Pontifícia Universidade Católica
do Rio de Janeiro

Álan Lívio Vasconcelos Guedes

Extending multimedia languages to support multimodal user interactions

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor: Prof. Simone Diniz Junqueira Barbosa

Rio de Janeiro
Setembro 2017

Pontifícia Universidade Católica
do Rio de Janeiro

Álan Lívio Vasconcelos Guedes

Extending multimedia languages to support multimodal user interactions

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the undersigned Examination Committee.

Prof. Simone Diniz Junqueira Barbosa
Advisor
Departamento de Informática – PUC-Rio

Prof. Sérgio Colcher
Departamento de Informática – PUC-Rio

Prof. Hugo Fuks
Departamento de Informática – PUC-Rio

Prof.ª Débora Christina Muchaluat Saade
Instituto de Computação – UFF

Prof. Carlos de Salles Soares Neto
Departamento de Informática – UFMA

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, Setembro the 29th, 2017

Álan Lívio Vasconcelos Guedes

The author received his Bachelor (2009) and M.Sc. (2012) in Computer Science from the Federal University of Paraíba (UFPB), where he worked as researcher in Lavid Lab. Since 2013, the author acts as researcher in TeleMídia Lab at PUC-Rio. During his academic career, he participated in several research projects about TV systems from funding agencies, such as RNP and FINEP. In particulary, his researches envolve Ginga and NCL specifications, which today are standards for DTV, IPTV and IBB.

Bibliographic data

Guedes, Álan Lívio Vasoncelos

Extending multimedia languages to support multimodal user interactions / Álan Lívio Vasconcelos Guedes; advisor: Simone Diniz Junqueira Barbosa. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2017.

v., 110 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Linguagens Multimídia;. 3. Interações Multimodais;. 4. MUI;. 5. Interações Multiusuário;. 6. Nested Context Language;. 7. NCL;. 8. HTML. I. Barbosa, Simone Diniz Junqueira. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

For Prof. Luiz Fernando Soares (*in memoriam*).

Acknowledgments

First, I would like to thank my family: my mother Rosa, my father Ednaldo, my grandmother Maria, my grandfather Abel, my aunt Socorro, my aunt Lucia, my aunt Tânia, my cousins Emmanuelle and Matheus, and my brothers Alysson and Adrian. They were always present in my life and contribute, in many ways, to what I am today. Without them I would not be able to fulfill this work.

I would like to deeply thank the Prof. Luis Fernando Gomes Soares (*in memoriam*) for having given me the honor of working with him. I do not have words to truly express my immense gratitude and admiration. His excellent guidance inspired and shaped me into becoming a better person and researcher.

I would like to thank my advisor Simone Barbosa for the guidance and patience. Moreover, I thank the support in an hard moment of my Phd and suddenly receive new an student.

I would like to thank everyone from TeleMídia Lab: Sergio Colcher, Alvaro da Veiga, Roberto Gerson, Guilherme Lima, Rodrigo Costa, Felipe Nagato, Rafael Diniz, Antonio Busson, Andre Brandão, Márcio Moreno and others. I am proud to by part of a research lab that focus not only in performs high level research projects but also training their member as researcher and as person. I share a lot moments, coffee and chocolates with them. In particularly, I deeply thank Roberto Gerson and Sergio Colcher for their support in my Phd.

I would also like to thank everyone from LAC lab, especially Marcos Roriz, Prof. Markus Endler,Prof. Francisco, André MacDowell and Luis Talavera. I also share a lot moments, coffee and chocolates with them.

I would like to thank a lot of people during the PhD journey and they made the journey with more joy and happiness. My roommates Marcos Roriz, Eduardo Araújo, Daniel Pires, Derlyane Simão, Katia Vega, Thais Abreu, Ruberth Barros, André Brandão and Julia Brandão. My friends in departament of informatics Aline Saettler, André Moreira, Hugo Gualandi, Lisseth Saavedra, Lívia Ruback, Luis Talavera, Paula Ceccon and Patrícia Carrion. My friends in departament of eletrical engenieering Andy Alvarez, Carlos, Roxana, Marcelo, Ted, Junior and Keila. So many other people.

I would like to thank all professors and staff from PUC-Rio that taught and helped me a lot during the PhD. More specifically, Regina Zanon.

Finally, I would like to thank CNPQ, for their financial aid.

Abstract

Guedes, Álan Lívio Vasoncelos; Barbosa, Simone Diniz Junqueira (Advisor). **Extending multimedia languages to support multimodal user interactions**. Rio de Janeiro, 2017. 110p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Recent advances in recognition technologies, such as speech, touch and gesture, have given rise to a new class of user interfaces that does not only explore multiple modalities but also allows for multiple interacting users. The development of applications with both multimodal and multiuser interactions arise new specification and execution issues. The specification of multimodal application is commonly the focus of multimodal interaction research, while the specification of the synchronization of audiovisual media is usually the focus of multimedia research. In this thesis, aiming to assist the specification of such applications, we propose to integrate concepts from those two research areas and to extend multimedia languages with first-class entities to support multiuser and multimodal features. Those entities were instantiated in NCL and HTML. To evaluate our approach, we performed an evaluation with NCL and HTML developers to capture evidences of their acceptance of the proposed entities and instantiations in those languages.

Keywords

Multimedia Languages; Multimodal User Interactions; MUI; Multiuser User Interactions; Nested Context Language; NCL; HTML

Resumo

Guedes, Álan Lívio Vasoncelos; Barbosa, Simone Diniz Junqueira. **Estendendo linguagens multimídia para suportar interações multimodais e multiusuário**. Rio de Janeiro, 2017. 110p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os recentes avanços em tecnologias de reconhecimento, como fala, toque e gesto, deram origem a uma nova classe de interfaces de usuário que não apenas explora múltiplas modalidades de interação, mas também permite múltiplos usuários interagindo. O desenvolvimento de aplicativos com interações multimodais e multiusuários trazem desafios para a sua especificação e execução. A especificação de uma aplicação multimodal é comumente o foco das pesquisas em interação multimodal, enquanto a especificação de sincronismos audiovisuais geralmente é o foco das pesquisas em multimídia. Nesta tese, com o objetivo de auxiliar a especificação de tais aplicações, buscamos integrar conceitos dessas duas pesquisas e propomos estender linguagens multimídia com entidades de primeira classe para suportar recursos multiusuário e multimodais. Essas entidades foram instanciadas nas linguagens NCL e HTML. Para avaliar nossa abordagem, realizamos uma avaliação com desenvolvedores NCL e HTML para capturar indícios de aceitação das entidades propostas e suas sintaxes nessas linguagens.

Palavras-chave

Linguagens Multimídia; Interações Multimodais; MUI; Interações Multiusuário; Nested Context Language; NCL; HTML

Table of contents

List of figures

List of tables

Listings

List of Abreviations

ASR	Audio Speech Recognition
BML	Behavior Markup Language
DOM	Document Object Model
DTMF	Dual-Tone Multi-Frequency
EMMA	Extensible MultiModal Annotation markup language
GDL	Gesture Description Language
GML	Gesture Markup Language
GUI	Graphical User Interfaces
HTML	Hypertext Markup Language
InkML	Ink Markup Language
MDE	Model-Driven Engineering
MUI	Multimodal User interfaces
NCL	Nested Context Language
NCM	Nested Context Model
RDF	Resource Description Framework
SALT	Speech Application Language Tags
SCXML	State Chart eXtensible Markup Language
SELD	Sensory Effect Description Language
SMIL	Synchronized Multimedia Integration Language
SMUIML	Synchronized Multimodal User Interaction Modeling Language
SPARQL	SPARQL Protocol and RDF Query Language
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
TTS	Text-To-Speech
UAProf	User Agent Profile
UsiXML	User Interface eXtended Markup Language
VoiceXML	Voice eXtensible Markup Language
WIMP	Windows, Icons, Menus, and Pointers
XHTML	eXtensible Hypertext Markup Language
XISL	eXtensible Interaction Scenario Language
XML	eXtensible Markup Language

1
Introduction

We naturally interact with the world and with other human beings by experiencing multiple senses (*e.g.* sight, hearing, and touch) and using different communication activities (*e.g.* such as voice, writing, and gestures) [1]. Also, we usually experience those multiple communication modes simultaneously, and we make sense of the overall environment through their combination. For instance, during a conversation audio cues allow us to identify a speaker's identity and location, while the conversation itself is commonly extended with gestures.

In contrast with the real complexity of multiple and coordinated interaction modes we experience in the real world, everyday human–computer interaction still focuses primarily on a single interaction input mode. Indeed, although everyday human-computer interaction technologies have supported some forms of multimodal interaction (*e.g.* by combining text entry, mouse movement and clicks, and by providing audiovisual output) the model of a single primary channel for data input, and a single primary channel for data output has been the norm [2]. Advances in recognition technologies such as speech, touch, and gesture recognition, however, have given rise to new human-computer interaction possibilities, such as MUI (Multimodal User Interfaces) and multiuser interactions.

A MUI [2] processes two or more combined user input modalities (*e.g.* speech, pen, touch, gesture, and head and body movements) in a coordinated manner with output modalities [3]. An input modality is a mode of communication [1] that conveys information generated by human communication activities (*e.g.* speech, gestures) and captured by input devices (*e.g.* microphone, pen) or sensors (*e.g.* motion sensors). An output modality is a mode of communication that conveys stimuli to be perceived by the human senses (*e.g.* hearing, vision). A MUI can uses synthesized audiovisual content (*e.g.* speech synthesis and avatar) and sensorial effects though actuators, which can be useful for increasing the QoE [4].

Figure 1.1 illustrates Bolt's seminal work, "Put-That-There" [5], which is an example of MUI-based interactions. His system enables users to use gestures and voice commands for planning tactical activities of military units over a

map. The user can move a military unit by first pointing his/her finger to a unit on the battlefield while saying "put that"; and then pointing his/her finger to the desired location and saying "there". As can be seen in this example, MUIs may support more natural human-computer interactions since, in many cases, it is possible to emulate the human-human communication. Bolt's work also evidences that developing MUIs brings many new challenges when compared to traditional Graphical User Interfaces (GUIs). Indeed, when developing MUIs, programmers are required to handle multiple device configurations and content specifications for both input and output modalities, as well as the coordination and synchronization among those modalities.

