XTemplate 4.0: Providing Adaptive Layouts and Nested Templates for Hypermedia Documents

Glauco F. Amorim(✉), Joel A.F. dos Santos, and Débora C. Muchaluat-Saade

MidiaCom Lab, Computer Science Department,
Fluminense Federal University - UFF, Niterói, Brazil
{gamorim,joel,debora}@midiacom.uff.br,
http://www.midiacom.uff.br

Abstract. A hypermedia composite template defines generic structures of nodes and links that can be reused in different hypermedia compositions. XTemplate is an XML-based language for the definition of hypermedia composite templates. XTemplate can currently be used to create templates for NCL documents, but other hosting languages can also be used.

In current versions of hypermedia document template languages, including XTemplate, there is no facility for defining template layouts. This work extends XTemplate, incorporating the concept of adaptive layouts. *Adaptive layouts* enable the definition of generic presentation characteristics for multimedia documents that are instantiated at processing time and adapted to the number of media objects declared in a given document that uses a template.

Another important facility that this work incorporates in XTemplate is hypermedia composite template nesting. *Template nesting* enables the inclusion of template components inside other hypermedia composite templates, thus making the use of multiple nested templates transparent to the document author that uses templates.

1 Introduction

Currently, XTemplate 3.0 [1] and TAL [2] are the main languages used to build hypermedia composite templates (or just template for simplicity) for multimedia documents specified with Nested Context Language (NCL) [3]. Both XTemplate and TAL are used to reduce the complexity of creating multimedia applications, mainly when the document contains a large number of media objects and synchronization relationships between these objects.

Although the use of templates reduce the authoring effort in defining the structure and synchronization of NCL documents, there are some points that need to be improved.

Hypermedia composite templates are used to define the generic structure of nodes and links for one single hypermedia composition. Therefore, whenever one wants to use multiple composite templates inside a document, it has to be done explicitly by both importing all desired templates and associating each (desired) composition to one imported template. The drawback of such a scenario is that

© Springer International Publishing Switzerland 2016
Q. Tian et al. (Eds.): MMM 2016, Part I, LNCS 9516, pp. 642–653, 2016.
DOI: 10.1007/978-3-319-27671-7_54

the author using a group of templates must have the knowledge about each template and how they can be put together. TAL allows a main template to declare inner templates, but the communication interface between them is not well-defined. XTemplate 3.0 allows template extension, but does not provide template nesting in the main template definition. Then, the first contribution of this work is to enable templates to be nested in another template, maintaining compositionality. Thus, for the scenario previously described, an author would just need to use one template that nests all other templates.

The second contribution of this work is related to multimedia documents layout. Typically, NCL applications involve the presentation of various types of media objects in devices ranging from smartphones to digital TVs. For each media object to be presented at a given device screen, its presentation characteristics include its spatial coordinates (x, y) on the player device screen, along with its size. One may notice that, as well as for relationships, the amount of presentation characteristics to be declared tends to grow when the number of media objects in a given application grows. The process of defining presentation characteristics in NCL in such a scenario is both cumbersome and prone to errors. Template authoring languages for NCL, such as XTemplate 3.0 and TAL, do not provide a generic way to define visual presentation characteristics. So, the second contribution of this work is to provide an approach for the generic definition of document presentation characteristics through the so-called *adaptive layouts*.

This work presents an extension to the XTemplate 3.0 language, called XTemplate 4.0. In this new version of XTemplate, we provide both facilities for defining *adaptive layouts* and *nested templates*. Through testing it was observed that this version reduces the authoring effort in creating and using templates created with the XTemplate language. This paper extends the work presented in [4], where an XML language for defining adaptive layouts was proposed.

The remaining of the paper is structured as follows. Section 2 describes both adaptive layout and template nesting features. Section 3 discusses related work together with comparison with XTemplate previous version. Section 4 presents XTemplate 4.0, and the features it adds to the language. Section 5 describes evaluation tests done with XTemplate 4.0 and discusses its results. Section 6 finishes the paper with conclusions and future work.

