Noema: a metalanguage for scripting Versionable Hypertexts

Ioannis T. Kassios and m. c. schraefel
Department of Computer Science, University of Toronto
Toronto ON M5S 3G4 Canada
{ykass, mc} @cs.toronto.edu

Abstract

In this paper we present Noema, a new intensional hyper-text metalanguage based on both intensional HTML and XML. The design of Noema aims at the improvement of the expressiveness of intensional HTML and introduces principles such as *intensional XML-like entities*, *versioning / hypertext unification* and *implicitly created dimensions*. The bigger expressive power of Noema doesn't mean that one has to program hypertext, as is the case with ISE/IML, so Noema is proposed as a better solution to the problems encountered in intensional HTML.

1 Introduction

In 1997, the Intensional Programming community was first introduced to the concept of treating the Web as a series of possible worlds, and to web pages as instances of a demand driven data flow stream. Since that time, there have been three core versions of intensional markup for bringing user-determined versioning to web site use: IHTML-1[10], IHTML-2[1], and currently ISE/IML [7]. While the first two versions of the markup language to support versioning were similar, the third was a significant departure. The first two attempted to use a markup approach that would only add a few HTML-like tags to HTML, making versioning accessible to the majority of non-programmer Web page authors.

As the limits of this approach became apparent through various test sites, a more robust programming-like solution was sought. The result was ISE/IML. This approach, while robust, lost the immediate accessibility for non-programmers to build their own page effects, forcing them to rely on prefabricated intensional macros instead.

As the authors of these systems have stated [1],[3], neither the IHTML nor ISE/IML approach is optimal for bringing flexible version control to non-programmer web authors. In this paper, therefore, we present Noema as meta-language, middle ground approach to the problems posed by both IHTML and ISE/IML. In the following sections, we briefly overview IHTML and ISE to situate the problem space. From here, we introduce Noema and demonstrate how we can use this as a markup interface for versioning primitives which will let authors have more direct, but interpretable control of the versioning functionality.

2 Overview of IHTML and ISE/IML

Intensional HTML (IHTML) was a first step implementation towards proposing the Web as an intensional set of possible worlds. That is, each site could be treated in the manner of Plaice and Wadge's Intensional Programming [5]. That is, sites, like software, could be represented as versionable components, rendered on the fly to satisfy a user's request for a particular version. Three of the key benefits of creating demand-driven pages are (a) the reduction of duplication/cloning if maintaining multiple versions/localizations of a particular site; (b) the possibility for a wide range of version combinations po-

tentially not anticipated by a site designer, and (c) the concept of "best match" for any given version request.

The first implementation of an intensionalized web, IHTML, took the form of an extended HTML markup. IHTML markup was relatively straightforward for an author to create and for another human to interpret. For instance

```
<a href="page.html"
   version=lang:french%canadian>
french version </a>
```

meant render this particular page with all attributes on the language dimension set to French Canadian-where those components are available. Where French Canadian is not available, default to French.

Simple statements like the above made IHTML immediately appealing. There was however a high cost for creating the version components and labeling them. The markup itself, for things no more complex than simple case statements for offering possible versions of a page became cumbersome.

For instance, the intensional concept would make it possible to create a "slide show" as a series of possible versions of a page. The markup costs are high, however, as the following sample demonstrates:

```
<ISELECT>
  <ICASE version="slide:1">
    <IMG src="sld1.jpg"><P>
    <A vmod="slide:2">Next</A>
  </ICASE>
  <ICASE version="slide:2">
    <IMG src="sld2.jpg"><P>
    <A vmod="slide:3">Next</A>
  </ICASE>
     ...
  <ICASE> I'M DONE AT LAST!!
  </ICASE>
</ISELECT>
```

A programmer would be frustrated if s/he was forced to write in a language that can not take advantage of the fact that all 250 cases above are instances of the same generic template, namely

```
<ICASE version="slide:X">
  <IMG src="sldX.jpg"><P>
  <A vmod="slide:(X+1)">Next</A>
</ICASE>
```

In other words, it was difficult for IHTML to handle simple functionality. To address this problem, a new approach was taken with the development of ISE and IML. ISE is an intensionalized version of the Perl language. As such it is custom-designed to handle any of the functional requirements missing from IHTML. IML was developed as a kind of front end for ISE. It would provide a set of predefined macros (in TROFF) for page authors to use to embed intentional functionality into a page. For instance, Nelson's drop text could easily be embodied as an ISE function through a macro call:

```
.bdrop - 1 Info about fish

The fish will be spawning here in three weeks
[whatever HTML markup the author wishes]

.edrop
```

The result of this call would be

```
>  Info about Fish
```

where the > is a link. Click the > and the page reveals the material under the header, re-orienting the page to the header (-1) of this point (info about fish).

