

Towards Active Web Clients

Vincent Quint
INRIA Rhône-Alpes
655 avenue de l'Europe
38334 Saint Ismier Cedex, France
vincent.quint@inria.fr

Irène Vatton
INRIA Rhône-Alpes
655 avenue de l'Europe
38334 Saint Ismier Cedex, France
irene.vatton@inria.fr

ABSTRACT

Recent developments of document technologies have strongly impacted the evolution of Web clients over the last fifteen years, but all Web clients have not taken the same advantage of this advance. In particular, mainstream tools have put the emphasis on accessing existing documents to the detriment of a more cooperative usage of the Web. However, in the early days, Web users were able to go beyond browsing and to get more actively involved. This paper presents the main features needed to make Web clients more active and creative tools, by taking advantage of the latest advances of document technology. These features are implemented in Amaya, a user agent that supports several languages from the XML family and integrates seamlessly such complementary functionalities as browsing, editing, publishing, and annotating.

Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: User Interfaces—*Graphical user interfaces, Interaction styles*; I.7 [Document and Text Processing]: Document Preparation—*Languages and systems, Markup languages, Standards, XML*

General Terms

Design, Experimentation

Keywords

Web user agent, authoring, XML documents, compound documents, style languages

1. INTRODUCTION

The Web was originally conceived as a shared, writable, information space [5], i.e. a universe where users can not only consume existing information, but also produce some new information, without any fence between consumers and

producers. This was very clear with the first Web client developed in 1990 by Tim Berners-Lee [4]. It had the ability to simultaneously read and write the Web: it was both a browser and an editor. As an editor it was designed to facilitate the task of creating and updating Web pages. It provided a WYSIWYG style of interface that allowed users to author documents.

Then came Mosaic, which played a key role in popularizing the Web. But, while it made the Web usable by a very large community, it put a strong restriction to the way people could use the Web. It offered only half of the functionality of the original Web client, the browsing feature. It did not provide any help to create, update or publish Web pages. Authoring and browsing became separate tasks, performed with separate tools. The term Web client became a synonym of browser [3].

A number of authoring tools emerged on the Web, ranging from simple text editors for direct manipulation of the HTML syntax, to sophisticated environments with a graphical user interface. Among these tools, structured editors such as HoTMetaL and Grif [18] deserve a special mention. For many Web editors, HTML pages were just plain text that could be freely interspersed with tags to achieve some visual effect. Structured editors considered HTML from a different perspective, much closer to its real nature, i.e. as an SGML application that allows text to be structured according to well defined rules. Doing so, they paved the way to XML.

The cooperative dimension of the Web was developed at the same time, but independently. One approach was to create specific applications that enable cooperation through the Web. The BSCW shared workspace system [2] is a typical example of these applications. Another approach was to tackle the problem at the lowest layer of the architecture. The HTTP protocol was extended with WebDAV to offer an infrastructure that makes resource sharing easier. With these various approaches the emphasis was put on sharing information, but no help was specifically provided to produce and update that information. Wikis address this issue to some extent. They support a very simple mean to edit shared Web pages by allowing users to enter text through Web forms, and using some simple typing convention to structure text.

Another way to cooperate on the Web consists in sharing annotations. Web annotations first appeared in Mosaic under the form of private comments that users associate with the pages they visit. The next step was to store these annotations on servers, thus allowing groups and communities

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng '05, November 2–4, 2005, Bristol, United Kingdom.
Copyright 2005 ACM 1-59593-240-2/05/0011 ...\$5.00.

to share comments. Many Web annotation aware tools or servers have seen the light since then, such as CritLink and ThirdVoice for instance.

Information on the Web is not provided only by humans entering it through some software tool. A large amount is dynamically generated from a variety of sources or extracted from databases. Converters translate various formats to HTML, or something close to it. But while all these methods are available and widely used, there is still a strong need for authoring Web documents.

In this paper we try to reconcile the multiple evolutions reviewed above. We believe that the original idea of an *integrated*, user-friendly tool that gives access simultaneously to the many facets of the Web is worth being explored further, especially in the light of the latest advances in document technology. This is what we do in the rest of this paper. We review the main features that a Web client could support and we consider the benefits that can be drawn from their integration. We illustrate this with the recent evolution of Amaya [21], a Web client that implements the latest Web technologies and standards. Amaya started in 1997 [19] as a testbed and demonstrator for new technologies developed by W3C. Since that time, its original combination of technologies has attracted enough users to motivate a new orientation towards a production tool. Previous papers have already presented some aspects of Amaya (see [21]).

The paper is organized as follows. After this brief introduction, the next section addresses the issue of authoring documents for the Web; it focuses specifically on XML formats and their combination in compound documents. It is followed by a discussion of the issue of style: for information that is intended for human consumption, style is the necessary complement to the content and structure represented by XML. After this part on authoring, the following section addresses the cooperation issue and considers the advantages that can be drawn from a tight integration of authoring with others functionalities. Finally, a conclusion summarizes the contribution of the paper.

2. AUTHORING CONTENT AND STRUCTURE

The most important step toward an advanced Web client that allows a user to play an active role in the Web is certainly the ability to create and update Web pages and other information resources.