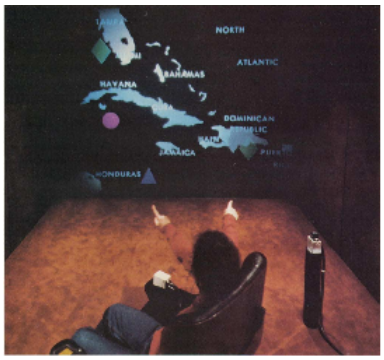


Figure 1.1: Bolt's Put-That-There from [2].

Regarding multiuser interactions, but without considering multimodal features, some applications can increase the number of interacting users. However, increasing the number of users does not necessary imply that the system has become able to identify or distinguish each one of them, *i.e.* , it does not mean that the system is fully aware of multiuser interactions. Stefik [6] proposes the early paradigm of WYSIWIS (What You See Is What I See). This paradigm enables users to collaborate using the same GUI across multiple users' screens. Applications in this paradigm commonly use shared view tools (*e.g.* VNC–Virtual Network Computing–), and even when multiple users are interacting with the system, the interacting users are not distinguished and are handled as if they were a single. More recently, research on Tabletop [7] and DUI (Distributed User Interfaces) [8] has been studying multiuser interaction over shared GUIs, but only few [9] of them truly consider multiuser interactions.

Truly multiuser applications are those in which the system can distinguish, and the programmer is aware of, the different users who are interacting with the system [10]. Some authors [11] also call the interface of such applications as Identity-aware User interfaces (IAUI). Examples of application of these truly multiuser interactions are present in gaming and virtual reality contexts. In these contexts, users are uniquely identified in each interaction. In game

contexts, for instance, users use gamepad or motion sensors (as illustrated in Figure 1.2).

Figure 1.2: Multiuser games with GameCube gamepads[3]and Microsoft Kinect[4]

This thesis addresses development of scenarios that use of human-computer interaction possibilities, namely MUI and multiuser. To highlight such scenarios, we present as follows some envisaged ones and their requirements.

1.1 Envisaged scenarios and requirements

Based on Bolt's scenario, Figure 1.3 presents the eight envisaged scenarios of applications that use both multimodal and multiuser interactions. We organize them by two categories, namely: "Put-That-There" (Figure 1.3-A to D) and "I-Get-That-You-Put-It-There" (Figure 1.3-E to H). Descriptions at the bottom of each scenario follow the scheme: <number of output modalities, number of input modalities, and number of interacting users>. We discuss each of them in what follows.

The "Put-That-There" category is illustrated in Figure 1.3-A to D. It aims at varying the number of input and/or output modalities for a single user interacting with the system. While Figure 1.3-A and B use GUI-based interactions, Figure 1.3-C and D use multimodal interactions. In Figure 1.3-A, the user interacts using a mouse and gets feedback on a screen (one input and one output modality). Figure 1.3-B extends A with voice feedback (one additional output modality). In Figure 1.3-C, the user interacts using gestures and voice commands (two input modalities and one output modality). Figure 1.3-D extends 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 supporting multimodal input/output for a single user.

[3]`https://images-na.ssl-images-amazon.com/images/I/91Bs1LePe4L._SL1500_.jpg`

[4]`http://kinectmediaplayerassets.blob.core.windows.net/assets/contexts/adventures/thumb/thumb_kinect_adventures.jpg`

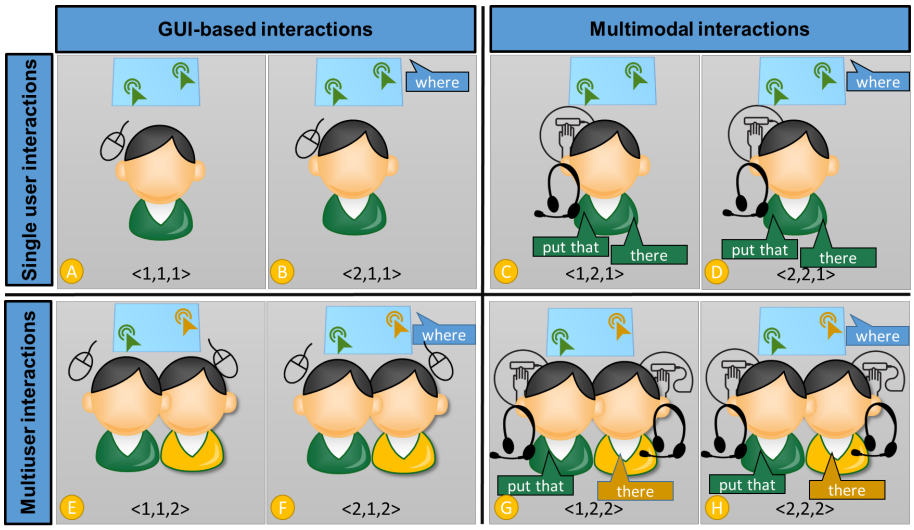


Figure 1.3: Scenarios based on Bolt's Put-that-there with single/multiple output/input modalities and interacting users.

The "I-Get-That-You-Put-It-There" category is illustrated in Figure 1.3-E to Figure 1.3-H. It is similar to the previous "Put-That-There", but the task must be performed by two different users, who are uniquely identified when interacting with the system. When Figure 1.3-E and Figure 1.3-F use GUI-based interactions, Figure 1.3-G and Figure 1.3-H use multimodal interactions. Figure 1.3-E shows each user interacting through individual mice. Figure 1.3-F extends the previous one with voice feedback. Figure 1.3-G shows each user interacting through gestures and voice commands. Figure 1.3-H extends the previous one with voice feedback (*i.e.* , it is similar to the original "Put-That-There", but for two users). The "I-Get-That-You-Put-It-There" scenario can be extended to handle users that are intentionally defined at runtime. For instance, one may define that the first interaction (*i.e.* point and say "put that") is carried out by any user and the second interaction (*i.e.* point and say "there") by a different user. We name this variation as "Anyone-Get-That-Someone-Else-Put-It-There".

The development of applications for the above scenarios brings both specification and system requirements.

System must support:

- The synchronized presentation of audiovisual media objects. This synchronized presentation is required to maintain a coherent visual feedback of a MUI-interface;
- The presentation of synthesized media objects. MUI interfaces use output modalities that are synthesized in presentation time such as synthetized audio, avatar or send actions to actuators;

– Different input devices. As mentioned, devices such as microphone, motion sensors and gamepad may be used by the interacting users;
– The matching of the required user characteristics defined by the developer with the interacting users. Given the specification of the users capable of interacting with application, the system must verify whether a user can interact with the application.

The specification must support:

– Abstractions to use different output and input modalities, besides the traditional GUI-based ones. MUI interfaces use output modalities such as: synthesized audio, animated avatar, and sensorial effects. Those interfaces also use input modalities, such as gesture and speech recognition;
– The specification of combined behavior of output and input modalities. This support enables the orchestration of both input and output modalities;
– Abstractions to define how users should be capable of interacting with the application. This support enables the developer to define the characteristics users need to have to be able to interact with the application. In the Put-That-There scenario, for instance, the developer needs to specify that a user needs to have a gesture sensor and a microphone.

In this thesis, we focus mainly on the specification requirements above to define our research goal.

1.2 Research goal

Given the aforementioned usage scenarios and their specification requirements, this thesis addresses the following general research question:

> *RQ1: How can we support the specification of applications that handle both multimodal interactions and multiple interacting users?*

Some researches [12, 13, 14] in Human-Computer Interaction (HCI) also address this question, but they suffer from some relevant drawbacks (discussed in Section 2). In particular, they lack support for fine synchronization among modalities. Synchronization among modalities is an issue mainly addressed by Visualization and Multimedia (VMM) research. Thus, we address this question by integrating efforts from both HCI and VMM.

In fact, other researchers, such as Rowe [15] and Turk [2], also share our motivation. Rowe's 2013 ACM SIGMM Report [15] claimed that multimedia

applications with MUI will be one of main themes for Multimedia research in the next few years. Additionally, Turk [2] argued that MUI is a multidisciplinary object of study. The specification of recognizers and usability of MUIs are commonly the focus of HCI research, while the synchronization and development of output modalities are usually the focus of VMM research. In particularly, VMM research addresses the specification of synchronization by studies in multimedia languages.

Traditionally, those languages focus on specifying multimedia applications with synchronized audiovisual media and limited user interactions. Examples of these languages are: HTML5 [16], NCL (Nested Context Language) [17], and SMIL (Synchronized Multimedia Integration Language) [18]. The developer who uses such languages is usually called author. Figure 1.4 illustrates an author creating a multimedia application, and the multimedia system executing it. At creation time, the author can use abstractions for: media, such as text (*e.g.* HTML's <p>, SMIL's <text>), graphics (*e.g.* HTML's <img>), and videos (*e.g.* HTML's <video>); synchronization; and user interactions mainly using mouse (*e.g.* HTML's onClick, NCL's onSelection) and keyboard (*e.g.* HTML and SMIL's keyPress, and NCL's onKeySelection). At execution time, the multimedia system presents the media's resulting content using audiovisual devices and reacts to pointer and key-based user interactions.

