified.

**Definition**

The description in this paper is limited to the generation principle for the Windows terminals, and a brief comparison is made with what can be generated on a VT100 workstation. It is obvious that the specification of dialogues\(^{13}\) implemented for the project is unique whatever the target systems. Nonetheless, all the modules required for generating the user interface are specific according to the desired target.
A dialogue instance allows the management of screen flows, dynamic management of drop-down menus, the setting of event-driven actions, and inter-area integrity controls.

Such an instance is characterized by a grammar that allows for a set of dialogues composed of one or more dialogue boxes.

- A dialogue box can be either a list or a simple screen. Each dialogue box is associated with the name of a message data group that allows mapping of the message data on the HTML page or the VT100 screen.
- A dialogue box consists of a set of data or a data group. The type and format of the field are deduced from the corresponding group of message data.
- Each dialogue box can be associated with buttons or drop-down menus that are dedicated to event management.
- Specific tags allow triggering of precise actions when an HTML page or a VT100 screen is loaded.
- Specific tags allow dynamic configuration of the screen depending on the context.

All these functions are possible by integrating Python code sequences into the dedicated tags.

As far as the buttons, drop-down menus, or specific tags are concerned, the associated Python code is simple and allows manipulation of data groups, navigation between the dialogue boxes, primary controls that could not be specified at message level, sending and receiving messages by means of elementary macro-instructions. The designer of the interface merely describes the management of screen flows and the events to be taken into account, without having to worry about the technical aspect of the client workstation modules.

This approach is similar to the definition of powerful configuration and encoding languages that are directly accessible to the experienced user rather than to an experienced programmer.

**Principle**

From message instances and dialogue instances generation-parsers allow, depending on the target aimed at (PC, VT100, Motorola), the implementation of standardized and stabilized structures in C/C++, Python, and HTML, that are likely to be interpreted by a unique engine that offers the same look-and-feel regardless of the target system.

A DTD allows validation of the structure of the dialogue instance. The LKD that integrates an extended application language generates an HTML template for each listed dialogue containing Javascript code and Java applets, as well as references to Python variables that will be substituted upon execution of the application that displays an entire HTML page. In addition, a Python file of variables is generated that allows display on a screen of the events to be driven as well as the associated code. Upon execution, depending on the screen that is displayed and the event (button or drop-down menu), the client workstation will drive the corresponding portion of the code and will execute the action required.
VT100 Generation

For workstations in character mode, the modules generating the user interface process the same dialogue instance, but adapt certain classical functions for a Windows workstation. Then the buttons or drop-down menus become function keys and the combo-boxes become screens that contain the list of all possible values. Scrolling of a screen page is represented by a second screen, etc.

The code inserted into the dialogue instance is executed with the help of a Python library.

For the VT100 or Windows NT stations, the problem therefore is to have a generic module that interprets the dialogue screen without worrying about the application itself or any future applications of the FO.

Client Workstation

Objective

The role of the client workstation is to make a stable, generic tool, that is capable of offering, on the basis of files generated from a formal specification of dialogues, a look-and-feel that is similar to that of a specifically developed client workstation.

Definition

For this paper only client workstations that were developed for the Windows NT stations are considered.

The client workstation is a system based on the Internet Explorer version 4 browser that allows display of HTML pages built from previously generated templates.

The client workstation uses a set of specific modules that are dedicated to the following functions:

- initialization of the application, connection to the server, and identification of the first screen to be displayed;
- mapping of the message fields with the associated HTML template in order to build an HTML page;
- display of HTML pages (Internet browser);
- code execution linked to the page related to a user-driven event;
- encoding and decoding of messages;
- sending and receiving messages between client and server.
**Principle**

Upon execution, the client workstation displays an initial screen that is listed in the initialization module. From this screen, the management of dialogue flows, controls, and events to be managed, in short the behaviour of the application, is described in the files resulting from processing the dialogue instance.

The structures generated in Python enable building, splitting the incoming or outgoing XML message, and execution of the mapping of the message group data on the HTML page.

**Advantages of the SGML Solution**

There are two principal advantages of the SGML solution for INGEPOL.

**Technical Environment**

At the server level, the use of SGML for validating message instances, generating a set of data groups that are characteristic of the database and a set of classes and methods for uniformity, has made standardization of the project easier. Moreover, it has reduced the risk of errors because the components generated were stable and has relieved the developer of much of his development work and from carrying out tests.

The generic modules that form the client workstation have allowed the interpretation of a dialogue instance that respects the grammars used.

**The User Interface**

The content-processing encapsulation principle which is possible with SGML has allowed, along with the object-oriented languages (C++, Python), a better correspondence between the needs of the end-user and the constraints of the producer, in particular through prototyping.

Within the context of INGEPOL, the prototype, based on the principles mentioned, has shown the strengths and weaknesses of the system as originally conceived. Indeed, as soon as the technical environment that allows the generation of components is stable, in addition to the further development/evolution of the server, the project essentially consists of the description of the user interface in more simple terms. Prototyping is the next process where the system is described in relation to a pilot object, validation occurring at regular intervals by processing concrete cases. Major modifications taking place at the user-interface level mean that the impact of a modification requested by the client is limited to modifying dialogue instances and generating templates. Once the pilot system has been validated, it is easy to generalize on all the objects and to obtain an end-product. A considerable amount of time can be saved with complex client/server applications, and on multi-platforms that use a considerable number of screens and messages.
By facilitating a dialogue between the client and the supplier, the SGML strategy adopted contributes towards delivering, at the end of the implementation, a product which conforms to the expectations and the needs.

**Conclusions**

Through allowing a unique definition of the user interface in spite of the different platforms targeted, by authorizing the automatic generation of services dedicated to the databases, achieved by using intrinsic programming possibilities associated with specific SGML parsers, SGML has allowed the simplification and improvement of the development process as far as costs and deadlines are concerned. Above all, SGML permitted command of the complexity which is inherent in any project that calls for different workstations, distant databases that are conceptually modelled according to their needs, and heterogeneous networks, in short everything which characterizes large present-day projects.

SGML plays an important part in the world of documents and it is possible, indeed effective, to use it in a project such as INGEPOL. It has been shown that the flexibility of SGML associated with object-oriented languages allows the description of the behaviour of an application without having to worry about the client platform, and to standardize the encoding process to a greater extent by allowing the abstraction of services generated by the concept.

**Notes**

1. Forces de l'Ordre, grouping the local police and the state police force.
2. Encoding and deciphering data transported on the network.
3. Document Type Definition.
4. Link Type Definition.
5. String, integer, date, boolean, or reference to a nomenclature.
6. The format for the dates can be YYYY, MM/YYYY, DD/MM/YYYY; in the other cases the length must be indicated.
7. This principle means that the components generated are stable and, by way of a bonus development work easier, thereby reducing development time.
8. Python is a high-level programming language which is object-oriented, belongs to the public domain, and is used in the application to introduce portions of code into the instance.
9. See Generation of Database Services, above.
10. Create, modify, or delete.
11 TUXEDO.

12 The interface consists of converting data into a format that can be recognized by the distant system, of executing the process desired, and, after conversion, of transferring the information received to the INGEPOL system.

13 The term used henceforth is dialogue instance (singular) notwithstanding the fact that there are several instances that describe the application.

