Extending NCL to Support Multiuser and Multimodal Interactions

Álan L. V. Guedes[1,2]
alan@telemidia.puc-rio.br

Roberto G. de A. Azevedo[1,2]
razevedo@inf.puc-rio.br

Sérgio Colcher[1,2]
colcher@inf.puc-rio.br

Simone D. J. Barbosa[1]
simone@inf.puc-rio.br

[1] Department of Informatics, PUC-Rio
[2] TeleMídia Lab, PUC-Rio
[1,2] Rua Marques de Sao Vicente, 225
Rio de Janeiro, RJ, 22451-900, Brasil

ABSTRACT

Recent advances in technologies for speech, touch and gesture recognition have given rise to a new class of user interfaces that does not only explore multiple modalities but also allows for multiple interacting users. Even so, current declarative multimedia languages—e.g. HTML, SMIL, and NCL—support only limited forms of user input (mainly keyboard and mouse) for a single user. In this paper, we aim at studying how the NCL multimedia language could take advantage of those new recognition technologies. To do so, we revisit the model behind NCL, named NCM (Nested Context Model), and extend it with first-class concepts supporting multiuser and multimodal features. To evaluate our approach, we instantiate the proposal and discuss some usage scenarios, developed as NCL applications with our extended features.

CCS Concepts

• **Human-centered computing→Hypertext / hypermedia**
• **Applied computing→Markup languages**

Keywords

Multimodal interactions; Multiuser interactions; Multimedia Languages; Nested Context Language; NCL; Ginga-NCL

1. INTRODUCTION

Multimedia languages—e.g. HTML, SMIL [5], and NCL (Nested Context Language) [26]—are declarative programming languages for specifying interactive multimedia presentations. Traditionally, they focus on synchronizing a multimedia presentation (based on media and time abstractions) and on supporting user interactions for a single user, usually limited to keyboard and mouse input. Recent advances in recognition technologies, however, have given rise to a new class of multimodal user interfaces (MUIs) [28], as well as novel ways to allow the system to be aware of multiple users, interacting with each device.

In short, MUIs process two or more combined user input modalities (e.g. speech, pen, touch, gesture, gaze, and head and body movements) in a coordinated manner with output modalities [22]. An individual input modality corresponds to a specific type of user-generated information captured by input devices (e.g. speech, pen) or sensors (e.g. motion sensor). An individual output modality corresponds to user-consumed information through stimuli captured by human senses. The computer system produces those stimuli through audiovisual or actuation devices (e.g. tactile feedback). Bolt's seminal work on MUIs, "Put-That-There" [4], illustrates a MUI system. He proposes to use gestures and voice commands for planning tactical activities of military units over a map. The user can move a military unit by: (1) pointing his finger to a unit on the battlefield while saying "put that"; and then (2) pointing his finger to the desired location and saying "there".

Specifically regarding multiuser interactions, but without considering multimodal features, Stefik [27] proposes the early paradigm of WYSIWIS (What You See Is What I See). This paradigm enables users to collaborate in face-to-face meetings using the same GUI (Graphical User Interface) across multiple users' screens. More recently, Tabletop [20] and Distributed User Interfaces (DUI) [8] research has been studying multiuser interactions over shared GUIs. Increasing the number of interacting users, however, does not necessary imply that the system has become able to identify or distinguish each of them, i.e., it does not mean that the system is aware of multiuser interactions [13]. For instance, in a "shared screen" scenario (e.g. WYSIWIS, tabletop) even when multiple users are interacting with the system, they are handled as if there was only one interacting user. Truly multiuser applications are those in which the system can distinguish, and the programmer is aware of, the different users that are interacting with the system.

In this paper, we aim at extending the NCL multimedia language to take advantage of multimodal and multiuser scenarios—e.g., involving shared displays (e.g. public displays, tabletop, and TV) and many simultaneous users interacting using different modalities. NCL aims at specifying interactive multimedia presentations, but, as will be discussed in the remainder of the text, its current version does not allow authors to adequately create presentations for the aforementioned scenarios.

To achieve our goal, we propose to include first-class abstractions in the language for: (a) describing agnostic input/output modalities; (b) describing user and groups of users; and (c) relating the interacting users and the input/output modalities. Regarding multimodal features, our proposal builds on the previous work of Guedes et al. [11]. Whereas some previous work focused on multimodal interactions [6, 7, 11, 32, 38] and others on multiuser interactions [10, 12, 19, 25], to the best of our knowledge, our proposal is the first to support both multimodal and multiuser features in a multimedia language.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
WebMedia '16, November 08-11, 2016, Teresina, PI, Brazil
© 2016 ACM. ISBN 978-1-4503-4512-5/16/11 $15.00
DOI: http://dx.doi.org/10.1145/2976796.2976869

For the presentation and discussion of our proposal, the paper continues as follows. Section 2 briefly describes basic scenarios we want to support. Section 3 discusses related work. Section 4 presents an overview of the model behind NCL: the Nested Context Model (NCM) [24]. Section 5 presents our solution and highlights the extensions we propose to both the NCM model and the NCL language. Section 6 discusses how one can implement the scenarios of Section 2 using our proposal. Finally, Section 7 brings our conclusions and discusses some future work.

2. ENVISAGED SCENARIOS

Based on the previously mentioned Bolt's "Put-That-There" system, Figure 1 presents the three categories of scenarios we aim to support, named: "Put-That-There", "I-Get-That-You-Put-It-There", and "Anyone-Get-That-Someone-Else-Put-It-There". We discuss each of them in what follows.

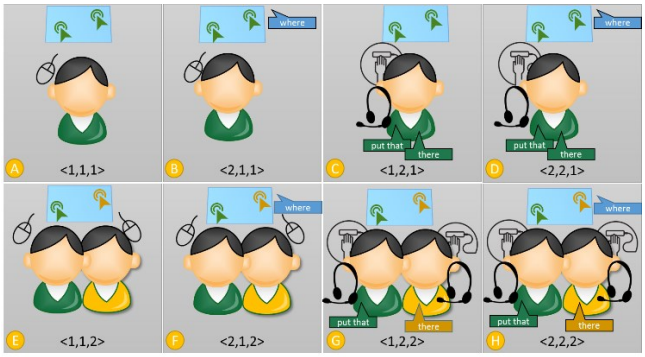

Figure 1: Scenarios based on Bolt's Put-that-there with additional output/input modalities and numbers of interacting users. Descriptions at the bottom of each scenario follow the scheme: *<number of output modalities, number of input modalities, number of interacting users>*.