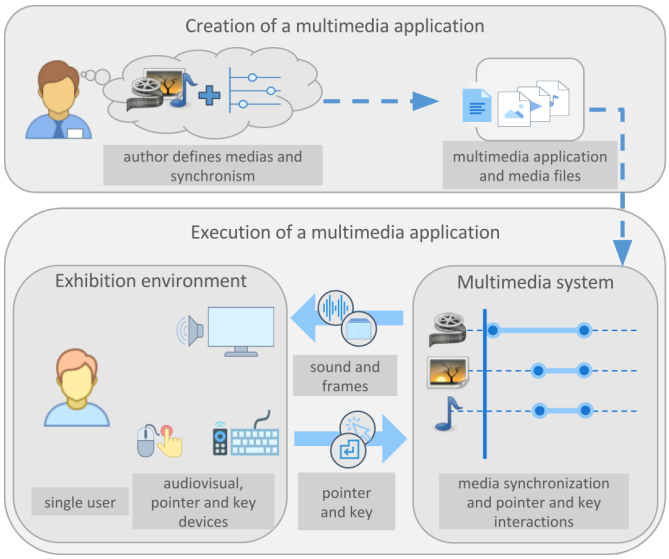


Figure 1.4: Creation and execution of a multimedia application.

Given this context of multimedia languages, we argue throughout our research that the support required by the *RQ1* can be achieved by extending multimedia languages to support multimodal and multiuser interactions. Therefore, we define a more specific question to be addressed in this thesis:

RQ2: How can we extend the output-oriented development in mul-

timedia languages to handle multiple modalities of user interactions, besides the ordinary GUI-based ones, and multiple interacting users?

As discussed in Chapter 3, our approach proposes extensions to multimedia languages with first-class entities to support both multimodal and multiuser features. Figure 1.5 illustrates our approach as a new version of the previous figure. At creation time, the author can define not only the media objects and the synchronization among them, but also the multimodal and multiuser interactions. For instance, for multimodal interactions, the author can specify a multimodal description, such as speech recognition and gesture recognition descriptions. At execution time, the multimedia system continues to use audiovisual devices to display the content of the media and pointer/key devices to capture user interactions, but it also uses multimodal interaction devices. For new output modalities, the multimedia can also use actuators, which perform sensorial effects, and sensors, which perform recognitions given multimodal descriptions.

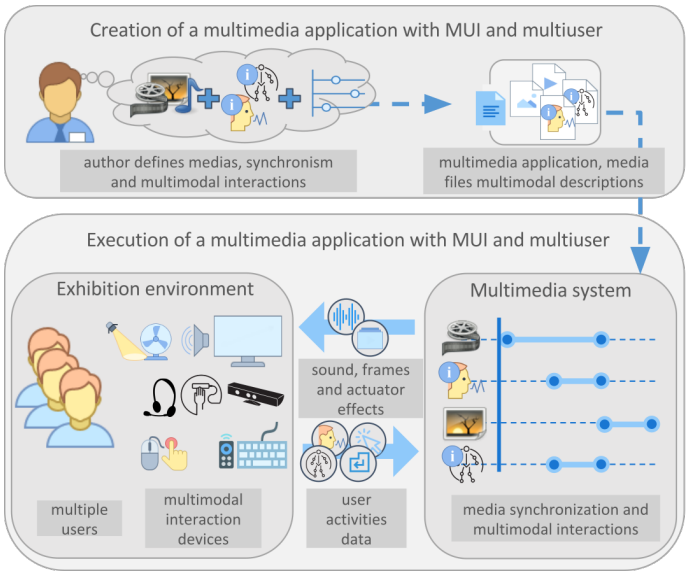


Figure 1.5: Creation and execution of a multimedia application with multimodal and multiuser interactions.

Some works have aimed at extending multimedia languages (*i.e.* , those exemplified in Figure 1.3-A to 1-D). However, those works usually add only one new modality to the language, usually speech [19, 20, 21, 22, 23]. The next chapter briefly discusses those works. To the best of our knowledge, no previous work has proposed abstractions to support the specification of applications using both multimodal and multiuser features.

1.3 Thesis structure

The remainder of this thesis is structured as follows. Chapter 2 presents the related works, languages, and frameworks for the development of multimodal and multiuser user interfaces, and highlights the main drawbacks of current approaches. Chapter 3 details our proposed approach to extend multimedia languages, which overcomes the identified drawbacks. Chapter 4 presents instantiations of the proposed approach in NCL and HTML languages. Chapter 5 presents an evaluation of our proposal. Finally, Chapter 6 presents our final remarks.

2
State of art

In this section, we present previous works that also aim at supporting the development of multimodal (Section 2.1) and multiuser (Section 2.2) interactions. Then, we summarize their drawbacks (Section 2.3).

2.1
Support for multimodal interactions

In order to guide our discussion about the support of multimodal interactions, we present Dumas's abstract architecture for MUI systems [24] in what follows.

Dumas's architecture (illustrated in Figure 2.1) presents a MUI as a perceptions-actions cycle between the multimodal system and the user. The user performs actions through human communication activities (*e.g.* speech, gestures, touch) and perceives the (result of) system actions through stimuli to her/his senses (*e.g.* sight, hearing). We detail the key components of the architecture in what follows.

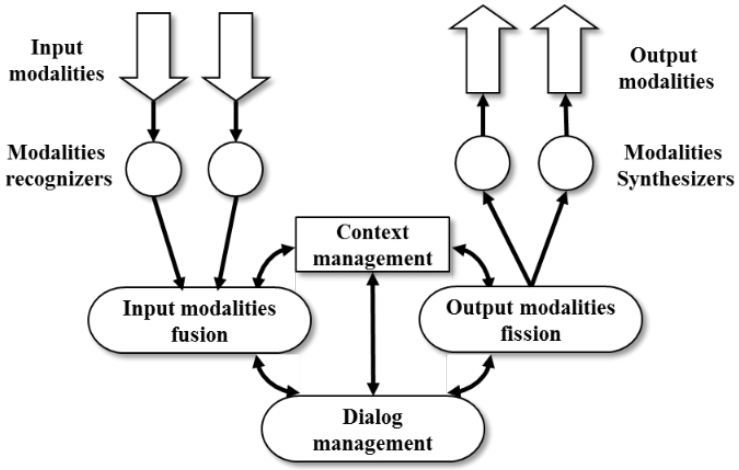


Figure 2.1: Conceptual architecture of a multimodal system, adapted from [24].

Modalities recognizers are capable of decoding *input modalities* through input devices and sensors (*e.g.* keyboard, video camera, and microphone). For example, an ASR (Audio Speech Recognition) recognizer identifies sentences in audio samples captured by a microphone. *Modalities synthesizers* are capable of producing *output modalities* through audiovisual output devices and actuator

devices (*e.g.* monitor, speakers, haptic). For instance, a TTS (Text-To-Speech) synthesizer produces audio to be played by a speaker.

Input modalities fusion is the process of combining the recognizers' results to interpret the user actions. For instance, fusion should interpret a user text conveyed in either a speech or an ink modality. In addition, the fusion should consider contextual information; for instance, it should not expect voice commands from a speech-impaired user or when the user is in a loud environment.

Output modalities fission is the generation of a system message –a combination of actions in one or more synthesizers– conveyed to the user through certain output modalities. The choice of which modalities are used in the message is called *modality selection*, which must also consider contextual information —*e.g.* it would not make sense to use visual modalities to convey information to a blind user.

Dialog management maintains the communication flow between the user and the system. The dialog defines a combination of fission and fusion processes, performed at each moment of application behavior. Finally, *context management* is responsible for storing the user context (*e.g.* car, mobile, home) and profile (*e.g.* blind, deaf, mute), which enables the fission and fusion processes to be adapted. In view the of Dumas's architecture, we discuss four groups of works in the next subsections (from 2.1.1 to 2.1.4). As illustrated in Table 2.1, each group supports some parts of the Dumas's architecture.

Approach	Recognizers	Fusion	Dialog management	Context management	Fission	Synthesizers
Language used by either recognizers or synthesizers						
SRGS [25], InkXML [26], GDL [27], and GML [28]	specialized modality	sequential-only				
SSML [29], BML [30], and SEDL [31]					specialized modality	sequential-only
Form-based dialog languages						
VoiceXML [32] and SALT [33]	speech	sequential-only	form-based		sequential-only	speech
XISL [13]	agnostic <input> element	SMIL *seq*,*par* and *alt* operators	form-based		SMIL seq, par and alt operators	agnostic <output> element
Frameworks						
MMI [14]	MCs with LifeCycle messages	DoneNotification with EMMA	state machine (SCXML)	SCXML (ECMAScript)	sequence of LifeCycle messages (optionally inside if-then-else)	MCs with LifeCycle messages
HephaticsTK [12]	SMUIML agnostic <recognizer> element	CARE properties based operators	state machine (SMUIML)		SMUIML sequence of <result> elements	ad-hoc message
Multimedia languages						
HTML+SALT [23] and HTML+ VoiceXML [22]	key/pointer+ VoiceXML		low-level (JavaScript)			text, image, video, and audio elements
SMIL+Rex [19]	key/pointer+ VoiceXML		low-level (par and seq time containers)		seq, par, excl time containers	text, image, video and audio elements
NCL+VoiceXML [21]	key/pointer+ VoiceXML		low-level (causality links)	user and system variables	seq operators with media, switch, and anchors	agnostic media element

Table 2.1: Comparison among the features supported by the different MUI development approaches.

2.1.1 Languages used by recognizers and synthesizers

The first group of related work consists of languages that handle either only *recognizers* or only *synthesizers*. None of them supports dialog or context management.

SRGS (Speech Recognition Grammar Specification) [25], InkXML (Ink Markup Language) [26], GDL (Gesture Description Language) [27], and GML (Gesture Markup Language) [28] assist in describing *recognizers*. SRGS is a grammar format for speech recognition. InkXML is a representation for electronic ink created with a stylus or other pointing devices, useful for handling text input. Finally, GDL and GML focus on describing user movements: GDL describes body joint movements captured by sensors (*e.g.* Microsoft Kinect), and GML focuses on touch gestures captured by touchpad devices.

SSML (Speech Synthesis Markup Language) [29], BML (Behavior Markup Language) [30], and SEDL (Sensory Effect Description Language) [31] assist in describing *synthesizers*. SSML is a representation for pronunciations focused on text-to-speech engines and can control speech aspects (*e.g.* pronunciation, volume, rate, pitch, and rhythm). BML enables controlling the verbal and nonverbal behavior of embodied conversational, useful for children- or elderly-oriented MUIs. Finally, SEDL, part of the MPEG-V framework [34], supports the description of sensory effects such as light, wind, fog, and chair vibration, which can be useful to enhance the consumer's experience of an audiovisual content.