When the author finishes the markup, they run a process to convert the file into ISE. They create a link to the .ise file, such as `"fish.ise"`, and a server-side process renders the ise into the appropriate HTML for a given version request of that file. Any link requests from the page are sent to the server as ise parameter requests causing a new html page to be sent to the client that represents the new version request. Like the original IHTML, ise pages are always represented to a browser as HTML, so they are accessible from any browser. The browser is only limited by the javascript/html or whatever effects the author adds to the page. IHTML and ISE are otherwise platform agnostic.

With macro calls like this available to an author, clearly the weight of having to write Perl (or javascript or for that matter reams of IHTML) is

highly diminished, allowing the author to concentrate on their design rather than on programming effects. On the down side, the IML approach does lose some of IHTML's more intuitive clarity and also requires someone proficient in both TROFF and ISE's specialties to create new macros.

Indeed, several papers on IML have lamented the "ugliness" of IML [9],[4] and have expressed a desire to convert it into a more XML-like form for better integration into the markup of the page - recovering some of what was lost from IHTML.

The challenge, therefore, is to find a middle ground between the full-fledged programming functionality of ISE and the benefits of more markup-like integration of IHTML. The solution should allow the author to create customized constructs (missing from IHTML) without escaping from the authoring language (as IML requires by backing up through TROFF to ISE). Such a solution is achievable by consideration of what is to be gained by adding functionality at the markup level. In the following paper, we present one possible solution: Noema. Noema is a hypertext metalanguage that generalizes IHTML in describing intensional hypertext. The three main ideas underlying Noema are: the unification of hypertext and versions, the use of versionable XML-like entities (and the version changes in pieces of a hypertext file) and the implicit dimensions as a substitute for state. We describe each of these concepts below and conclude with a discussion of future work.

3 Introduction of Noema

As demonstrated above, a concern with IHTML is the absence of markup reuse and of textual manipulation of versions. The missing features are not as general as programming functionality, nor do we need the imperative paradigm with states and the like to implement them. However they are very broad to be implemented by defining new constructs, because our ultimate goal is generation of specific markup from generic markup (that is, certain text manipulation facilities)[1].

[1] Arithmetic may be regarded as text manipulation too

In this section, we propose a different approach to the problem. The goal for our design is to create a meta-language (not necessary an HTML, or IHTML superset), that is going to encompass the enhanced handling of versions we require:

- Manipulation of versions as hypertext
- Text reuse in a parameterizable fashion
- Some notion of state
- As friendly to the author as possible; markup rather than imperative

In the following sections, we present the design of Noema, such a meta-language. It is important to note here that this isn't a complete presentation of the language. Many design details have been deliberately left out. The focus here is on the ideas that underlie the design and Noema syntax is meant to help demonstrate these ideas in a more concrete way. The ideas under discussion are:

- Use of hypertext to denote versions: We will exploit the tree-like structure of hypertext to express the tree-like structure of versions. In this way, we may use both interchangeably and this results in a nice uniformity when it comes to parameter passing.
- Versioned entities: Entities are an XML concept missing from HTML and IHTML. With entities one names a piece of hypertext to be used several times in the document. What we are going to define is versioned entities. This makes Noema be to IHTML what XML is to HTML. Versioned entities can be seen also as *parameterizable* entities, because we can invoke any version of an entity we want. Versioning is enough to pass parameters, so we will not use any novel machinery for that.
- Implicit versions as a substitute for state: An invocation of an entity generally implies that we are going to use a specific dimension in our version space as "state". We make the existence of this dimension implicit and this choice makes the language strictly more expressive.

Noema is a metalanguage that describes hypertext. A Noema file prescribes various version-dependent hypertexts. We therefore need two levels in our language: that of meta-hypertext, and that of literal hypertext. Our conventions will be the following:

- A Noema file consists of a declaration section and a meta-hypertext section (divided by `%%`).
- When writing meta-hypertext we may include literal hypertext within braces ({}).
- When writing literal hypertext we may include a meta-hypertext expression preceded by `&` and succeeded by `;` (like XML entities)
- Special characters such as `&` and {} within literal hypertext, are preceded by `&` if they are to be taken literally

3.1 Versions themselves are hypertext

Version space in IML/ISE is currently very advanced, if compared to the first tries in [5]. The current version space (see [7]) supports nesting, as the following abstract syntax shows (V is the non-terminal for version, s is just any string):

$$V \rightarrow \epsilon \mid s : V \mid V + V$$

(also assume that s will be equal to the version $s : \epsilon$).