2.1 Structured Web pages

With the advent of XML, Web pages can now be considered with the structured approach. The era of “HTML tag soup” is over. To populate the Web with meaningful and useful information, we should now produce rigorously structured documents. The first step in this direction is to move from crappy HTML to clean XHTML [1]. This is a smooth transition, as XHTML provides the same functionality as HTML, but it brings all the advantages of XML, while remaining compatible with legacy browsers. A modern Web client must clearly allow users to produce XHTML pages.

XHTML pages are structured documents. Even if some pages involve only a simple sequence of paragraphs, headings and images, many other pages include deeper structures such as nested lists or tables. For this reason XHTML pages must be handled as structured objects. Fortunately, previ-

ous research brought techniques for editing structured documents [17] and some Web clients have adopted this approach for editing XHTML. Mozilla [16], for instance, includes an editing tool, Composer, that works that way. Nvu follows the same way. This is also the approach we have taken in designing Amaya.

An important issue with structured editing is the user interface. Manipulating a possibly complex structure is not often something that can be done in an intuitive way. In particular, following too closely the intrinsics of XML structures does not lead to a user friendly interface. An XHTML page, for instance, is not simply a tree assembling named elements which may have attributes. Authors have a more “semantic” representation of their documents. They do not consider nested lists and tables the same way, for instance, although both types of objects are represented by very similar XML structures. Therefore a specific user interface has to be provided for XHTML, that allows users to interact with the editor according to their own representation of the document. Specific commands should be available for editing rows, cells and columns in tables, that are different from commands for editing lists and their items. Amaya follows this approach. While some generic commands allow authors to manipulate all types of elements the same way, additional commands are dedicated to the most complex structures. Details on the user interface for structure manipulation can be found in [20].

Structure complexity is not the only reason for providing specific editing commands. The main motivation is rather the gap between the cognitive model of an object for the author and its realization in an XML structure. Hypertext links constitute a typical example. For the user a link is an association between some location in a document (the source) and some other location in the same or another document (the destination). In XML terms, this association is represented by elements and attributes. The source is an element called **a** (anchor), the destination is another **a** element or any other type of element with an **id** attribute; the association is represented by the value of the **href** attribute of the source anchor which contains an URI composed of the address of the destination document and the value of the **id** attribute of the destination element, plus some cryptic characters. Instead of asking users to create these elements with their attributes and to type the URI, it is certainly better to provide a higher level command that allows them to simply select the source and the destination with a pointing device; the tool makes then the association: it computes the URI, creates the appropriate elements and attaches the relevant attributes with the right value. Given the key role of hypertext links in XHTML and in the Web in general, such a command is a must for a user friendly Web client. It is worth noting that most Web authoring tools ask users to type URIs, making the process cumbersome and error-prone.

Another example of a high-level editing command is the automatic generation of a table of contents. This command finds all headings (**h1**, **h2**, etc.) in a XHTML document, copy their content in document order, and inserts it as a nested list at a location chosen by the author. It also turns existing headings into destinations and establishes links between the generated copies and the original headings. The result is a clickable table of contents that can be inserted very simply anywhere the author feels it appropriate.



Figure 1: Synchronized views of an XHTML document

While it is important to provide the author with convenient commands for creating or modifying the document structure, this is not enough. The user should also be able to comprehend the existing structure and to check the changes s/he is making to it. For that purpose, the concept of *views*, following the model-view-controller (MVC) paradigm [14], may be very useful. The idea is that several graphical representations of the document, with different points of view, help the user to perceive the various aspects of a complex structure. An author may then choose the view that s/he feels the most appropriate for performing each editing task.

This concept is implemented in Amaya which supports different views (see Figure 1). The *formatted* view (top of Figure 1) displays the document the way most Web browsers would display it. It is actually the view that Amaya opens by default when opening a document. The *structure* view (bottom of Figure 1) shows the DOM tree and thus gives access to the actual structure of the document. It may be used to manipulate the document according to its hierarchical structure. The *source* view (not shown on the figure) displays the document representation in the XML syntax. If the user is knowledgeable in XML, s/he can edit the source code as plain text there. As there are different ways to render an XHTML page, the formatted view is complemented with the *alternate* view, which displays the document with a very simple layout, similar to the way a text browser such as Lynx [15] would render it. This is helpful to anticipate the problems a user may have if s/he is visually impaired or when reading the document with a less powerful device. To present a page through its hypertext aspect, the *links* view collects all source anchors of the document with the full URI of their destination. The last view available is the *table of contents*, which shows an outline of the page by displaying only headings (**h1**, **h2**, etc.). With this outline an author can grasp the organization of a long document in the blink of an eye.

Following the MVC model [14], all open views are “synchronized”: the result of any editing action performed in one view is also visible in the other open views. This includes selecting. Clicking in any view lets the user see the same part of the document in the other views. This is useful, for instance, to move around in a long document through the table of contents view, or to check the source code cor-

responding to an element selected in the formatted view, or to locate the source of a link to a given URI noticed in the links view. Experience has shown that multiple synchronized views are a great benefit to authors when manipulating complex structured documents.

