XConnector & XTemplate: Improving the Expressiveness and Reuse in Web Authoring Languages

Débora Christina Muchaluat-Saade¹,²
¹Departamento de Engenharia de Telecomunicações – UFF
R. Passo da Pátria, 156 – São Domingos – 24210-240, Niterói, RJ, Brazil
e-mail: deborams@telecom.uff.br, debora@telemidia.puc-rio.br

Luiz Fernando Gomes Soares²
²TeleMídia Lab – Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225 – Gávea – 22453-900, Rio de Janeiro, RJ, Brazil
e-mail: lfgs@inf.puc-rio.br

Despite recent efforts made by the W3C, web-authoring languages still need to be enhanced. Aiming at this goal, this paper presents proposals for improving their expressiveness and reuse. The proposals are based on an XML language called XConnector, which provides the creation of complex referential and multimedia synchronization relations. XConnector can be used for improving the expressiveness of either linking languages, such as XLink, or linking modules of hypermedia authoring languages, such as XHTML or SMIL. The novel contribution of this paper is another XML language called XTemplate, which provides the creation of hypermedia composite templates. A composite template specifies types of components, types of relations, components and relationships that a hypermedia composition has or may have, without identifying who all components and relationships are. Templates are traditionally used for improving reuse. Composite templates allow the definition of common structures, which can be seen as representing types of compositions with specific semantics given by the set of defined relationships. Therefore, composite templates could be used to provide new time containers in web languages, besides the well-known par, seq and excl provided by SMIL 2.0. The paper also presents how composite templates are used in the HyperProp hypermedia system and proposes an extension to XLink to incorporate facilities provided by XConnector and XTemplate, improving its expressiveness and reuse.

1. INTRODUCTION

Recent efforts have been made by the W3C – World Wide Web Consortium – in order to improve expressiveness in web authoring languages. As examples, the SMIL language (33) enables authoring of interactive multimedia presentations and the XLink recommendation (37) provides the creation of sophisticated links among XML (7) resources.

Another concern of W3C is augmenting language reuse. The newest recommendations of XHTML (35) and SMIL were proposed following a modular structure. This type of language structure allows the combination of modules generating distinct profiles. The XHTML+SMIL profile (36) is an example of reusing SMIL syntax and semantics for integrating timing into XHTML.

Despite those efforts, expressiveness and reuse in web languages still need to be improved. Hypermedia authoring languages must be rich in their semantic capabilities to express different kinds of relationships among document components. Besides the traditional referential relationship where a user selects an anchor to navigate to another part of a document, authoring languages should provide structuring relationships, which capture the logical structure of a document, such as a book and its
chapters, chapters and their sections and so on; and synchronization relationships, which define both
temporal and spatial ordering of document components. Other types of relationships that should also
be provided are discussed in reference (32). In order to support relationships with different semantics,
hypermedia languages must use different entities to represent each type of relation. Links and
compositions are the most used entities. If links alone were used for expressing all types of
relationships, a language would need to provide several different types of links. The same would
happen if compositions alone were provided. Thus, usually both compositions and links are used,
each of them representing different relationships.

Links may represent any kind of relation, such as referential relations as in XHTML, XLink and
SMIL, synchronization relations as in NCL (2), ancestor-descendant relations as in version support
systems (31), aggregation relations as in Labyrinth (5), semantic relations, etc. This work focuses on
the use of links for representing referential and synchronization relationships. The traditional
referential relationship, usually represented by a link, can be considered a special case of a causal
synchronization relationship, specifying that an action must be fired when a condition is satisfied. In
this particular case, the triggering condition occurs at a non-deterministic time instant: the user
selection of an anchor. Therefore, the definition of hypermedia links can be generalized to model
other types of synchronization relationships, as it will be seen later on this paper.

On the other hand, compositions are the most used approach for defining synchronization
relationships. Usually, each composition type has a specific semantics. This approach makes the
authoring task easier, since the author can specify with a single composition what would be
alternatively specified using several links (27). However, most languages offer a limited set of
compositions to capture synchronization semantics. In SMIL, for example, there are only three
temporal composition types, which are parallel, sequential and exclusive. As a consequence, complex
relationships must be built through a hierarchical nesting of basic compositions.

The document logical structure can also be captured by the use of nested compositions. For example,
a book can be represented by a composition that contains other compositions representing its chapters
and so on. Some compositions will contain not only composite nodes but also some media nodes. A
more general approach is to allow compositions to specify relationships, defined by links, among its
nodes. When compositions contain nodes and links, contextualization of links will be possible, that
is, defining different sets of links to the same node, depending on the nesting of compositions the
node is in (31, 32). Furthermore, when composition nesting is provided, link “overlay” becomes
possible, providing the addition of new links to a pre-existent document structure. As an example,
suppose a composition that represents chapter one of an electronic book (composition $B_1$). For a new
book given to a reader (composition $B_2$), it could be desirable to introduce a relationship between
sections one and two of chapter one, to give a hint of related matters; for another advanced reader,
the original book should be delivered without that relation. Note that composition $B_2$ could be
defined as a new composition containing $B_1$ (reusing its pre-existent structure) and introducing the
new link (32). This type of structuring composition is not provided by any of W3C standards. As
stated by Hardman (11), although space-time dependent compositions are very useful, a hypermedia
model should also provide space/time-independent composition, allowing an application to be
consisted of more than one multimedia presentation fragments, but without merging them into the
same time base. Using only space-time dependent compositions, the authoring language obliges the
document logical structure to match its presentation structure, what sometimes may not be desirable.