2.1.2 Form-based dialog languages

The second group of related work consists of languages that focus on specifying the dialog management through a form-based approach. More precisely, developers specify MUI systems through questions, to be conveyed by the system through synthesizers, and expected answers from the user, interpreted through recognizers. None of the works in this group supports context management.

VoiceXML (Voice Extensible Markup Language) [32] and SALT (Speech Application Language Tags) [33] are limited to speech modalities and the widely used development of vocal interactions focusing on telephony conversations. They can use synthesized speech and digitized audio (voice recordings) as output. In addition, they can recognize speech and telephony DTMF (Dual-Tone Multi-Frequency) digits as input. Both focus on speech-only con-

versations, providing elements such as <listen> and <prompt>. In those languages, the author only combines a sequence of input or a sequence of output modalities.

XISL (eXtensible Interaction Scenario Language) [13] introduces an agnostic treatment for modalities through the <input> and <output> elements. It is inspired by the VoiceXML dialog but uses SMIL <par> and <seq> elements for defining the modalities synchronization. The XISL dialog uses a <prompt> element for system questions, <operation> for the fusion of user inputs, and <action> for the fission of outputs —*i.e.* , the system response. Inside fusion (*i.e.* <operation>) or fission (*i.e.* <action>), XSI supports temporal relationships through the <par> and <seq> SMIL elements, which are children of the <operation> and <action> elements.

2.1.3 Frameworks

The third group of related work consists of frameworks that focus on specifying the dialog management, usually using state machines. Those frameworks delegate the fission to be handled by a multimedia system. For instance, fine media synchronization, such as lip-syncs, are delegated to the multimedia system.

W3C's MMI (Multimodal Interaction) [14] framework is defined by Modality Components (MCs) and an Interaction Manager (IM). Inspired by XISL, an MC is modality agnostic and it is responsible for one or more input or output modalities, which it can handle by nesting other MCs and IMs. The IM controls the dialog flow of the application by coordinating the MCs by message exchange using an API named Life-Cycle [35]. These messages include the activation of an MC (*e.g.* Start, Pause, Cancel) and the result of an MC by a DoneNotification (*e.g.* end of speech recognition). MMI is instantiated by *control* and *presentation* documents, which implement IM and MCs, respectively. MMI recommends the use of SCXML [36] for control documents, and VoiceXML, HTML, or SMIL for presentation documents. Figure 2.2 illustrates the message exchange between control and presentation documents. An SCXML document describes the multimodal dialog using state machines, in which a set of <state> and <transition> elements defines the possible states of the multimodal application and the transitions between them. When transitioning between states, SCXML sends life-cycle events to MCs. Additionally, these transitions can use "if-then-else" constructions and ECMAScript variables. MMI recommends EMMA (Extensible MultiModal Annotation markup language) [37] to describe DoneNotification messages

exchanged between presentation documents and their IM. EMMA defines a semantic interpretation for a variety of modalities; for instance, presentation documents can interpret a text input in either a speech or an ink modality.

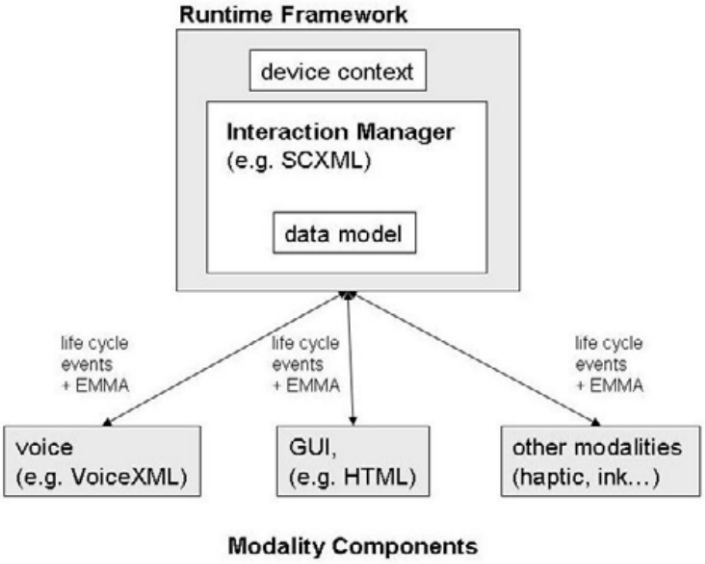


Figure 2.2: MMI overview, from [38].

Aiming at implementing his architecture, Dumas proposes the HephaticsTK framework. This framework is instantiated by one SMUIML (Synchronized Multimodal User Interaction Modeling Language) [12] document and its client application. Illustrated in Figure 2.3, SMUIML is a state-machine-based language that focuses on the specification of the fusion process. HephaticsTK delegates the fission to the client application in charge of interpreting the message and generating one or more output modalities.

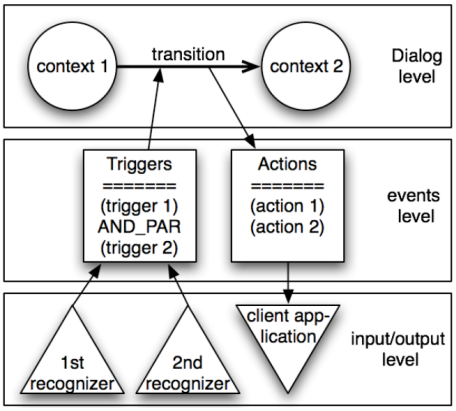


Figure 2.3: SMUIML overview, from [39].

In an SMUIML document, a state transition (<transition> element) is defined through a combination of recognizers' results (<triggers>) and messages to be sent to the client application (<action>). This combination

```
<transition leadtime="1500">
<seq_and>
  <trigger name="put_trigger" />
  <transition>
  <par_and>
    <trigger name="that_trigger" />
    <trigger name="object_pointed_event" />
  </par_and>
  </transition>
  <transition>
  <par_and>
    <trigger name="there_trigger" />
    <trigger name="object_pointed_event" />
  </par_and>
  </transition>
</seq_and>
<result action="put_that_there_action" />
</transition>
```

Listing 2.1: "Put-that-there" example illustrating CARE properties, expressed in SMUIML. Adapted from [41].

process of SMUIML considers the CARE (*Complementarity*, *Assignment*, *Redundancy*, and *Equivalence*) properties [40].

The formal expressions of the CARE properties rely on the transitions between two states of a given multimodal system. Considering S' following S, *Assignment* expresses the absence of choice: a single modality is available in S to reach S'. *Equivalence* means the availability of choice between multiple modalities, *i.e.* , the use of any one of the modalities is enough to reach state S' from state S. *Redundancy* means that a set of modalities are used redundantly to reach S' from S, *i.e.* , when they are used within the same temporal window without increasing the user's expressive power. A set of modalities is used in a *Complementary* way within a temporal window if all of them must be used to reach S' from S, either sequentially or in parallel. Section 2.1.5 discusses the state of the art in terms of combination of modalities and their expressiveness.

SMUIML implements CARE assignment by using only one <trigger> per <transition>. To implement the remainder CARE properties, SMUIML proposes four elements to combine the <trigger>s: <par_and> (CARE *parallel Complementarity*); <seq_and> (CARE *sequential Complementarity*); <par_or> (CARE *Redundancy*); and <seq_or> (CARE *Equivalence*). Listing 2.1 shows the "Put-that-there" in SMUIML, which uses <seq_and> and <par_and> combinations.

2.1.4 Multimedia languages

Similar to our approach, some works [19, 20, 21, 22, 23] extended XHTML (eXtensible Hypertext Markup Language) [42], SMIL, and NCL to support multimodal interactions. The main drawback of these works,

however, is that they do not provide a way to seamlessly integrate new modalities, because they incorporate recognizer specification by overloading existing elements or adding directly into a multimedia language body. In particular, they are limited to adding only a speech modality by incorporating VoiceXML and SALT elements.

Wang [23] proposes the integration of SALT elements (*e.g.* <salt:prompt>, <salt:grammar>) directly into XHTML. Likewise, W3C [22] proposes the integration of VoiceXML elements (*e.g.* <vxml:prompt>,<vxml:grammar>) directly into XHTML. Both proposals use DOM events or JavaScript code to support relationships between voice recognition/synthesis with XHTML content. Those scripts allow developers to control, in an imperative manner, the activation of recognizers and synthesizers, as well as the notification of the recognizers' result.

Beckham [19] proposes elements for integrating voice recognition and synthesis in SMIL. For speech synthesis, Beckman proposes a <TTS:render> element. For speech recognition, Beckman proposes a reactive language, named REX (Reactive XML), which has <raise>, <handle>, and <await> as its main elements. The <raise> element specifies the ASR (Audio Speech recognition) grammar to be recognized; <handle> defines actions that must be performed when a recognition grammar is accepted; and <await> acts as a <par> composition which includes <raise> and <handle>. As with other related work, Beckman's proposal merges elements related to recognition and speech synthesis with the specification of the multimedia document. Another drawback is the fact that it only adds supports to recognition and synthesis of a single modality (voice).