The "Put-That-There" category is illustrated in Figures 1-A to 1-D. In this scenario, there can be different input and/or output modalities for a *single user* interacting with the system. In Figure 1-A, the user interacts using a mouse and gets feedback on a screen (one input and one output modality). Figure 1-B extends A with voice feedback (one additional output modality). In Figure 1-C, the user interacts using gestures and voice commands (two input modalities and one output modality). Figure 1-D extends 1-C with voice feedback (two input and two output modalities); it is similar to the original "Put-That-There". Note that these scenarios focus on multimedia applications supporting multimodal input/output for a *single user* only.

The "I-Get-That-You-Put-It-There" category is illustrated in Figures 1-E to 1-H. It is similar to the previous "Put-That-There", but the task must be done by *two different users*, who are uniquely identified by the system (e.g. User1 and User2). Figure 1-E shows each user interacting through individual mice. Figure 1-F extends Figure 1-E with voice feedback. Figure 1-G shows each user interacting through gestures and voice commands. Figure 1-H extends Figure 1-G with voice feedback (i.e., it is similar to the original "Put-That-There", but for *two users*).

Finally, the "Anyone-Get-That-Someone-Else-Put-It-There" category extends the "I-Get-That-You-Put-It-There" scenario with users that are *intentionally defined* during runtime. For instance, one may define that the first interaction (i.e. point and say "put that") is done by any user and the second one (i.e. point and say "there") by another user, who must be different from the first one.

3. RELATED WORK

Given the scenarios envisaged in Section 2, in this section, we discuss related work that focus on specifying multimodal or multiuser interactions. With regards to multimodal interactions, we limit the discussion to works that extend multimedia languages. With regards to multiusers, we discuss works on multiuser interactions in gaming contexts and others that indirectly address multiuser interactions by supporting distributed user interfaces.

Carvalho et al. [6, 7] and Guedes et al. [11] propose some level of multimodal interactions for NCL.

Carvalho et al. [6, 7] integrate VoiceXML elements inside an NCL document. In [7], the VoiceXML code is inserted into the <port> NCL element; whereas in [6] it is inserted into the <link> element. Voice recognition events are mapped onto keyboard-based events of NCL. Besides only supporting speech recognition, these works compromise the separation between the structure and the content, which is favored by the NCL model.

Differently, Guedes et al. [11] do not compromise the separation between the content and the structure of NCL by proposing a new <input> element that handles input modalities in an agnostic way. Indeed, Guedes et al. can implement the single-user multimodal scenarios defined in Section 2 (Figures 1-A to 1-D). Our work builds on Guedes et al.'s proposal to support multimodal interactions and extends it to support multiuser interactions. Moreover, we revisit the NCM model and discuss both multimodal and multiuser supports in the model behind NCL.

W3C (2001) [38] and W3C (2012) [32] propose multimodal interactions in HTML.

Similar to Carvalho et al., W3C (2001) directly integrates VoiceXML elements into the HTML document. Voice recognition events are controlled in an imperative manner using JavaScript code. That work, however, supports only one additional modality, the audio modality.

To support other input modalities, W3C (2012) [32] proposes a complementary specification, named SCXML (State Chart XML) [37]. SCXML is a state machine-based language responsible for combining multiple input modalities; HTML is responsible for presenting the output modalities.

Microsoft [19] and Google [10] propose multiuser support in gaming contexts. Both enable multiuser interactions by imperative APIs to handle gamepad controllers. Microsoft supports multiuser in DirectX applications by the XInput controller API. Similarly, Google supports multiuser interactions in Android applications by the GamePad API. These both APIs use callback events with an identification parameter informing the source controller.

Guerrero-Garcia [12] and Soares et al. [25][3] support multiple users through distributed user interfaces, respectively, in UsiXML and NCL.

UsiXML [18] is a task-oriented GUI description using an MDE (Model-Driven Engineering) approach to be deployed to different device configurations (e.g. desktop, web, and mobile). Guerrero-Garcia [12] extends UsiXML by modeling the coordination of multiple users in task-oriented systems. In particular, the work models GUIs for *group tasks* in UsiXML, in which users or groups of users can interact with one another. Then, each grouping task is deployed to each device of a user or a group of users.

Soares et al. [25] propose a hierarchical distribution of media in NCL. The distribution specification uses the abstraction of *device groups*, called *device class*. The author of a multimedia application

distributes it by sending and orchestrating the media presentation for the different device groups. When sending a media (e.g. image) to a device class, Soares et al. do not define how the different users who may interact with the devices can be identified. Indeed, an expected interaction over a media object in a specific device class will be triggered when any of the users do it. Soares' work—which specified fixed device classes (*passive* and *active*)—is extended by Batista et al [2, 3]. In [3], the authors define new *device classes* using a description based on UAProf [21] description. In [2], they propose that the author may use a document variable called *child.index*, which can be consulted by the author inside each NCL document sent to each device.

4. NCM 3.0

NCM (Nested Context Model) [24] is the conceptual model behind NCL. Figure 2 partially[1] illustrates the NCM entities; the main are: *Node*, which represents information fragments; and *Link*, used to define relationships between *Nodes*.

An NCM *Node* is defined by its *Content*, a *Descriptor*, and a list of *Anchors. Content* is the collection of information of a node. *Descriptor* defines properties for how a node should be exhibited, such as position for graphics, frame rate for video, and volume for audio. The *Anchors* in the list of *Anchors* can be of two types: *ContentAnchor*, which represents portions of the *Content* of the node; or *AttributeAnchor*, which represents a property of the node. *ContentNode* and *CompositeNode* are specializations of *Node* and detail the semantics of *Content* and *ContentAnchor*. The *Content* of a *ContentNode* or its subclasses (e.g. *TextNode, AudioNode, VideoNode*) are audiovisual media objects (e.g. text, audio, video, respectively). *ContentAnchor*s of *ContentNode* are spatial or temporal portions of these media objects. For instance, the *Content* of a *VideoNode* may be defined via references to a video file (e.g. file URI) and its temporal anchors via references to the presentation times.