2 Adaptive Layouts and Nested Templates

2.1 Adaptive Layouts

NCL documents are (typically) composed by several media objects, where a (great) part of those objects are presented on a device screen. How media objects are presented in the screen is specified in an NCL document by *region-descriptor* pairs in the following way, a media object in NCL is represented by a *media* element. A media element refers to a *descriptor* element for defining how it will be presented on a device screen, such as its transparency, sound level (if applicable), navigation specification, and the screen area it will occupy. The latter is defined by referring to

a *region* element. Although such separation is intended to improve definition reuse - several *media* nodes may refer to the same *descriptor* and several *descriptors* may refer to the same *region* - when specifications are different and no reuse is possible, the number of *region* and *descriptor* elements is the same as the number of *media* elements.

What we call **adaptive layout** is an approach that enables document authors to create generic presentation characteristics that adapt itself to the number of media objects in a given document, thus diminishing authoring effort regarding how media objects will be displayed.

An adaptive layout relies on (possibly several) **layout models**, where a layout model specifies general directives for media object presentation. For example, a layout model may specify to place objects inside a grid.

Layout model instances are declared by **layout components**. A layout component specifies the device screen area where media objects referring to it should be placed, besides specific information about its layout model. For example, a layout component using a grid layout model specifies its number of lines and columns. A layout component can also have as children one or more **layout items**. A layout item declares a subset of presentation characteristics to be associated to media objects, for example a specific size. Media objects, therefore, may refer to specific layout items inside a layout component or to the layout component as a whole.

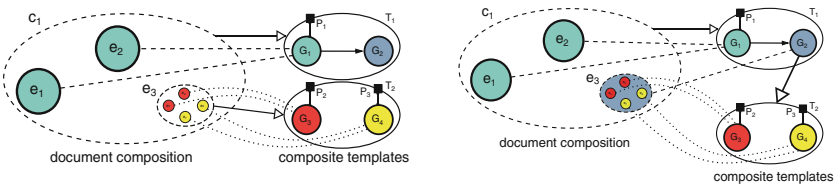
A **layout template** defines one or more layout components. Hypermedia composite templates define *components* that represent groups of media objects. While using a composite template, an author associates specific media objects to generic template components by labeling media objects (*xlabel* attribute) with a given template component identification. Likewise, specific media objects can be associated to layout components by labeling them (*layout* attribute) with a given layout component identification. Alternatively, a specific media object can be associated to a template component that, by its turn, is associated to a layout component.

2.2 Template Nesting

Composite templates define generic structures to be inherited by a composition that uses it. It means that synchronization specification provided by a composite template will take into account the child nodes of a given composition. Suppose that a composition c_1 declares three elements e_1, e_2 and e_3 inside it. Composition c_1 uses composite template T_1, which defines components G_1 and G_2 along with a synchronization specification relating both components.

Suppose, for example, element e_3 is another composition nested in c_1. If a document author also wants to embed semantics into e_3, this has to be done by making e_3 use another composite template (T_2). This approach is depicted in Fig. 1a.

An NCL author creating a document with multiple compositions, therefore, is supposed to associate all compositions that are intended to use templates to their corresponding templates. The drawback of such a scenario is that an author



a) Nested composition using composite templates b) Template component using nested template

Fig. 1. Composite templates without and with nesting template

using a group of templates must have knowledge about each template separately and how they can be put together. Once the user of composite templates is not supposed to have previous knowledge about the template authoring language (in our case XTemplate), using more than one template in the same document can be difficult for him.

What we call **template nesting** is an approach where template components can be associated to other templates. Therefore when a document composition is associated to a given template component, it will have the embedded semantics specified in the composite template that component uses. This idea is depicted in Fig. 1b. Notice that a hypermedia composite template cannot nest itself.