Carvalho *et al.* [20, 21] propose two approaches for integrating VoiceXML elements into NCL. They both incorporate VoiceXML directly into the XML tree of an NCL document and then map VoiceXML elements for voice recognition onto keyboard-based events of NCL. In [20], a VoiceXML dialog is inserted into the <port> NCL element (Listing 2.2) and is activated in the beginning of the document, whereas in [21] the dialogue is inserted into the <link> element (Listing 2.3) and is activated when the media related with the <link> is occurring. Besides overloading the concept of the <port> and <link> elements, these approaches compromise the separation between the structure and the content of the application, which is favored by the NCL model, namely NCM [43]. Moreover, the proposals do not allow developers to control the internal behavior of events occurring inside the VoiceXML dialog, which prevents the creation of relationships between the speech synthesis and recognition with other media objects.

```
<ncl>
  ...
  <body>
  <port id="pInicio" component="video">
    <voice>
      <menu scope="dia1og">
        <prompt>
          Video inicializado. Se deseja finalizar o video
          clique no botão<emphasy>vermelho</emphasy> ou diga
          <emphasy>SIM</emphasy>.
        </prompt>
        <choice next="#rec1">Sim</choice>
      </menu>
      <optionChoice id="rec1" action="RED" />
    </voice>
  </port>
  <media id="video" src="media/video.mpg" descriptor="dIV">
    <link id="iniciarVoz" xconnector="onSelectionStop">
      <bind component="video" ro1e="onSe1ection">
        <bindParam nae="keyCode" va1ue="RED" />
      </bind>
      <bind component="video" ro1e="stop"/>
      <voice>
        <prompt>Video finalizado</prompt>
      </voice>
    </link>
  </body>
  ...
</ncl>
```

Listing 2.2: Code fragment from [20], which uses VXML inside an <port>.

```
<link xconnector="onKeySelectionStartStop">
  <bind component="prato3" role="onSelection">
    <bindParam name="keyCode" value="GREEN"/>
  </bind>
  ...
  <vncl>
    <prompt>
      To choose the dish with vegetables and fish, say:
      <break size="medium"/>Green<break size="small"/>
      or meat and fried potatoes<break size="small"/>
    </prompt>
    <choice text="Dish with meat and fried potatoes.">
      <grammar type="text/gsl">[green vegetables fish]</grammar>
      <object action="GREEN" />
    </choice>
  </vncl>
</link>
```

Listing 2.3: Code fragment from [21], which uses VXML inside an <link>.

The aforementioned works may specify dialog management through low-level (multimedia-oriented) constructions of their languages. More specifically, to support dialog management, HTML can use JavaScript, SMIL can use time containers, and NCL can use causality links. Another characteristic of the proposals in this group is that they already support audiovisual synthesizers. In NCL, an agnostic <media> element is used, which is similar to the XISL <output> element. Moreover, NCL supports context management using global variables.

Although they do not directly address multimodal interactions, Moreno

et al. [44] aim at extending NCM 3.0 and NCL 3.0 to support some kind of recognition. Their proposal is inspired by semantics descriptions such as RDF (Resource Description Framework) [45] and proposes to use <media> elements to describe abstract concepts (*e.g.* soccer player and piece of art). More precisely, in their proposal, an author may define abstract concepts by: defining <media> elements with a string-based *concept* attribute and relating them using <spoConnector> (subject-predicate-object) relations; or by defining <media> that refer to an RDF description (src attribute). Once defined, these concepts can be recognized (*inferFrom* <link> role) in other <media> elements. For instance, Listing 2.4 illustrates the NCL code fragment of application that shows a museum tour video, and an additional content is presented when some specific piece of art appears in the video.

```
<ncl>
  <head>
    <connectorBase>
      ...
      <spoConnector id="arts">
        <subject nole="subject" max="1"/>
        <compoundObject qualifier="seq">
          <simpleObject role="born" max="1"/>
          <simpleObject role="painted" max="unbounded" qualifier="seq"/>
        </compoundObject>
      </spoConnector>
      <spoConnector id="appearance">
        <subject nole="subject" max="1"/>
        <simpleObject nole="appears" max="unbounded" qualifier="seq"/>
      </spoConnector>
      <causalConnector id="onBeginInferFromStart">
        <compoundCondition>
          <simpleCondition role="onBegin"/>
          <inference role="inferFrom" qualifier="from"/>
        </compoundCondition>
        <simpleAction role="start" max="unbounded"/>
      </causalConnector>
    </connectorBase>
  </head>
  <body>
    <port id="p1" component="museum"/>
    <port id="p2" component="expo"/>
    <media id="museum" src="museum.mp4"/>
    <media id="expo" src="VanGoghExpo.mp4"/>
    <media id="img1" src="sn1.jpg"/>
    <link id="l1" xconnector="onBeginFromstart">
      <bind role="onBegin" component="starry_night"/>
      <bind role="inferFrom" component="museum"/>
      <bind role="start" component="img1"/>
    </link>
    ...
    <media id="starry_night" concept="Starry Night"/>
    <media id="van_gogh" concept="Vincent Van Gogh"/>
    <media id="sn_canvas" src="images/starrynight.jpg"/>
    <media id="sn_museum" src="images/snmuseum.jpg"/>
```

```
41    <link id="l5" xconnector="arts">
42      <bind role="subject" component="van_gogh"/>
43      <bind role="born" component="zundert"/>
44      <bind role="painted" component="starry_night"/>
45    </link>
46    <link id="l6" xconnector="appearance">
47      <bind role="subject" component="starry_night"/>
48      <bind role="appears" component="sn_canvas"/>
49      <bind role="appears" component="sn_museum"/>
50    </link>
51  </body>
52</ncl>
```

Listing 2.4: NCL code fragment for recognitions in video, adapted from [44].

2.1.5 Expressiveness analysis

Based on the Dumas's architecture for multimodal systems, the previous subsections discussed the support provided by current approaches for developing MUIs. To highlight gaps of the related works in terms of their Expressiveness power, in this section we analyze them based on Allen's temporal relations. Allen's relations were initially proposed to express temporal relations among intervals in the database field [46]. In the Multimedia research, they are commonly used to describe the temporal arrangement among media objects in a multimedia presentation [47]. Table 2.2 illustrates the seven temporal relations proposed by Allen.

Different from previous analyses [47], which only use Allen's temporal relations for analyzing media objects (output modalities), we use them for analyzing both output and input modalities. Here, the temporal interval of an output or input modality means the interval in which it is activated, *i.e.* the time between its activation and its deactivation.

Allen's relation			Visual representation	Semantics
T1	before	T2		T1 takes place before T2
T1	equals	T2		T1 is equal to T2
T1	meets	T2		T1 meets T2
T1	overlaps	T2		T1 starts before T2 and they overlap
T1	during	T2		T1 is fully contained within T2
T1	starts	T2		T1 starts together with T2
T1	finishes	T2		T1 finishes together with T2

Table 2.2: Visual representations of Allen's temporal relations. Adapted from [46].

Table 2.3 presents the analysis of the expressive power of the selected approaches based on Allen's temporal relations. In this analysis, for each relation, we use every possible combination of input modalities (iM) and output modalities (oM). Some of the works discussed in this Chapter 2 were excluded from the analysis because they do not handle fission and fusion together. The excluded works are: the unimodal languages, such as SSML and SRGS; and the form-based approaches specialized in only one modality, such as VoiceXML, SALT, and MIML. In addition, we have excluded the HTML-based approaches, because they lack an explicit specification of synchronization between output and input modalities, delegating it to JavaScript.

XISL is a form-based language that expresses output modalities (*i.e.* system questions) followed by input modalities (*i.e.* user answers), or vice-versa. This means that it is possible to implicitly specify the "iM *meets* oM" and "oM *meets* iM" relations. Moreover, XISL uses the SMIL operators (*seq* and *par*) to combine a set of either output modalities or input modalities (but not both input and output within a single set). The seq operator in input modalities enables "iM *meets* iM." The *seq* operator in output modalities enables "oM *meets* oM." The *par* operator in input modalities enables "iM *starts* iM." The *par* operator in output modalities enables "oM" *starts* oM". Despite using the SMIL containers, XISL does not offer anchor attributes, such as *begin* and *end*, which prevents it from expressing the *before*, *overlaps*, and *during* relations. Finishes and equals are not supported either.

The MMI framework uses the SCXML language to define dialog management, which provides a state-machine abstraction. SCXML allows synchronizing actions to the activation and deactivation of states, through the <onentry> and <onexit> elements. Examples of actions include: the assign action (<assign> element), which edits the data of an ECMAScript; and the send action (<send> element), which sends LifeCycle messages to the client application. This way, MMI can express in a declarative way the starts, finishes, and meets temporal relations. More specifically: starts may be achieved through Start messages in an <onentry> element; finishes may be achieved through Cancel messages in <onexit>; and meets may be achieved through one state sending a Cancel message to an MC in <onexit>, and another state sending a Start to another MC in <onentry>. MMI cannot express before, equals, overlaps, and during in a declarative way. Before and equals would require the use of ECMA script variables and their evaluation in ECMA script expressions.

The HapticsTK framework uses SMUIML, another state-machine-based language, to define dialog management. SMUIML, however, does not handle fission and is limited to sending ad-hoc messages to the client application. Input

	Allen's relation			XISL	W3C MMI	SMUIML	SMIL+ Rex	NCL+ VoiceXML
	iM	before	iM					
	iM	equals	iM			Yes		
	iM	meets	iM	Yes	Yes	Yes		
fusion	iM	overlaps	iM					
	iM	during	iM					
	iM	starts	iM	Yes	Yes	Yes	Yes	
	iM	finishes	iM		Yes			
	oM	before	oM				Yes	Yes
	oM	equals	oM				Yes	Yes
	oM	meets	oM	Yes	Yes		Yes	Yes
fission	oM	overlaps	oM				Yes	Yes
	oM	during	oM				Yes	Yes
	oM	starts	oM	Yes	Yes	Yes	Yes	Yes
	oM	finishes	oM		Yes		Yes	Yes
	iM	before	oM					
	iM	equals	oM					
	iM	meets	oM	Yes	Yes	Yes	Yes	Yes
relating iM-oM	iM	overlaps	oM					
	iM	during	oM					
	iM	starts	oM		Yes			
	iM	finishes	oM		Yes			
	oM	before	iM					
	oM	equals	iM					
	oM	meets	iM	Yes	Yes		Yes	
relating oM-iM	oM	overlaps	iM					
	oM	during	iM					
	oM	starts	iM		Yes	Yes		
	oM	finishes	iM		Yes			

Table 2.3: Multimodal synchronization analysis based on Allen's relations.

modalities may be composed using CARE-based properties: <seq_and> can express "iM meets iM"; <par_end> can express "iM equals iM"; <par_or> can express "iM starts iM". At the end of each relationship, it is possible to send one ("iM meets oM") or multiple ("oM starts oM") ad-hoc messages to the multimedia system.