There are other subclasses of *Node* that are not directly related to media content. *ApplicationNode* represents imperative code (e.g. a Lua script [23]). *TimeNode* defines timers useful in temporal relationships. *SettingsNode* is a *Node* with a list of global variables as properties. It does not need to be manually started and is available for every *Link*. These variables are useful for authors to consult environment characteristics (e.g. *screenSize*) or store auxiliary values. Finally, a *CompositeNode* has a collection of *Node*s as *Content*. *CompositeNode* also defines a list of *Ports*, which map to internal *Nodes* (or a *Node's Anchor*) of the *CompositeNode*. Ports and anchors have similar purpose and extend a common type named *Interface*.

A *Link* is defined by a *Connector* and a list of *Bind*s. *Connector*s define the link semantics, independently of the participating *Nodes*. More precisely, a *Connector* defines the relation between *Roles* and not between specific *Nodes*. When instantiating a *Connector*, one *Link* must define the association (i.e. *Bind*) of each connector *Role* to a node interface (*ContentAnchor*, *AttributeAnchor* or *Ports*).

A *Role* is defined by the attributes: *RoleID*, *EventType*, *minCardinality,* and *maxCardinality*. *RoleID* must be unique in the *Connector*. *EventType* refers to a specific event related to an *Anchor.* NCM 3.0 has three main event types: *PresentationEvent*, meaning the exhibition of a *ContentAnchor*; *AttributionEvent*, meaning the modification of a node property (*AttributeAnchor*); and *SelectionEvent*, meaning a mouse click or key-based event while a specific *ContentAnchor* is occurring. Also, events have attributes. The "state" attribute defines the occurrence status of the event and may have the values "sleeping", "paused" or "occurring". The "occurrences" attribute means the number of times that the event has passed by the "occurring" state during the multimedia presentation. In particular, the "selection" event has a *key* attribute that defines the key (e.g. keyboard or remote control) that may trigger the event.

Whereas a connector may represent any kind of relationship between anchors of nodes, NCL 3.0 defines only the *CausalConnector* relationship. *CausalConnector* specifies that conditions (*ConditionRole*s) shall be satisfied to trigger actions (*ActionRoles)*.

A *ConditionRole* can be a *SimpleCondition* or a *CompoundCondition*. *SimpleCondition*s act over *Node*s and may test the occurrence of an event (e.g., when the event state changes to "occurring"). *CompoundCondition*s represents a logical expression (using "and" or "or" operators) through *SimpleCondition*s and *AssessmentSatement*s. An *AssessmentSatement* is used to compare event attributes values, e.g. using a comparator (e.g. =, ≠, <, ≤, >, ≥). For instance, *SelectionEvent* has a key attribute, which can be tested in *SimpleCondition* or *AttributeAssessment*.

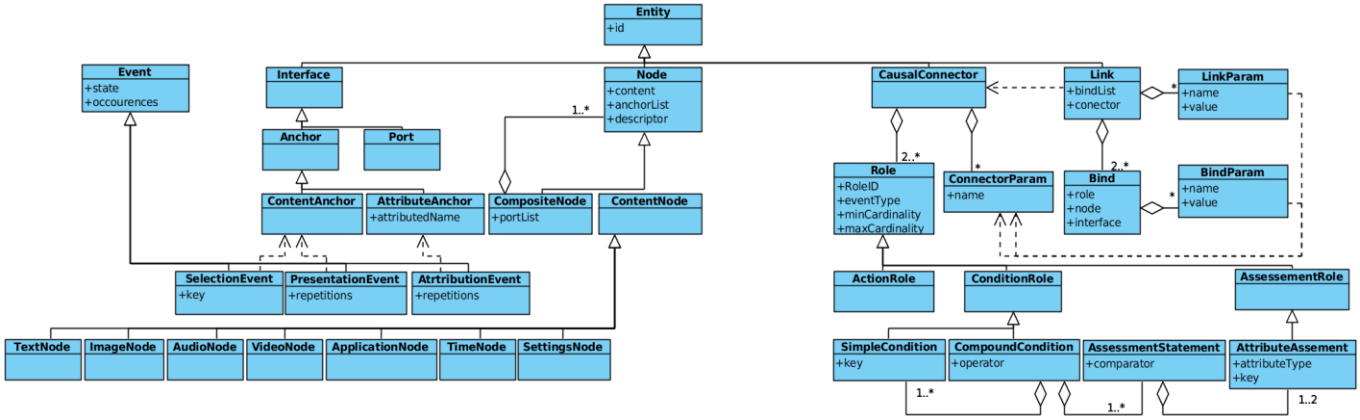

Figure 2: NCM 3.0

[1] Some NCM entities are omitted from the discussion here for simplicity (e.g. *Descriptor, DescriptorSwitch*, *SwitchPort*, *Rule, Interface,*) or because they are not used in NCL 3.0 (e.g. *ConstraintConnector, ConstraintGlue*). For a complete description of NCM, we refer the reader to [24].

Finally, *ActionRole*s may be used to change the presentation state of *Anchor*s or the properties of a *Node*, using an attribution event.

Using only the current NCM concepts, authors cannot represent modalities different from the GUI-based ones, neither can they be aware of the users who interact with the application. In the next section, we present our approach to extend NCM aiming at addressing these limitations.