Amaya not only supports several views, but it can also edit several documents simultaneously, and several views for each document. This is important when splitting a document or when merging several documents. This is also useful for copying or moving contents across documents. Obviously, the edited documents are independent from each other: they may come from different Web servers and can be saved on different servers.

Another important advantage of the structured approach is that it allows the editing tool to closely follow the definition of the document type. The editor manipulates an internal representation of documents, a DOM tree, that is built rigorously according to the document type. The external representation, the XML file, can be generated directly from that representation. As a result, documents are always well-formed (in the XML sense) and valid. They are compact and do not contain extra code, as it is often the case with documents produced by tools based on a different model.

Regarding structured editing, Amaya is significantly different from other (X)HTML editors. Many of them (Composer, Nvu, FrontPage, Dreamweaver, etc.) allow an author to manipulate the document structure, but very few of them go as far as Amaya in that direction. The generation of the table of contents or the complex structure transformations described in [20] are typical examples. Views are also available in other tools (in general, formatted view, source code and DOM tree), but the user can often see only one view at a time; when two can be displayed, only one is editable, and lacking an efficient synchronization mechanism, multiple views are not as useful as they could be. Finally, as opposed to FrontPage or Dreamweaver for instance, the code generated by Amaya is always valid and well-formed. This is the best way to make sure that documents will be readable with any browser.

2.2 XML applications

An important reason for moving from legacy HTML to XHTML is to take full advantage of the XML technology. In particular, when XML is supported other document types may be used. In addition to XHTML, many XML applications have been developed for the Web, which involve more complex structures. Let us consider two of them which are of particular interest for Web pages in the scientific community, MathML [9] and SVG [10]. MathML was designed to represent mathematical expressions on the Web while SVG is dedicated to structured, animated graphics. To allow authors to create and edit such objects, it is important to provide convenient tools. Obviously the converter approach can be used here, which makes many existing tools usable for producing objects in these formats. But many important Web aspects of these languages are not supported by tools that have not been designed specifically for them. MathML and SVG use hypertext links and CSS style sheets for instance, as many other Web formats do. To benefit from the advances of the latest features introduced in recent Web formats, tools that support these formats natively are necessary.

As languages such as MathML and SVG are built on the same basis as XHTML, it would make sense to handle mathematical expressions and structured graphics with the same kind of tools and techniques as XHTML Web pages. As for XHTML, a pure XML approach would not work. In graphics and mathematics, structures may be very complex and the author's representation is often very different from the DOM tree and its XML encoding. In SVG for instance, a geometric shape (a rectangle, an ellipse, a curve, etc.) is represented by a single XML element with attributes which contain the coordinates of the control points. This is not something a user would like to type in. An interface similar to the one offered by usual drawing programs is much more acceptable: the user draws the shape with a pointing device and the system creates the element that represents it and computes the attribute values from the positions clicked by the user.

The same applies to mathematics. The MathML structure is indeed fairly detailed and even a very simple mathematical expression may involve a number of elements and attributes that authors are reluctant to provide explicitly. For instance, an expression such as $y = 2x + n$ is made of seven elements: three identifiers (y , x , and n), a number (2), two operators (= and +), and an invisible operator (times) between 2 and x . An authoring tool for MathML should be able to generate this structure automatically.

An efficient method to achieve this is implemented in Amaya. It consists in parsing the character strings entered by the user. To enter the above expression, an author just hits six keys ($y=2x+n$). The character string entered that way is then parsed, characters are classified (identifiers, numbers, operators, etc.) and the appropriate MathML elements are generated according to this classification. It is only when the linear sequence is interrupted that constructs such as superscripts, fractions, integrals, roots or matrices need to be explicitly entered by the author. This is done through short control sequences, in order to allow the user to keep his/her hands on the keyboard. In addition, in regular structures like matrices, new repeatable elements (cells, columns and rows) can be added by a single key stroke [20].

A few additional features help users to manipulate complex mathematical structures. Structure navigation through the usual arrow keys is such a feature. Structure transformation is another one. Authors can select any part of an existing expression and turn it into an element of a different type. For instance, in the expression above, when the sequence of elements $2x+n$ is selected, typing Ctrl+F makes it a fraction: a `mfrac` element (fraction in MathML) is created with two sub-elements. The first one represents the numerator and contains the selected elements, while the second, the denominator, is left empty. Such structure transformations are very useful [6]. They allow authors to enter mathematical expressions quickly and to go back and change any part that was not entered correctly in the first place. This is also what makes automatic structuring usable, as in some cases the structure produced automatically could not be exactly what the author expected. In these cases, restructuring commands can be used to fix the structure.

With this kind of user interface, it is no more complicated to edit highly structured objects than using a text processor. But without it, structure editing is very cumbersome, and average Web users would be reluctant to use the latest Web formats.