14 The name of the dialogue is the name of the HTML page or the VT100 screen that will be used following an action by the user.

15 Creation, deletion, or updating.

16 Profile control depends on the officer connected, checking that an item has been selected in a list, etc.

17 The macro-instructions are defined in the modules integrated into the client workstation.


19 For fields that refer to nomenclatures.


21 The client workstation is written in Visual C++ version 5 (browser-based application).

22 The HTML page is equivalent to the dialogue box described in the dialogue instance.


24 Only the VT100 workstations in the distant units.


26 See Generation of Database Services, above.

Please e-mail your comments to Jacques Kasprzak at jka@sgmltech.com.

This paper was first published in the Conference Proceedings of XML Europe'99, April 1999, pp 207-15.

© The SGML Technologies Group 1999
SGML Joins the Battle Against International Art Robbery

Author

Jorge Leal Portela

Keywords

- SGML
- Applications

Abstract

Many organizations, such as museums, police forces, and insurance companies, are faced with the problem of identification of stolen and recovered objects of art and have difficulties in sharing relevant information. GRASP (Global Retrieval, Access and information System for Property items) is a project which addresses the problem of sharing information by demonstrating how descriptions of objects can be captured, stored in a heterogeneous database, and widely distributed across a network environment.

This paper addresses the issue of how SGML (Standard Generalized Markup Language) was successfully used for numerous aspects of the project, ranging from data storage and specification of the exchange structure, to distributed database synchronization control, in combination with a programmable processing tool.

Biographical Note

Jorge Leal Portela is a project manager at ACSE sa/nv, a member of the SGML Technologies Group. He is a software engineer and systems architect specializing in object-oriented distributed applications and document management applications. All these systems use SGML either as a document storage and exchange medium or as a formal message specification tool for communications between distributed application processes. A graduate of engineering specializing in nuclear physics, he studied at the Free University of Brussels. He may be contacted at jlp@sgmltech.com.
Introduction

Many organizations, such as museums, police forces, and insurance companies, are faced with the problem of identification of stolen and recovered objects of art and have difficulties in sharing relevant information. GRASP is a project which addresses the problem of sharing information by demonstrating how information about objects can be captured, stored in a heterogeneous database, and widely distributed across a network environment.

GRASP AD1008 is a Telematics for Administrations project funded by the European Union. It started in January 1996 and is currently (summer 1998) in the final validation phase. The list of partners includes prestigious organizations in the field of crime, cultural heritage, and museums, such as the Metropolitan Police Services (Scotland Yard) in London, the Spanish Ministry of Culture, and the Dutch Central Information Research Department CRI (Centrale Recherchle Informationdienst).

Key Options and Requirements

Resulting from the requirements analysis phase (see [GRASPD51] for full functional specifications), the GRASP partners decided to focus on several functionalities. In particular, the system should:

- store descriptions of items of property, especially objects of art;
- assist the user in the description process;
- deal with existing data;
- be multilingual and distributed;
- allow for future extensions and the inclusion of new participating organizations.

Technologies

The originality of the approach of the GRASP partners was to combine relevant existing technologies in order to meet the project requirements.

Some of the technologies include:

- Prolog as an artificial intelligence programming language;
- relational databases as an efficient back-end storage system;
- CORBA (Common Object Request Broker Architecture) as a flexible inter-component messaging system;
- C++ as an object-oriented programming language;
Internet as a Wide Area Network (WAN);

SGML.

This paper will focus on SGML. The following sections describe how SGML, in combination with the other technologies, helped the project requirements to be met.

**The GRASP Distributed Architecture**

The GRASP network is organized in terms of nodes interconnected through a WAN. Typically, each organization will have one GRASP node. This node may, in turn, serve one or multiple local workstations equipped with the GRASP front-end user interface.

The system consists of various modules:

- the Adaptor module handles the storage and distribution of the object descriptions across the GRASP network;
- the HCI (Human Computer Interface) module is the front-end user interface of the system;
- the Ontology module is in charge of assisting the user during the description process;
- the Image Acquisition System module handles the picture acquisition and subsequent manipulations;
- the Communication System ensures safe (encrypted and authenticated) exchange between nodes.

An object-oriented approach is used for the GRASP design; there are a number of nodes composed of software modules that are distributed across a heterogeneous network of workstations and high-end servers. All interoperations among these components are handled by a CORBA-compliant Object Request Broker.

Actual content of the messages being exchanged between these components is specified and manipulated in SGML. As will be seen in a later section, SGML provides an elegant way of manipulating object descriptions.

This component-based architecture enables the user to reconfigure the system by replacing individual components according to his needs.

**The Ontology**

Each art object is described by a set of fifty-five descriptors, each of which describes one particular distinguishing feature of the object. Some descriptors are factual (colour, size, shape, signature text and position, and so on), while others are more subject to interpretation (style, region, period in time when the object was manufactured, subject, and so on). Because
GRASP has to provide access to a heterogeneous database a common vocabulary is required. This standardized vocabulary is the Ontology.

The GRASP ontology is based on the AAT (Arts and Architecture Thesaurus) developed at the Getty Research Institute. It contains 28,400 main terms and categories and about 100,000 terms including alternative spelling, synonyms, etc. The AAT is structured according to 33 hierarchies for different classes of concepts (material, processes and techniques, and styles and periods, for example). Each hierarchy contains a root concept and a tree of sub-concepts.

To support multilingual use the descriptor values are transformed into integer values before storage in the Adaptor and transformed back into actual words on any another node depending on the local language settings. Currently supported languages include English, Dutch, and Spanish (partially).

Consider the descriptor category. It is composed of three values which define the object category according to a classification. The first level of the classification groups objects in three types: instruments, fine art objects, and decorative arts. If a decorative art object is being described there will be, at the second level, the choice between ceramics, textiles, costumes, and glassware. And finally, if it is a textile that is being described, then at the third level the choice will be among tapistry, laceware, silkware, etc.

The user assigns values to descriptors with the HCI module; during this description process the Ontology system guides the user by computing on-the-fly smaller and smaller sets of values on the basis of the description introduced so far.

**Typical Usage Scenario**

This section will briefly describe a typical usage scenario.

- A police officer located in London describes a Ming vase which was reported stolen by its owner.

- The object is described in the English version of the HCI module with the assistance of the Ontology module.

- The description is forwarded to the Adaptor module which handles the storage and the distribution. The object is marked as Lost/Stolen.

- A few days later a Dutch police officer finds an object of suspect provenance in an abandoned van in the outskirts of Amsterdam - our Ming vase.

- Back in the office he introduces a description of the object into the GRASP system using the Dutch version of the HCI module.

- The object is forwarded to the Adaptor module and marked as Found/Recovered.
The system reports a list of possible matching candidates for the newly-introduced object, among which is the Ming vase previously reported stolen. The objects are illustrated by a thumbnail sketch.

- The Dutch officer identifies the vase.
- The Dutch officer contacts his colleagues in London and checks whether the two vases are indeed one and the same object.
- The Ming vase is returned to its rightful owner.

**SGML for Object Descriptions**

The descriptors are organized in a hierarchical structure: the description tree. Additionally, description trees must conform to a set of rules that define the structure and occurrence of each descriptor.

The description structure is, from a conceptual viewpoint, very close to the hierarchical structure of a document. SGML thus appeared the obvious candidate for structuring, validating, versioning, and handling this kind of data.

