A Complementary Approach for Adaptive and Adaptable Hypermedia: Intensional Hypertext

W.W. Wadge[1] and m.c. schraefel[2]

1 Department of Computer Science, University of Victoria, Victoria, B.C., Canada
wwadge@csr.uvic.ca
2 Department of Computer Science, University of Toronto, Toronto, ON, Canada
mc@cs.toronto.edu

Dear AHH01 Reviewers,

The attached paper describes the intensional approach to versionable hypertext, whether that text is driven by an adaptable or adaptive interaction.

The paper describes the current implementation to support this model-neutral approach to authoring a front end for versionable interaction.

Consequently, though it's a fast read, and even though there's no page limit for the workshop, the paper may be a tad long in terms of specific detail. We hope this is ok for the workshop for a couple of reasons. Intensional Hypertext is a new-ish approach in this context. We hope by this paper to provide the reviewers and the workshop participants with as much info in this space as possible for good discussion and your feedback.

Thank you for your consideration. Should our paper be accepted, we'll of course be more than happy to pare back for the printed proceedings as required.

Thanks again.

monica schraefel
Bill Wadge

A Complementary Approach for Adaptive and Adaptable Hypermedia: Intensional Hypertext

W.W. Wadge[1] and m.c. schraefel[2]

1 Department of Computer Science, University of Victoria, Victoria, B.C., Canada
wwadge@csr.uvic.ca
2 Department of Computer Science, University of Toronto, Toronto, ON, Canada
mc@cs.toronto.edu

Abstract. In this paper describe a methodology and an authoring/publishing tool for adaptable Web documents (user-determined adaptable Web pages) as a complementary approach to Adaptive Hypermedia. Our approach is based on intensional logic, the logic of assertions and expressions, which vary over a collection of contexts or possible worlds. In our approach the contexts are sets of values for parameters which specify the current user profile as supplied by the current Web page URL, and the latest user input. The author produces generic (multi-version) source in the form of HTML with extra markup delimiting parts that are sensitive (in various ways) to the parameters. This source (in what we call Intensional Markup Language) is translated into a Perl-like language called ISE (Intensional Sequential Evaluator). To generate the appropriately adapted individual pages, the server runs the ISE program in the appropriate context. The program produces HTML that, when displayed in the user's browser, is rendered into the desired adaptation of the requested page. Although this intensional approach was originally designed to work without any explicit user model, we can extend it (and make the documents adaptive as well as adaptable) simply by incorporating a user model that monitors the user and computes some of the profile parameters.

1 Introduction

By the implementation of a user model, adaptive hypermedia systems select the appropriate components of a hypertext space to serve to their users. Such systems depend upon some kind of evaluation of an actual user against a user profile in the model [3]. This means that users need to identify themselves to the system in terms of traits the system will recognize (age, grade in a course, sex, favorite TV program, performance on an online test, etc) before the system can present them with the appropriate information. To gain this user profile data, the system must either watch users interact with the system [6], or ask the user for a variety of information either before the user proceeds into the site or regularly evaluate the user [13] during visitation. A combination of these methods can also be used. In each case, the adaptive system will select which links a user may have available, which content the user is best suited to see, and so on [1].

In 1994, Eco referred to the differences between Mac and DOS-based machines (and their users) as the differences between Catholic and Protestant approaches to salvation: in the one, everyone is saved by Grace; in the other it is only by one's own work that one reaches heaven.[1] In hypertext, the web itself in its current graceless state may be seen as the Protestant, Calvinist analog, which implies only by dint of one's own surfing and searching may one reach the Journey's End. The above-described user model-based, adaptive approach to customized documents of select and occluded links, however, sounds remarkably reminiscent of the medieval Catholic Church, with its list of banned books, which only elite liturgists were allowed to access, or its sacred texts, which were interpreted for the common folk in Mystery Plays rather than through direct reading of the sources.

If that is the case, then what we present in the following paper may be seen as perhaps an Anglican middle ground between the protectiveness of adaptive hypermedia and the cold indifference of the raw web. That middle ground, open to user models, domain-models, adaptable and raw web hypertext is IML-authored, Intensional Hypertext. In [12] we presented an overview of Intensional Hypertext as a class of adaptable hypertext which allows the user to determine at any time which version of a document they wish to see. The previous intensional hypertexts created by IML do not require that an explicit user model be managed by the system. Rather, adaptable hypertexts hold the user model implicit and present more explicitly a model of a domain which users refine in their own way (as touched on in [8]). The approach is based on the heuristic that users know what they don't know when they see it or read it, and can, if given the opportunity which adaptable hypertexts afford, refine a document themselves, possibly better than an automated user model can.