SMIL and NCL focus on multimedia synchronization and can implement all Allen's relations for output modalities. SMIL supports them by using time containers (seq and par) and their attributes (*e.g.* begin and end). NCL supports them using causality links and temporal anchors. Therefore, the SMIL+Rex and NCL+VoiceXML approaches —which extend NCL and SMIL, respectively, with speech modality— also support Allen's relation over output modalities.

Regarding input modalities, the SMIL+Rex and NCL+VoiceXML approaches are limited. SMIL+Rex proposes an <await> element, but it does not support begin and end attributes. Therefore, the <await> element can be used inside SMIL containers, but the containers cannot use <await>'s begin and end. The <await> element and a media inside a seq container enable meets relations between input and output ("iM followed by iM", "iM followed by iM", "iM followed by oM", "iM followed by oM"). Additionally, the <await> and a media in a par container enable starts relations ("oM starts iM" and "oM starts iM"). NCL+VoiceXML maps the speech recognition inside the VoiceXML onto

key-based events in NCL. This way, it is not possible to activate or de-activate recognizers, and it is not possible to achieve temporal relation between the activation of input modalities with other modalities.

2.2 Support for multiuser interactions

In this subsection, we discuss works aimed at supporting the development of multiuser interactions. Illustrated in Table 2.4, those works investigated focus on gaming and DUI (Distributed User Interface) [8].

Approach	**Application specification**	**User abstraction**	Context
Microsoft [48] and Google [49]	Imperative (*i.e.* C#, Java)	User is a GamePad parameter	Gaming
Guerrero-Garcia [50]	UsiXML	User is coupled with his/her device	DUI
Batista *et al.* [51, 52]	NCL	User is coupled with his/her device	DUI

Table 2.4: Comparison among the features supported by the different multiuser development approaches features.

Microsoft [48] and Google [49] APIs, among other game APIs, propose multiuser support in gaming contexts. Both enable multiuser interactions by imperative APIs to handle gamepad controllers. Microsoft supports multiuser interactions in DirectX applications by the XInput controller API. Similarly, Google supports multiuser interactions in Android applications by the GamePad API. Both of these APIs use callback events with an identification parameter informing the source controller.

UsiXML [53] is a task-oriented GUI description which adopts an MDE (Model-Driven Engineering) approach to be deployed to different device configurations (*e.g.* desktop, web, and mobile). Guerrero-Garcia [50] extends UsiXML (USer Interface eXtended Markup Language) 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.* [54] propose a hierarchical distribution of media in NCL. The distribution specification uses the abstraction of types of devices, called device class. The developer of a multimedia application distributes it by sending and orchestrating the media presentation for the different device classes. When sending a media (*e.g.* image) to a device class, Soares *et al.* do not define how the users who 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 interact. Soares *et al.*'s work —which specified fixed device

classes (passive and active)— is extended by Batista *et al.* [51, 52]. In [51], the developers define new device classes using a description based on the UAProf (User Agent Profile) [55] description. In [52], they propose that the developer may use a document variable called child.index, which can be consulted by the developer inside each NCL document sent to each device.

2.3 Drawbacks

We were able to identify four main drawbacks in the approaches presented in this chapter:

- *Lack of support for fine synchronization among modalities.* The CARE properties have been used in the fusion process to combine input modalities. However, when using audiovisual output modalities in fission processes —such as voice synthesizers, videos, or sounds— a multimedia system must handle many synchronization issues (*e.g.* lip-sync between a synthesized avatar and its speech synthesis). For instance, MMI and HephaticsTK consider the multimedia system as a "black box" which, as aforementioned, has the drawback of not supporting a fine synchronization between the input and output modalities. As stated before, only the approaches based on multimedia languages do not suffer this problem.
- *Strong encapsulation between fusion and fission.* According to Tulk [2], one of the main goals of Multimodal Interaction research is to enable the development of multimodal systems supporting bi-directional communication between humans and machines. However, the languages discussed here focus more on supporting either fusion or fission processes. On the one hand, the multimedia-based approaches encapsulate the fusion process; they delegate the fusion by using scripts (*e.g.* XHTML+VoiceXML) or by mapping input modalities to keyboard-based events (*e.g.* NCL+VoiceXML). On the other hand, languages such as MMI and SMIUML encapsulate the fission and delegate it to a multimedia system (*e.g.* HTML player, in the MMI case).
- *Lack of support for modality selection considering sensory capabilities.* Modality selection over output modalities is defined as part of the fission process [56, 24]. Such a selection should consider the presentation context and the user's profile (*e.g.* capabilities, skills). Adapting the input/output media based on human sensory capabilities is an issue investigated often by Multimedia research [4]. XISL and SCXML propose conditional structures —switch and if-then-else, respectively—, which can be used

for selecting the output modalities. However, they do not support the description of individual human sensory capabilities and, as a consequence, they do not directly support modality selection based on the users' sensory capabilities.

– *Interacting users are second-class citizens.* For both gaming and DUI contexts, the user interaction is identified by the source device, so that a user interacting with two devices will be viewed by the system as two different users.

Schnelle-Walka *et al.* [57] discuss the first two drawbacks when trying to use MMI for implementing a virtual assistant that helps a user to perform a task. IM implements the virtual assistant using two MCs: a BML document, for avatar rendering; and a VoiceXML document, for speech synthesis. To implement their usage scenario, however, they had to violate the MMI architecture twice. The first violation happened when they had to connect the BML MC and the VoiceXML MC directly. That was required because the MMI LifeCycle API does not provide enough support for lip-sync features (in this case, between the avatar and its speech). Such a workaround shows the lack of support for fine synchronization in the MMI framework.

According to Schnelle-Walka *et al.*, it is impractical to employ the MMI framework for synchronizing MCs with continuous output [57]. The second violation occurs when the virtual assistant IM needs to use an MC for a motion sensor managed by another IM. MMI defines a tree-based organization of IMs and MCs, which leads to a strong encapsulation between fusion and fission IM. In other words, the motion sensor MC, which is used in one of the fusion branches, cannot be (re-)used in another branch, the virtual assistant one (a fission branch).

3
Proposed approach

As previous mentioned, this thesis aims at answering the question of *how to support the specification of applications that handle both multimodal interactions and multiple interacting users* (RQ1). Some researches [12, 13, 14] in HCI also address this question, but they suffer from some relevant drawbacks (discussed in Section 2.3). In particular, they *lack support for fine synchronization among modalities*. The specification of output modalities synchronization is addressed by VMM by studies in multimedia languages. Thus, we address this question by integrating efforts from both HCI and VMM and we aim at extending multimedia languages to offer such support.

By so extending multimedia languages, we also aim to answer *how to extend the output-oriented development in multimedia languages to also handle these interaction, besides the ordinary GUI-based ones, and multiple interacting users* (RQ2). This question considers that the state of art suffers from a drawback of *strong encapsulation between fusion and fission*. More precisely, multimedia specification approaches delegate the input modalities and fission process, whereas MUI specification approaches delegate output modalities and fusion process. Thus, we address this question by proposing a multimedia document model with a set of entities that multimedia languages should instantiate.

Figure 3.1 shows the entities of our proposed model, which are: *Media* (detailed in Section 3.1), for defining output modalities to be presented in audiovisual devices and actuators; *Recognizer*, for identifying an expected input modality captured from an input device or sensor (detailed in Section 3.1); *UserClass*, to enable multiuser interactions and user-based contextual information (detailed in Section 3.2); and *Relationship*, which uses causal relationship to combine both input and output modalities using CARE-compatible conditions (detailed in Section 3.3).

The model is based on the NCM entities [43]. In particular, we follow a structure-based [5] paradigm, which means that the multimedia document is decoupled from the modalities contents. A language following our model acts as glue language and does not restrict or define what kinds of *Media* and *Recognition* are supported. Figure 3.2 depicts this structure, in which a

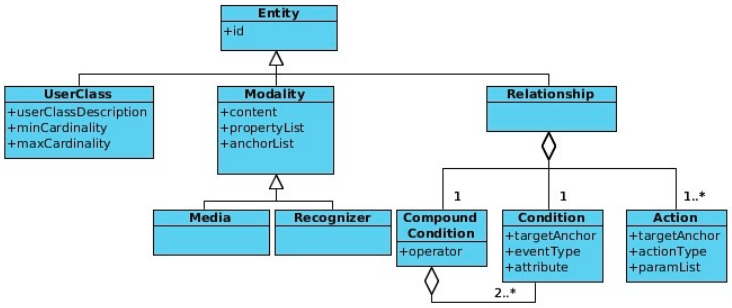


Figure 3.1: Class diagram for the proposed model.

multimedia document should define how *Media*, *Recognition*, and *UserClass* are related in time, using Relationships, not carrying their actual contents or descriptions. Then, if one changes the contents of an entity, from a *Media*, *Recognizer*, or *UserClass*, this will not change the document structure. For instance, in the "Put-That-There" scenario, the speech synthesis can be replaced by a video or the gesture recognition can be replaced by speech recognition.

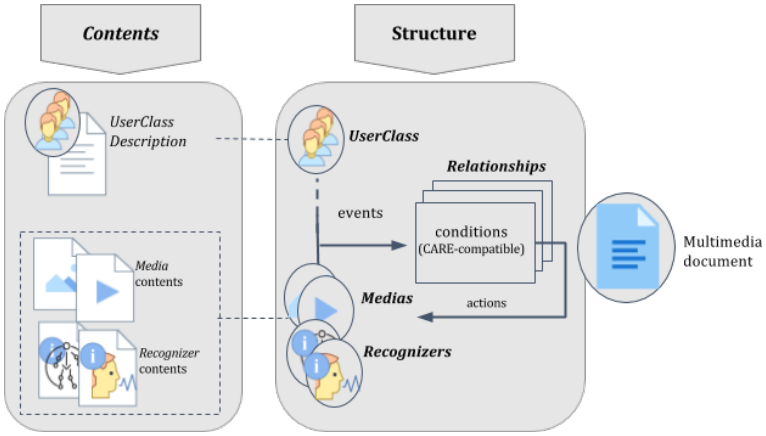


Figure 3.2: Schematic view of the proposed multimedia document.