A start was made by specifying the structure of the description tree (document) in a DTD (Document Type Definition). To do so the descriptors were mapped to elements, their structure and occurrence to element models, and the descriptor values to numerical attributes. The Ontology thesaurus takes care of the translation of the descriptor's terms (actually words) into numerical values according to local language settings.

Owing to the similarity between documents and object descriptions, SGML proved to be far more compact and elegant as a data definition language compared to a classical structured data definition language such as CORBA's Interface Definition Language (IDL). The resulting architecture uses CORBA as a low-end inter-component messaging system, while SGML takes care of defining the actual content of the messages that are exchanged.

SGML was also chosen for the storage of the descriptions themselves. The alternative technology, a relational database, is very efficient for handling many to many relationships but very clumsy for manipulating linear and hierarchical structures like the GRASP descriptions. The system still uses a relational database but only as a back-end storage system; the descriptions are stored as SGML instances.

The object description DTD of the GRASP project and an example instance are given in the annexes.

**Ranking Object Descriptions**

Ranking is used to find matching candidates for a certain reference object. A successful ranking will therefore present the objects that are in the database in such a way that the
objects closest (admitting a distance between two descriptions) to the reference object will appear first.

The ranking strategy used for GRASP consists of three phases.

- A pre-selection phase excludes most descriptions according to a very coarse classification based on a few elements of the description.

- For each of the remaining candidates the system computes the distance to the reference object.

- The list of descriptions is sorted and presented to the user.

The main difficulties in finding the distance between two descriptions are:

- the fact that the relative distance between two descriptions is difficult to assess and thus even more difficult to measure;

- descriptions are generally incomplete or are made up of different sets of descriptors;

- element values are numbers associated with concepts (again the relative distance between two concepts is difficult to assess and therefore even more difficult to measure);

- the algorithm should be able to compare descriptions that are at different levels of detail (that is, the algorithm should be able to match rich descriptions, such as the ones made by an expert, with poorer ones made by a police officer, for example).

To summarize, and without going into too much detail, the following assumptions were made:

- the distance calculation will only take into account the descriptors present in both descriptions;

- values of descriptors are reduced to a common level of detail and an equality test is done at that level.

As an example to illustrate the second assumption, suppose two values for the colour descriptor are being compared. The system will reduce the detail level of colour to a common level; comparing light brown with dark brown will therefore give a positive result because both colours are derived from brown; on the other hand comparing light brown with dark green will give a negative result because the first colour is derived from brown and the second from green. This simple technique was applied to all descriptors.

Although the test base is still rather limited, preliminary tests gave good results in terms of selectivity and resistance to perturbation (ranking of incomplete or slightly perturbed object descriptions).

It is possible that the distance calculation algorithm could be extended to other classes of documents.
The distance calculation is processed by an SGML parser. A reference object and a list of object descriptions are given as input; on output the application produces a list of relative distances. Owing to the stream-based way of processing the data, a large number of descriptions can be processed efficiently.

SGML for Inter-node Transaction Exchange

As stated before, SGML is used for handling object descriptions that are stored in the Adaptor module. But to ensure distribution, once the descriptions are stored in the Adaptor of one node, they have to be replicated in other nodes so as to ensure data availability to all nodes. This is performed by a transaction record and replay system.

All operations on a node - insertion, modification, and deletion of object descriptions - are logged (recorded) in the Adaptor. These transactions can be requested by other nodes that will repeat (replay) them in order to synchronize their database contents.

SGML was the natural choice as an exchange format for these transactions. Additionally, the generation and the processing of these transactions are carried out by means of an SGML parsing system coupled with an application language. Owing to the stream-based way of processing the information, large transaction files can be processed efficiently.

Conclusions

GRASP is not considered to be a typical document management application. However, SGML proved well suited because of the similarities in structure between GRASP object descriptions and traditional documents. The benefits of using this technology were clear, particularly in terms of reduced system complexity and reduced development time.
Annex A: The GRASP DTD

```xml
<!DOCTYPE description SYSTEM "GRASP.dtd">

<!--****************************************************************
Copyright (c) GRASP Consortium/ACSE, 1996-1998
*******************************************************************-->

<!--DTD of the SGML description message that allows to describe any art object
and containing the case properties of the item.-->-->

<!--****************************************************************
History:
*******************************************************************-->

<!--****************************************************************
Ranking
*******************************************************************-->

<!ENTITY %RANK-MODE-ON "IGNORE"> <!-- Do not modify! -->
<!ENTITY %RANK-MODE-OFF "INCLUDE">-->

<!--****************************************************************
Pseudo types
*******************************************************************-->

<!-- ISO-Latin with entities for Greek alphabet -->
<brick % date "CDATA"> <!-- +/-yyymmdd -->
<!ENTITY % ontology "CDATA"> <!-- integer -->
<!ENTITY % closed-list "CDATA"> <!-- integer -->
<!ENTITY % living-list "CDATA"> <!-- integer -->
<!ENTITY % millimeter "CDATA"> <!-- integer -->
<!ENTITY % text "#PCDATA">

<!--***********************************************************
Textual elements definition
*******************************************************************-->

<!ENTITY % langCodes "(FR|SP|EN|NL|IT)">

<!ENTITY % textDscr
  "(%short_description; | %full_description; | %title; | %serial_number; | %engravings; | %inscriptions; | %distinctive_features;)">

<!ELEMENT %textDscr; - - (%text;)

<!ATTLIST %textDscr; %language; %langCodes; #REQUIRED>

<!--****************************************************************
onthology descriptors definition
*******************************************************************-->

<!ENTITY % dsc-onthology
  "%technique; | %period; | %style; | %main_material-1; | %main_material-2; | %additional_material; | %region; | %place; | %form; | %visual_texture; | %intended_location; | %main_colour; | %background_colour; | %other_colour; | %object_type; | %functional_context; | %subject_matter_type; | %pattern;">-->

<!ELEMENT (%dsc-onthology;) - - EMPTY>

<!ATTLIST (%dsc-onthology;)
  %value;  %ontology; #REQUIRED
  %baseterm; %ontology; #REQUIRED>

```

SGML Joins the Battle Against International Art Robbery

The Exploitation of SGML
SGML Joins the Battle Against International Art Robbery

<!ELEMENT %subject_matter_content-l; - -     (%subject_matter_content;)*>  <!-- SWI -->
<!ELEMENT %component-l; - -     (%component;)*>    <!-- SWI -->
<!ELEMENT %pattern-l;        - -     (%pattern;)*>
<!ELEMENT %subject_matter_content;      - -     EMPTY>
<!ATTLIST %subject_matter_content;   %subject_matter_content_contents;   %ontology;  #REQUIRED      -- SWI --
%subject_matter_quantity;           CDATA       "1"      -- SWI --
%subject_matter_property;           %ontology;  #IMPLIED>     <!-- SWI -->

<!ELEMENT %component;      - -     EMPTY>
<!ATTLIST %component;      -- SWI --
%component_type; %ontology;  #REQUIRED      -- SWI --
%component_quantity;  CDATA       "1"      -- SWI --
%component_property;  %ontology;  #IMPLIED>     <!-- SWI -->