There are obviously cases, however, when either a user or a designer may wish to take advantage of the benefits of user-modeled support in a hypertext. Therefore, we have extended IML to act as a general tool which can support either user-selected, user-model selected, or a combination of both through its parameterized, intensional (perhaps agnostic) versioning model.

In the following paper, therefore, we present a more detailed look at the markup language for intensional hypertexts, IML. We will present a general model of an adaptive/adaptable architecture to situate the potential for the intensional approach in particular to adaptable/adaptive hypertext. We will therefore review intensional versioning which underlies our approach and we will describe key attributes available to authors for creating intensional hypertexts, which can function adaptably "as is," or be combined with user-model input, or natural language generated dialogs (as pre [7]) to support an adaptive approach.

[2] and [10] present compelling work on the creation architectures for managing the entire adaptive hypermedia authoring process, from UM to front end. We hope to contribute to this kind of approach by adding a tool for a specific aspect of this process: a kind of model-neutral front end authoring tool, IML, that can manage input from adaptive and adaptable sources, but that specifically embodies an intensional approach to managing these versionable sources. We hope by this presentation to investigate where, through use of tools such as these, we might (a) be able to define a

[1] From Umberto Eco's back-page column, "La bustina di Minerva," in the Italian newsweekly *Espresso* (September 30, 1994).

heuristics for building complementary adaptable, adaptive texts for enhanced user experience and (b) consider how this approach may suit an Open Adapative/Adaptable Hypermedia System.

2 General Adaptive/Adaptable Architecture and the Role of Parameters

The goal of adaptive and adaptable hypertext[2] is of course to make reading easier for the readers; but at the same time it can greatly complicate the task of authors and publishers.

By the publisher we mean the hardware and software (usually, a web server and/or processes that pass information to that server), which produces the currently needed parts of the document, sends these to the reader, and handles reader requests. The publisher of an adaptive document in particular must, by definition [1], build and maintain a user model. Each time the publisher sends another part of a document to the reader, the publisher takes the current state of the reader model into account and modifies the document part accordingly.

In general there are many different possible states (configurations) of a reader model - many different possible reader profiles. In adaptable hypermedia, there are likewise multiple possible instances (or versions) of a domain object, such as the large color view of an image with English captions or the small, black and white one with French captions. The publisher is therefore responsible in each case for delivering a whole family of variants of a document (one for each possible profile or domain instance), rather than one single monolithic document.

Before the publisher can deal with this content, however, the content itself must be generated. Even if natural language processing is brought to bear to synthesize a variety of components for a specific context, such as the HealthDoc work [4], the multiple components to serve that synthesis must still be generated or retrieved from somewhere. Authors of adaptable and adaptive “documents” are responsible for writing a whole family of (admittedly related) documents. Because these documents themselves are hypertexts, this means that the link topology of the document should be subject to adaptation, and not just the contents of individual pages. While this concept of link versioning and multipointing links is familiar in adaptive hypermedia [3] it is not commonly represented in web-based hypertexts, and we are particularly interested in the Web case. We are also interested in making it easy for authors to incorporate combinations of adaptive and adaptable approaches as appropriate.

[2] To be clear, by adaptive hypermedia, we refer to hypermedia systems which rely on a user model to support the delivery of user-specific content. By adaptable, we mean that users select from a variety of parameters to adapt the hypertext to their needs. A stateless version of adaptable hypertext can be created in client-side JavaScript, for instance, in which users can change the color of the background of a page dynamically. IML as a server side process can maintain the state of such a change across any other page selections made on that site.

A (hardly original) way to support these varieties of inputs is to provide the publisher system with some kind of generic meta-document source. The publisher then generates actual parts from this source based on the current user profile, the current page, and the latest user input.

In the following sections of this paper, we describe a methodology and an authoring/publishing tool which can be understood as the result of using intensional logic to formalize these general observations about adaptive/adaptable architecture. We hope by this to provide a general front-end or author-based tool, which will facilitate the authoring of versionable, web-based hypermedia which can then be supported by either adaptive, adaptable or both kinds of inputs.

3 The Intensional Approach

Our approach is based on intensional logic [16], a natural choice for versioning, since it is the logic of assertions and expressions which vary over a collection of contexts or possible worlds. In our approach the contexts are sets of values for parameters which specify the user profile (if one exists), current page, and user input. It is important to note two aspects of this intensional approach which distinguish it from other forms of parameter selection. One is *version refinement* and the other is *best fit.* We describe these attributes and their significance in detail below.

3.1 Intensional Versioning

The intensional (possible-worlds) approach to versioning was originally developed by Wadge and Plaice [11] for use in configuring families of programs from families of components. Most software configuration tools work bottom-up, and allow the user to create a variant of the program by selecting different variants of the components. In [11], by contrast, each variant is described/determined by "version expression" assigning values to parameters. For example, the expression

```
processor:ppc+OS:8+language:french
```

might indicate the french language version for a PPC macintosh running OS 8.