In order to ensure communication between two templates (let's call them the external and the internal templates), it is necessary to establish a communication interface. This interface is defined by *port* elements. The internal template must declare in its vocabulary one or more *port* elements to work as its communication interface. In the external template, the same *port* elements are declared inside the component using the internal template.

3 Related Work

Some works that address the creation of documents based on templates are discussed in LimSee3 [5] and Stamp [6].

The LimSee3 [5] model uses templates for multimedia document authoring. It defines an authoring language, independent of the existing languages, focusing on the logical structure of a document and not in the target language element semantics. With this approach, LimSee 3 provides great flexibility and reuse. The template created with that language describes a generic hierarchy among its components. A template has to be edited in order to become a complete document or another template.

The STAMP model [6] proposes a solution for the adaptation of multimedia presentations. That model is applied when the presentation content comes from a database, focusing on web systems. STAMP uses templates for the automatic generation of presentations. The presentation model and structure can be adapted according to the number of elements retrieved from the database.

However, STAMP does not work with template extension and cannot embed spatio-temporal semantics into a composition.

XTemplate 3.0 [1] is the current version of XTemplate. It is an expressive template authoring language, and it also allows for the definition of presentation (layout) characteristics for template components. However, XTemplate 3.0 does not have features such as nesting templates and generic specifications of media object presentation characteristics, as the concept of adaptive layouts. This work extends XTemplate 3.0 including both new facilities.

TAL [2] is described by its authors as a modular declarative language that supports the specification of templates: incomplete hypermedia compositions. As XTemplate, TAL can define a set of documents that share the same specification for their compositional structure. However, TAL does not allow the specification of layout information for its components, nor provides any facility to create generic definition of document presentation characteristics. TAL has nesting templates as one of its features, but it does not define template interfaces for nesting templates, so template compositionality is not satisfied.

In [7], the authors propose a system for creating and presenting grid-based documents to suit various viewing conditions and content selection. The system can display static content or dynamic content from several different sources. A set of templates is proposed for layouts that were inspired by models of traditional newspapers. Each template organizes a collection of content in the region determined by the model. In another work [8], the authors present a solution for publishing interactive digital documents, which is based on document authoring instead of programming. The solution is generic and describes how the definitions of templates and variable content elements can be used to decrease redundancy and increase the flexibility for those applications. Some ideas proposed in that related work to define adaptive layouts were used in the model proposed in our work.

There are tools that facilitate the definition of layout features in web documents, such as the CSS Regions Module [9]. The CSS Regions module allows content from one or more elements to flow through one or more boxes called CSS Regions. Besides, it allows dynamic magazine layouts that are flexible in placement of boxes for content flows.

Although those layout managers are proposed for different types of applications, the idea of organizing the layout suggested by them matches the dynamic management of layouts and, thus, were used as a basis for the adaptive layout modeling proposed in this paper.

4 XTemplate 4.0

XTemplate 4.0 keeps the facilities added to the template authoring language by XTemplate 3.0 and incorporates two new facilities for defining *adaptive layouts* for a given template and also *nesting templates*, so that multiple templates can be used together (in a main template).

Different interactive multimedia applications created by the NCL community, available at the NCL Club[1], were analyzed in order to determine different useful layout models to be provided. After analyzing them, we identified presentations that resembled the characteristics described by *FlowLayout* and *GridLayout* managers in Java language. Other layouts found did not have a well-defined structure or could be represented by compositions of *FlowLayout* and *GridLayout*. XTemplate 4.0 currently provides two layout models, which are:

- **FlowLayout:** layout items are placed from left to right, row by row. When there is no more space in a row considering the size of a given layout item, another row is created below it and the same principle is used again.
- **GridLayout:** layout items are placed from left to right, row by row, in a grid format. The grid is always built considering the whole area specified by the layout component.