5. PROPOSED APPROACH

To achieve both multimodal and multiuser features in NCM, we extend it by defining:

- a new subclass of *ContentNode*, named *MediaNode*. This class is reserved to represent output modalities. Previous audiovisual-oriented *ContentNode*s (e.g. *Text, Image*, and *Video)* and other proposed types of output modalities (e.g. *TTSNode*) became *MediaNode* specializations;
- a new subclass of *ContentNode*, named *RecognitionNode*. This class is used for representing input modalities, different from the previous ones, which are limited to keyboard/mouse. Examples *RecognitionNode* specializations include *SpeechRecognitionNode*, *GestureRecognitionNode*, *InkRecognitionNode*;
- a new kind of *Anchor*, called *RecognitionAnchor*, and an associated "recognition" event type. Two specializations of "recognition" event include *PointerEvent* and *KeyEvent;*
- a new *UserClass* entity, aiming at representing a *group of users* interacting with the multimedia presentation;
- new event attributes ("user_id" and "excluded_user_id") for *Link*-related entities (e.g. *SimpleCondition* and *BindParam*) which allow the specification of complex behaviors for multimodal and multiuser presentations.

Figure 3 brings our main modifications in NCM highlighted in light green. The remainder of this section details each of our proposed extensions to the model, whereas Section 5 is reserved to discuss how they are instantiated in NCL.

5.1 Multimodal Specializations of *Node*

As previously mentioned, the *ContentNode* entity may be specialized to different output modalities. It has been specialized mainly for 2D or 3D [1] audiovisual media modalities, such as *TextNode, ImageNode,* and *VideoNode*. In our proposal, we group the audiovisual modalities as specializations of the new *MediaNode* class, which is itself a specialization of *ContentNode.* Also, we propose three new *MediaNode* specializations for representing output modalities: (1) *TTSNode,* representing a TTS (Text-To-Speech) content (as described in W3C SSML [36], for example), useful for visually impaired users; (2) *AvatarNode,* representing embodied conversational agents (e.g. described using BML [29]), useful for deaf-, children- or elderly-oriented interfaces ; and (3) *SensorialNode,* representing sensorial effects (e.g. described in MPEG-V SEDL [15]), useful for increasing the QoE [9]of the multimedia presentations.

For the representation of input modalities, we propose the new *RecognitionNode*, which is a subclass of *ContentNode* and can be used in *Link* relationships. The *Content* of a *RecognitionNode* is also a collection of information. Different from the *MediaNode*, however, the information is expected to be captured, not presented. Some examples of *RecognitionNode* specializations include: (1) *SpeechRecognitionNode*, used for speech recognition, such as recognizing words and phrases spoken by the user(s); (2) *GestureRecognitionNode*, used for gesture recognitions; (3) *InkRecognitionNode,* used for pen writing ("ink") recognitions; (4) *PointerRecognitionNode,* used for recognizing interaction from a pointer device; and (5) *KeyRecognitionNode,* used for recognizing interactions from keyboard devices. Some examples of how the *Content* of those nodes may be represented include: W3C SRGS [35] for *SpeechRecognitionNode*; GDL (Gesture Description Language) [14] for *GestureRecognitionNode*; and *InkXML* [31] for *InkRecognitionNode.*

Since *RecognitionNode* is indeed a specialization of *ContentNode*, it is also possible to define *Anchor*s in it. A special type of *Anchor*, the *RecognitionAnchor*, specifies a portion of the recognition content. For instance, an anchor may refer to expected speech tokens defined in an SRGS file. A *RecognitionAnchor* is associated to a "recognition" event. Moreover, an anchor may refer to events of that *RecognitionNode*, for instance, a "click" anchor to a *PointerRecognitionNode.* The "recognition" event indicates that the system has recognized the expected information defined in a *RecognitionAnchor*. It is important to highlight that the occurrence of events issued by a *RecognitionNode* are not coupled with events from a *MediaNode. For instance, KeyEvent* is different from the *SelectionEvent* with key parameter, because the latter only happens during a *Media* occurrence. Similarly, we can contrast the *Pointer* event with the *PointOver* event.

Since the use of *MediaNode* and *RecognitionNode* entities are similar (they are both subclasses of *ContentNode*), it is now possible to use both "selection" and "recognition" events in *Link*s. Therefore, it is possible to create relationships such as: "when a *SpeechRecognitionNode* recognizes a specific word or sequence of words (e.g. "Put that"), a *TTSNode* should render a phrase (e.g. "where?"). However, authors are still unaware of the multiple interacting users. To support applications in which authors may be

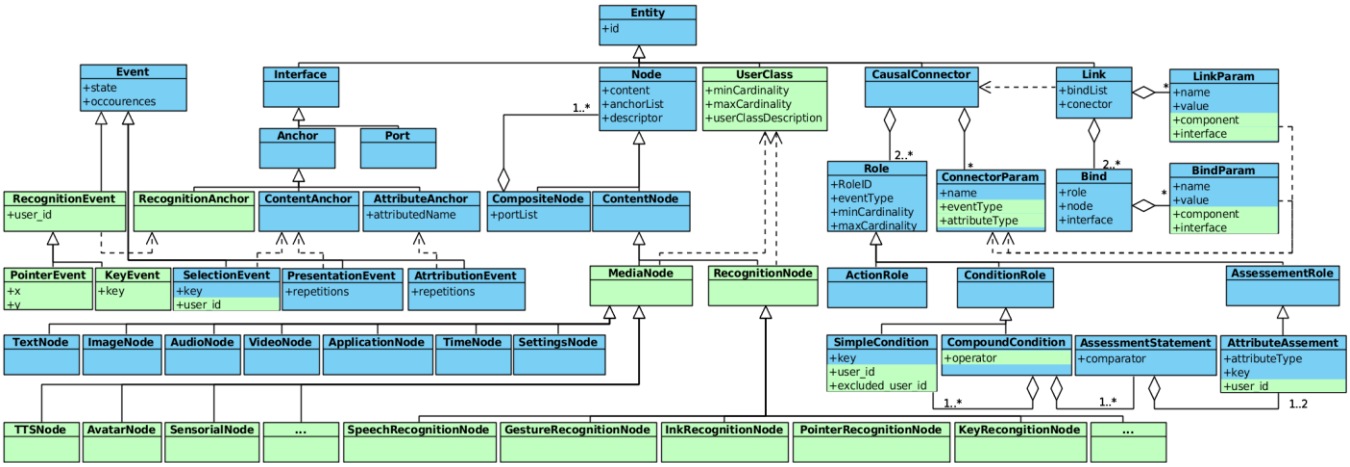

Figure 3: NCM 3.0 with proposed extensions.

truly aware of multiuser interactions, we propose the new *UserClass* entity, discussed in next subsection.