This paper presents proposals for improving expressiveness and reuse in web authoring languages
using links to represent referential and synchronization relationships and compositions to represent
grouping of nodes and links, with logical structuring and synchronization purpose. It recognizes the
importance of having compositions with synchronization semantics, or even other different types of semantics. One of the main contributions of the paper is to show how composition semantics can be obtained through the use of templates. Through composite templates, compositions with different desired semantics can be created and authors do not need to be limited to a predefined set of compositions offered by the authoring language, as usual. Moreover, in the case of templates representing temporal synchronization relationships, the presentation structure of a document may now be adapted to its logical structure, and not the opposite.

The definition of link used in the paper is based on the concept of hypermedia connector, already introduced by (18). A connector is a first-class hypermedia entity that allows the relation specification independent of which resources are related. Different links of the same type can be created reusing the same connector and defining different sets of participating resources. Previous work proposed a language for the definition of connectors, called XConnector (17), providing the creation of complex referential and synchronization relations. XConnector can be used either by linking languages, such as XLink, or by linking modules of other hypermedia authoring languages, such as XHTML or SMIL.

The concept of connector was brought from the Architecture Description Language (ADL) domain (30) to hypermedia (19). Following a similar approach, the novel contribution of this paper is applying the concept of ADL architectural styles to hypermedia, resulting in the so-called hypermedia composite templates. A composite template specifies types of components, types of relations, components and relationships that a hypermedia composition has or may have, without identifying who all components and relationships are. An XML language called XTemplate is proposed for the creation of this kind of templates. Although the main utility of templates is incrementing reuse in authoring languages, another significant advantage for using composite templates is the already mentioned capability of representing compositions with special semantics. From this point of view, if a language to create templates is provided, authors can create other templates representing different types of compositions and use them to introduce the desired semantics inside their documents. Thus, users can create the types of compositions they need, store them in libraries and reuse them whenever they want to. Using composite templates, the introduction of synchronization in existent languages (36, 23) becomes more natural, since any composite element can assume a specific synchronization behavior given by the template semantics. Conceptually, a composite template can specify any type of relationship among its components, however, in the current proposal, emphasis will be given to spatio-temporal relationships, since XTemplate uses XConnector to specify relations. As a validation of the ideas presented, this paper also shows how XTemplate was used in the HyperProp hypermedia system (32). Moreover, a further extension to XLink taking advantage of some facilities provided by XTemplate is also proposed.

The paper is organized as follows. As XConnector was used as a basis for this work, Section 2 summarizes reference (17) briefly describing its main concepts. Section 3 introduces the concept of hypermedia composite templates and shows how they can be created using the XTemplate language. Section 4 explains how an authoring language can use composite templates, showing how they were used in HyperProp. The same section also presents how XLink can be further extended to incorporate some of XTemplate facilities. Section 5 compares the proposal to related work. Finally, Section 6 is reserved to conclusions and future work.
2. THE XCONNECTOR LANGUAGE

The key for improving expressiveness and reuse in link specification is detaching the definition of relations from the definition of relationships and giving both first-class statuses. Most hypermedia models and languages treat relationships as first-class entities, but few of them treat relations the same way. Following the concept of hypermedia connector proposed in (18), a separate entity called \textit{xconnector} is used for defining the relation. As an xconnector does not specify document components, another entity is needed to define the relationship, which is the hypermedia link referring to that xconnector.

Figure 1 shows an xconnector named \textit{R}, representing a relation with three different participant roles. For example, xconnector \textit{R} may represent a relation specifying “if a participant anchor is selected (first role) or the presentation of another participant finishes (second role), start the presentation of a third participant (third role)”. The figure also illustrates two different links, \textit{l}_1 and \textit{l}_2, using \textit{R}. A link is defined by an xconnector and by a set of binds relating xconnector roles to resources (node anchors or attributes) (18). While xconnector defines the relation type, the set of binds specifies the interacting resources. Link \textit{l}_1 defines three binds connecting resources \textit{A}, \textit{B} and \textit{C} to the roles of xconnector \textit{R}, meaning that “if an anchor of \textit{B} is selected or the presentation of \textit{A} finishes, start the presentation of \textit{C}”. Link \textit{l}_2 also defines three binds, but it connects a different set of resources (\textit{B}, \textit{C} and \textit{D}), meaning that “if another anchor of \textit{B} is selected or the presentation of \textit{C} finishes, start the presentation of \textit{D}”. Links \textit{l}_1 and \textit{l}_2 define different relationships, as they relate different node sets, but represent the same relation type, as they use the same xconnector. One may consider links as instantiations of xconnectors.

XConnector (17) is an XML-based language providing the definition of xconnector elements. Each xconnector defines a set of roles (that will be played by participating resources in a relationship) and how they actually interact, through a child element called \textit{glue}. The definition of roles is based on events that happen during document presentation. Mouse clicks, beginning, ending or pausing the presentation of document components, changing values of attributes of document components, etc. may be used to define roles. The glue specifies interaction among roles. Suppose an author wishes to specify a causal multipoint relation such as “if a certain part of a video is playing and an icon is selected, pause the video and start the presentation of an XHTML form”. This relationship is a classic example in an interactive TV program, where the viewer may order a pizza during a movie play. In order to specify this relationship, a causal xconnector with two condition roles and two action roles is required. One condition role would specify that a presentation event is occurring (video is playing) and the other would specify that a mouse click event occurred (icon was selected). One action role would specify action pause over a presentation event (pause the video) and the other would specify action start over another presentation event (start the presentation of an XHTML form). The glue defines a compound trigger expression, which relates both condition roles with the AND logical operator, specifying the condition that should be satisfied in order to fire a compound action expression, which relates both action roles with the parallel (\textit{par}) operator. Besides causal relations,
as the examples previously given, XConnector also provides the specification of constraint relations, which have no causality involved.