This syntax is readily expressible with markup, under the following translation scheme ($[V]$ is the markup that corresponds to version V):

$$[\epsilon] \;=\; \texttt{<!--EMPTY MARKUP-->}$$

$$[s : V] \;=\; \texttt{<}s\texttt{>}[V]\texttt{</}s\texttt{>}$$

$$V_1 + V_2 \;=\; [V_1][V_2]$$

Versions are a limited form of hypertext because not all hypertext can count as versions under this translation scheme. For example`<A><B/></A>` represents version `A:B`. But `<A><B/></A><A><C/></A>` doesn't count as a version, because the `A` dimension appears twice.

This scheme offers a uniformity, with certain advantages. For example, later we introduce parameter passing to an entity via versioning. Because versions and hypertext are considered the same, we can thus pass pieces of hypertext as parameters. We can also version versions and so on.

Since we express versions as hypertext, our metalanguage should have some operators that handle versioning. To that end we also borrow the semantics of most of the original IHTML constructs:

- `cv` is a nullary operator, that returns the current version
- `vmod` is a binary operator. `a vmod b` means version `a` modified by modifier `b` (modifiers are also expressed as hypertext). We may also use it as a unary operator, where `vmod b` means `cv vmod b`. Version modification is exactly the same as in IHTML
- `++` stands for version algebra + and `:` stands for version algebra :
- `(/)` is the vanilla version (or the empty hypertext)
- `'` followed by an identifier denotes that version name, e.g. `'A:'B` stands for hypertext `<A><B/></A>`[2]

Note that the + operator of version algebra is not exactly the same as hypertext concatenation, since for example it is idempotent[3]. Operator `,` denotes pure hypertext concatenation.

3.2 Limited scope version modifiers

We introduce the `@` binary operator whose purpose is to modify the current version within its scope. If `a` and `b` are meta-hypertext expressions, then `a@b` denotes `a` at version `b`. For example,

```
{someHypertext}@(vmod someModifier)
```

changes the current version by `someModifier` so that `someHypertext` is evaluated in the changed version. Similarly,

[2] Note that we use the XML abbreviation `<X/>` for `<X></X>`

[3] In general, any version is hypertext, but not any hypertext is version

```
{someHypertext}@(someVersion)
```

sets the version of `someHypertext` into `someVersion` altogether. The difference of `@` and `vmod` and `version` attributes of IHTML is that in Noema we may change the version of an arbitrary piece of hypertext, instead of just one link.

3.3 Versioned entities

The main problem is to make text re-usable. Here's where the entity idea comes: we name a piece of text, so that we can reproduce the text by using only its name. This exactly what XML entities are doing [2]. But we want more than that: as we've already seen in our examples, the text to be re-used is not exactly the same, but rather it depends on parameters. Adding simple XML entities to IHTML won't solve the problem. Instead, we need to make parameterized entities possible. Instead of new machinery for parameters, we prefer to use something already existing: versions. Not only does this avoid the formal clutter by not introducing new constructs that are not needed, it also has the well known advantages of versionning, such as the use of version refinement etc.

Noema introduces entities and entity references (in the XML sense) that have versions. For that matter, we use the `entity` meta-construct in Noema, for the creation of entities in the declaration part of a file:

`entity` *name_of_entity*{*noema_code*}

where in *noema_code* we have a Noema file again (declarations / meta-hypertext etc.).

What is contained in this construct defines exactly what markup is going to be generated when the entity is referenced. The markup depends on a version, so by referencing the right version of the entity[4], we get the exact markup we want. Entity declarations may be nested, but then the inner one is regarded as "local" to the entity definition and is not available out of it. Recursion is allowed. To reference the entity within meta-hypertext we just use its name.

[4] We can do that using the `@` operator

3.4 Other primitives

Except from what we've shown so far, Noema also supports as meta-hypertext numeric values (that stand for their corresponding decimal representation) and all usual mathematical operations, and it also supports the logical operations `and`, `or` and `not`. These operate on "boolean" hypertext, that is, meta-hypertext that either equals to `'TRUE` or to `(/)`. There are comparison operators, that return such boolean hypertext:

- Hypertext comparisons: `=` and `!=`
- Numerical comparisons: `>=` etc. (numerical equality and inequality coincide with their hypertext counterparts)
- Version comparisons: version equality `~` is weaker than hypertext equality. The appropriate notions of version inequality `!~` and version refinement `[~` and other derived operators such as `[,` `]`[5] and `~]` are also supported

The *conditional hypertext* takes a boolean meta-hypertext expression and two other meta-hypertext expressions. Its semantics is obvious. For example:

```
if cv [~ 'language:'french
then {bonjour}
else {good morning}
```

To get the hypertext within a tag we use `get` as a binary operator. `a get b` gets the value of version `b` within hypertext `a`, for example

```
('language:'french) get 'language
```

evaluates to `'french`. Or, similarly,

```
{
  <GREETING>Hi!</GREETING>
} get 'GREETING
```

evaluates to `Hi!`. `get` will work on `cv` when used as a unary operator. `strip` is a unary operator that strips off the outermost tag in its parameter.