However, there is an exception to this principle in Amaya, to allow authors to edit XML structures that are not known to the editor. These structures are called *generic XML*. The interface presented above is indeed made possible because the editor "knows" the document type. It includes specific code and user commands that handle XML languages according to their semantics. With generic XML, this is not possible, but it is worth accepting other XML languages than those known to the editor, even if it is a bit more complicated to manipulate them. At least a few minor modifications could be done if needed, in particular when generic XML is part of compound documents that also use languages natively supported by the editor.

2.3 Compound documents

Compound documents are XML documents that mix several XML languages. Thanks to the XML namespace mechanism [8], a single XML document may contain parts that are structured according to different XML applications. A XHTML document that includes SVG graphics and MathML expressions is a typical example of a compound document. This XML feature is something important for Web users. It opens up new possibilities for complex documents. It is no longer mandatory to divide a document into a number of different resources that are integrated by transclusion. With namespaces, a single resource can contain everything in a consistent structure. There is no need, for instance, to make a separate file for each drawing or formula in a document. These objects can be part of the document itself.

To create and edit such composite documents, several approaches could be taken. One is to develop a separate tool for each type of object. While this could be considered for instance for graphics, it does not work well for mathematics. In a document that contains several mathematical expressions in each line of text, it is not acceptable to switch tool so often. Plugins provide an alternative. When using plugins, several tools are still involved, but an author sees them as integrated because they share the same display space. However, plugins pose problems. The user interface lacks consistency when switching from one tool to the other. Integrating the graphical representation of different objects handled by different tools is problematic too: the programming interface between tools is usually not sufficient to properly handle inheritance of style properties and to propagate formatting constraints from one tool to the other. With existing plugin interfaces it is impossible to correctly align mathematical expressions processed by a specific tool with the surrounding text handled by another tool.

For these reasons we prefer a different approach, where priority is given to integration and consistency. It follows the principle of XML namespaces. A compound document is represented by a single DOM tree and editing functions operate on all parts of the tree, whatever the type of object they represent. Formatting is performed the same way: text is formatted and displayed by the same functions, be it the content of a paragraph, a mathematical variable or a label within a drawing.

This approach is implemented in Amaya. As we have seen above, the editor is specialized for several XML applications and provides higher level operations for manipulating different kinds of objects. That is the reason why Amaya's architecture is made of a kernel that performs basic functions, plus extensions that implement the specifics of differ-

ent kinds of objects. The kernel and extensions work on the same data structures: the DOM tree and the formatted representation of each view. Obviously, specialized functions are performed only on the parts of the DOM tree that represent the objects they are supposed to manipulate. But these specialized functions may also access and update any information they need about the context of the data they manipulate. This solves the problems mentioned above regarding plugins.

A consequence of this approach is that the same system is used to manipulate all parts of a compound document. This allows users to evolve freely in a continuous space, to manipulate simultaneously a structured object and the surrounding structure, to access the various facets of a compound document in a consistent way. Hypertext links, for instance, are edited in exactly the same way when their source and destination are located in graphics, mathematics or text. That is the same for style: changing the color, size or font of characters is done through the same commands in all parts of a compound document and inheritance of style properties across the DOM tree works as expected.

Another advantage of this approach is that it does not put any limitation to the nesting of various XML languages. A MathML expression or a piece of XHTML text can be inserted and edited within an SVG figure, even when this figure is itself included in an XHTML page. That way, authors can fully benefit from the advantages of compound XML documents.

As compared with the most popular Web authoring tools, this feature of Amaya is unique. Usual tools can only edit XHTML. They allow “foreign” objects such as SVG drawings or MathML expressions to be included into Web pages, but these objects have to be edited separately, with different tools.

3. EDITING STYLE SHEETS

XML introduces a clear separation between content and structure on one side and style and presentation on the other. HTML has benefitted from this separation by moving to XHTML. The strict profile of the XHTML specification contains only elements and attributes that represent the document structure. Presentation-oriented elements and attributes have been removed from the language and are replaced by style information associated with the document through style sheets.

In a client intended to handle documents on the Web, it is then important to provide a style editing feature, in addition to the structure editing feature. Even if style is separated from structure in the document format, authors often manipulate both aspects of documents simultaneously. Two style languages have been designed for XML: XSL and CSS. In this article we only consider CSS [7].

CSS style has many aspects. First, there are style *properties* (`color`, `font-size`, `margin`, etc.) whose *values* determine different aspects of the presentation of elements in a document. Then, there are mechanisms for associating properties and their values with the elements of a document. In CSS this is done through *selectors* which indicate in what structural context a property and its value are applied to an element. At this point *inheritance* enters into play: most properties that are associated with an element are inherited by its descendants in the XML structure. With inheritance, if a section is red, the paragraphs it contains are red too, as

well as the character strings contained in these paragraphs, unless different rules are associated with these sub-elements to set different colors.

A selector, a property and a value make a rule. Rules are collected in style sheets, which give a consistent look to a whole document. These style sheets are independent, sharable Web resources associated to documents by various ways (user preferences, links from the document, Processing Instructions). The same style sheet can be used by several documents and several style sheets may be associated with a single document. The combination of multiple style sheets for a single document is called *cascading*. When several conflicting style rules from the same or different style sheets are candidates for the same element, well defined priority rules select only one of the conflicting rules in the cascade.