A particular version of a program is configured, as usual, by assembling particular versions of components. In intensional versioning, however, the component choices are not arbitrary; instead, they are determined by the expession for the version requested. For this to work it is necessary that each variant of each component be labeled with a version expression. In the simplest case, when configuring version V of a program, we use version V of each component. However in general we do not require that each component have a version whose label is exactly V. If there is no such component, we are allowed to choose a component whose label is strictly more generic than V, in that it omits some of the parameter values prescribed by V.

For example, suppose that we wanted the `French PPC OS 8` version described above. It may be that there is no version of the (say) windowing component which has exactly that label; but that there is one with the label `processor:ppc+OS:8`.

Then we can use this component, the justification being that the windowing component is language independent.

In general, although we do not require that there be a component labelled V, we insist that there among the more generic alternatives there is one which is best in that it is the most specific. If there is a most specific alternative component, that one must be taken. If there is no such best alternative, the configuration attempt is abandoned.

The net effect of the best-generic-alternative or *best-fit* rule is that a relatively small number of component variants can form the basis for a much larger family of program variants. The leverage comes from the fact that different versions of the program can share the same versions of particular components (for example, as the `French` and `English` programs `PPC OS 8` share the `PPC OS 8` windowing component). The more-generic relationship between versions is thus an inheritance relationship, so that more specific versions inherit (that is, use by default) more generic versions of components.

3.2 IHTML, ISE, and IML

The system presented here is the latest stage in an effort, as described in [12], begun in 1996, to produce an authoring/publishing system based on an intensional versioning scheme analogous to the one just described for programs.

In these systems authors define Web sites as linked collections of pages, each of which can exist in many different versions. A request issued by a browser consists of a conventional URL together with a version expression. The server software generates the requested version of the requested page by combining the particular version expression with the (usually generic) source.

The first such system was Intensional HTML [18], a minimal extension of ordinary HTML. Initially, the most important feature of IHTML was that it allowed multiple source files for the same page, each labeled (as above) with a version expression. When a request arrives for a particular version of a particular page, the IHTML server (an Apache server plug-in) uses the best-alternative rule described above to select the appropriate source file.

In generating the actual HTML all the copy and most of the mark up is duplicated verbatim. Links, however are specialized, so that they connect analogous versions of pages. For example, the IHTML source for page A might contain an ordinary-looking link to page B. When a request arrives for the HTML source of (say) version `language:french+level:expert` of page A, the link is changed to lead to the `language:french+level:expert` version of page B.

IHTML also allowed what we call transversion links: these are links which connect one version of a source page with a different version of the target page. For example, in the source for A, the author can include a link to B in which it is specified (in the tag attributes) that target version has the language parameter set to english. When the `language:french+level:expert` version of A is requested, the IHTML link is turned into an HTML link to version `language:french+level:expert` of B. With transversion links the author can allow the user to change parameter values (and therefore adapt the target page) simply by following a link.

IHTML worked well enough for small sites but turned out to be impractical for larger projects. Maintaining a large number of small files proved to be a real headache. We added a version-sensitive conditional tag (iselect) [15] that allowed us to refine components within documents themselves, rather than refer to external files. This case statement gave us two things: a way for authors to define within-page version selections easily, and, as a side effect of this, a way to create Nelson-like stretchtext sections with markup rather than, for instance, a great deal of browser-specific JavaScript code. That said, for large files with many degrees of stretchtext, we sometimes needed a proportional number of additional lines of markup to define the regions. This required no great expertise, but it was tedious to do. The real problem was that IHTML was not author extensible, so we could not create a simple wrapper for the stretch text case, for instance. Furthermore, adding new tags for genuinely new features involved extending the specialization software.

In 1998 Paul Swoboda, then an MSc student, decided to abandon markup as the basis for intensional Web versioning.[3] He designed and implemented ISE, which is to Perl as IHTML is to HTML [14]. Since ISE is a full featured algorithmic language, there is no *a priori* limitation on the kinds of features it can support. Also, since it has functions and procedures, sophisticated versioned Web sites do not necessarily translate into extremely large ISE programs.

ISE, however, is still a programming language, and therefore is inherently more complex than a markup system. We therefore added IML as a kind of front end to retrieve the obvious advantages of markup authoring. IML offers all of the power of IHTML but, since it is implemented using macros, it is easily extensible, even, to a limited extent, by authors without any programming skills or knowledge of the language ISE (we explore this in more detail in section 4.8 below). Adding new constructs involves adding definitions for new macros to a file. These definitions can be intricate, but only have to be written once, and then can be used by authors who have no knowledge of ISE or (for that matter) of any other programming language.