The next subsections detail each entity of the model.

3.1 Media and Recognizer

An interaction modality can be specialized in two main entities: *Media* and *Recognizer*. It is defined by its Content, a list of *anchor*s, and a list of *properties*.

Media entities are capable of producing output in certain modalities through audiovisual devices and actuators. For instance, a TTS (Text-To-Speech) synthesizer produces audio speech to be played through a speaker. Similar to the recognizer *content*, the media *content* is also decoupled from the multimedia document and it is referenced by a location property. The

media *content* can be an ordinary audiovisual content —such as audio, video, image— or a document described in a unimodal synthesizer language, *e.g.* an SSML file. *Anchor*s are portions of the content presentation. For instance, an *anchor* may define a temporal portion of a video (*e.g.* a segment of the video presentation ranging from 30 to 50 seconds). Additionally, an *Anchor* may refer to the identifier of an internal element in the synthesizer document, *e.g.* an SSML element to be synthesized. *Properties* define parameters for *Media*, such as the transparency of video, the volume of the audio output produced by an SSML speech synthesizer, the frame rate of a BML avatar rendering animation, etc.

Recognizer entities are capable of identifying an expected input in a certain modality captured from an input device or sensor. For example, an ASR (Audio Speech Recognition) recognizer identifies sentences in audio samples captured by a microphone. The *content* of a *Recognizer* is decoupled from the multimedia document and it is referenced by a location property (the URL of the content). The content can be a document described in a unimodal recognizer language, *e.g.* an SRGS file. *anchor*s are portions of the recognizer content. They define when or what to expect, within a document, a certain input modality to be recognized. For instance, an SRGS defines one rule element for each expected speech input (*e.g.* a certain rule may define that acceptable input tokens are "put that" and "there"). An anchor enables the multimedia document to know the moment when the recognition of a token is complete. *propertie*s define parameters for the *Recognizer* or dynamic values inside the *Recognizer*. For example, an ink recognizer can have properties to indicate the current position of the ink pointer.

3.2 UserClass

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, a developer can create applications that are aware of the users who are interacting with them, and how. To support multiuser features, the *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 its *minCardinality* and *maxCardinality* attributes, and by a *userClassDescription*. The id attribute uniquely defines a UserClass. The cardinality attributes allow us 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 who can interact with the system must have a *profile description*. The user profile may include, for example, information such as whether he/she is a student or a professor. Moreover, it may include the devices he/she is using to interact with the application. The *userClassDescription* then should specify which users, based on their profiles, are part of each *UserClass*.

Note that the specification of *userClassDescription* is tied to the specific vocabulary used to describe the user's profiles. Our model is agnostic and does not prescribe the *userClassDescription* specification details, which should be defined by the language instantiation. For instance, we propose in our language instantiations (discussed in Chapter 4) the use of an RDF description for user *profile description* and *SPARQL*, which enables queries over RDF entities, for *userClassDescription*.

Runtime properties related to users in a *UserClass*, such as the number of users registered in a class and user context, should be stored as global variables. In particular, regarding the user context, the language must support the description of human sensory capabilities, which can be used during modality selection. Examples of such variables are *canHear*, *canSee*, and *canSpeak*. Therefore, a modality selection can be specified through conditional control structures, such as *switch* or *if-then-else*, which evaluate these runtime properties.

Table 3.1 shows examples of modality selection considering users' sensory capabilities.[1] For instance, when the *canSee* variable is *false*, audio or speech modalities should be used instead of visual and GUI modalities; and when *canHear* is *false* an avatar-based modality should be selected instead of audio modalities.

This thesis focuses on specification issues of multiuser interactions. More precisely, we focus on how the author defines user interaction requirements and uses context information. We do not directly address, nor prescribe, how the multimedia system should gather the users' *profile description* and retrieval of their runtime context variables. These tasks can be achieved by explicit user identification (*e.g.* via log in) or by recognition techniques using sensors (*e.g.* via facial recognition).

[1]The recommendation of modalities given users' sensory capabilities should draw on accessibility research results. Table 3.1 is simply an illustration that our approach makes it feasible to build MUIs considering those capabilities.

Variables	When false	When true
canSee	– Audio modalities (*e.g.* audio/*) – Speech modalities (*e.g.* SRGS, SSML)	– Visual modalities (*e.g.* video/* img/*, text/plain) – GUI modalities (*e.g.* key presses, mouse
canHear	– Avatar modalities (*e.g.* BML)	– Audio modalities (*e.g.* audio/*)
canSpeak	– Audio modalities (*e.g.* audio/*)	– Speech modalities (*e.g.* SRGS, SSML)

Table 3.1: Comparison among the features supported by the different MUI development approaches features.

3.3 Relationship

Relationship entities enable developers to express the combined use of the modalities perceived by users through *Media* and the modalities generated by users through *Recognizers*. In the "Put-That-There" scenario, for instance, Relationship entities express the combined use of speech syntheses and visual objects (*Media*) with the speech and gesture recognition (*Recognizers*). *Relationship* entities are defined by a set of *conditions* and *actions*. When the *conditions* are satisfied, then the *actions* are performed. The actions and *conditions* in a *Relationship* are detailed as follows.

Regarding the *actions* of the *Relationship*, they enable developers to change the properties and to control the activation/deactivation of *Recognizers* and *Medias*. A *set* action changes the value of a property. Four actions can manipulate the activation/deactivation of *Recognizers* and *Media*: *start*, which activates a recognizer or synthesizer; *stop*, which deactivates a *Recognizer* or *Media* and releases its resources; *pause*, which deactivates a *Recognizer* or *Media*, but does not release its resources; and *resume*, which reactivates a previously paused *Recognizer* or *Media*. Table 3.2 presents the semantics of those actions over *Recognizers*, *Media*, and their *anchors*.

In a *Recognizer*, the activation of an anchor "a1" means that the portion of the *Recognizer* content related to "a1" is enabled to be recognized. By default, activating a *Recognizer* without specifying a specific *anchor* is equivalent to activating all of its *anchor*s. The recognition of the *anchor* content itself only occurs when the *Recognizer* identifies the required pattern in the user input in the corresponding modality. For instance, consider a speech recognizer that defines two anchors, for the tokens "put that" and "there". When the recognizer is activated without specifying an anchor, both anchors are activated and, thus, either one can be recognized in the audio input (*e.g.* captured by a microphone). Moreover, the developer can restrict the recognizable content by

starting the anchors individually.

Action	**Over**	**Semantics**
start	*Media*	*Media* starts rendering its content from the beginning.
	An anchor of a *Media*	*Media* starts rendering its content from that specific anchor.
	Recognizer	*Recognizer* is activated and it is able to recognize all of the anchors of the recognizer content.
	An anchor of a *Recognizer*	*Recognizer* is activated and it is able to recognize only the content related to that specific anchor.
stop	*Media*	*Media* stops rendering its content.
	An anchor of a *Media*	*Media* stops rendering the content of that specific anchor.
	Recognizer	*Recognizer* is deactivated and it is unable to recognize any content.
	An anchor of a *Recognizer*	*Recognizer* is deactivated and it is unable to recognize the content related to that specific anchor.
pause	*Media*	*Media* pauses rendering/synthesizing its content, but the system does not release its resources.
	An anchor of a *Media*	*Media* pauses rendering/synthesizing its content from that specific anchor, but the system does not release the synthesizer's resources.
	Recognizer / anchor of a *Recognizer*	Acts as stop.
resume	*Media*	*Media* restarts rendering/synthesizing its content from the previously paused state.
	Recognizer	*Media* restarts rendering/synthesizing its content from the previously paused anchor.
	Recognizer / anchor of a *Recognizer*	Acts as start.

Table 3.2: Semantics of the actions over media, recognizers, and their anchors.

In a *Media*, the activation of an anchor means that the *Media* is rendering the content related to that *anchor*. While a synthesizer is activated and being rendered, its anchors are automatically activated/deactivated as the presentation content of the *Media* reaches the content portion related to each *Anchor*. For instance, consider an *anchor* "a1" of an audiovisual media "m1" from 300 to 360 seconds. This *Anchor* will be automatically activated when the presentation of "m1" reaches 300 seconds, and deactivated when it reaches 360 seconds. This behavior is similar to other types of *Media*. For instance, consider an *anchor* "a2" that points to a specific token of an SSML document associated to a Media "s1". In this case, when the presentation of the SMML document —*i.e.* the speech synthesis— reaches the token related to "a2", then "a2" is activated.

A *condition* may test:

- the activation of *Recognizers* and *Media*, or their anchors. For instance, we can define a *Relationship* to be triggered when a (portion of a) video is activated with given attributes;
- the recognition of portions of the *Recognizer* content. For instance, we can define a *Relationship* to be triggered when the recognition of "hello", defined in an SRGS file, is complete;
- values of properties of *Recognizer* or *Media* and runtime of a user's context. For these value tests, the multimedia language must provide comparators, such as: is equal to, greater than, and less than. For instance, we can define a Relationship to be triggered when the volume of a video becomes greater than 70% or when the *canHear* value is equal to "false".

*CompoundCondition*s specify the combined use of *conditions* over the modalities. For instance, a *Relationship* can be defined to be triggered when portions of a video are activated and the recognition of a sentence is complete. The following operators are proposed here:

- *or*: the *Relationship* will trigger when any one of the conditions is satisfied;
- *and*: the *Relationship* will trigger when all the conditions are satisfied, independently of the order in which they are satisfied;
- *seq*: the *Relationship* will trigger when all the conditions are satisfied in the specified order. This operator was not present in the relationship entity of NCM.