XConnector syntax uses several constructs from existing W3C standards, such as XHTML and SMIL, and introduced some new ones (17). However, the concept of event used in the definition of roles is the one defined by (24), that is, an event is an occurrence in time that can be instantaneous or can occur over some time period. So, different types of SMIL and XHTML events, which are always instantaneous, are treated as transitions of a same XConnector event. For example, SMIL beginEvent and endEvent correspond to transitions starts and stops in the XConnector presentation event state machine. Readers should refer to (17) for details about XConnector. The complete XML Schema (39) definition of XConnector is available at http://www.telemidia.puc-rio.br/specs/xml/XConnector.xsd.

As the definition of connectors is not simple to be done by ordinary users, because they would need the notion of event states and transitions, the idea is to have expert users defining xconnectors, storing them in libraries (connector bases) and making them available to others for creating links. An example of connector base unambiguously specifying the causal or constraint semantics of Allen’s temporal relations (1, 6) can be found at http://www.telemidia.puc-rio.br/specs/xml/allen_base.xml.

The main feature of XConnector is that it can be used in conjunction with hypermedia linking languages regardless of how they define their document components. Reference (17) proposed how XLink can be extended to use XConnector facilities.

3. HYPERMEDIA COMPOSITE TEMPLATES

Specifying relations using xconnectors can be seen as a way of defining link templates. This idea can be generalized to composite templates. The concept of composite templates came from ADL architectural styles (30), which represent families of software systems with common properties. Specific software architectures following a style, that is, pertaining to a family of systems, inherit all definitions made by the style and consequently also satisfy all its properties. Styles provide a way of reusing common or well-known structures.

As defined in Section 1, a hypermedia composition is made up by components representing media objects or other compositions and relationships among them, possibly represented by links. A composite template specifies types of components, types of connectors and interface points; that a composition has or may have, without identifying who all components and relationships are. This task is responsibility of specific hypermedia compositions using the template.

XTemplate is an XML-based language that provides the definition of hypermedia composite templates. The complete definition of XTemplate using XML Schema is available at http://www.telemidia.puc-rio.br/specs/xml/XTemplate.xsd.

The definition of hypermedia composite templates is similar to the definition of ADL styles, adding the possibility of defining specific instances of components and connectors. The definition is done in two distinct parts, where only the first one is required:

1. vocabulary: defining types of components, types of connectors and interface points;
2. set of constraints on vocabulary elements and on the inclusion of component instances and relationships among components.
The main advantages of using templates are reusing structure and embedding semantics into a composition. As previously mentioned, SMIL (33) provides three types of compositions with temporal semantics, which are par, seq and excl. These types of composition could be seen as templates, as shown in Figure 2. This figure shows a composite template with the sequential temporal semantics (same as SMIL seq) using links to specify synchronization behavior among its components. This template is used by a composition containing components A and B. The final document generated after the template is processed will contain A, B and also synchronization links giving the sequential semantics.

Through composite templates, compositions with different desired semantics can be created and authors do not need to be limited to a predefined set of compositions offered by the authoring language. Conceptually, a composite template can specify any type of relationship among its components, however, in the current proposal, emphasis will be given to spatio-temporal relationships that can be specified using XConnector.

In order to illustrate how a composite template is created, assume a composition representing the temporal synchronization among an audio object (music or lecture) with its corresponding subtitles (music lyrics or lecture text) and also the logo of the copyright-owner company. Figure 3 illustrates the structural and temporal view of the example. Relation type $L$ specifies that “the start of a presentation event causes the start of another presentation event”. Relation $L$ is used to create relationships between the audio and the logo (link $L_1$) and also between each audio track and its corresponding subtitle (links $L_2 \ldots L_{n+1}$). Relation type $P$ specifies that “the end of a presentation event causes the end of another presentation event”. Relation $P$ is used to create relationships between the audio and the logo (link $P_1$) and again between each track and its corresponding subtitle (links $P_2 \ldots P_{n+1}$).

If this hypermedia composition is specified as a template, the same temporal specification can be reused by different audio tracks with their corresponding subtitles. When a specific composition uses this template, it should refer to the template and instantiate the audio node, its tracks and the text nodes (subtitles) assigning the respective labels defined in the template. After the composition using the template is processed, all relationships defined by the template will automatically be created. Section 4 gives an example of how this template is used by a specific document, reusing the synchronization behavior. Prior to Section 4, the following three subsections present in more details how a template is specified, using the same example as illustration.
3.1 Specifying the Template Vocabulary

A composite template can define specific types of components, such as text, audio, video, image, composite, etc. Each type may be assigned a label attribute and also minimal and maximal number of instances.

Labels are used to distinguish component types and specify constraints and relationships among them. If it is necessary to specify relationships among specific anchors/attributes of a component type, a template can determine labels for these interface points, called ports (18).

In the template vocabulary, relation types (connectors) used for creating links among components can also be defined. Each connector is specified by an src attribute set to the URI of an xconnector (see Section 2), its label and the minimal and maximal number of instances.