Cascading is a way of combining style sheets. Inclusion is another, different way. A piece of CSS style can be stored in a Web resource to be shared between several style sheets. These style sheets include the shared resource which is then considered as if it was part of the including sheet.

All these aspects of CSS offer many possibilities to a style designer, but they also make his/her task complex. Complexity does not lie in the CSS syntax, which is very simple, but in the combinatorial aspect of style (multiple sources of style, inheritance, cascading, inclusion). This is where an authoring tool for the Web should provide assistance. In Amaya, this assistance is provided at three different levels: style rule editing, style sheet manipulation, and debugging.

3.1 Editing style rules

Style may be entered with Amaya either through a graphical interface or by typing the CSS syntax directly. The graphical interface is similar to the one of a word processor. The user selects an element in a document and assigns it some style through a dialog box. This attaches style properties with values to the selected element. When the element has the desired look, the user can create a full rule by entering the selector that is then combined with the style properties of the element. The rule is then applied to all elements of the document that match the selector. The user can thus easily check whether the rule works as expected. In a XHTML document, the rule is finally put in the `style` element of the document (the internal style sheet), where it can still be edited further.

3.2 Manipulating style sheets

Style sheets are external resources which affect the presentation of a document. When focusing on document presentation, it is important to be able to access, consult, modify, create, and link style sheets. For this purpose, Amaya displays CSS style sheets as separate documents (except the internal style sheet) that can be edited as plain text. It also provides commands that perform several manipulations on style sheets.

The user can see the list of all style sheets involved in the presentation of a document and s/he can freely open any of them. The list of style sheets referred by a document can also be used to deactivate or reactivate selected style sheets. Deactivating a style sheet consists in showing how the document would look if the style sheet would not be associated with that document. When several style sheets apply to a document, this is helpful to understand how each one affects the final presentation.

To act on the document style, it is also important to be able to open style sheets that are not linked to a document. This allows authors to inspect other style sheets and decide which ones could be associated with the document. When linking a style sheet to a document the user immediately sees how the document looks with this style sheet, and s/he can safely decide whether to keep the link or not. If the desired style sheet is not available on the Web, the author may choose to create a new one and to enter its content. Another way to get a new style sheet is to export the content of the `style` element of an XHTML document and to make it a separate style sheet.

When a style sheet is open, its content can be edited as plain text. Given the simplicity of the CSS syntax, this is enough for many of the usual changes one makes to style sheets and it is often easier than using a convoluted graphical interface. When the changes are made, the style sheet can be saved or simply “synchronized”. In both cases it is applied to all open documents that use it and the user can see whether the latest changes produce the expected result. Syntactic errors, if any, are also reported.

3.3 Debugging style sheets

Looking at the document is a way to check whether the result is correct or not. But if it is not, the issue is the same as in programming: it may be difficult to find the faulty rule (statement), especially when a number of included and cascaded style sheets are used by the document. Some kind of debugging aid is required.

One method consists in deactivating style sheets one after the other to locate the questionable style sheet. But this may be tedious when there are many style sheets and it is not precise enough: the exact rule has still to be found in the sheet.

To locate a rule more precisely, another method is available. It is based on a single command (Show applied rules) that opens a small window (see Figure 2) showing all the rules that are actually applied to the selected element itself, i.e. it does not include properties that are inherited from ancestor elements. When the current selection changes, this window is updated accordingly. Each line in this window displays a rule applied to the selected element: the name of the property, its value, and the URI of the style sheet it belongs to. Each line is actually a link to the rule in its style sheet. Clicking it opens the style sheet if it is not already on the screen and highlights the rule that was applied to the element.

When an unexpected look is visually detected on an element, the user asks for the applied rules. If the property that has the wrong value is not listed, this is because that property is inherited from some ancestor element. The user then selects the parent element (a single key stroke is enough to move to the parent) several times until the desired property appears. Then, clicking it shows the faulty rule in its style sheet. With that information, the bug can be fixed very easily by editing the rule and the result can be instantly checked by synchronizing the style sheet.

This debugging technique has been extensively used for developing style sheets with Amaya. It lets the system handle the complexity of inheritance and multiple, cascading style sheets with inclusions. It was considered to be important enough to change priority in the development schedule. Indeed, it quickly appeared more important to develop this

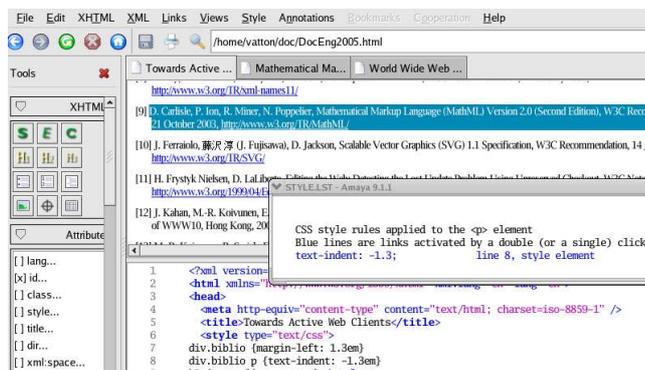


Figure 2: Debugging a CSS style sheet

feature than creating a graphical user interface for every CSS property.