<!ELEMENT %production; - - ((%maker-l;)?, (%technique-l;)?, (%period;)?, (%time_period;)?, (%year_of_make;)?, (%style;)?, %main_material-1;, (%main_material-2;)?, (%additional_material-l;)?, (%place;)?, (%region;)?)>
<!ELEMENT %maker-l; - - (%maker;)*>
<!ELEMENT %maker;   - - EMPTY>
<!ATTLIST %maker;   %value; %living-list; #REQUIRED>

<!ELEMENT %technique-l; - - (%technique;)*>
<!ELEMENT %year_of_make; - - EMPTY>
<!ATTLIST %year_of_make;   d %date;  #REQUIRED>

<!ELEMENT %time_period; - - EMPTY>
<!ATTLIST %time_period;   %from; %date;  #REQUIRED
%to;  %date;  #REQUIRED>

<!ELEMENT %additional_material-l;  - - (%additional_material;)*>

<!ELEMENT %visual_texture-l; - - (%visual_texture;)*>
<!ELEMENT %main_colour-l;    - - (%main_colour;)*>
<!ELEMENT %other_colour-l;   - - (%other_colour;)*>
<!ELEMENT %colour_cardinality;  - - EMPTY>
<!ATTLIST %colour_cardinality;   - - EMPTY>

<!ELEMENT %physical; - - ((%form;)?, (%visual_texture-l;)?, %measurements;,(%intended_location;)?, (%main_colour-1;)?, (%background_colour;)?, %colour_cardinality;,(%other_colour-1;)?, (%markings;)?)>
<!ELEMENT %visual_texture-l; - - (%visual_texture;)*>
<!ELEMENT %main_colour-1; - - (%main_colour;)*>
<!ELEMENT %other_colour-1; - - (%other_colour;)*>
<!ELEMENT %colour_cardinality; - - EMPTY>
<!ATTLIST %colour_cardinality;   - - EMPTY>
SGML Joins the Battle Against International Art Robbery

%value; %closed-list; #REQUIRED> <!-- 0|1|2|9=many -->

<!ELEMENT %measurements; - - EMPTY>
<!ATTLIST %measurements; 
%weight; CDATA #REQUIRED 
%height; %millimeter; #REQUIRED 
%length; %millimeter; #REQUIRED 
%width; %millimeter; #REQUIRED>

<!-- weight in grams  
height: when object is in its natural position (could be 0 for lying objects ...) 
length: longest other dimension  
width: shortest other dimension (could be 0 if wall-suspended ...)
-->

<!-----------------------------------------------------------------------
markings definition                                                        
----------------------------------------------------------------------->

<!ELEMENT %markings; - - 
(%seals_marks-l;)?, 
(%title;)*, 
(%serial_number;)*, 
(%signature;)?, 
(%engravings;)?, 
(%inscriptions;)?)> 

<!ELEMENT %seals_marks-l; - - (%seals_marks;)*> 
<!ELEMENT %seals_marks;   - - EMPTY> 
<!ATTLIST %seals_marks; 
%value;    %living-list; #REQUIRED>       <!-- SWI -->
<!--   %baseterm; %ontology; #REQUIRED--       -- SWI!!! -->

<!ELEMENT %markings-1; - - (%seals_marks;)*> 
<!ELEMENT %seals_marks; - - EMPTY> 
<!ATTLIST %seals_marks; 
%value;   %living-list; #REQUIRED> 
<!ELEMENT %signature; - - EMPTY> 
<!ATTLIST %signature; 
%position;           %closed-list; #REQUIRED 
%year;               CDATA         #IMPLIED 
%signature_name;     CDATA         #IMPLIED>

<!-- position 
0 = no signature  
1 = bottom right  
2 = bottom left  
3 = yes, undefined position  
4 = yes, on the back  
5 = top right  
6 = top left  
-->

<!-----------------------------------------------------------------------
others definitions                                                        
----------------------------------------------------------------------->

<!--  
url where item is further described  
(not replicated with ptyitem on other nodes  
as an url may change locally out of GRASP control)
-->

<!ELEMENT %url;    - - EMPTY> 
<!ATTLIST %url;    name      CDATA #REQUIRED>

<!--  
Images and thumbnails                                                       
-------------------------------  
Images and thumbnails are referred by IDs and are not directly included in the SGML flows. 
IDs could be file names in a certain context, TBD by ASTRA
-->

<!ELEMENT %imgs;       - - EMPTY> 
<!ATTLIST %imgs;       folderid CDATA #REQUIRED>
Annex B: A GRASP Instance

```
<PIT AID="Test Org." DHAP="19980202" DINS="19980223" DMOD="19980610"
DREP="19980202" GID="46" GND="0" ITT="WHOLE" OPC="t" PID="se"
STS="FR"><SUM LAN="EN">a brown walnut queen anne kneehole desk</SUM><WHL DID="0" QTY="1"><TOP
V="60000139"/><MGR V="50037335"/><GRP
V="50037680"/><TOP
B="50021047" V="50021047"/><TPR P="1700"
T="1720"/><MM1 B="50132451" V="50012476"/><RGN
B="50020656" V="60000187"/><PHY><MGR
B="50127490" V="50127490"/><COLCAR
V="1"/></PHY><OBT
B="50136379" V="50136379"/></WHL></PIT>
```

Reference

- [GRASPD51] The GRASP Consortium: Project Deliverable 5.1 Functional Specifications; public project deliverable available on GRASP's Web site (http://www.arttic.com/GRASP/)

Please e-mail your comments to Jorge Leal Portela at jlp@sgmltech.com.

This paper was first published in the Conference Proceedings of Markup Technologies '98 US, November 1998, pp 145-52.

© The SGML Technologies Group 1998
Experience of EDI for Documents: The Role of SGML

Author

Philippe Vijghen

Keywords

- SGML
- XML
- System Integration
- EDI
- Workflow

Abstract

This paper describes the use of SGML in the EDIDOC project for the European Space Agency. The project involved the creation of a flexible framework for exchanging different types of documents, being a gateway for workflow, document conversions, security, and communication. It is used for calls for tenders, working documents, and press releases, and also covers WWW publication.

SGML was used for many aspects including attaching the different envelopes of the messages exchanged and as a technology for defining workflow scenarios. Benefits and challenges of using SGML or XML at different levels are highlighted, both technically and organizationally.

Biographical Note

Philippe Vijghen is a project manager at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. He is a software engineer and systems architect specializing in object-oriented distributed applications and complex document-oriented Electronic Data Interchange systems; in addition to structuring documents, these systems make use of SGML at other levels, such as for external application programming interfaces. He obtained a degree, specializing in electromechanical engineering, at the Free University of Brussels (ULB). He may be contacted at phv@sgmltech.com.
The EDIDOC Project

More and more companies are now moving towards Electronic Document Interchange (EDI) with their partners, in the broad meaning of the term, including collaboration on document processing. However, it is still challenging to reach an agreement on technical choices at different levels (for example document formats, electronic security mechanisms, and protocols). This can be especially so when dealing with complex workflow on a WAN, across borders.

This paper results from the experience gained during the development of a flexible framework for document-oriented EDI. This framework, called EDIDOC, has been developed for the European Space Agency. The project started in 1994 and entailed about 10,000 hours of work, leading to the implementation of a system responsible for document exchange occurring in several distinct applications of the agency, including:

- working documents exchanged among the delegations;
- calls for tenders sent to the potential bidders;
- press releases and information notes sent to the press and public;
- WWW publication.