Using the example of Figure 3, the template vocabulary is given in Figure 4. Three types of components are defined: an “audio” type that has just one instance, but may have several ports called “tracks”; a text type called “subtitle” that may have several instances; and an image called “logo” that has just one instance. Besides components, two connectors are defined (L and P), considering that they are stored in a library called “connector_base.xml”.

```xml
<xtemplate id="audio-with-subtitles">
  <vocabulary>
    <component label="audio" type="audio" maxOccurs="1">
      <port label="track" maxOccurs="unbounded" />
    </component>
    <component label="subtitle" type="text" maxOccurs="unbounded" />
    <component label="logo" type="image" maxOccurs="1" />
    <connector src="../connector_base.xml#L" label="L" maxOccurs="unbounded" />
    <connector src="../connector_base.xml#P" label="P" maxOccurs="unbounded" />
  </vocabulary>
</xtemplate>
```
3.2 Specifying Template Constraints

The specification of template constraints is done in three distinct parts:
- a) constraints on components, connectors and interface points,
- b) definition of component instances identifying specific resources,
- c) definition of connector instances defining specific links among components.

The vocabulary specifies that template components must satisfy certain types and number of elements. However, additional constraints are required. For example, a template can specify constraints on components, connectors and interface points, indicating that there is no other type of component besides the ones defined in the vocabulary, or that two types of components must have the same number of occurrences, etc. XPath (38) expressions are used for this purpose, generalizing the types of constraint that can be specified. The use of XPath also makes the validation of constraints easily done with XSL transformations (40).

Turning back to the example of Figure 3, three constraints can be defined, as shown in Figure 5. Each constraint has a description attribute that can be used as an error message if the constraint is not satisfied in the XSL transformation process.

```xml
<xtemplate id="audio-with-subtitles">
  ...
  <constraints>
    <constraint select="count(child[@label!='audio' AND @label!='subtitle' AND @label!='logo']) = 0" description="All components must be audios, subtitles or logos"/>
    <constraint select="count(child[@label='audio']/child[@label='track']) = count(child[@label='subtitle'])" description="The number of tracks must be equal to the number of subtitles"/>
    <constraint select="count(child::link[@xconnector='L']) = count(child::link[@xconnector='P'])" description="The number of links of type L must be equal to the number of links of type P"/>
    ...
  </constraints>
</xtemplate>
```

Figure 5. Example of template constraints

Different from ADL styles, hypermedia composite templates can also define specific component and connector instances. Therefore, a composite template can determine specific web resources as component instances and specific links as connector instances. These definitions are also done inside the set of constraints of a composite template.

The definition of a specific component instance is done by resource elements specifying the content URI; the type of component, which must be set to a label already defined in the vocabulary; and another label for this specific resource. As example, consider the definition of a specific copyright-owner logo given in Figure 6 for the same template “audio-with-subtitles”.

```xml
<xtemplate id="audio-with-subtitles">
  ...
  <constraints>
    ...
</xtemplate>
```

Figure 6. Example of template substraints
The definition of specific connector instances (links) gives the behavior semantics of a composite template, since they provide specific temporal or spatial synchronization behavior among components. In the definition of links, components and connectors are identified by labels specified in the template vocabulary or in the definition of specific component instances (label attribute of a resource element).

As connectors should be specified using XConnector (see Section 2), each link defines a set of binds relating component labels to connector roles. In a bind specification, a connector role is identified by its id, and the template component(s) is (are) identified by an XPath expression that selects it (them), using its (their) label(s). Variable declaration instructions (variable), iteration instructions (for-each) and even XPath functions added by XSLT (40), as for example current(), can be used for specifying template relationships.

Figure 7 presents the definition of links of the same template example illustrated in Figure 3, where each link identifies the label of an xconnector declared in the vocabulary.

```xml
<resource src="http://www.telemidia.puc-rio.br/img/logo.jpg" type="logo" label="logotele"/>
...
</constraints>
</xtemplate>

Figure 6. Example of defining a specific web resource as a template component instance

3.3 Other Considerations about Composite Templates

Templates can be defined as a specialization of other templates, inheriting all definitions and also introducing new ones. Let us introduce this feature through an example.

As discussed in (27), parallel and sequential composition semantics may be given by links specifying synchronization relationships. A parallel composition with identical SMIL par semantics represented
by synchronization links is shown in Figure 8. Link $L_{in}$ causes the start of the presentation of the composition components triggered by the start of the presentation of the composition itself. Link $P_{out}$ causes the stop of the presentation of the components triggered by the stop of the presentation of the composition. Figure 10 illustrates the template that represents this case. Note that any type of component may be included in the composition, since the template does not determine any specific type.

Figure 8. Parallel composition represented by links

Figure 9. Parallel (a) finished by a specific component; (b) finished by the first; (c) finished by the last.

Now, xtemplate par can be extended to add the semantics of finishing the composition by the end of some specific component. In this case, a link causing the stop of the composition presentation triggered by the specific component presentation end is added, as illustrated by $P_{sync}$ link in Figure 9(a). In the case of a composition finishing by the first or last component, links $P_{first}$ and $P_{last}$, represented respectively in Figures 9(b) and 9(c), need to be added. Figure 11 illustrates xtemplates par-sync, par-first and par-last representing the three types of compositions previously mentioned. Note that all of them extend xtemplate par. The xtemplate par-sync defines a special component labeled “endsync” to represent the specific component that determines the stop of the composition.

Figure 10. xtemplate par

Figure 11. xtemplates par-sync, par-first and par-last

Now, xtemplate par can be extended to add the semantics of finishing the composition by the end of some specific component. In this case, a link causing the stop of the composition presentation triggered by the specific component presentation end is added, as illustrated by $P_{sync}$ link in Figure 9(a). In the case of a composition finishing by the first or last component, links $P_{first}$ and $P_{last}$, represented respectively in Figures 9(b) and 9(c), need to be added. Figure 11 illustrates xtemplates par-sync, par-first and par-last representing the three types of compositions previously mentioned. Note that all of them extend xtemplate par. The xtemplate par-sync defines a special component labeled “endsync” to represent the specific component that determines the stop of the composition.
Finally, it is worth to mention that one of the goals of XTemplate is to provide compositionality, allowing the definition of interface points and the nesting of templates. In the current proposal, however, compositionality was limited to one template interface point, which represents the whole composition.

4. HOW A COMPOSITE TEMPLATE IS USED

In order to illustrate how a composite template can be used by an authoring language, this section presents how XTemplate was used as a module of the NCL language (2), the declarative language of the HyperProp hypermedia system (32). Different from SMIL, NCL composite nodes have no embedded presentation semantics. A composite node can contain other composite nodes and/or simple media objects and also links among them. NCL media objects and anchors are defined similarly to how they are in SMIL.

Suppose we would like to create a document containing an audio synchronized with its lyrics, exactly with the same synchronization relationships as described in the structure shown in Figure 3 (see Section 3). The audio is a Brazilian samba music coded in a wave file and each part of the samba lyrics is a separated html file. Using NCL, this document would be described as illustrated in Figure 12. A composite node identified by “samba-document” makes reference to xtemplate “audio-with-subtitles”. Each component of the composite node declares a label attribute, whose value must be set to the corresponding label defined in the template. The document contains four media objects, one audio and three text objects. Just three music tracks and their corresponding lyrics were used to
shorten the example. This document should also contain information about spatial layout, although it is omitted in the figure.