In the next section of the paper, we describe some of the most commonly used macros in IML. Some can be used simply to add stretch text easily to Web pages; others can be used as conduits for more sophisticated versioning applications.

4. Common IML authoring patterns

IML is designed to make the full power of ISE available to authors without obliging them to become programmers; so that they remain authors in the colloquial sense of the word. IML can be understood as an extension of HTML. An IML source file consists of text together with markup which is either ordinary HTML tags, interpreted in the usual way, or special IML tags (we call them intensional tags), which are context sensitive. The two kinds of markup can be mixed and nested.

[3] As described by Yannis Kassis, there is some debate within the Intensional Hypertext group about the IHTML-like markup approach vs the IML/ISE approach described here [5]. For our purposes, however, the effects both approaches wish to support are the same: efficient, effective authoring of parameterized, versionable documents.

In this section we will use, for the sake of clarity, a (hypothetical) XML-style syntax for the IML markup. At present, however, IML is based not on XML but on the macro language of Groff. The (primarily pragmatic) reasons for using Groff instead of XML will be discussed in section 6, Future Directions, below.

4.0 Plain HTML

Since IML is an extension of HTML, the simplest IML documents are plain HTML documents, recast as IML documents. The recasting process is trivial: the `<html>` and `</html>` tags are replaced with `<iml>` and `</iml>` tags. Everything between the two remains unchanged, including JavaScript. The IML implementation by default treats markup as text and does not try to analyze non-IML tags.

One of the simplest ways to create an IML document is to modify an existing HTML document. The first step is to replace the `<html>` and `</html>` tags as described, and the result can be fed to the IML implementation, which will produce a corresponding ISE program (consisting mainly of a large statement). The page produced by running this program should be identical to the original.

4.1 Parameter Substitution

One of the simplest (but also most useful) features of IML is the ability to retrieve the value of a parameter, in almost any context. For example, suppose the page was a message and we want to insert the adressees surname in the salutation. Using the XML-style syntax, we can write

```
 <p> Dear Mr/Ms <impl-param name="surname">,
pleaseconsult … </p>
```

Then the generated ISE program is run in an environment in which the surname parameter has the value Jones, it will produce

```
<p> Dear Mr/Ms Jones please consult … </p>
```

The 'real' groff-based IML implementation allows a more concise notation: the parameter name preceded and followed by ##. This nonstandard notation has the advantage that it can be used even inside HTML tags. For example, <body bgcolor=##EGG##> will set the background color to the current value of the EGG parameter (whatever that may be).

4.2 Conditional Inclusion

Of course, it would be very strange to personalize a message by including the adressee's name, and then take no account of their gender. If the current context has a title parameter we can include it directly; but if necessary we can choose the title on the basis of the gender parameter (if there is one).

We can make the choice using IML's <iselect> construct, as follows:

```
<iselect>
<icase version="gender:F">Ms</icase>
<icase version="gender:M">Mr</icase>
</iselect>
```

In general, there can be any number of <icase…>…</icase> sections between the opening and closing `iselect` tags. Each tag specifies a version attribute. Only those icase sections whose attributes are consistent with (can refine too) the current context are considered. If there is exactly one such *eligible* section, the enclosed copy is included. If there is no such section, the iselect clause contributes nothing (and is essentially ignored). If there is more than one eligible section, the best fit rule is used to select the a single icase section.

Suppose, for example, that there is also a parameter mstatus with value Y to indicate the individual is married, N to indicate not married, and vanilla to indicate no information. The following iselct clause, together with the best-fit rule, specifies the traditional rules for choosing a title:

```
<iselect>
<icase version="gender:F">Ms</icase>
<icase version="gender:F+mstatus:Y">Mrs</icase>
<icase version="gender:F+mstatus:N">Miss</icase>
<icase version="gender:M">Mr</icase>
</iselect>
```

4.3 Levels

One common method for adapting documents is to allow one or more integer-valued parameters which specify some kind of depth of treatment. This parameter could measure the (estimated) level of the reader's expertise, or the degree of detail deemed to be appropriate.

The author indicates in the source document either at the document level, or within each section, a minimum level required for that section to appear. In publishing the document, the value the parameter has in the publishing context is used to producer a version which is longer or shorter, according to whether the value in question is greater or smaller.

4.3.1 Stretch Text. The simplest approach is to supply the author with tags, say `<iml-incdepth>` with the understanding that any text enclosed beteween `<iml-incdepth>` and `</iml-incdepth>` has an associated depth which is one greater than that associated with the enclosing text.