Operator `linkto` creates a link to a Noema file described by its parameter. We usually write a filename

[5] Strict refinements

in braces for literacy (but the filename could also be an arbitrary meta-hypertext expression!) and we define a specific version of the file using the `@` operator. `link` is a parameterless version of the operator, that refers to the same file it appears in.

A further abbreviation: Suppose we want to create an entity `a` equal to the hypertext that lies under dimension `d`[6]. We normally include in the declaration part:

```
entity a { %% get d }
```

We may abbreviate this declaration to:

```
par a d;
```

3.5 The drop markup example

The drop text effect could be implemented in Noema (figure 1)

The few nested `par`'s just give convenient names to the parameters of the entity. The value of dimension `'MODE` can be either `'EXPANDED` or `'HIDDEN`. `'CONTENT` and `'ABSTRACT` contain the two versions and `META` describes the version of the document to be modified, if the user clicks on the link `[+]` or `[-]`. So a "client" of the `drop` entity would do something like:

```
drop @ vmod
(  'CONTENT:{Blah blah}
++ 'ABSTRACT:{Blah}
++ 'META:'MODE
)
```

This code says that the state of the drop markup depends on the version dimension `'MODE`.

3.6 Implicit dimensions as state

The code we created is still unsatisfactory, since we need to write all this information about the `'META` dimension (on which depends whether the markup is "hidden" or "expanded"). It would be nicer if the inclusion of a drop markup control within the document implicitly created such a dimension, already known to the control, which is then able to adjust itself. That among others would spare us the clutter of creating the `'META` dimension ourselves in the previous example.

To achieve this we introduce the *implicit dimension* concept. Each reference to an entity in the document implicitly creates a new version dimension which can be used to change the status of the document. We access this dimension *within the entity definition* by `i(x)` where `x` will be the entity name. A command `this(x)` re-invokes entity `x` *using the same implicit dimension*. The drop markup example is now as seen in figure 2.

The point of this example is, apart from showing the actual Noema code, to demonstrate that it is very easy to write a Noema entity definition. When defining an entity, unlike in IML, the user has at his or her disposal the full power of the language[7]. The language is only one and it is the same for both definitions and invokations of entities. The definition syntax is also considerably less cluttered than the IML counterpart. It is therefore believed that the Noema framework is better in many ways than the IML front-end. It remains to be proved that it is actually so.

Now the client is:

```
drop @
(  'CONTENT:{Blah blah}
++ 'ABSTRACT:{Blah}
++ 'INITIAL:'HIDDEN
)
```

Note that the number of implicit dimensions added is not fixed. This is because the number of actual entities invoked is not fixed in a Noema document: it is a variable that depends on the version, as shown in the code below:

```
entity multiply {
  par num 'NUM;
  par markup 'MARKUP;
%%
  if num = 0
```

[6] Thus i.e. we name "formal" parameters within an `entity` declaration

[7] In IML, we either don't have ISE at our disposal, or we have to resort to programming

```
entity drop {
  par mode 'MODE;
  par content 'CONTENT;
  par abstract 'ABSTRACT;
  par meta 'META;
%%
  if mode = 'EXPANDED
  then (link @ meta:'HIDDEN):{[-]} , content
  else (link @ meta:'ABSTRACT):{[+]} , abstract
}
```

Figure 1: The drop-markup in Noema

```
entity drop {
  par initialmode 'INITIAL;
  par content 'CONTENT;
  par abstract 'ABSTRACT;
%%
  if get i(drop) = 'EXPANDED
  then (link @ vmod i(drop):'HIDDEN):{[-]} , content
  else if get i(drop) = 'HIDDEN
       then (link @ vmod i(drop):'EXPANDED):{[+]} , abstract
  else this @ vmod i(drop):initialmode
}
```

Figure 2: The drop-markup with implicit dimensions

```
  then (/)
  else markup ,
       multiply @ vmod 'NUM:(num - 1)
}
```

This construct is given a markup m and a number n and its output is n consecutive copies of m. n may depend on version, so there is no way for the author to determine how many entities s/he has in the document if it contains a `multiply`.