```xml
<composite id="samba-document" xtemplate="http://www.telemidia.puc-rio.br/specs/xml/examples/templates/audio-with-subtitles.xml">
  <audio id="samba" label="audio" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/samba.wav">
    <area id="part1" begin="8.4s" end="18s" label="track"/>
    <area id="part2" begin="18.5s" end="28s" label="track"/>
    <area id="part3" begin="29s" end="39s" label="track"/>
  </audio>
  <text id="lyrics-part1" label="subtitle" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics01.htm"/>
  <text id="lyrics-part2" label="subtitle" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics02.htm"/>
  <text id="lyrics-part3" label="subtitle" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics03.htm"/>
</composite>
```

Figure 12. NCL document example using xtemplate “audio-with-subtitles”

In order to generate the final NCL document to be played by the HyperProp formatter, an XTemplate processor was implemented. The main purpose of the XTemplate processor is transforming an xtemplate into a traditional XSLT stylesheet, allowing a standard XSLT processor, such as Saxon (15), to generate the final NCL document. The final document contains the same composite node with its components and also automatically generated components and links. If any error occurs during constraint validation, an error report is generated. The complete processing of an NCL document using a composite template is illustrated in Figure 13.

Figure 13. Complete processing of an NCL document using an xtemplate

The final NCL document generated from the previous example is shown in Figure 14. Note that this document does not need to be handled by the final user, but only internally by the HyperProp formatter. Elements and attributes generated automatically after the template processing are shown in
NCL links are defined using hypermedia connectors whether or not the document uses templates.