An important feature of MUIs is the combination of interaction modalities. According to Nigay and Coutaz [16], this combination can be: *redundant*, when only one of the interactions is needed; *complementary*, when all interactions are needed; or *sequentially complementary*, when all interactions are needed in a specific order. To support these combinations in NCL, authors may use *CompoundCondition*s with *RecognitionNode*s. Using the "or" operator, authors can define alternative (redundant) ways in which the user may interact. Using the "and" operator authors can define complementary interactions. In addition to the operators already defined by NCM, we include a new "seq" operator, through which authors can define a required sequence of interactions. In the "Put-That-There" scenario, for instance, authors must use a "seq" operator to guarantee that the interactions must occur in the specified order (first, the "put that"; then, the "logo" selection, etc.).

5.2 Multiuser Support

In multiuser applications, a user may assume different roles. For instance, it is possible to create an application that behaves differently if the user is a professor or a student. Moreover, both professors and students can use the application at the same time. By supporting multiuser features in the authoring model, authors can develop applications that are aware of who and how users are interacting with the application. With current NCM concepts, however, authors cannot model such possibilities. To support multiuser features in NCM, we create a new *UserClass* entity. *UserClass* is used to model the different types of interacting users—i.e., different roles that users can play when using an application.

A *UserClass* is defined by an *id*, a *minCardinality* and a *maxCardinality* attributes, and a *userClassDescription*. The *id* attribute uniquely defines a *UserClass*. The cardinality attributes allow to limit the number of users that can be part of that *UserClass*. The *userClassDescription* specifies how users are identified as being part of the *UserClass*. To do that, each user that is interacting with the system must have a *profile description*. The user profile may include, for example, information such as if he (or she) is a student or a professor. Moreover, it may include the devices he (or she) is using to interact with the application. The *userClassDescription* then should specify which users, based on their profiles, are part of a *UserClass*.

Note that the specification of *userClassDescription* is tied to the specific vocabulary used to describe the user's profiles. NCM, however, should be agnostic and not prescribe the *userClassDescription* specification details, which should be defined by NCM instantiation (discussed in Section 5.4). Moreover, runtime properties related to a *UserClas*s, such as the number of users registered in a class, can be consulted as properties in the *SettingsNode*.

5.3 Linking Multiple Modalities and Users

To support multiuser-oriented authoring, we define a new event attribute, named "user_id". The value of the "user_id" (e.g. *BoltLikeUser* (1)) attribute defines which specific user from a *UserClass* was responsible for generating the event. This approach is similar to how Soares [24] uses the "key" attribute to define which key from "selection" events may trigger a link. Indeed, the "user_id" attribute is defined for both "selection" and "recognition" events. As in the "key" case, the "user_id" attribute may also be tested by authors, in *SimpleConditionRole*s and *AttributeAssessment*s, when creating *Connectors*. By doing this, it is possible to limit which specific users can trigger a link. Moreover, we define another optional attribute in *SimpleConditionRole,* called "excluded_user_id". In this attribute, authors can define a list of users that are not allowed to trigger a *SimpleConditionRole*.

It is also interesting to give authors access to the proposed attributes through *Links*. To do that, we defined extensions to *ConnectorParam, BindParam*, and *LinkParam*. Besides an arbitrary string value, *ConnectorParam* can now receive an interface as well. To define that a *ConnectorParam* should receive an *Interface*, we propose the *eventType* and *attributeType* attributes, which are analogous to those of *AttributeAssessment. BindParam* and *LinkParam* can pass an *Interface* as a parameter to *Connectors* through the *component* and *interface* attributes. For instance, in the "Put-That-There" scenario, the author should save the position pointed by the user.

5.4 Instantiating the Proposed Extensions in NCL

The previously proposed entities were instantiated in NCL. Table 1 shows the XML elements with their corresponding attributes and content, which can be used in NCL documents.

Table 1: NCL elements related to the proposed NCM extensions

NCL element	Main attributes	Content
media	id, src, type, descriptor	(area|property)*
input	id, src, type	(area|property)*
connectorParam	name, eventType, attributeType	empty
bindParam	name, value, component, interface	empty
linkParam	name, value, component, interface	empty
userClass	id, min, max, userClassDescription	empty

MediaNode and *RecognitionNode* are instantiated using <media> and <input>, respectively. They are both defined inside the <body> element and may participate in NCL <link>s. They have *id*, *src* and *type* attributes. Different from <media>, <input> does not have a *descriptor* attribute, which is used in <media> for starting and ending a presentation (e.g. transitions, border details). Specific attributes for recognizing user input are defined as properties in the <input> element.

Moreover, <bindParam> and <linkParam> are extended with the *component* and *interface* attributes (similar to those of <bind>) and <connectorParam> is extended with *eventType* and *attributeType* attributes (similar to those of <attributeAssessment>).

The <userClass> element is defined inside <userBase>, in the document head (<head> element). It has *id*, *min*, *max*, and *userClassDescription* attributes. A <userClass> can be used by <input> and <media> elements by the "userClass" property, which value must be the *id* of a <userClass>. The *userClassDescription* is an URL to an external document describing the *UserClass*.

As previously mentioned, *userClassDescription* is tied to how user profiles are specified. In our instantiation, user profiles are described in RDF (Resource Description Framework) [33], and a *userClassDescription* is a SPARQL [34] query. Each user is a foaf:Person element from the FOAF [30] RDF vocabulary, used for user profiles. Additionally, this foaf:Person may define prf:name

elements from UAProf [21] RDF vocabulary, used to define device profiles. The SPARQL query is responsible for selecting which users should be part of the *UserClass*. Listing 1 shows a SPARQL query defining that a *UserClass* is composed of the users whose emails belong to the "inf.puc-rio.br" domain and who have a Leap Motion device.