More than 2,000 people are involved in the document exchanges.

At the heart of the EDIDOC system a central server acts as a clearing house, giving a potential legal value to the documents exchanged by logging them and by keeping a structured trace of the exchange in a robust relational database.

This server integrates in a very generic and flexible way the key concepts needed in electronic document exchanges:

- document standards (conformance checking and format conversions), eg SGML, EDIFACT, PDF, ASCII, RTF, and WordPerfect;
- security packages (information confidentiality, and authentication), eg PGP and MAC MD-5;
- access protocols (including network aspects), eg WWW, Internet e-mail, X.400, and FTP.

At each of those levels the server makes sure that the documents are delivered in accordance with preferences of the recipients: in the right format, with the right security package, and with the right communication protocols. It really plays the role of a gateway.

For example, a telecommunications company could use EDIDOC to send invoices to a customer. According to its profile, the customer may get the invoice as:

- a secured EDIFACT file sent by X.400;
• a PostScript document sent by Internet e-mail;
• as a secured WWW page;
• and why not soon as an information note sent by Web-push technology?

The server also includes provision for integrating specific application logic in document exchanges. This is typically used for implementing workflow scenarios such as support for document review cycles or a chain of approvals for payment. At ESA a scenario has been developed so that when a new press release is sent to registered subscribers (in ASCII, HTML, or as a URL), EDIDOC also publishes it on the WWW and adds hyperlinks on the organization's home page.

Many utilities have been developed around the server, for example system management tools and graphical clients.

Several different groups within an organization have the need to exchange formal documents with those outside the organization. The benefit of having them all going through a single point, in a three-tier architecture, are obvious:

• new document formats, security schemes, and communication protocols can be defined without any impact on the information producers;
• the staff in charge of the operation control and configure the document exchanged in a single place.

The following diagram illustrates the functional architecture of EDIDOC.
As can be appreciated, such a project involved many aspects that, although interesting, lie outside the scope of this paper. Where SGML is concerned, it proved to be the technical cornerstone of the implementation.

**SGML: The Cornerstone of EDIDOC**

SGML impacted the project at various levels:

- as a way to structure the documents exchanged;
- as a method to attach the descriptive envelopes of the messages;
- as the pivotal concept for implementing complex conversions;
- as the glue for the integration of external applications, and more specifically as the interface for the implementation of workflow scenarios;
- as a technology for defining workflow scenarios, including their implementation and documentation.

The benefits and challenges of using SGML at those different levels will be highlighted, both from a technical point of view and from an organizational point of view. When appropriate, it will be explained how far SGML is competing with other concepts, in the light of personal experience.
Although the EDIDOC application was used as a basis for this paper, a few of the statements and ideas put forward result from experience gained on other projects.

**SGML for Structuring Source Documents**

As the documents sent through EDIDOC are checked and then converted to various formats according to the profile of the recipient, the benefits of having original documents in SGML format seem obvious.

Although format conversions of unstructured documents are possible, the set of target formats is often limited and, most importantly, the result of the conversion is often not satisfactory. Also, such conversions should preferably occur on the desktop instead of on a server.

Anyone familiar with SGML and the earth's gravitational forces will easily understand why translations to SGML are called *up-translation* and translation from SGML *down-translation*. It is indeed difficult to automate the enrichment of document structures, and *a fortiori* to exploit unstructured documents. Converting correctly structured documents to other target formats is a lot easier.

Even in the case where up-translation to SGML sounds feasible on the server, it has the disadvantage of detecting mistakes very late. Using structured editors for producing SGML at the source is better.

However, although SGML has technical advantages, the reality is that for many organizations (although not specifically for ESA) it is very difficult to impose SGML authoring tools on authors, for various reasons. This is especially so when the authors are employed by other organizations. Moreover, even if SGML is accepted, it is challenging to agree on a DTD when many partners are involved.

EDIDOC does not enforce the use of SGML for documents. At ESA, a radically different approach was even selected for sending calls for tenders to potential contractors. Complete flexibility is offered to authors, provided that they generate PDF files as output. Of course, the problem of having PDF as a source for distribution is that it then limits the flexibility for further conversions (only ASCII versions of documents are produced from PDF in our case). But this drawback is negligible in comparison with the problem that the set of authors would face if SGML were imposed.

However, authoring using SGML is still recommended in the following cases:

- when *numerous target formats* are required;
- when the documents must be *protected against the future* (the information will be used for a very long time, probably using new technologies);
- there is a *limited set of authors*;
- the partners manipulating SGML documents are involved on a very *regular basis*;
- document structure is needed by some specific *electronic processing* applications.
If few of those conditions are fulfilled, SGML should not be considered; it would be too much of a challenge for little benefit.

**SGML for Labelling EDIDOC Envelopes**

As EDIDOC messages can be transported by many different vehicles (for example e-mail, X.400, and WWW), we had to define an envelope for the documents in order to specify the details of the exchange: originator, list of recipients, unique reference, subject, time stamps, document types and formats, security mechanisms, delivery options, groupware context, remote management options, error messages, and so on.

Below is a sample header for an EDIDOC message:

```xml
<esh version="2">
  <secpack mode="clear">none</secpack>
</esh>
<eh version="2">
  <user level="process" notif="always" clearmsg="safemsg" clearnot="safenot" nondeliv="negack" destin="most">
    <sndreq>
      <from>phv@acse.be</from>
      <ref>phv@acse.be - 10/15/97 10:46:11</ref>
      <subject>PDF version of EDIPress manual</subject>
      <docinfo>
        <type>FLAT</type>
        <format>PDF</format>
      </docinfo>
      <to>sansari</to>
    </sndreq>
  </user>
</eh>

%PDF-1.2 document is included here...
```

These envelopes are generated by the producers of information to send requests to EDIDOC and are parsed by the recipients who do want to exploit the EDIDOC context. This means that people should be able to generate and process them on many different platforms, using different tools. In fact we developed the following APIs and tools for parsing or producing the envelopes:

- **specification**
  - SGML DTDs à la XML (not even empty elements);

- **developers' APIs**
  - a Simple SGML-Like Parser (named SSLP but what I would now call a light XML parser),
  - a C++ class corresponding to the data structured in the SGML envelopes, this class being used from C++ code, for example to make the link with relational databases representations,
  - a Unix shell interface based on environment variables;
• **end-user tools**
  - graphical clients for Windows,
  - WWW CGI scripts and Java code,
  - server-side generation of pretty MIME e-mail
  - ASCII pretty printing.

For those integrating their applications with EDIDOC, it was easier to generate and process the envelopes directly by looking at the simple SGML specification. When security algorithms are required, the API-level interfaces are preferred. Of course, the end-users just use tools, much as an Internet surfer would use a browser for viewing HTML.

Using valid XML files (called light normalized SGML at that time) for structuring data exchanged between loosely coupled applications has the following selling points:

• it is an relatively accessible format for the person who looks at it;

• it is standardized;

• it has the big advantage of offering automatic implementation of parsers by using SGML or even XML tools and loading them with a standardized meta-specification (DTD);

• it is so easy to generate and even to parse that you can deal with it if you simply have read/write access to files on your platform.