```
<composite id="samba-document">
  <audio id="samba" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/samba.wav">
    <area id="part1" begin="3.4s" end="4s"/>
    <area id="part2" begin="10s" end="18s"/>
  </audio>
  <text id="lyrics-part1" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics01.htm"/>
  <text id="lyrics-part2" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics02.htm"/>
  <text id="lyrics-part3" src="http://www.telemidia.puc-rio.br/specs/xml/examples/samba/lyrics03.htm"/>
  <img id="logotele1" src="http://www.telemidia.puc-rio.br/img/logo.jpg"/>
  <linkBase>
    <link xconnector="/connector_base.xml#L">
      <bind role="source" component="samba"/>
      <bind role="target" component="logotele1"/>
    </link>
    <link xconnector="/connector_base.xml#P">
      <bind role="source" component="samba"/>
      <bind role="target" component="logotele1"/>
    </link>
    <link xconnector="/connector_base.xml#L">
      <bind role="source" component="samba" port="part1"/>
      <bind role="target" component="lyrics-part1"/>
    </link>
    <link xconnector="/connector_base.xml#P">
      <bind role="source" component="samba" port="part1"/>
      <bind role="target" component="lyrics-part1"/>
    </link>
    <link xconnector="/connector_base.xml#L">
      <bind role="source" component="samba" port="part2"/>
      <bind role="target" component="lyrics-part2"/>
    </link>
    <link xconnector="/connector_base.xml#P">
      <bind role="source" component="samba" port="part2"/>
      <bind role="target" component="lyrics-part2"/>
    </link>
    <link xconnector="/connector_base.xml#L">
      <bind role="source" component="samba" port="part3"/>
      <bind role="target" component="lyrics-part3"/>
    </link>
    <link xconnector="/connector_base.xml#P">
      <bind role="source" component="samba" port="part3"/>
      <bind role="target" component="lyrics-part3"/>
    </link>
  </linkBase>
</composite>
```

Figure 14. Final NCL document example after processing xtemplate "audio-with-subtitles"

If we import the final “samba-document” into the HyperProp authoring system, its graphical structural view will be the one shown in Figure 15. In the figure, number (2) near the arrows means that each arrow represents two links. After playing the document using the HyperProp formatter, Figures 16, 17 and 18 show three snapshots of the presentation when each of the three text nodes representing the first three parts of the music lyrics are presented in synchronization with the audio
(see the audio control progress bar). Although not shown in the paper, the document spatial layout specification positioned the image logo at the right side of the same window of the music lyrics.

The same way XTemplate and XConnector were used as modules of the NCL language, they could be used in conjunction with other authoring languages. The next section shows how XLink could use some of XTemplate facilities.

Figure 15. Graphical structural view of the final “samba-document” in the HyperProp browser.

Figure 16. First snapshot of the “samba-document” presentation.

Figure 17. Second snapshot of the “samba-document” presentation.

Figure 18. Third snapshot of the “samba-document” presentation.

4.1 Applying Templates to Improve Reuse in XLink

XLink (37) allows describing sophisticated relationships, with an arbitrary number of participating resources, through extended links. An extended link is composed by a set of participants and a set of traversal rules. Each participant has a label attribute that is used in the specification of traversal rules. An XLink traversal rule (arc-type element) relates a source label to a target label, besides specifying when navigation must occur and how target node presentation must be done. Navigation may be triggered by user request or may be done automatically when the starting resource is loaded, providing a very simple type of synchronization link. As several participants may share the same label, it is possible to define multipoint relationships.

XLink provides the definition of link repositories, called linkbases, facilitating link management by gathering several linking elements. The standard also provides semantic attributes to describe the meaning of resources within the context of a link, but they are out of the scope of this paper.

Despite the fact that XLink allows describing more sophisticated relationships than simple HTML hyperlinks, it also has limitations as a hypermedia linking language, as discussed in (17). The usefulness of extending XLink with XConnector is that it improves XLink expressiveness, allowing the specification of synchronization relationships, it maintains the possibility of defining relationships separately from the linked resources and it adds the possibility of having relation
libraries (connector bases) besides linkbases. Moreover, it also facilitates link maintenance, since modifying an xconnector automatically updates all links referring to it.

The extension to XLink proposed by (17) suggests a new arc-type definition that can comprise the current XLink arc-type definition or, instead, refer to an xconnector and associate its roles to resource labels. This is indeed a significant difference from XLink, allowing the definition of the relation type separated from the relationship. In the new arc-type element definition, XLink attributes from, to, show and actuate become deprecated and a new attribute, named xconnector, used for identifying a valid xconnector URI, is introduced. Moreover, arc-type elements must define a set of bind-type child elements. Each bind-type element must have a label attribute to identify an XLink participant label and a role attribute to identify one of the xconnector roles. This new arc-type element specification gives more flexibility to link definition because the number of labels associated to the same traversal rule is not limited to two, as it is in XLink. However, this is not an XLink limitation, because, as already mentioned, multipoint relationships can be done with several binary traversal rules.

This paper proposes a further extension to XLink to incorporate the concept of composite templates. Since an extended link may define several traversal rules, each extended link actually represents a set of relationships (hypermedia links) among participants, instead of just one. Thus, an extended link can be considered a hypermedia composition containing components and a set of links among them. The central idea of the new extension is to create a template to define a set of XLink traversal rules, independently of which resources are participants. Referring to the same template, several extended links defining different participating resources can reuse the same set of traversal rules. This will increase reuse in XLink specifications even more.

In order to provide the definition of templates, a new XLink type is proposed, called template. A template-type element is very similar to a standard extended-type element, but must declare an id attribute to allow being referred by an extended link. Template-type elements must declare child arc-type elements, specifying a set of traversal rules. Each arc-type element follows the extensions proposed by (17), having an xconnector attribute that refers to the URI of a hypermedia connector and declaring bind-type child elements, relating labels to roles. Figure 19 illustrates an example of XLink template definition representing the relationships among a course, the professor and its students.