XTemplate 4.0 adds a new element called *layoutBase* where the layout components for a given template are declared. A layout component declaration is done through element *layout*. The *layout* element has a child element called *format* that defines the area the layout component covers by its attributes *width*, *height*, *top* and *bottom*. Layout item arrangement is specified with attribute *align*, while items spacing is specified by attributes *hspace* and *vspace* of element *format*.

The layout model a given layout component instantiates is declared through attribute *type*. The *layout* element has specific attributes for each kind of layout model it instantiates. A *layout* element with type *flowLayout* does not need to define a number of rows and columns, on the other hand, it has to define the alignment for layout items. A *layout* element with type *gridLayout* defines a number of rows and columns, but does not define layout item alignment or size. Layout items are declared by the *item* element. Whenever a layout component instantiates a *flowLayout* model, it is possible to include different sized items in the same layout component. So each *item* element can define its own *width* and *height*.

Besides size and positioning, a layout component is capable of describing the navigation behavior among its items and between components. Each *layout* element may define a *focus* child element with a *focusIndex* attribute that determines a unique navigation index for that element. Whenever the *focus* element is defined for a given *layout* element, it establishes the possibility of navigation among its items. Therefore, media objects referring to a given *layout* element will be associated to navigational definition, depending on their relative position to each other.

Navigation among layout components is declared through attributes *moveUp*, *moveDown*, *moveLeft* and *moveRight* of element *focus*. It is not mandatory to have all those attributes, but at least one is necessary for establishing navigation among layout components. Each attribute indicates the *focusIndex* of the layout component to receive focus when the corresponding remote control

[1] http://clube.ncl.org.br.

(or keyboard) key is pressed while that layout component is in focus. Whenever navigation among layout components is established, layout items in the border of a given layout component will inherit a given navigation attribute from the layout component, thus allowing navigation to items in other layout components. For example, items in the bottom border of a layout component will inherit the *moveDown* attribute. The ones in the left border will inherit the *moveLeft* attribute and so on. It is important to notice, however, that when navigation among layout components is not established, the default behavior takes place, i.e. navigation from one border moves to the opposite border of the same layout component (cyclic navigation).

In addition, it is possible define media exhibition parameters in the layout component. Suppose, for example, that a media object should be reproduced with 90 % transparency. Then, it is necessary to include a child element, called **layoutParam**, in the related layout item. This parameter is represented by a tuple $< name, value, item >$.

While creating a layout template, the author associates layout components to template components through its new *layout* attribute, whose value is the layout component *id*.

Template nesting is achieved by extending the *component* element with a new *xtemplate* type. Thus, a *component* element with *xtemplate* type contains the same structure of the internal template it references. The internal template a *component* element references is indicated along with its type as *xtemplate/alias*, where *alias* is the unique identification of a given template in the template base. To enable using components of a nested template, the *component* element declares *port* child elements with the same *xlabels* as ports defined in the vocabulary of the internal nested template. An example of template nesting is presented in Sect. 4.2.

It is possible to define more than one level of nesting among templates. The template processor will process nesting in a recursive way, from the most inner to the root template element.

4.1 Template Processing

Template processing is done in two main steps. At the first processing step, elements referencing a given template component will be associated to synchronization relationships related to that component and declared in the template *body*. If a given template component represents a nested template, composite elements referring to that component are associated to the given template and processed before continuing the main template processing. At the end of the step, media objects inherit from template components their reference to layout components. At the second processing step, each layout component is translated to an NCL *region* representing the whole area declared by it. Each layout item is translated into a *region* with the same size and positioning attributes and an NCL *descriptor* with the same navigational attributes. Media objects are associated to *region-descriptor* pairs representing layout items in the order they are declared in the NCL document, i.e. the first media object declared in the NCL

document will be associated to the first *region-descriptor* pair and so on. When no more *region* can be created inside the *region* representing the whole layout component, media objects are associated to existing *region-descriptor* pairs starting from the beginning. If a specific *item* element of the layout component is indicated, media objects are associated to a specific *region-descriptor* pair.