Several Web authoring tools provide a CSS editing function. Amaya contrasts with these tools through two main features: integrated browsing and synchronization. The browsing function of Amaya allows remote style sheets to be freely searched, tested, edited and linked to a document under editing. Synchronization makes debugging possible. Composer and Dreamweaver, for instance, provide a DOM (or CSS) inspector, which only allows an author to see what style property is associated with an element, but does not relate the property with the style sheet it comes from. Amaya goes further by maintaining this relationship, even if it involves a remote sheet.

4. INTEGRATION AND COOPERATION

Editing documents, their structure and their style is only one of the services expected from a Web client. To efficiently support the multiple tasks a user performs on the Web and to take into account the multiple facets of the Web, more features are needed. In this section, we review such important features as browsing, publishing, annotating, and bookmarking. We will see that their combination and their integration provides more benefits than all of them considered separately.

4.1 Browsing and editing

When browsing is added to an editor, it helps setting the context of the document being edited. Browsing becomes part of the authoring environment and extends it. It allows an author to locate the resources to be included in the document: images, objects, pieces of structured text, etc. It is also used to get the resources that will be linked to the document: other, remote Web pages, style sheets, transcluded objects, etc. By downloading these resources first, the author can check them before linking. This method is much more secure than typing a blind URI, what most Web authoring tool require. It is also easier: to set the link a simple click on the resource displayed on the screen is enough (see section 2.1)

Downloading remote resources in the editor environment is also helpful for borrowing material from existing documents. With a browsing function integrated in the editor, any Web page can be downloaded and then any part of it can be copied and pasted. With structured document this is especially interesting, as the editor can paste a piece of

DOM tree and then all the details of the structure. When the pasted structure contains Web links, the corresponding URIs, if they are relative, can be modified by the editor to refer to the same resource from a different place.

The relationship between editing and browsing can also be considered in the opposite direction. Authoring is then considered as a service that complements a browser. When reading a Web page a user may notice a typo, some obsolete information, a dangling link, something that is missing or would need to be changed. It is then very useful to be able to make the change immediately, just by editing the page that the browser is displaying, without starting another tool.

This is done in Amaya. We have seen that the formatted view of a document is the same as a browser view. In addition, Amaya provides the usual browsing commands: clicking a link downloads the destination document in the current tab, in a new tab, or in a new window; commands Back and Forward allow the user to navigate the browsing history, etc. Amaya can be used to surf the Web and access local documents in the same way as any browser. Multiple documents, images, style sheets, etc. can be open simultaneously, and all these resources are under the control of an editor. Then any editing operation can be performed on them and every combination of browsing and editing is made possible. This is a key feature that is missing in all Web authoring tools we know.

4.2 Publishing

When remote resources are downloaded and modified, it is necessary to be able to save them back to a remote server. Therefore publishing is a required feature in an advanced Web client. The minimum functionality is saving an edited resource to the location it comes from. If it is a local file, it should be possible to save it in the same file. If it is a remote resource, it should be possible to save it to its Web server. But it should also be possible to create a new document or style sheet locally and to store it on a remote server, or to download a resource from a server and to save it on another server. Local files could be saved remotely, and conversely. In fact, the publishing feature is quite complex if all possible moves are considered, but this is required to allow users to take advantage of the distributed nature of the Web: many users have to create and update resources on several servers, especially when working collaboratively.

Additional complexity comes from the fact that some Web resources depend on other resources, and when moving one of these resources, some others may have to be moved accordingly. A usual case is an XHTML page that includes images. When such a page is copied or moved from one location (local or remote) to another location, it may make sense to copy the associated images too, or at least some of them.

Publishing is more than saving a file and some associated resources. It may also require the URIs used in the saved document to be updated. This is like the Paste command mentioned above: relative URIs have to be changed if the document is saved to a different location. Obviously this is not something that an author wants to do “by hand”. The application has to do it.

There are other changes that should be made to the document at publication time. Adding or updating a timestamp at a position chosen by the author is something that may be useful for an evolving document.

The publishing feature of Amaya implements these requirements. It is based on the HTTP protocol. While all Web editors use FTP to save files remotely, HTTP was preferred for Amaya because it provides greater control over the information that is transmitted to the server, thanks to the HTTP header. Thus, Amaya includes full support for HTTP 1.1. Browsing uses the Get method, and publishing is based on the Put method. This symmetry makes sure that, when a document has been changed by the editor, it can be moved from one server to another while preserving all the meta-information associated to it: MIME type, character encoding, etc.

With this publishing feature, an author really gets the feeling that s/he is editing the Web. Working on remote documents is exactly the same as working with local files. A single click on the Save icon immediately updates the document on its server. Creating a new document on a remote server just requires the author to enter the address of its location. Obviously, to write a file remotely, the user must be allowed to do so, and an authentication process is involved the first time a file is saved on a remote server during a session. The Web server has to support the HTTP Put method (but a conformant server is supposed to provide full support of the HTTP protocol) and to be configured for allowing authorized users to write in some of its areas. Also, to make sure the changes saved by one user are not lost when another user saves his/her own changes made to an earlier version of the document, the lost update feature of HTTP [11] is mandatory.