```
<template xlink:type="template" xlink:id="courseload-template">
  <go xlink:type="arc" xlink:xconnector="xlink-replace-onRequest">
    <bind xlink:type="bind" xlink:xconnector="student" xlink:role="from"/>
    <bind xlink:type="bind" xlink:xconnector="professor" xlink:role="from"/>
    <bind xlink:type="bind" xlink:xconnector="course" xlink:role="to"/>
  </go>
  <go xlink:type="arc" xlink:xconnector="xlink-replace-onRequest">
    <bind xlink:type="bind" xlink:xconnector="course" xlink:role="from"/>
    <bind xlink:type="bind" xlink:xconnector="professor" xlink:role="to"/>
  </go>
</template>
```

*Figure 19. Example of XLink template-type element*

The definition of xconnector “xlink-replace-onRequest” used in the arc-type elements of Figure 19 is given in Figure 20. This xconnector has one condition role specifying that a mouse click event have to occur in order to fire action start over a presentation event, modeling the same behavior of a standard XLink traversal rule with attributes show and actuate respectively set to “replace” and “onRequest”.

15
One might think that the complexity to use XConnector is much higher than XLink. If you consider that most users would have to define xconnectors, this conclusion could make sense. However, the idea is to have expert users to write xconnector libraries representing several types of hypermedia relations, such as the one shown in Figure 20. Common users will just need to use them to create links, writing XML code like the one shown in Figure 19, which is very reasonable.

In order to use an XLink template, extended-type elements need a new attribute called xtemplate, which should refer to the URI of a template-type element. Following current XLink rules, each participant of an extended link must define a label that will be used in traversal rule definition. Thus, participants that define labels used by the template referred by the extended link inherit traversal rules defined in the template. An XLink extended link may continue defining other traversal rules independently of the ones defined by the template. Figure 21 shows two XLink extended links using the proposed extensions. These two links represent two distinct courses that reuse template “courseload-template” and declare labels “student”, “course” and “professor” for their participants, reusing the set of traversal rules already defined by the template.

Note that this extension proposed to XLink used just a subset of facilities provided by XTemplate. The initial goal of this extension was to allow traversal rule sets to be reused in several extended
links with the same navigation behavior among their participants, increasing even more the reuse in XLink definitions. Now it is also possible to define XLink template libraries, storing traversal rule sets independently of participating resources.

Therefore, in addition to the advantages pointed out in (17), extending XLink in order to incorporate the concepts of XTemplate and XConnector brings:

- Facility of use – Applying the same idea of linkbases and connector bases, template bases can be created and reused to define extended links.
- Facility of link maintenance – when an xtemplate is modified, all extended links referring to it are automatically updated.

5. RELATED WORK

Although still few works in the literature address XLink (3, 4, 10), standardizing linking constructs in XML languages is very important and useful. When analyzing the link definition in details, we can recognize that a link is defined by a relation type and by a set of related components. Although the definition is conceptually divided in two parts, hypermedia languages/models usually have a unique entity to capture both parts, the link itself.

The relation type captured by a link depends on the expressiveness of the authoring language used. In HTML\(^b\), links are uni-directional and single-headed representing the traditional hyperlink navigation behavior. SMIL (33) links are also uni-directional and single-headed, but they can be triggered by either user interaction or other triggering conditions, such as temporal events. In Dexter (9) and AHM (11), links may be bi-directional and multi-headed, but are used only to capture hypermedia relationships that depend on user interaction. Dexter has no support for temporal relationships and AHM provides point-to-point synchronization arcs to capture temporal constraints between two parts of a presentation. Labyrinth (5) and NCM (31, 32) provide the definition of complex multipoint relationships among components based on different concepts of event. In NCM, a link is defined as a causal or constraint relationship among events that happen over document nodes, analogous to the ideas used in XConnector. In fact, many of XConnector features came from NCM links. Reference (13) compares links and events as different approaches for defining activation and deactivation information of document components. NCM and XConnector combines both, since NCM links and xconnectors are defined as relationships among events that happen over document components. In Labyrinth, an event is treated as a separate model entity that can be attached to a link in order to describe causal relationships. A Labyrinth event defines a condition and a list of actions, similar to a causal xconnector behavior.

Hypermedia languages/models also differ in the way the link entity is treated. In HTML, for example, links are embedded in node content, preventing reuse of the same resource without inheriting previously defined links. Other models allow the definition of a link as an independent entity, such as Dexter, AHM, Microcosm (16), XLink, Labyrinth and NCM. In Dexter and AHM, links are considered a type of document component and, different from most models, links can even be used to link other links (11). In NCM, Microcosm and XLink, links can only connect nodes. In NCM, links are contained in composite nodes and must relate nodes recursively contained in the composite node. NCM composite nodes can be reused, but links alone cannot. Microcosm and XLink provide independent link repositories that can be reused in several documents.
Although most hypermedia languages/models provide a unique entity to capture relationship definitions, some of them already divide the definition in two parts, the first specifying the relation and the second, the related participants. In XLink, for example, the definition of participating resources and the definition of traversal rules are done with different child elements of an extended link. In NCM, a link is defined by sets of source and target endpoints and another attribute called meeting point, which specifies the compound traversal behavior. However, in both cases, traversal behavior is embedded in the link definition and cannot be handled independently.

Other hypermedia models provide predefined sets of relation types that can be used for creating relationships in a document. These models actually break the definition of a relationship in two parts and provide relation types separately from relation instances. Madeus (14) high-level constraints and the typed transitions of Petri-net based models like I-HTSPN (34) and caT (20) are examples. However, these models do not provide the definition of user-defined relation types.

Addressing relation type definition, reference (18) proposed another first-class entity, called hypermedia connector, whose main purpose is describing the relation independent of which participants are related. XConnector (17) uses and extends this idea and proposes an authoring language for defining connectors to represent referential and synchronization relations. XConnector alone does not provide the definition of links, as it only allows specifying the relation type. Another language must be used to complete the link definition, specifying its set of participants. That language can be XLink, following the proposal presented in (17), or any other hypermedia authoring language willing to take profit of XConnector facilities, as it was done with NCL (see Section 4).

Note that features like automatically computing link sources and destinations, offered by Microcosm generic links and the virtual links of NCM and Labyrinth, are concerns of the linking language and not of XConnector. Another issue is the definition of the link context (12). XConnector does not have the duty to define link specifiers, identifying component anchors, and consequently link specifier contexts, as proposed by AHM (11). This is responsibility of the linking language using XConnector.