For example, suppose that at the topmost level the source contains

```
<p>At the southern end of Vancouver Island we find
Victoria<inc-depth parm="detail">, the provincial
capital</inc-depth>.</p>
```

The simplest way to produce this kind of telescoping text is to take an existing (mono-level) document and add `<iml-incdepth>` tags. Our experience is that well

written text lends itself readily to this kind of factoring, even though the authors never intended to produce less than the full-length version.

One minor difficulty is that omitting sections may cause grammatical errors, for example, verb forms which have to change from plural to singular in the shorter versions. This problem is easily solved by allowing <iml-incdepth> to have an extra alternate attribute of text to be included in case the enclosed copy is to appear. For example, the source

```
<p>At the southern end of Vancouver island we find
Victoria<iml-incdepth>, Saanich, and Oak Bay</iml-
incdepth>, which <iml-incdepth alt="enjoys">enjoy</iml-
incdepth> a very mild climate.
```

4.3.ii Drop Text. By drop text we mean a single block of text that can be made to appear or disappear separately, without affecting any other part of the document. Drop text can be considered (and implemented) as a simple kind of telescoped text in which the dropped block has its own private two-valued depth dimension.

A very basic kind of drop text consists of an anchored heading and a body, with the heading always visible. When the text is absent, following the heading link causes the text appear, and when the text is present, following it makes it disappear (with nothing else changed). Sections like this can be specified by the iml-drop tag which encloses the disappearing text. The "heading" attribute determines the anchored heading. For example,

```
<iml-drop heading="Other cities to visit">
Nanaimo, Sooke, Esquimault, Duncan. </iml-drop>
```

4.4 Stereotypes

Depth parameters can be considered a special case of "stereotype parameters" [1] that have a discrete set of values (not necessarily ordered) each of which represent a kind of user profile. For example, we might have a "purpose" parameter which specifies the reason for a visit to an attraction, with values such as "tourism", "shopping" and "business."

We supply authors with a tag `iml-stereo` that includes enclosed text when the value of the "purpose" tag has the value specified. For example,

```
<p>The parliament buildings are located just east of
the Empress<iml-stereo param="purpose" type="tourist">,
and are open to visitors most mornings</iml-stereo>.
</p>
```

The parameter selection can be set up to be determined by the user selecting a link to establish the "tourist" version of the page, or by a user model, passing through the tourist value to this version of the page, or by both. The parameter values associated with the reader queries can be combined with those computed by a UM and passed on an equal footing to the ISE interpreter running the ISE version of the adaptive/adaptable document.

The same, of course, can be said of all the other constructs here: they do not take into account the origin of the parameter values they examine. That *should* mean more flexibility for authors and system developers: it becomes easier to determine where and under what conditions parameters are to be interpreted by user, whether selection or user model (see the diagram in section 5 below).

4.5 Transversion Links

As we explained earlier, IML supports adaptable hypertext, in which the users make explicit choices about the form and content of the document they are reading. These choices must be captured as parameter values but the reader (user) need not know anything directly about IML version expressions. Instead, the author can arrange that particular hyperlinks be associated with changes in parameter settings.

The simplest form is what we called in IHTML a transversion link: an ordinary a tag with an extra vmod attribute which specifies updates. For example, suppose the IML source has the following.

```
Si vous preferez, on peut vous presenter ces
renseignments <a href=info vmod="lang:french">en
francais</a>.
```

When the document is rendered, this becomes a link to the version of the "info" page that is identical to the current version except that the `lang` parameter is set to `French`. Note that the values of the other parameters are carried across the link unchanged: if the reader was looking at the `purpose:tourism+lang:english` version of the page on which the link appears, clicking it will take the reader to the `purpose:tourism+lang:french` version of the info page.

Similar tags allow authors to enable reader-adaptations using menus, forms, image maps and the like. Notice that transversion links may alter more than one parameter. For example, the author could offer a link

```
<a href=expl vmod="lang:french+level:newbie"> J'ai rien
compri</a>
```

that allows the user to self-identify as an inexperienced francophone.

It is also worth pointing out that the reader is not necessarily aware of the existence of some of the parameters being changed, and may not immediately see any effect of their change. In fact, the document might be completely insensitive to the values of these parameters. Instead, these values might be meant for the User Model, which monitors all the user requests.

4.6 Nesting

It goes without saying that the constructs described can be nested - but we'll say it anyway. For example, we can have telescoping text in which deeper level copy has both tourist and business variants; or variants of one page corresponding to different stereotypes can offer different sets of options (as transversion links).

The advantage of nesting, of course, is that it gives us a component model: the author can construct very elaborate adaptive/adaptable documents using a relatively small set of primitives. This is a vital part of what makes IML a general-purpose tool.