In the classic EDI context, EDIFACT has the big advantage of having well specified and standardized formats for many types of EDI messages, such as invoices and orders. The semantics of all the fields of those messages are then clearly specified.

However, we believe that SGML, and more specifically its little child XML, is more appropriate when no existing EDIFACT message fulfils the needs. Indeed, if a new message specification has to be defined, XML has the following advantages over EDIFACT:

• DTDs not only specify the structure of the document but also do it in a standard way and are processable by any SGML-compliant parser (the definition of EDIFACT messages also has to be modelled in the parser);

• valid XML is easy to parse, from many programming languages;

• SGML documents are more intelligible from a developer's point of view (the structure and the name of the elements are easier to understand).

One of the most famous examples is that Microsoft selected a light SGML specification, *à la* XML, for its Open Financial Connectivity, used in the electronic dialogues between Microsoft Money and banking institutions.
SGML for Implementing Convertors

In its generic way for configuring convertors, EDIDOC defined, for each type of document, a pivot format. Conversions (up or down) and conformance-checking scripts are defined relative to this format. Although EDIDOC does not impose its use, SGML is the obvious choice for being a pivot format. In fact SGML not only offers a way to specify and structure documents but it even offers a few mechanisms for processing the information.

The following features are particularly useful for processing non-SGML documents with full-blown SGML-aware toolkits.

- OMITAG allows for the automatic closing of open elements when some new elements are recognized. For example all open bullet lists, paragraphs, and subsections are automatically closed when a new section is encountered, simply because of the SGML specification of the document model being used at the heart of the converters.

- SHORTREF offers a nice mechanism for generating or preserving SGML entities or for recognizing special characters. With some non-standard extensions, SHORTREF can in fact even be used for adding sophisticated pattern-matching recognition in a similar way.

- LINK allows the context-dependent attachment of code to the occurrence of some elements, in a standard way.

- CONCUR is a must when processing documents that are formatted using various concurrent formatting styles. For example bold and italic markers can typically cross paragraph boundaries. Having a tool that can present the formatting possibilities as many different concurrent grammars is very useful. One of the convertors developed by the SGML Technologies Group uses twenty-five different grammars.

Glue for Linking External Applications

The EDIDOC server offers the possibility to plug-in customized application logics, such as a workflow scenario for supporting document review cycles. The server keeps track of the processing evolution, dispatches the various relevant events and, most importantly, offers transparent access to the various document formats, security mechanisms, and communication schemes.

Technically, so-called groupware scenarios can be plugged into EDIDOC by implementing external scripts called event handlers.

We thought first about including a generic workflow engine in EDIDOC. This engine would have been processing the definition of finite-state machines or Petri networks. However, we realized that there were annoying limitations for the developer in each of those models (for example ‘What if a new reviewer is added in our scenario?’). Also, we preferred to have loosely coupled external event handlers, interacting with EDIDOC through a clean interface.
This interface had to impose as few constraints as possible on the tools used for developing programs defining the customized behaviour of the server for a workflow scenario. Also, the interface between EDIDOC and the groupware event handlers has been defined as a simple SGML (XML-compliant) interface.

Those event handlers get various kinds of event from the server:

- groupware requests sent by a partner, generally including a document (whatever the network protocol, the security options, and the document formats, it will be transparent for the event handler);
- expired time-outs registered for the scenario instance;
- non-delivery notifications.

In response to those events, the external script can request the EDIDOC server to:

- register time-outs;
- update data to be stored in the server, usually for keeping track of the context;
- produce error notification messages;
- log some specific traces;
- send messages, using any of the options that EDIDOC offers.

All of those requests are passed between the script and the server as XML-formatted data flows that comply to specific DTDs: one for the standard input, one for the standard output. This gives complete flexibility to the developers for the languages and tools to be used for implementing the external scripts; if they can control the standard input and output, they have all what is needed. Of course, using an SGML toolkit gives the advantage of having robust parsing with little effort.

**Implementation of Workflow Scenarios**

This is the most complicated part of the paper and covers a methodology that our Group is using for several projects. This methodology helps us to define, document, and implement applications based on multiple state diagrams. This is particularly useful for implementing workflow solutions.

**State Diagrams Specification**

We have been developing a meta-DTD for specifying classes of state diagrams. From this meta-DTD, the following documents are automatically generated:

- TeX documentation of the classes, state diagrams, and events.
Experience of EDI for Documents: The Role of SGML

- implementation-DTDs that define the implementation of state diagrams by means of standard SGML features.

In the meta-DTD, classes are defined as sets of finite state diagrams, including states and events.

**SGML Representation of Events**

In the generated implementation-DTD, each event is mapped to an SGML element. That element can have specific attributes and can include structured data as child elements; this allows for the passing of complex structured information with each event.

**Linking Code to Events**

SGML link rules allows for the attachment of external information to SGML elements in a standard context-dependent way. We use this mechanism to attach application logic to SGML events. Thanks to the implementation-DTD, different link sets are used for each state in the diagram; they are automatically activated by standard SGML means when a new state is reached. Specific application code can then be linked to the occurrence of the various events by the customization of the SGML link rules of the link set, for example for accessing a database and forwarding events to other processes. The code to execute when an `event` element occurs is selected by the standard use of SGML LINK.

**Implementation of the State Transitions**

When a new `event` element is input, the SGML parser also knows the new state of the state diagram by standard SGML means. This is due to the appropriate definition of the SGML group models and the OMITAG feature in the generated implementation-DTD. No specific code has to be written for doing this.