This publishing feature makes Amaya very different from usual Web authoring tools. For editing remote documents, there is no need to make a local copy of the remote site first, nor to save this copy back when all the changes are made. The user can change and save any document independently of any other, and the tool takes care of keeping things consistent. This also makes Amaya usable for cooperative editing of remote documents, just by using its publishing feature.

4.3 Cooperation and annotations

The combined use of browsing, editing and publishing in a single, consistent software tool provides a strong basis for cooperative work on the Web. Documents can be shared and accessed remotely in read/write mode by several users. HTTP 1.1 with the lost update feature handles writing conflicts but it only allows an user to be warned that his/her changes will overwrite another user's update. A more advanced way to handle conflicts is to prevent them from occurring using WebDAV: an author can temporarily lock a resource under modification to avoid other users to access it in write mode.

While these basic functionalities ease cooperation, they are not sufficient for effectively supporting the cooperative efforts of a distributed group of users. For such a group, it is important not only to share documents, but also to discuss these documents. People need to make comments, to propose changes, to submit alternative content, etc. Obviously this could be done by email, but with severe limitations. When the discussion is about specific parts of a shared document, unambiguous references have to be made to the document. When pieces of content are submitted, they should follow the document structure. All these dependencies to the document require a more suitable mechanism, which annotations can provide.

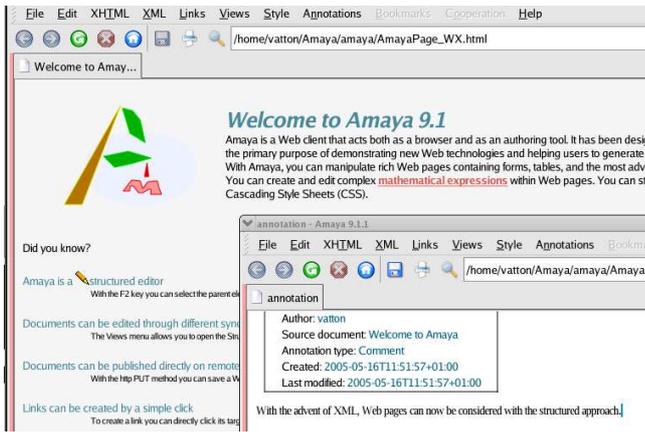


Figure 3: Creating an annotation

In its simplest form, an annotation is a chunk of information that is attached to a specific part of a document. For supporting cooperative work, annotations should be shared within the group interested in a document. Different types of annotations can be made by different users. Each user may want to see only a selected subset of these annotations, depending on the task being performed. A filtering mechanism is then needed that could select annotations according to different criteria: date, author, type, etc. Users should be free to annotate any part of any document and to put any kind of information in annotations.

Adding this functionality to a Web client that can edit and publish complex documents enhances its collaborative features. It has been done in Amaya by introducing support for Annotea [12], a shared annotation system based on a general-purpose open RDF infrastructure. Actually, Annotea was first developed in Amaya, taking advantage of its rich set of features, before being implemented in other Web tools. In Annotea, annotations are external to the documents and can be stored in one or more annotation servers to be shared by several users. Thanks to XPointer, annotations can be attached to any part of any XML document without making any change to the annotated document. In its Amaya implementation, the merging of annotations with documents takes place within the client. An icon is temporarily inserted in the document at each position where an annotation is attached (see the pencil icon in Figure 3). This occurs after filtering: only the annotations that are currently of interest to the user are shown. The user is then free to click any of these icons to display the corresponding annotation.

Annotations are first class Web resources. Each annotation is associated with a URI and may contain anything a Web resource may contain. They also have RDF metadata that are used to search and filter them. In the Amaya implementation the full editing feature is used to edit the content of an annotation. Annotations could then contain a (usually short) compound document with drawings, mathematical expressions, hypertext links, style, etc. This gives users a great flexibility in annotating Web resources.

Bookmarks are a well-known service provided by most Web clients. With the availability of Annotea in Amaya, this usual service was significantly extended. By considering bookmarks as a special case of annotations, shared book-

marks [13] have been introduced. It is a tool that permits users to informally classify Web resources and share these classifications in the same way annotations are shared.

5. CONCLUSION

In this paper we advocate an extended functionality of Web clients. A dynamic and rich information space such as the Web deserves powerful user agents that enable a more creative activity, allowing users not only to consume existing information but also to produce new information and to interact with other users.

For this purpose, a Web client should provide more services than the usual browsing functionality. In particular, editing is required. It allows users to create and update documents and to feed the Web with new information. But editing needs to be combined with browsing and publishing, to make users able to work directly on the Web, without any intermediary, local copies, complex transfers, etc. To take advantage of the latest developments of document technology, the editing feature must be able to manipulate compound XML documents and their associated style sheets. As a complement to editing, an annotation system is needed to efficiently support cooperative work through the Web.