4.7 Extensibilty

The other *sine qua non* of a component model is the ability to define new components. In IML this takes two forms, which we might call "author-extensibility" and "implementer-extensibility." The first is simply abstraction (sometimes called encapsulation), the ability to give a name to combination of already existing constructs and invoke the combination using this name, possibly with parameters.

In terms of XML-style syntax, this means the ability to define new tags by expressions involving existing ones. A very simple example would be to define the tag `</hello>` as appropriate to the word or phrase in the language specified by the value of the `lang` parameter. We have already seen how to do this, but the resulting source expression can be very long, especially if there are a number of alternate languages. Abstracting this expression as `</hello>` shortens the source and protects part of it from needing alteration in case we change our language options.

IML's author-extensibility comes directly from groff's macro definition facility. Unfortunately this feature is tied closely to groff's syntax and can't be described without going into details of groff, which we would rather avoid (those interested can consult [17]). At the moment it is not clear to us what a full XML-style syntax would look like, so we won't offer even hypothetical examples of parameterized definitions (we return to this point in section 6, below).

IML is also implementer-extensible. This means simply that people who understand ISE and its versioning system can write macros, which produce fragments of ISE programs, rather than just more IML. The IML-to-ISE macros tend to be short but dense but there is no other way to define the primitive constructs. Fortunately, with nesting and abstraction, a few complex implementer-extensions go a long way. These have been used currently to specifiy a variety of versioning effects, as per [12].

In addition, ISE itself is extensible - it allows procedures to be written in C but called in ISE. This allows the implementer, for example, to provide authors with tags, which access a database. The implementer writes the access primitives in C then defines the tags with macros which generate ISE source which calls these primitives.

4.9 Best Fit

One of IML's most useful features, and perhaps the trait which distinguishes it the most from other markup based systems for version support like [9] is its ability to combine a special kind of author-extensibility with the *best fit* protocol as introduced in section 3.1 above. An IML author is able to give a name to a block of IML code, invoke the code using the name, but define the abstraction incrementally in a case-by-case manner, with inherited defaults. The extra definitions are labeled with version expressions, much like the branches of an iselect. But they can be added one by one

and scattered throughout the source. This kind of abstraction is based on ISE procedures, which themselves can be defined incrementally and with defaults.

For example, we do not have to give a single monolithic definition for the tag `</hello>` mentioned above. We can start by giving default definitions, say "hello" in English, then use it wherever we want. If we do nothing else, `</hello>` always expands to "hello."

Next, we can add special case definitions of `</hello>` as "Bonjour" and "Ola" labelled with expressions `lang:french` and `lang:spanish` respectively. These definitions will now override the default, which nevertheless remains in force for languages other than French and Spanish, Later, we may decide that Hispanic business people might prefer something more formal and add fourth definition of `</hello>` as "Buenos Dias", labelled with `lang:spanish+purpose: business`. This overrides the "Ola" default for hispanics, but Spanish speaking tourists will still see the more informal greeting.

In this way authors can concentrate their efforts to produce documents that are more carefully adjusted for certain audiences. And they can improve the adaptability gradually, without large-scale revisions at each step. Indeed, authors can encode these versions in advance of a specific UM taking advantage of them, should that approach be appropriate. Until the UM is ready to take these into account, they may not appear to the user.

5. Publishing Modes

One great advantage of the approach described here is that it is possible to produce useful, version-based sites with only a simple UM, or none at all. All of the documents produced earlier using IHTML fall into this last (UM-free) category. This kind of site is referred to as "adaptable" rather than "adaptive" because the variations are the result of the user making explicit choices (mainly by choosing links). For example, in an adaptable site the user might identify herself as a tourist by following a link anchored to an image of sunglasses, request French language material by selecting that language from a menu choice, and open up the drop section on "Other Cities" by following the link anchored to that heading.

Recall that transversion links carry over, unchanged, all parameter values that are not altered by the link. The result is that the URLs of the pages the reader views normally accumulate the choices made (and hence act as a very simple implicit user model). One very simple case of this publishing mode results if the author uses only regular HTML tags. Then no parameter values at all are generated, the site is neither adaptive nor adaptable, but the author still has the advantage of IML's author/implementer extensibility.

On the other hand, even a very simple UM can greatly improve the effectiveness of the site. For example, the UM could simply guess at the reader's language preference by examining the IP address associated with the reader's request. Or it could supply the parameter values which specify the date and time (definitely something the reader should not have to provide). In either case, the author could take advantage of this

extra UM-supplied information to, say, select the kind of visitor information displayed. A more sophisticated model could have access to a database - a regular DBMS or perhaps just a UNIX text file. This UM could extract the reader's identity (they might be asked to provide it), look up the individual in the database, then pass on (to the ISE interpreter) associated parameter values, for example the reader's degree program or midterm marks.