The following example illustrates a definition of state elements similar to the ones generated in the sample DTD. When an event is encountered in the input stream, the SGML parser implicitly selects the new state.

```
<!ELEMENT stateA O O ((event1,stateB)|(event2,stateA))>
<!ELEMENT stateB O O (event3,stateC)|(event4,stateA)>
...`

**Concurrent State Diagrams**

At run-time, for each class of diagrams, a single SGML parsing process is in charge of managing all the state diagrams and the associated logic; there is one concurrent parsing of the various SGML DTDs defined for each state diagram. The application language has concurrent access to the various contexts.

**Benefits of the Approach**

Such an approach offers key benefits:
the formalization of the application logic as state diagrams leads to cleaner, more robust code;

it means that the code includes less explicit tests and concentrates more on the implementation of specific processing;

moreover, the code relative to the application logic, for example a workflow, is less subject to errors;

if the workflow is slightly changed, for example by allowing a user action in a new state or by defining some new actions, the impact on the code is limited;

there is an important productivity gain by using a common, consistent way to work across projects, if appropriate.

Conclusions

This paper has demonstrated where SGML is invaluable for system development and integration, based on the EDIDOC experience.

XML can be considered as a candidate of choice for structuring data in EDI applications, when no EDIFACT message fulfils the needs. It is also very useful as a way of structuring inter-process communications when integrating distributed applications.

Full-blown SGML toolkits are a must for data processing. They are particularly useful for implementing data convertors, just by using standard features such as OMITAG, SHORTREF, LINK, and CONCUR. Finally, SGML and related development tools offer a nice way for addressing, at the same time, the definition, documentation, and implementation of workflow scenarios.

Acknowledgements

I wish to thank Jordi Farrès, of the European Space Agency, for having defined the EDIDOC requirements and for the numerous open-minded technical discussions during the analysis and design phases.

My thanks are also due to Stéphane Bidoul for his explanations regarding the implementation of state diagrams by the means of SGML technology.

Reference

Experience of EDI for Documents: The Role of SGML

Acronyms

CGI     Common Gateway Interface
DTD     Document Type Definition
EDI     Electronic Data Interchange
EDIDOC  Electronic Data Interchange for Documents
EDIFACT Electronic Data Interchange for Administration, Commerce and Transport
ESA     European Space Agency
FTP     File Transfer Protocol
HTML    HyperText Markup Language
MAC     Message Authentication Code
MIME    Multipurpose Internet Mail Extensions
PGP     Pretty Good Privacy
SGML    Standardized Generalized Markup Language
URL     Uniform Resource Locator
WAN     Wide Area Network
WWW     World Wide Web
XML     eXtensible Markup Language

Please e-mail your comments to Philippe Vijghen at phv@sgmltech.com.

This paper was first published in the Conference Proceedings of SGML'97 US, December 1997, pp 213-18.

© The SGML Technologies Group 1997
Cost-Effective EDI Using XML? A Pivot-Oriented Approach

Author

Philippe Vijghen

Abstract

This paper proposes the use of a pivot format when developing EDI applications, based on the experience of three operational projects.

The role of SGML/XML, as pivot, is presented in a broader context, with regard to other relevant candidates for structuring data.

Biographical Note

Philippe Vijghen is a project manager at ACSE sa/nv, Brussels, a member of the SGML Technologies Group. He is a software engineer and systems architect specializing in object-oriented distributed applications and complex document-oriented Electronic Data Interchange systems; in addition to structuring documents, these systems make use of SGML at other levels, such as for external application programming interfaces. He obtained a degree, specializing in electromechanical engineering, at the Free University of Brussels (ULB). He may be contacted at phv@sgmltech.com.

Introduction

More and more companies, together with their partners, are moving towards EDI (Electronic Data Interchange). Traditional EDI, however, has the reputation of being inflexible and expensive.

This paper calls on the experience gained from the use of SGML/XML in various EDI projects. It illustrates how the use of a central pivot model for implementing an EDI application is cost-effective, owing to its reusability and scalability. It also demonstrates that in terms of data modelling, SGML/XML offers the flexibility required for such a pivot role in the EDI system.

The first section gives an overview of the various syntaxes that have been defined in the past for exchanging data.

A pivot-oriented approach, and its benefits, are explained in the following section.
In the final section, three EDI projects are presented, where XML has been used as a pivot.

Various Syntaxes for Exchanging Structured Data

Several conventions have been defined for structuring electronic information in the past. Listed below are some well-known abstract and concrete data structuring mechanisms, and their respective application targets.

- **ASN.1 (Abstract Syntax Notation One) and BER (Basic Encoding Rules)**

  ASN.1 is a common platform and a language-independent way of defining abstract data structures. The associated BER specify the concrete representation of an ASN.1 structure when it transits between computers.

  ASN.1 is very useful for documenting structures at a conceptual design level, especially when working at network level. However, the only meta-information carried in the concrete syntax is for the identification of low-level data types. Therefore, the identification of the data semantics relies on an implicit knowledge of the structure.

- **RPC (Remote Procedure Call)**

  The various implementations of RPCs (eg by SUN, DCE, and Microsoft) address the need of software developers to call a function or procedure that resides on another machine. The RPC mechanisms include a platform-independent way to marshal the data (encoding/decoding before and after network transmissions) but its lack of meta-information only makes it suitable for volatile low-level inter-process communications.

- **CORBA (Common Object Request Broker Architecture) and IIOP (Internet Inter-Object Request Broker Protocol)**

  CORBA, like RPC, also targets software integration and the development of network-enabled applications, but this time in an object-oriented environment. Like ASN.1, it includes a mechanism for specifying data structures in an abstract, language-independent way called IDL (Interface Description Language). IDL also includes an abstract specification of the functional interfaces. As for RPCs, there are associated marshalling mechanisms (such as the one specified for Internet, IIOP) for passing data structures between objects. No meta-information about the data semantics is included in the marshalled data, as CORBA relies on the volatile object interfaces for addressing such needs.

- **CSV (Comma Separated Values)**

  The simple syntax of CSV has been defined for the exchange of tabular data, exported from databases or spreadsheets. The first row of the file, often including the titles of the various columns, can be considered as a meta-specification.
Cost-Effective EDI Using XML? A Pivot-Oriented Approach

- **EDIFACT (Electronic Data Interchange for Administration, Commerce, and Transport)**

  EDIFACT aims at specifying standard messages for exchanging electronic data in trade applications, eg orders and invoices. The syntax is built on top of separators and allows for messages to be structured as groups of segments that are themselves made up of data elements.

- **SGML (Standard Generalized Markup Language)**

  This standard was designed to structure documents. It provides a flexible mechanism that enables the user to model document structures and to encode them so as to enhance the manipulation, exchange, and publication of the documents.

- **XML (eXtensible Markup Language)**

  XML, a recently defined subset of SGML, brings the concepts behind SGML into the popular arena of Internet browsing. The idea behind XML is to maintain eighty per cent of the benefits of SGML, with only twenty per cent of its complexity. This enabled the development of easy XML support in tools as common as Web browsers. XML does not really target the SGML market; its major goal is the Internet distribution of structured documents. In terms of application target, Microsoft would also like to position XML as the next-generation CSV format, for interchanging data between databases, spreadsheets, and a few EDI applications.

A Brief Look at These Syntaxes

Although they are all aimed at defining a structure below the level of file granularity, it is clear that all of these different specifications and syntaxes target different needs in terms of application environments.

If the purely technical aspects of the various syntaxes are examined, however, it must be concluded that SGML/XML allows for more complex modelling.

- **ASN.1/BER, RPCs, and CORBA/IIOP** mechanisms are difficult to use for defining complex recurrent structures with optional branches and various lengths of data. Moreover, as no meta-information about the data semantic is encoded, it would be impossible to figure out which information is carried by looking only at the data: the encoding/decoding relies on implicit knowledge of the model. In any case, they are not really suitable for the archiving of information as they lack the set of tools that could be expected for achieving it.

- **CSV** has the advantage of being extremely simple and is used everywhere. But it is limited to the export of data that can be represented as a list of rows.

- **EDIFACT** syntax is based on four different levels: messages, groups, segments, and data elements. It is very appropriate for structuring information that has been defined in the standardized catalogues of messages and segments. However, it is difficult to
use EDIFACT as a generic syntax for modelling arbitrary information, especially when dealing with structured text. Moreover, EDIFACT lacks a formal meta-language.