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX prf:
3 <http://www.wapforum.org/profiles/UAPROF/ccppschema-
4 20010430>
5 SELECT ?person
6 WHERE {
7  ?person foaf:mbox ?email FILTER regex(?email,
8 "@inf.puc-rio.br$") .
9  ?person prf:component ?component .
10  ?component prf:name ? name FILTER regex(?name,
11 "Leap Motion")
12 }
```

Listing 1: SPARQL code fragment for *UserClassDescription*

Finally, runtime properties related to a *UserClass* can be accessed in <media> elements of the type "x-ncl-settings" using the scheme: "userClass(UserClassName).propertyName" for the "user" scope. For instance, if we want to know (e.g. using a Link) the number of users registered in a class named "BoltLikeUser" we can use the property "user.userClass(BoltLikeUser).count".

6. USAGE SCENARIOS

This section discusses how the usage scenarios presented in Section 2—the "I-Get-That-You-Put-It-There" and the "Anyone-Get-That-Someone-Else-Put-It-There" scenarios—can be implemented using the proposed extensions. For simplicity, we replaced the military unit of the "Put-that-there" scenario with an image (the Ginga logo), which can be moved by two users using a combination of voice and gesture commands.

Both scenarios use the same <userClass>, <media> and <input>, elements, which are depicted in Listing 2. The code fragment is composed of one <userClass> ("BoltLikeUser"), two <media> elements ("logo" and "sentences"), and two <input> elements ("pointer" and "asr"). "BoltLikeUser" (lines 1-6) defines a *UserClass* that must have two users and which description is in the "boltlikeuser.sparql" file.

Both scenarios use "logo" and "sentences"<media> elements and "pointer" and "asr" <input> elements. The "logo" media (lines 13-17) is an *ImageNode* using "ginga.png" file as *Content.* The "sentences" media (lines 18-21) is a *TTSNode* using an SSML specification in the "sentences.ssml" file as *Content.* An *Anchor* (<area> element) is defined in the "sentences" media pointing to the SSML fragment responsible for defining the word "where". The "pointer" input (lines 22-26) is responsible for recognizing the point on the screen at which the user's finger is pointing to. The "asr" input (lines 27-32) enables voice commands recognition using the SRGS "commands.sgrs" file as *RecognitionContent*. Two anchors are defined to the SRGS rules specifying the "put that" and "there" sentences. Both scenarios begin by starting the "logo", "pointer", and "asr" elements, as defined in the <port> elements (lines 10-12). In particular, the "asr" media is started by the "put_that" interface, which defines the word expected in the first interaction.

```
1 <head>
2  <userBase>
3   <userClass id="BoltLikeUser" min="2" max="2"
4    userClassDescription="boltlikeuser.sparql">
5  </userClass>
6 </userBase>
7 ...
8 </head>
9 ...
10 <port component="logo">
11 <port component="pointer">
12 <port component="asr" component="put_that">
13 <media id="logo" src="ginga.png">
14  <property name="userClass" name="BoltLikeUser"/>
15  <property name="top"/>
16  <property name="left"/>
17 </media>
18 <media id="sentences" type="application/ssml+xml"
19  src="sentences.ssml">
20  <area label="where"/>
21 </media>
22 <input id="pointer"
23  type="application/x-ncl-pointer">
24  <property name="userClass" name="BoltLikeUser"/>
25  <area label="pointer_click"/>
26 </input>
27 <input id="asr" type="application/srgs+xml"
28  src="commands.sgrs">
29  <property name="userClass" name="BoltLikeUser"/>
30  <area label="put_that"/>
31  <area label="there"/>
32 </input>
```

Listing 2: Code fragment of <userClass>, <media> and <input> elements

6.1 I-Get-That-You-Put-It-There

The "I-Get-That-You-Put-It-There" scenario considers interactions of two specific users from a *UserClass*. Listing 3 shows the code fragment of "I-Get-That-You-Put-It-There" that controls the logo movement, based on voice and gesture input modalities. We use two <link> elements, which are described in what follows.

The first <link> (lines 2-13) says that when the first user from the *BoltLikeUser* class performs the voice command "put that", followed by the selection of the "logo" media, the application will synthesize the word "where". Here, the selection of the "logo" is equivalent to the user pointing his finger at the logo but using the "user_class" and "user_id" in <connectorParam>.

The second *Link* (lines 14-30) says that when the second user from the *BoltLikeUser* class performs the voice command "there", the "logo" must be moved to the new position where the user's finger is pointing at. For that, it reads the "x" and "y" properties of the "pointer" input, by defining a <LinkParam> with names "user_event_to_getX" and "user_event_to_getY". Then, it uses those values to set the new "logo" position ("top" and "left" properties).

```
1 ... <!-- code from Listing 2 -->
2 <link xconnector="onRecognizeByUserOnSelectionByUser
3 Start">
4 <linkParam name="user_id" value="BoltLikeUser(1)"/>
5  <bind role="onRecognizeByUser"
6   component="asr"
7   interface="put_that"/>
8  <bind role="onSelectionByUser" component="logo"/>
9 <bind role="start" component="asr"
10   interface="there"/>
```

```
11  <bind role="start" component="sentences"
12   interface="where"/>
13 </link>
14 <link
15 xconnector="onRecognizeByUserSetXFromUserEventSetYFr
16 omUserEvent">
17  <linkParam name="user_id" value="BoltLikeUser(2)"/>
18  <linkParam name="user_event_to_getX"
19    component="pointer"
20    interface="click"/>
21  <linkParam name="user_event_to_getY"
22   component="pointer"
23   interface="click"/>
24  <bind role="onRecognizeByUser" component="asr"
25   interface="there"/>
26  <bind role="setXFromUserEvent" component="logo"
27   interface="left"/>
28  <bind role="setYFromUserEvent" component="logo"
29   interface="top"/>
30 </link>
```

Listing 3: Code fragment of Links from I-Get-That-You-Put-There

6.2 Anyone-Get-That-Someone-Else-Put-It-There

The "Anyone-Get-That-Someone-Else-Put-It-There" scenario considers interactions of any two users from a *UserClass*. More precisely, it first considers the interaction of any user and, in sequence, the interaction of another one, who must be different from the first one. We access the first interacting user by a <connectorParam> and store, in the *SettingsNode*, his (or her) "user_id". Listing 4 shows the code fragment of the <causalConnector> using our extended <connectorParam>. First, we access the "user_id" from a "recognition" event of the given *Node* passed in the <connectorParam>. Then, the link sets the "user_id" value as value of the *AttributeAnchor* passed in the *setUserId* role.