3.7 The slide example

Figure 3 is how we do the slide example in Noema utilizing everything introduced so far[8]. We notice that it takes a very short and easy definition to create something impossible in IHTML and very hard in ISE. The difference from ISE lies at the functional-like approach of Noema which makes the definition of the construct more natural and compact. This is also true for the previous examples.

4 Conclusion

Noema as presented focuses on and achieves improvement which the IML community has sought for versioning hypertext:

- First, Noema is an authoring language as opposed to a programming language. It is not all-capable, so it cannot perform tasks unrelated to web authoring and potentially harmful. It is not as involved and complicated as a full-fledged programming language. However, its functionality has not been compromised. Text reuse, parameterization, recursion and states are supported.
- Versions and hypertext are considered the same in Noema. The flexibility gained by this syntax unification is very important. A version may contain hypertext in it (we saw that in the drop markup example where the `<CONTENT>` and the `<ABSTRACT>` versions are full-fledged hypertext) and parts of hypertext may be used (or compactly written) as versions. Versions can themselves be versioned.
- Noema allows for parts of the versioned document to have different versions. It also allows XML-like entities. The combination of these features offers everything needed for fully parameterized hypertext
- Noema can be viewed as a language that prescribes how a hypertext document is going to be, independent of the target language, that could be for example XML.
- Another idea introduced in this paper, somewhat orthogonal to the previous ones, is the concept of implicit dimensions. We saw that the effect of this is that entities within a document are given state. Since the number of entities invoked in a Noema document depends on the version, the version space is not definable in authoring time.
- Noema can be a substitute for TROFF macros. Its syntax is more readable than TROFF. In the Noema framework, there is but one language. In IML, if one needs more than what is already implemented, they might need to resort to a second language, ISE.

The next step should be the creation of a formal specification for the language and an efficient implementation. From the language design point of view there is more work to be done to study the strengths and weaknesses of the language and to discover new ways of using it or new properties of interest and areas of improvement. As far as applications of Noema are concerned, it seems that it has potential to be used as a versioning tool for XML, for example XML components of software systems, taking versioning back to its roots, where it was designed for software component versioning.

[8] This example specifies a series of slides found in jpeg files, where the name of the file for slide X is "$prefixX$.jpg", where $prefix$ is a parameter to the `slide` entity

References

[1] G. D. Brown. Intensional HTML 2: A practical approach, MSc Thesis, Computer Science Department, University of Victoria, 1998.

```
entity slide {
  par prefix 'PREFIX;
%%
  {
    <IMG src = "&prefix , get i(slide);.jpg" /><P/>
  } ,
  (link @ vmod i(slide):(get i(slide) + 1)):{ Next }
}
```

Figure 3: The slideshow in Noema

[2] World Wide Web Consortium. Extensible markup language (XML). Available on-line at http://www.w3.org/TR/REC-xml.

[3] m. c. schraefel and W. W. Wadge. Two cheers for the web. In M. Gergatsoulis and P. Rondogiannis, editors, *Intensional Programming II*, pages 31–39. World Scientific Publishing Co. Pte. Ltd, 2000.

[4] m. c. schraefel and W. W. Wadge. Complementary approach for adaptive and adaptable hypermedia. In *Intensional Hypertext. 3rd International Workshop on Adaptive Hypermedia.* Springer-Verlag, 2002. Forthcoming.

[5] J. Plaice and W. W. Wadge. A new approach to version control. *IEEE Transactions on Software Engineering*, 19(3):268–276, March 1993.

[6] P. Rondogiannis and W. W. Wadge. Intensional programming languages. In *Proceedings of the First Panhellenic Conference on New Information Technologies (NIT'98), Athens, Greece*, pages 85–94. National Documentation Center of Greece, October 1998.

[7] P. Swoboda. Practical languages for intensional programming. MSc Thesis, Computer Science Department, University of Victoria, 1999.

[8] B. Wadge, G. Brown, m. c. schraefel, and T. Yildirim. Intensional HTML. In E. V. Munson, C. Nicholas, and D. Wood, editors, *Principles of Digital Document Processing*, volume 1481 of *Lecture Notes in Computer Science*, pages 128–139. Springer-Verlag, 1998.

[9] W. W. Wadge. Intensional markup language. In P. Kropf, G. Babin, J. Plaice, and H. Unger, editors, *Distributed Communities on the Web*, volume 1830 of *Lecture Notes in Computer Science*, pages 82–89. Springer-Verlag, 2000.

[10] T. Yildirim. Intensional HTML, MSc Thesis, Computer Science Department, University of Victoria, 1999.