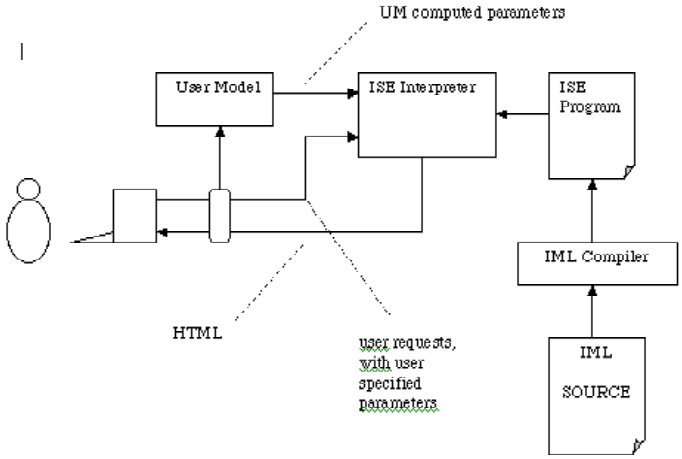

Figure 1: Model-neutral IML architecture for versioning

Of course, there is no limit to the sophistication of a user model. Our approach to publishing puts only one constraint on the nature of the model: that it be able to pass on the relevant parts of its current state as a set of values for parameters.

6. Conclusions and Research Directions

In this paper we have presented a mechanism for letting authors author versionable hypertext without being primarily constrained by how those versions will be delivered, whether by user-determined link choices or UM parameter determination. We anticipate that this approach offers ecumenical advantages for hypermedia researchers with either adaptable or adaptive biases since it frees adaptable/adaptive hypermedia researcher from designing their own author layer for their system. They can simply pass their parameters to the IML/ISE front end. The IML/ISE tools for creating the adaptable/adaptive author layer also foreground the intensional approach to versioning, which offers particular advantages not available in other version-markup systems, namely, version refinement and best fit. One of the chief advantages of version refinement is that it allows a large number of potential versions to be created from a relatively small set of primitives, or for that matter, from a small set of source files. Likewise, the abstraction/encapsulation facility collaborates with the version best fit protocol to allow incremental refinement of the version families, in which the author specifies general defaults first, then overrides where appropriate. We suggest that this intensional approach to handling versionable content, as embodied in

the IML methodology, can act as general front-end tool for preparing versionable content to be delivered to a user, whether the versioning is managed by an adaptive or adaptable approach. That said, there are certain attributes of the current IML implementation that need further refinement and evaluation for the mehod and tool to be successful in this open context. We describe these below.

Current IML Syntax. The most urgent task in IML is to abandon groff in favor of an XML-compatible syntax, like that used in our hypothetical examples. For example, the drop text example presented earlier is actually, currently, written as

```
.biml-drop “Other cities to Visit”
Nanaimo, Sooke, Esquimault, Duncan.
.eiml-drop
```

Groff’s quaint line-oriented syntax is simple enough once you get use to it. Groff is distributed with every Linux implementation and its implementation is solid. Also, nesting and abstraction are completely straightforward. The current groff-based markup is, in a sense, very successful and has allowed a number of people (beyond the authors) to experiment with the ideas presented here.

Nevertheless, it is syntactically incompatible with XML and on those grounds alone hopelessly obsolete. And even if we ignore syntactic issues, there are also serious problems with groff’s form of abstraction. The arguments of groff macros are listed in a fixed order, rather than assigned as values of parameters. This makes it practically impossible to arrange sets of default values or generate different output according to the presence or absence of particular arguments. Instead, one has to define whole families of related macros with separate complete definitions; a very tedious process and one, ironically, at odds with the IML philosophy of versioning.

The major challenge in replacing groff is designing (not to mention implementing) an XML-compatible abstraction notation. This effort is only just under way.

Evaluation. We have not yet implemented a large-scale intensional hypertext project and exposed it to rigorous user evaluation. We are in the process of doing this for two reasons in particular. The first is that we wish to understand better how we need to support authors in developing intensional hypertext documents, since version management and version visualization or tracking are not insignificant additional considerations for authors or teams developing content. The second is that we wish to see if we can, in participation with authors and site users, develop heuristics for a versionable site’s evolution, from plan, to integration with adaptable or adaptive techniques, to maintenance and refinement. We are engaged in a project with Biomedical Communications at the University of Toronto to develop a multi-user site on breast cancer and the sentinel node biopsy procedure which will also be the test bed for our proposed evaluation.

7 Acknowledgements

We wish to thank Michael Milton and Adele Newton of the Bell University Labs Research Program at BCE and the University of Toronto for their continued enthusiastic support of this research. Thanks also to Paul Swoboda and Neil Graham for their development efforts and insights throughout the lifecycle of IML.