The use of templates and its benefits in providing reuse in hypermedia authoring were already discussed in the literature (8, 22, 21, 28). However, the majority of work that discusses reuse in hypermedia does it in the design level and not in the specification/implementation level, as this proposal did. Reference (28) discusses use of design patterns and frameworks to improve the design of hypermedia applications based on the OOHDM model (29). Reference (21) presents a formalism called MCF – Media Construction Formalism – for the specification of multimedia scenarios based on a temporal model similar to SMIL 2.0, but again focusing on design problems.

The template proposed in this paper addresses the hypermedia document specification/implementation level and its main goal is to facilitate author work, allowing reuse of previous specifications and giving synchronization semantics to compositions. Nothing prevents that a design model is used and that, in the process of converting from the design model to the specification/implementation model, design patterns can be translated to composite templates. Reference (21) comments that authors tried to do this from MCF to SMIL and that they found difficulties because the design model was richer. This emphasizes the need to increase the expressiveness of hypermedia specification languages.

Reuse at the implementation level is also provided by caT (20), a Petri-net based hypermedia system that provides document structure reuse, besides content reuse. In caT’s representation, a transition in a higher-level net may be mapped to a separate subnet, and places in the higher-level net to places in the subnet, allowing the same definition of structure to be reused with distinct content elements. Although caT provides what they call templates, they are used to specify how content elements and
links, modeled as places and transitions, are displayed when a document is browsed on the Web. CaT’s template files are treated as simply another content type which is essentially an HTML page with placeholders indicating where information is to be embedded when the corresponding places and transitions are marked and enabled. This is very different from our composite template concept.

A work that is closer to this paper template proposal is discussed in (8, 22). It presents the concept of constructive templates, allowing automatic generation of template instances in a target document, providing reuse of well-known specific structures. Templates proposed by (8) allow the specification of causal relations among components through a “When” clause, which specifies a condition and an action to be executed with an optional delay. XTemplate proposal provides more expressiveness to define relationships, since it uses XConnector to specify multipoint relations with causal or constraint semantics. Besides that, XTemplate also permits the definition of constraints on template components, which is not offered by (8). An interesting feature of (8) is the possibility of defining variables that can be parameterized by template instances. This feature is seen as a future work in our proposal.

Comparing XTemplate methodology with SMIL 2.0 modularization, a profile called SMIL+XTemplate could be created, providing different types of time containers that could be used to author SMIL documents besides the known par, seq and excl. In order to support the definition of templates, XConnector and XTemplate modules should be added to the profile and two new language modules would have to be created, as follows. The first new module would comprise the definition of SMIL links using xconnectors. The other new module would provide the use of xtemplates. This latter module would create a new space/time-independent container in SMIL and add the xtemplate:label attribute to SMIL media objects, anchors and standard time containers. SMIL players would need to be adapted to handle xconnector synchronization specifications.

Other work should still be mentioned considering specification and validation of constraints. XCSL (26) – XML Constraint Specification Language – is a language for restricting XML document contents. It provides constructors that define actions triggered whenever a boolean expression, over the value of element content or attributes, evaluates to false. An XCSL processor generator is supplied to transform an XCSL document into an XSL stylesheet, allowing the use of a standard XSL processor to validate document semantics. Another reference (25) discusses the use of XPath and XSLT, besides XML Schema, to assert constraints on document content, applying a transformation-based approach to solving them. Similar ideas were applied in XTemplate. An xtemplate is transformed into an XSLT stylesheet and a standard XSLT processor is used for validating template constraints and for automatically generating specific components and relationships inside documents using the template.

6. CONCLUSIONS

This paper discussed how to improve web-authoring languages addressing the issues of expressiveness and reuse. It was based on previous work about XConnector (17), which can be used to extend existing linking languages or linking modules, in order to provide multimedia synchronization facilities for WWW resources using links. Reference (17) also proposed how XLink can be extended to incorporate XConnector facilities. The novel contribution of this paper was the definition of hypermedia composite templates provided by XTemplate. Composite templates allow specifying reusable composite structures possibly defining types of components and relations and how they are interconnected to give a special semantics for a composition. A further extension to
XLink was proposed incorporating few concepts of XTemplate, providing reuse of traversal behavior for extended links.

As a validation of the proposed ideas, XConnector and XTemplate were incorporated into the HyperProp hypermedia system (32) implementation. Using JAXP (Java API for XML Processing), an XML parser for XConnector was implemented, allowing users to import connector bases to the system and use hypermedia connectors to create causal and constraint NCM links. The HyperProp formatter was also adapted to understand and play documents with links using connectors. XConnector and XTemplate were incorporated as modules of NCL, the declarative language used in the HyperProp system. An XTemplate Processor was also implemented in Java to transform composite templates into XSLT stylesheets, allowing the use of Saxon (15), a standard XSL processor, to generate the final NCL document. After the final document is generated, it can be imported to the HyperProp authoring system using NCL parsers and then played by the formatter. A future work is implementing an XLink processor considering the extensions proposed in this paper, which can be done making some adaptations in the HyperProp formatter implementation.

The use of connectors also provides the possibility of defining relations as composite elements, augmenting even more the flexibility for expressing and reusing relation specifications. Composite connectors represent groups of several connectors and components, modeling more elaborated relationships among components of a document (18). Although composite connectors could be provided, they were not discussed in this paper, since their definition requires the definition of document components, which depends on the authoring language used.

Connectors could be used in a broader sense to represent any kind of relation, besides the synchronization relation types focused in this work. For example, we could have a version connector to represent the ancestor/descendant relation of versioned resources, or we could have a channel connector to represent the publish-subscribe paradigm used in notification mechanisms. A future work is to extend XConnector and create new connector bases to represent other kinds of relations found in the hypermedia domain. Based on these new connector bases, XTemplate could be used to create compositions with semantics different from those discussed for synchronization purpose. For example, we could have compositions to group versions of a same node and its derivation graph (31).

Another interesting future work is to combine XTemplate with the XHTML language, aiming at providing other time containers besides the well-known *par*, *seq* and *excl* introduced by XHTML+SMIL.

Another future work is introducing the possibility to parameterize some attribute values in XConnector and consequently in XTemplate. This would allow, for example, configuring delays in connectors with the same temporal synchronization semantics without the need to define distinct ones.

ENDNOTES

a  Note that the semantics of a *bind*-type element *role* attribute is different from the XLink homonym attribute.

b  HTML script programming was not considered for comparison.
REFERENCES