4.2 Template Example

To illustrate the new facilities provided by XTemplate 4.0, we modified the template "quiz.xml" presented in [1]. The "quiz.xml" template helps creating an interactive quiz that will be presented during a video presentation. "quiz.xml" uses the "screen.xml" template for presenting a question and its possible answers on the screen. Between a screen component and its successor, there are "change_screen" links. Those links are responsible for checking if one of the color keys of the remote control is pressed (red, green, yellow and blue). Once one of those keys is pressed, the link stops the presentation of the current screen and starts its successor. That link also passes a value representing the key pressed to a program (Lua counter node), which is a script written in the Lua language. That node will be responsible for testing if the answer is correct or not. If so, it updates one of its variable values, counting the correct answers.

Using XTemplate 3.0, where no layout components are available, "screen.xml" had to declare four answer types, because each answer will be presented in a different screen region (one below the other). Using XTemplate 4.0, just one answer component has to be defined in "screen.xml" using the *flowLayout* model, regardless of the number of answers for a question. This template was modified in the following way: (i) each *screen* component now represents the nested template "screen.xml"; (ii) "change_screen" links where modified to check if a given answer was selected and pass its position to the Lua counter node; and (iii) template "screen.xml" declares a *layout* element with type *flowLayout* for presenting possible answers one bellow the other together with their navigation attributes, regardless of the number of answers provided.

File "layout.xml" defines two layout components to be used for a question and its related answers. The question is presented on top of the screen (centralized) and answers are presented one bellow the other (centralized). Listing 1.1 presents a fragment of the definition of answer layout components.

```
1 ...
2 <layout id="ansFl" type="flowlayout">
3    <format align="center" height="640" hspace="10" left="10" top="300" vspace="0"
       width="200" zIndex="3"/>
4    <item id="c1" height="150" width="200" />
5    <focus focusIndex="1"/>
6 </layout>
7 ...
```

Listing 1.1. "layout.xml" layout components

File "screen.xml" defines how the question and answers for each question are presented. This template is nested inside the *quiz* template, thus it defines port *portAnswer* to work as its communication interface. A fragment of this template vocabulary is presented in Listing 1.2. As aforementioned, the template will be

processed in a recursive way, from the most inner to the root element. Then, when the "screen.xml" is processed, each *media* element with *answer* label will receive a *port* element. This element will be related to a *portAnswer* label to identify the communication interface.

```
<vocabulary>
   <port xlabel="portAnswer"/>
   <component xlabel="question" layout="lay#qstFl"/>
   <component xlabel="answer" layout="lay#ansFl"/>
</vocabulary>
<body >
   ...
     <variable name="i" select="1"/>
     <for-each select="child::media[@xlabel='answer']">
         <port id="port" select="current()" xlabel="portMenu"/>
         <variable name="i" select="$i + 1"/>
     </for-each>
     ...
</body>
```

Listing 1.2. "screen.xml" vocabulary fragment

File "quiz.xml" represents the main template. It nests template *screen* (alias "scn") for its *screen* component. To enable referencing nested template components, the *screen* component declares a *port* child element also named *portAnswer*. A Fragment of "quiz.xml" is presented in Listing 1.3.

```
<component xlabel="screen" xtype="xtemplate/scn">
 <port xlabel="portAnswer"/>
</component>
```

Listing 1.3. "quiz.xml" fragments

An example of NCL document using the *quiz* template is presented in Listing 1.4. Notice that, different from the example in [1], the NCL author only has to use the *quiz* template. The *context* element refers the nested template declaring a *screen* label.

```
<body xtemplate="quiz">
 <context id="screen_01" xlabel="screen">
   <media id="question_01" xlabel="question"/>
   <media id="answer_01_A" xlabel="answer"/>
   <media id="answer_01_B" xlabel="answer"/>
   ...
 </context>
</body>
```

Listing 1.4. NCL document (fragment) using *quiz* template

5 XTemplate 4.0 Evaluation

We performed tests to evaluate the *adaptive layout* feature. The test involved five activities, where in each activity authors should write code for one or more layout components and NCL code using layout components to define the application's presentation characteristics. The authors had to specify the layout template document in each activity and use this template within the NCL document.