- **XML** includes a formal way of specifying the model through the notion of a DTD. It offers a high-level of flexibility and extensibility and is particularly appropriate for modelling trees and graphs (by using links). It is simple to use. It is even possible to exploit an XML instance without knowing the details of the meta-specification that is the DTD. Unfortunately, it has very limited support for data types (the next revision of the standard will address this point).

- **SGML** includes many features that are not supported by XML. Those features can legitimately be considered as unnecessarily complex when the EDI data structures are considered. However, as explained in [Vijghen 97], many of those additional features are invaluable in a development environment used for processing the information.

This very straightforward comparison only takes into account the syntax and the modelling flexibility, of course. Although this comparison is irrelevant when the difference in application that is targeted by each of them is considered individually, the comparison is invaluable when considering the best candidate for representing pivot models.

Indeed, this paper addresses the need for consistent use of a *pivot format* when developing EDI applications. With a pivot format in mind, the use of the most flexible and scalable syntax is fundamental; there, the choice is purely technical and is independent of the external representation format that may be required by the users in function of the application field.

**A Pivot-Oriented Approach**

When developing EDI applications, one of the key tasks consists of implementing filters for processing the messages to be exchanged. Such filters aim either at converting the messages from one representation to another, or at using the information contained in the message in databases or other external applications.

Our experience demonstrated that using a pivot-oriented approach for developing such filters proved to be extremely cost-effective. The approach consists of using a single internal representation of the information, for the implementation of all the filters applied to EDI messages.

Note that the word 'pivot' here has a different meaning from that in the traditional EDI terminology, where it often designates more restrictively the representation used for loading and exporting messages to and from databases.

The cost effectiveness of such an approach is justified by the following facts:

- the code can be independent of the actual concrete representation of the information, by relying only on the pivot model;
the application can be adapted more easily if the public syntax of a message is modified;

the set of features available in the development environment, associated with the pivot syntax, becomes independent of the end-user's choice of public representation syntax;

the use of a consistent set of tools and techniques for manipulating the pivot syntax, not only across filters but also across projects, improves the productivity of the developers;

the number of filters that must be developed for implementing all the conversions between a set of formats is proportional to the number of formats, and does not grow exponentially as it does in cases where no pivot format is selected.

We, the SGML Technologies Group, have chosen to use SGML/XML and an integrated development environment based on this technology as the cornerstone for many projects based on a pivot-oriented approach, including those in the field of EDI.

**XML-Based EDI Applications**

Experience gained during the development of various EDI applications includes:

- G-EDI, a generic tool for parsing and processing EDIFACT messages, first developed for implementing an EDI system for the handling of telecommunication bills of a major Belgian bank;

- CLASET, an EDIFACT and SGML-based system for exchanging nomenclatures in the context of the European institution for statistics (EUROSTAT);

- EDIDOC, a generic framework for the Electronic Data Interchange of Documents, developed for the European Space Agency, that has been operational for a few years.

**G-EDI**

The G-EDI project, aiming at processing the telecommunication bills of a major banking institution in Belgium, initiated the development of a generic EDIFACT parser. The key point is that the parser was based on the notion of a pivot format. In practice, the implementation is based on SGML technology.

Although the SGML tags and syntax did not help as such for this implementation, the generic coding mechanisms that are part of SGML helped to keep the application independent of the actual syntax of the message. Indeed, XML offers all the possibilities that are required for modelling the information contained in EDIFACT, as it allows for the encoding of the documents with regard to arbitrary complex tree models and, if hyperlinks are considered, even graph models.
Although the actual EDIFACT syntax of the messages that are transmitted by the telecommunications company changed four times since the system was put into production, only the mapping to the generic underlying model had to be reviewed. Owing to the approach, it has been possible to reduce the application maintenance costs by a factor of five.

## CLASET

The goal of the CLASET (Classification Information Set Message) project, developed in the context of the European Programme for the Interchange of Data between Administrations (IDA), is rather ambitious. Take the example of the definition of `secondary school' in the various European countries. The reader will understand that there is no consistency at all. But the European Institutions still want to produce accurate statistics on such matters, across the internal borders. In order to achieve this, complex nomenclatures for statistics must be defined. CLASET includes the definition messages for exchanging such nomenclatures.

The CLASET message allows for the exchange of any hierarchical structure, such as nomenclatures or classifications, and has conceptually been defined as a result of a Merise model.

Different representations are used, each based on a distinct syntax:

- an EDIFACT representation of the messages is being standardized in official committees;
- an SGML representation is used as an alternative format to which the EDIFACT representation can be mapped;
- HTML representations of the messages can be produced, as generated from the SGML representation.

The EDIFACT representation is the most official representation of the information transported by CLASET messages, because of the standardization process.

SGML representation is recommended to people who are dealing with highly structured text, because the SGML representation offers more flexibility than EDIFACT for operations such as the attachment of footnotes or presentation styles to words that are part of the free text. Such structures can easily be encoded using an SGML mixed content group model, which has no equivalent in the EDIFACT layering of messages, groups, segments, and data.

The HTML representation is read-only: it enables people to view the messages using a tool such as a Web browser (used as a local viewer application in this case).

The actual implementation of the CLASET project is based on SGML, used as an internal pivot. This approach has lead to an application code that is independent of the actual details of the representation syntaxes and has been a major argument for the cost-effectiveness of the project. The benefit of such an approach is not due to the SGML syntax itself, but the tools associated with SGML offer the required flexibility and the relevant processing features.
EDIDOC

The EDIDOC (Electronic Data Interchange for Documents) project covers the design, implementation, and deployment of a flexible framework for document-oriented EDI at the European Space Agency (ESA).

The system is in charge of document exchanges occurring in several distinct applications of the agency:

- working documents exchanged with the delegations;
- calls for tender sent to potential bidders;
- press releases and information notes sent to the press and public (including their WWW publication).

At the heart of the EDIDOC system a central server acts as a clearing house, giving a potential legal value to the documents exchanged by logging them into a robust relational database.

This server integrates, in a very generic and flexible way, the key concepts needed for electronic document exchanges:

- **document standards** (conformance checking and format conversions), eg SGML, EDIFACT, PDF, ASCII, RTF, and WordPerfect;
- **security packages** (information confidentiality and authentication), eg PGP and MAC MD-5;
- **access protocols** (including network aspects), eg WWW, Internet e-mail, X.400, and FTP.

At each of these levels the server makes sure that the documents are delivered in accordance with the preferences of the recipients: in the right format, with the right security package, and the right communication protocols. It really plays the role of a gateway.

The EDIDOC generic envelopes have been defined in XML. They include the details of the exchange: originator, list of recipients, unique reference, subject, time stamps, document types and formats, security mechanisms, delivery options, groupware context, remote management options, error messages, and so on.

The filters that are plugged in the EDIDOC document standards' components are based on the notion of a pivot format for conversions. Although the use of SGML is not enforced, it is the best candidate for defining customized pivot formats for structured documents. Indeed, SGML includes a very consistent and generic way to model the information. Moreover, the use of SGML as a pivot format can help for the actual implementation of the converters because some of the SGML features, such as OMITAG, SHORTREF, LINK, and CONCUR, can be used for the actual implementation of the converters themselves.
EDIDOC has demonstrated how important and cost-effective it can be to have a system that uses a pivot format at the heart of an implementation, even when the format is not being exposed to users or external systems. In the context of EDIDOC, this pivot paradigm was applied not only to the messages themselves but also to all the surrounding services (communications, security, and workflow), owing to an object-oriented approach. This has given provision for reusability, scalability, and customizing.

Conclusions

This paper demonstrates that when using a common pivot format, SGML/XML is invaluable for the development and integration of EDI applications. It was illustrated by some operational experiences in the context of the G-EDI, CLASET, and EDIDOC projects.

Reference


Please e-mail your comments to Philippe Vijghen at phv@sgmltech.com.

This paper was first published in the Conference Proceedings of SGML/XML'98 Europe, May 1998, pp 517-22.

© The SGML Technologies Group 1998