In our experience with Amaya, tight integration of all these features is crucial. From the user perspective, working with the Web is a single task, which involves simultaneously editing, styling, browsing, publishing, annotating, bookmarking, etc. Web clients are tools that are supposed to support this task. Therefore, they should provide a uniform, homogeneous environment where all actions can be performed smoothly and consistently, without any fence. However, it should be acknowledged that the development effort required for such an integrated environment is significant. Implementing many different features is obviously costly, but integration allows features to share a large part of the software and thus avoid duplicated efforts. Actually, the main costs result from the implementation of multiple languages and standards that are evolving. Most of the languages available in Amaya are quite rich, and a full implementation of each of them requires a lot of effort.

In its current version Amaya implements all the features presented above, but some features of SVG, MathML and CSS are not yet available due to the effort needed. Future work will allow to increase the coverage of the technologies supported. In particular, advanced editing commands for SVG are planned, as well as support for entering mathematical symbols in MathML expressions more easily. Regarding style, the user interface will be extended to avoid users having to type the syntax of some CSS rules. Further work is also planned for enhancing the publication feature by taking into account more resources related to a document.

6. ACKNOWLEDGEMENTS

We are grateful to W3C for their support and contribution to the development and distribution of Amaya. We also acknowledge the valuable participation of a number of people in the work presented in this paper. In particular, we thank Laurent Carcone, Paul Cheyrou-Lagrèze, Pierre Genevès, Ramzi Guétari, Stéphane Gully, Jose Kahan, and Daniel Veillard. The whole Amaya community is acknowledged for their valuable contribution to the evolution of the software.

7. REFERENCES

- [1] M. Altheim and S. McCarron. Xhtml 1.1 - module-based xhtml. Technical report, W3C Recommendation, <http://www.w3.org/TR/xhtml11/>, 31 May 2001.
- [2] R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkell, J. Trevor, and G. Woetzel. Basic support for cooperative work on the world wide web. *International Journal of Human Computer Studies*, (46):827–846, 1997.
- [3] H. Berghel. The client's side of the world-wide web. *Commun. ACM*, 39(1):30–40, 1996.
- [4] T. Berners-Lee. The worldwideweb browser. Technical report, W3C, <http://www.w3.org/People/Berners-Lee/WorldWideWeb.html>.
- [5] T. Berners-Lee and M. Fischetti. *Weaving the Web*. Harper, San Francisco, 1999.
- [6] S. Bonhomme and C. Roisin. Interactively restructuring html documents. *Computer Networks and ISDN Systems*, 28(7-11):1075–1084, 1996.
- [7] B. Bos, H. Lie, C. Lilley, and I. Jacobs. Cascading style sheets, level 2. Technical report, W3C Recommendation, <http://www.w3.org/TR/CSS2/>, 12 May 1998.
- [8] T. Bray, D. Hollander, A. Layman, and R. Tobin. Namespaces in xml 1.1. Technical report, W3C Recommendation, <http://www.w3.org/TR/xml-names11/>, 4 February 2004.
- [9] D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical markup language (mathml) version 2.0 (second edition). Technical report, W3C Recommendation, <http://www.w3.org/TR/MathML/>, 21 October 2003.
- [10] J. Ferraiolo, J. Fujisawa, and D. Jackson. Scalable vector graphics (svg) 1.1 specification. Technical report, W3C Recommendation, <http://www.w3.org/TR/SVG/>, 14 January 2003.
- [11] H. Frystyk Nielsen and D. LaLiberte. Editing the web: Detecting the lost update problem using unreserved checkout. Technical report, W3C Note, <http://www.w3.org/1999/04/Editing/>, May 1999.
- [12] J. Kahan, M.-R. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An open rdf infrastructure for shared web annotations. In *Proceedings of WWW10, Hong Kong*, <http://www.w3.org/2001/Annotea/Papers/www10/annotea-www10.html>, May 2001.
- [13] M.-R. Koivunen, R. Swick, and E. Prud'hommeaux. Annotea shared bookmarks. In U. Sanibel, Florida, editor, *Proc. of the KCAP 2003 workshop on knowledge markup and semantic annotation*, <http://www.w3.org/2001/Annotea/Papers/KCAP03/annotaebm.html>, 25-26 October 2003.
- [14] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):2649, 1988.
- [15] *Lynx*. <http://lynx.browser.org/>.
- [16] *Mozilla*. <http://www.mozilla.org/>.
- [17] V. Quint. Systems for the manipulation of structured documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, pages 39–74. Cambridge University Press, 1989.
- [18] V. Quint, C. Roisin, and I. Vatton. A structured authoring environment for the world-wide web. *Computer Networks and ISDN Systems*, 27(6):831–840, April 1995.
- [19] V. Quint and I. Vatton. An introduction to amaya. *World Wide Web Journal*, 2(2):39–46, 1997.
- [20] V. Quint and I. Vatton. Techniques for authoring complex xml documents. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, pages 115–123, 28-30 October 2004.
- [21] I. Vatton. *Welcome to Amaya*. <http://www.w3.org/Amaya/>.