8 References

1. Brusilovsky, Peter, Methods and techniques of Adaptive Hypermedia. Adaptive Hypertext and Hypermedia. Kluwer Academic Publishers, Amsterdam 1998, 1-43,

2. De Bra, Paul, Licia Calvi. Towards a Generic Adaptive Hypermedia System. Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98, Pittsburgh, USA, June 20-24, 1998, 5-12.

3. DeBra, Paul. Design Issues in Adaptive Hypermedia Application Development. 2nd Workshop on Adaptive Systems and User Modeling on the WWW. Toronto (1999) 29-39.

4. Hirst, Graeme, Chrysanne DiMarco, Eduard Hovy, and Kimberley Parsons, "Authoring and generating health-education documents that are tailored to the needs of the individual patient." In Anthony Jameson, Cécil Paris, and Carlo Tasso, (editors), User Modeling: Proceedings of the Sixth International Conference, UM97, (Chia Laguna, Sardinia, Italy), Springer, 1997, 107–118.

5. Kassis, Yannis. Noema as Alternative to ISE/IML. Unpublished Hypermedia Graduate Course Paper, U of Toronto, Canada, 2001.

6. Kushmerick, Nicholas James McKee, and Fergus Toolan. Towards Zero-Input Personalization: Referrer-Based Page Prediction. P. Brusilovsky, O. Stock, C. Strapparava (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000, Trento, Italy, August 2000. Proceedings. Springer-Verlag, Lecture Notes in Computer Science, Vol. 1892, 133-143.

7. Moore, Johanna D. Participating in Explanatory Dialogues: Interpreting and Responding to Questions in Context. MIT Press, Cambridge, Mass. (1995).

8. Murray, Tom, Tina Shen, Janette Piemonte, Chris Condit, and Jason Thibedeau. Adaptivity for Conceptual and Narrative Flow in Hyperbooks: The MetaLinks P. Brusilovsky, O. Stock, C. Strapparava (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000, Trento, Italy, August 2000. Proceedings. Springer-Verlag, Lecture Notes in Computer Science, Vol. 1892, 155 –166.

9. Paradis, Franois and Anne-Marie Vercoustre and Brendan Hills. A Virtual Document Interpreter for Reuse of Information, Proceedings of Electronic Publishing '98, Saint-Malo, France, 1-3 April, 1998. Springer-Verlag, Lecture Notes in Computer Science 1375, 487-498.

10. Petrelli, Daniela, Daniele Baggio, and Giovanni Pezzulo Adaptive Hypertext Design Environments: Putting Principles into Practice P. Brusilovsky, O. Stock, C. Strapparava (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000, Trento, Italy, August 2000. Proceedings. Springer-Verlag, Lecture Notes in Computer Science, Vol. 1892, 202-213.

11. J. Plaice and W. W. Wadge, A New Approach to Version Control, IEEE Transactions on Software Engineering, March 1993, 268-276.

12. schraefel, m.c. "ConTexts: Adaptable Hypermedia." P. Brusilovsky, O. Stock, C. Strapparava (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000, Trento, Italy, August 2000. Proceedings. Springer-Verlag, Lecture Notes in Computer Science, Vol. 1892, 369-375.

13. Stern, Mia K. and Beverly Park Woolf Adaptive Content in an Online Lecture System P. Brusilovsky, O. Stock, C. Strapparava (Eds.) Adaptive Hypermedia and Adaptive Web-Based Systems International Conference, AH 2000, Trento, Italy, August 2000. Proceedings. Springer-Verlag, Lecture Notes in Computer Science, Vol. 1892, 227-238.

14. Swoboda, P. Practical Languages for Intensional Programming, MSc Thesis , (Computer Science), University of Victoria, Canada (1999).

15. Wadge, W. Intensional Logic in Context. Intensional Programming II, World Scientific Publishing, Singapore, 2000, 1-13.

16. Wadge W., G. Brown, m. c. schraefel, T. Yildirim, Intensional HTML, Proc 4th Int. Workshop PODDP '98, Springer Verlag LNCS 1481 (1998) 128-139.

17. Wadge, W. Intensional Markup Language. Peter Kropf, Gilbert Babin, John Plaice, Herwig Unger (Eds.): Distributed Communities on the Web, Third International Workshop, DCW 2000, Quebec City, Canada, June 19-21, 2000, Proceedings. Lecture Notes in Computer Science, Vol. 1830, Springer, 2000, 82-89.

18. Yildirim, Taner. Intensional HTML Masters Thesis, U of Victoria, Canada. (1997).

Paper contact: mc@cs.toronto.edu