Activity 1: authors should place media objects in a grid with three columns and two rows in the center of the screen; *Activity 2:* authors should place media objects in a flow with one item in the bottom of the screen; *Activity 3:* authors should place media objects in a flow with two different item sizes at the top of

the screen and a grid with four columns and one row in the bottom; *Activity 4:* authors should place media objects along two grids, one in the left and one in the right side of the screen, both with one column and four rows, and one flow in the bottom; *Activity 5:* authors should create any placement they wanted.

A total of 20 students participated in the current study: seven students of a high-school technical course in Telecommunications (Group 1) and thirteen students of a computer science graduation course in (Group 2). All students in Group 1 were enrolled in a digital TV applications course, therefore, authors had a good knowledge about NCL. No student in Group 2 had any knowledge about NCL, but had good knowledge about HTML language, which is also XML-based.

After completing all five activities, authors filled out a questionnaire with ten questions, nine related to the following cognitive dimensions [10]: *visibility*, *role-expressiveness*, *closeness of mapping*, *verbosity*, *premature commitment*, *hidden dependencies*, *error-proneness*, *consistency* and *viscosity*, and one to define a final score. Each question should be answered with a score of one to ten, which was used to evaluate the facility. The overall result of the tests can be seen in Fig. 2a.

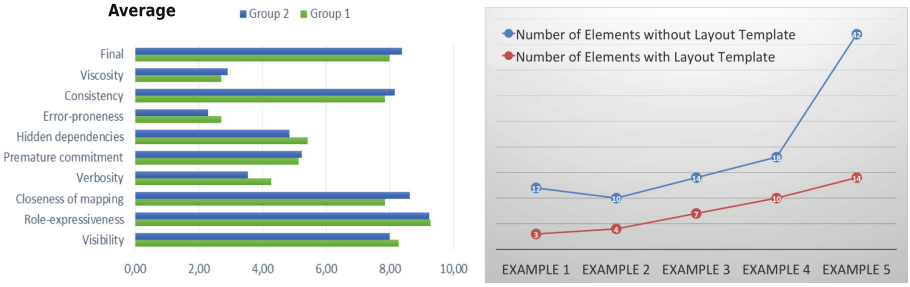
The first interesting conclusion about test results is that, although the two groups had different knowledge about NCL, results are quite similar in both groups. This result shows that the author experience with the adaptive layout authoring language is not influenced by the author expertise about the hosting multimedia authoring language.

One main concern was not to increase *verbosity* when using adaptive layouts. As it can be seen by test results, this goal was achieved. Another important result is in *role-expressiveness* and *visibility* dimensions. Average results indicate that the *adaptive layout* feature is self-explaining and therefore easy to use. Combined with the results for *error-proneness* and *viscosity* dimensions, we can conclude that the new feature does not insert any difficulty for those who want to use it, leading to new authoring mistakes, even with non-expert authors.

Although the evaluation results for both *hidden dependencies* and *premature commitment* dimensions were not very high, it was better than expected. *Hidden dependencies* is explained because of the navigation feature provided by layout components. Both the *id* and *focus* attributes of a layout component are used to define NCL descriptor focus index. This information needs to be provided to authors using that feature. *Premature commitment* is explained because authors did not know that the relative position among layout components in the layout template does not interfere in the final positioning of media objects in the device screen.

In general, the *adaptive layout* feature has been well evaluated. However, some adjustments can be made to improve its use. Suggestions given by test subjects include providing a graphical tool to create layout components and to provide other more sophisticated layout models. Both suggestions are work in progress.