```
1 <conectorBase>
2  <causalConnector id="onRecognizeSetUserId">
3   <connectorParam name="var_user_id"
4    evenType="recognition" attributeType="user_id"/>
5   <simpleCondition role="onRecognize"/>
6   <simpleAction role="setUserId"
7    value="$var_user_id"/>
8 </causalConnector>
9 ...
10 </connectorBase>
```

Listing 4: Code fragment of using new <connectorParam> extensions.

Listing 5 shows the code fragment of "Anyone-Get-That-Someone-Else-Put-It-There" that controls the logo movement. We use three <link> elements, which are described in what follows.

The first <link> (lines 2-7) says that when any user perform the voice command "put that", we then store the "user_id" of that user. Here, the any user is defined by not specifying the "user_id" in the <bindParam>.

The second *Link* (lines 8-17) says that when the stored user performs the selection of the "logo" media, the application will synthesize the word "where".

The third *Link* (lines 18-35) says that when another user performs the voice command "there", the "logo" must be moved to the new position, where this user is pointing at. Here, another user is defined by specifying "excluded_user_id" in the <bindParam>, which is the value of the previously stored "user_id", from the first interaction. For that, it reads the "x" and "y" properties of the "pointer" input, and uses those values to set the new "logo" position ("top" and "left" properties).

```
1 ... <!-- code from Listing 2 and Listing 4-->
2 <link xconnector="onRecognizeSetUserId">
3  <bind role="onRecognize" component="asr"
4   interface="put_that"/>
5  <bind role="setUserId" component="settings"
6   interface="first_user"/>
7 </link>
8 <link xconnector="onSelectionByUserStart">
9  <linkParam name="user_id
10   component="settings"
11   interface="first_user"/>
12  <bind role="onSelection" component="logo"/>
13  <bind role="start" component="asr"
14   interface="there"/>
15  <bind role="start" component="sentences"
16   interface="where"/>
17 </link>
18 <link xconnector="onRecognizeByExcludedUserSetXYFrom
19 User">
20  <linkParam name="excluded_user_id"
21   component="settings" interface="first_user"/>
22 <linkParam name="user_event_to_getX"
23    component="pointer"
24    interface="click"/>
25 <linkParam name="user_event_to_getY"
26    component="pointer"
27    interface="click"/>
28  <bind role="onRecognizeByExcludedUser"
29    component="asr"
30    interface="there"/>
31  <bind role="setXFromUserEvent" component="logo"
32   interface="left" />
33  <bind role="setYFromUserEvent" component="logo"
34   interface="top"/>
35 </link>
```

Listing 5: Code fragment of Links from "Anyone-Get-That-Other-Put-There".

7. FINAL REMARKS

This paper proposes extensions to NCL aiming at supporting multimodal and multiuser interactions in NCL. For that, we build on the multimodal extensions of Guedes et al. [11], review them based on the NCM model, and propose new entities for authors to be aware of multiple users interacting with a multimedia system. In particular, we have added the *RecognitionNode* and *UserClass* entities for supporting, respectively, input modality interaction and multiuser interactions.

As future work, we aim at improving our proposal following three main paths.

First, we intend to investigate the use of multiuser features in multimedia presentations for ubiquitous environments. In this scenario, each user can use or share different devices, such as tablets or smartphones. This scenario may bring up several issues such as device discovery and relationships between *UserClass* and these devices. The work of Batista et al. [3] proposes a form of

description for screen-based devices required in NCL applications. Therefore, a possible approach would be extending the Batista et al. description for supporting multiple users' concepts.

Second, we plan to evaluate our approach through observations [17] with developers. Such evaluation may help us to highlight the possible benefits and difficulties of using the proposed extensions.

Third, we consider that NCL extended with the *RecognitionNode* abstraction can enable other sensor/actuators-based scenarios, for instance, those based on IoT (Internet of Things). Given the imminent researches in IoT, we aim at studying how NCL could handle IoT applications using different ambient sensors, such as those measuring ambient conditions (e.g. temperature, humidity, and barometric conditions) or user conditions (e.g. heart rate).

ACKNOWLEDGMENTS

First, we are strongly thankful to Prof. Luiz Fernando Gomes Soares (*in memoriam*) for the profound guidance and friendship, essential to this work and its authors. We also thank Rodrigo Costa, Marcos Roriz and all TeleMidia Lab's researchers, who provided thoughtful discussions on this work. Finally, we thank the Brazilian National Council for Scientific and Technological Development (CNPq) for their financial support (project #309828/2015-5).

REFERENCES

[1] Azevedo, R.G.D.A. and Soares, L.F.G. 2012. Embedding 3D Objects into NCL Multimedia Presentations. *Proceedings of the 17th International Conference on 3D Web Technology* (New York, NY, USA, 2012), 143–151.

[2] Batista, C.E.C.F. et al. 2010. Estendendo o uso das classes de dispositivos Ginga-NCL. *WebMedia '10: Proceedings of the 16th Brazilian Symposium on Multimedia and the Web* (2010).

[3] Batista, C.E.C.F. 2013. *GINGA-MD : Uma Plataforma para Suporte à Execução de Aplicações Hipermídia*. Pontifícia Universidade Católica do Rio de Janeiro.

[4] Bolt, R.A. 1998. "Put-that-there": voice and gesture at the graphics interface. *Readings in Intelligent User Interfaces*. M.T. Maybury and W. Wahlster, eds. Morgan Kaufmann Publishers Inc. 19–28.

[5] Bulterman, D.C.A. and Rutledge, L.W. 2008. *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer Publishing Company, Incorporated.

[6] Carvalho, L. and Macedo, H. 2010. Estendendo a NCL para Promover Interatividade Vocal em Aplicações Ginga na TVDi Brasileira. *WebMedia '10: Proceedings of the 16th Brazilian Symposium on Multimedia and the Web* (2010).

[7] Carvalho, L.A.M.C. et al. 2008. Architectures for Interactive Vocal Environment to Brazilian Digital TV Middleware. *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems* (New York, NY, USA, 2008), 22:1–22:8.

[8] Elmqvist, N. 2011. Distributed User Interfaces: State of the Art. *Distributed User Interfaces*. J.A. Gallud et al., eds. Springer London. 1–12.

[9] Ghinea, G. et al. 2014. Mulsemedia: State of the Art, Perspectives, and Challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications*. 11, 1s (Oct. 2014), 17:1–17:23.

[10] Google (n.d.). Supporting Multiple Game Controllers | Android Developers. *https://developer.android.com/intl/pt-br/training/game-controllers/multiple-controllers.html*. Accessed on 2016-05-02.

[11] Guedes, A.L.V. et al. 2015. Specification of Multimodal Interactions in NCL. *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web* (2015), 181–187.

[12] Guerrero Garcia, J. et al. 2010. Designing workflow user interfaces with UsiXML. *1st Int. Workshop on User Interface eXtensible Markup Language UsiXML'2010* (2010).

[13] Haber, C. 2001. Modeling Multiuser Interactions. *Proceedings at the First European Computer Supported Collaborative Learning Conference, Maastricht, Germany* (2001), 22–24.

[14] Hachaj, T. and Ogiela, M.R. 2012. Semantic Description and Recognition of Human Body Poses and Movement Sequences with Gesture Description Language. *Computer Applications for Bio-technology, Multimedia, and Ubiquitous City*. T. Kim et al., eds. Springer Berlin Heidelberg. 1–8.

[15] ISO/IEC 2013. ISO/IEC 23005-3:2013 Information Technology - Media Context and Control - Part 3: Sensory Information. *www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=60391*. Accessed on 2016-08-08.

[16] Laurence Nigay and Coutaz, J. 1997. Multifeature Systems: The CARE Properties and Their Impact on Software Design. *Multimedia Interfaces: Research and Applications, chapter 9* (1997).

[17] Lazar, J. et al. 2010. *Research Methods in Human-Computer Interaction*. Wiley Publishing.

[18] Limbourg, Q. et al. 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. *ICWE Workshops* (2004), 325–338.

[19] Microsoft (n.d.). Getting Started With XInput. *https://msdn.microsoft.com/en-us/library/windows/desktop/ee417001#multiple_controllers*. Accessed on 2016-08-02.

[20] Müller-Tomfelde, C. and Fjeld, M. 2010. Introduction: A Short History of Tabletop Research, Technologies, and Products. *Tabletops - Horizontal Interactive Displays*. C. Müller-Tomfelde, ed. Springer London. 1–24.

[21] OpenMobileAlliance 2001. WAG UAProf. *http://www.openmobilealliance.org/Technical/wapindex.aspx*. Accessed on 2016-08-02.

[22] Oviatt, S. 2007. Multimodal Interfaces. *The Human-Computer Interaction Handbook*. CRC Press. 413–432.

[23] Sant'Anna, F. et al. 2008. NCLua: Objetos Imperativos Lua Na Linguagem Declarativa NCL. *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web* (New York, NY, USA, 2008), 83–90.

[24] Soares, L.F.G. et al. 2010. Ginga-NCL: Declarative Middleware for Multimedia IPTV Services. *IEEE Communications Magazine*. 48, June (Jun. 2010), 74–81.

[25] Soares, L.F.G. et al. 2009. Multiple Exhibition Devices in DTV Systems. *Proceedings of the 17th ACM International Conference on Multimedia* (New York, NY, USA, 2009), 281–290.

[26] Soares, L.F.G. 2009. Nested Context Model 3.0: Part 1 – NCM Core. *ftp://obaluae.inf.puc-rio.br/pub/docs/techreports/05_18_soares.pdf*. Accessed on 2016-08-02.

[27] Stefik, M. et al. 1987. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Trans. Inf. Syst.* 5, 2 (Apr. 1987), 147–167.

[28] Turk, M. 2014. Multimodal interaction: A review. *Pattern Recognition Letters*. 36, (2014), 189–195.

[29] Vilhjálmsson, H. et al. 2007. The Behavior Markup Language: Recent Developments and Challenges. *Intelligent Virtual Agents*. C. Pelachaud et al., eds. Springer Berlin Heidelberg. 99–111.

[30] W3C 2014. FOAF Vocabulary Specification. *http://xmlns.com/foaf/spec/*. Accessed on 2016-08-02.

[31] W3C 2011. Ink Markup Language (InkML). *http://www.w3.org/TR/2011/REC-InkML-20110920/*. Accessed on 2016-08-02.

[32] W3C 2012. Multimodal Architecture and Interfaces. *http://www.w3.org/TR/mmi-arch/*. Accessed on 2016-08-02.

[33] W3C 2014. RDF/XML Syntax Specification. *https://www.w3.org/TR/REC-rdf-syntax/*. Accessed on 2016-08-02.

[34] W3C 2008. SPARQL Query Language for RDF. *https://www.w3.org/TR/rdf-sparql-query/*. Accessed on 2016-08-02.

[35] W3C 2004. Speech Recognition Grammar Specification Version 1.0. *http://www.w3.org/TR/speech-grammar/*. Accessed on 2016-08-02.

[36] W3C 2010. Speech Synthesis Markup Language (SSML) Version 1.1. *http://www.w3.org/TR/speech-synthesis11/*. Accessed on 2016-08-02.

[37] W3C 2012. State Chart XML (SCXML): State Machine Notation for Control Abstraction. *http://www.w3.org/TR/scxml/*. Accessed on 2016-08-02.

[38] W3C 2001. XHTML+Voice Profile 1.0. *http://www.w3.org/TR/xhtml+voice/*. Accessed on 2016-08-02.