A graphical editor that supports templates is very useful for helping document authors using templates [11]. When template nesting is available, a graphical editor can make the use of several nested templates transparent to document authors.



a) Usability test results

b) Number of elements test results

Fig. 2. Test results

Using only textual edition, the document author will not be able to understand clearly the difference between using separate templates and using one main template with nested templates. Because of this, we did not apply usability tests to evaluate the nested template facility with subject groups 1 and 2. However, we intend to run those tests when a graphical editor that supports nested template is available.

In addition, another evaluation about the number of language elements used for creating templates with and without adaptive layouts was made. The five scenarios used in the previous tests were considered and the results can be seen in Fig. 2b.

The number of template language elements increases considerably when the complexity of the example document increases. This result is expected because they involve the creation of more region-descriptor pairs. With the use of layout models, this effort is mitigated because the author only needs to set the proper model and, for each model, define a few required elements.

In order to evaluate nested template authoring, several XTemplate examples using more than one template were redesigned to use the template-nesting feature. Given the small changes necessary to use that feature for those examples and our experience about authoring templates, we state that nesting templates do not increase the complexity when authoring templates.

6 Conclusions

This paper presented a new version of the XTemplate language, called XTemplate 4.0. In this new version, two new features are provided, *adaptive layouts* and *template nesting*. Both features represent the main contribution of this paper.

Adaptive layouts enable the definition of generic presentation characteristics for multimedia documents that are instantiated at processing time and adapted to the number of media objects declared in a given document. As a test case, we provided two layout models, *GridLyout* and *FlowLayout*, that are implemented for the NCL language. It is important to notice that this feature is not provided

by any other NCL template language. XTemplate 4.0 usability test results indicate that the *adaptive layout* feature is intuitive and does not increase verbosity.

Template nesting enables the association of template components to other templates. Our solution for template nesting specifies interface points (ports) for nesting templates, which does satisfy composionality for authoring hypermedia composite templates.

A future work is to develop a graphical editor that supports nested templates to help the NCL document author when using templates. With textual edition, the difference between using several different templates and one main template with nested templates is subtle and might not be clearly understood by template users. That explains why we have not run usability tests about using nested templates yet.

Another future work is to include new layout models and perform evaluations for each new model provided and develop a graphical editor to help specifying adaptive layouts for multimedia documents.

References

1. dos Santos, J.A.F., Muchaluat-Saade, D.C.: XTemplate 3.0: spatio-temporal semantics and structure reuse for hypermedia compositions. MTAP **61**(3), 645–673 (2012)
2. Neto, C.S., Pinto, H.F., Soares, L.F.G.: TAL processor of hypermedia applications. In: DocEng, pp. 69–78. ACM (2012)
3. Recommendation ITU-T H.761, Nested Context Language (NCL) and Ginga-NCL for IPTV Services (2011)
4. Amorim, G.F., dos Santos, J.A.F., Muchaluat-Saade, D.C.: Adaptive layouts for authoring ncl programs. In: 19th WebMedia. ACM (2013) (in Portuguese)
5. Deltour, R., Roisin, C.: The limsee3 multimedia authoring model. In: DocEng, pp. 173–175. ACM (2006)
6. Bilasco, I.M., Gensel, J., Villanova-Oliver, M.: STAMP: a model for generating adaptable multimedia presentations. MTAP **25**(3), 361–375 (2005)
7. Schrier, E., Dontcheva, M., Jacobs, C., Wade, G., Salesin, D.: Adaptive layout for dynamically aggregated documents. In: 13th IUI, pp. 99–108. ACM (2008)
8. Signer, B., Norrie, M.C., Weibel, N., Ispas, A.: Advanced authoring of paper-digital systems. MTAP **70**(2), 1309–1332 (2014)
9. W3C, CSS Regions Module Level 1. http://www.w3.org/TR/css-regions-1/
10. Blackwell, A., Green, T.: HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann, San Francisco (2003)
11. Mattos, D., Silva, J., Muchaluat-Saade, D.: Next: graphical editor for authoring NCL documents supporting composite templates. In: EuroITV (2013)