CARE properties	**Relationship operators (our approach)**	**SMUIML transition operators**
Equivalence	Compound condition using the or operator	Recognizers trigger using the seq_or operator
Assignment	A simple condition	Only one trigger
Redundancy (use temporal window)	Compound condition using the or operator and activated within a temporal window	Recognizers trigger using the par_or operator
Sequential complementarity	Compound condition using the seq operator	Recognizers trigger using the seq_and operator
Parallel complementarity (use temporal window)	Compound condition using the and operator and activated within a temporal window	Recognizers trigger using the par_and operator

Table 3.3: Expressiveness of the multimodal *Relationship* operators compared to SMUIML.

Indeed, the combination of interaction modalities is an important feature of MUIs. To analyze the expressiveness of the proposed operators, Table 3.3 compares them to the CARE properties. *Equivalence* can be expressed by a compound condition using the or operator. *Assignment* can be expressed by a simple condition. *Redundancy* can be expressed by compound conditions using the *or* operator defined within a temporal window. The temporal window may be defined by activating/deactivating modalities within a specific period of time. *Sequential complementarity* is expressed by a compound condition using the *seq* operator. Finally, *parallel complementarity* is expressed by a compound condition using the *and* operator defined within a temporal window. As highlighted in Table 3.3, the proposed operators have the same expressiveness as the CARE properties and SMUIML operators. However, CARE properties and SMUIML operators consider only input modalities, whereas our approach supports simple and compound conditions relating any combination of input and ouput modalities. In addition, our proposal supports conditions for evaluating the user's context.

To support multiuser-oriented development, the recognition *condition* in a *Relationship* can be parameterized with a "user_id" attribute. The value of this attribute defines which specific user from a *UserClass* can be responsible for generating the event. By doing this, it is possible to limit which specific users can trigger a *Relationship*. Moreover, we define another optional attribute in conditions, called "excluded_user_id". In this attribute, developers can define a list of users who are not allowed to trigger a condition.

3.4 Discussion

We argue in this thesis, that an multimedia language that instate our entities will support both multimodal and multiuser interactions. We believe that these extensions can contribute to the state of the art by overcoming the drawbacks and same time increase multimodal specification expressiveness.

We handled the lack of support for fine synchronization among modalities mainly by reusing the multimedia players' infrastructure, which keeps media object presentations finely synchronized. In our approach, the multimedia language and its player should also handle recognizers for input events, in the same infrastructure. The recognizers are responsible for interacting with the user input devices and sensors, and for triggering anchor events that enable synchronization by the multimedia player.

Regarding the strong encapsulation between fusion and fission, we solved this issue by using a structure-based [58] paradigm and using the NCM

anchor concept. In addition, we have extended this concept to be used not only by output modalities (*e.g.* instantiated as <media> in NCL), but also by input modalities (<input>). By so doing, the multimedia document is decoupled from the content for recognizers and synthesizers. If necessary, however, the internal behavior of those recognizers and synthesizers can be known and controlled by the multimedia application through their anchor events. Conversely, the languages and frameworks discussed in Chapter 3 delegate one of them to another entity. For instance, SMUIML sends ad-hoc messages to a HephaticsTK framework client.

To solve the lack of support for modality selection considering sensory capabilities, the framework uses context variables that are able to express user capabilities. Those variables could be used in conditional constructions to enable the right modality selection based on the user's sensory capabilities.

Finally, we overcame the interacting users are second-class citizens through enabling the specification of interacting users, and the association of a user with recognition events.

Besides overcoming the above drawbacks, our proposal also achieves more expressiveness when combining both output and input modalities. It does so by using the causal relationship and anchor concepts from NCM. Those concepts enable NCM express Allen's temporal relations among output modalities. In our approach, we enable such concepts for input modalities which enable it express Allen's temporal relations among both output modalities and input modalities.

4
Language instantiations

The model presented in the previous chapter was proposed to be agnostic to multimedia languages and their syntax. In this section, we discuss the instantiation of the model entities as first-class citizens and their proposed syntax in two multimedia languages: NCL (Section 4.1) and HTML (Section 4.2).

4.1
NCL instantiation

NCL 3.0 is an XML multimedia language based on the Nested Context Model (NCM) 3.0, a model for hypermedia document specification which allows defining temporal and spatial relationships among media objects. The current entities of NCM 3.0 do not focus on representing either modalities different from the GUI-based ones, or interactions aware of the users who interact with the application. To overcome these limitations, we first instantiate our entities in NCM 3.0. Figure 4.1 illustrates the current NCM entities, highlighted in light blue, and our extensions, highlighted in light green. Appendix A presents XML schemas, which detail our NCL 3.0 syntax extensions.

The main entities of NCM 3.0 are *Node* and *Link*. We briefly detail them in what follows.

An NCM 3.0 *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 *Anchor*s in the list of *Anchor*s 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*. *ContentNode* has been specialized mainly for 2D audiovisual media modalities, such as *TextNode*, *ImageNode*, and *VideoNode*. *ContentAnchor*s of *ContentNode* are spatial or temporal portions of the corresponding 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 by references to its presentation times.

An NCM *Link* is defined by a *Connector* and a list of *Bind*s. *Connectors*

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

A Role is defined by the attributes: *RoleID*, *EventType*, *minCardinality*, and *maxCardinality*. The *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. In particular, the *SelectionEvent* has a key attribute that defines the key (*e.g.* keyboard or remote control) that triggers the event.

Whereas a connector may represent any kind of relationship between node anchors, NCL 3.0 defines only the *CausalConnector* relationship. *CausalConnector* specifies which conditions (*ConditionRoles*) need to 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 represent a logical expression using "and" or "or" operators through *SimpleCondition*s and *AssessmentSatement*s. An *AssessmentSatement* is used to compare event attributes. For instance, *SelectionEvent* has a key attribute, which can be tested in *SimpleCondition* or *AttributeAssessment*. Finaly, *ActionRole* may define changes in presentation state or properties of *Node*s.

In the remainder of this section (subsections 4.1.1 and 4.1.2), we present our extensions to NCM and NCL aiming to address these limitations.

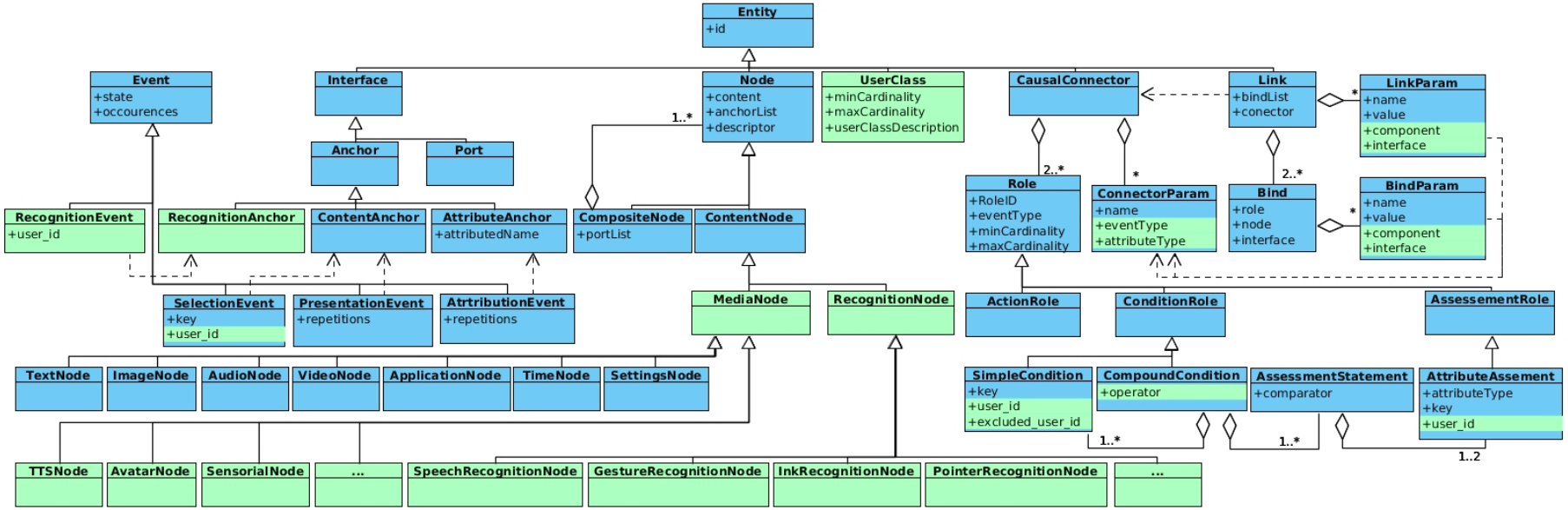


Figure 4.1: NCM 3.0 and proposed extensions.

4.1.1 Multimodal specializations for Node

To add our *Media* and *Recognizer* entities to NCM 3.0, we propose multimodal specializations for Node, named *MediaNode* and *RecognitionNode*.

In a *ContentNode* of NCM 3.0, 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 [29], for example), useful for visually impaired users; (2) *AvatarNode*, representing embodied conversational agents (*e.g.* described using BML [30]), useful for deaf-, children- or elderly-oriented interfaces; and (3) *SensorialNode*, representing sensorial effects (*e.g.* described in MPEG-V SEDL [31]), useful for increasing the QoE (Quality of Experience) [4] of multimedia presentations. Other specializations to *MediaNode* representing other modalities can be seamlessly integrated in the future.

For the representation of input modalities, we propose the new *RecognitionNode* as a specialization of *ContentNode*, which can be used in Link elements. 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 [25] for *SpeechRecognitionNode*; GDL (Gesture Description Language) [27] for *GestureRecognitionNode*; and InkXML [26] for *InkRecognitionNode*. Also, other specializations to *RecognitionNode* representing other input modalities can be seamlessly integrated in the future.

Since *RecognitionNode* is indeed a specialization of *ContentNode*, it is also possible to define Anchors in it. A special type of Anchor, the *RecognitionAnchor*, specifies a portion of the recognition content and is associated to a "recognition" event. For instance, a *RecognitionAnchor* may refer to expected speech tokens defined in a *SpeechRecognitionNode* or a "move" or "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 is not intrinsically coupled with *MediaNode* events.

Based on the above extensions to the NCM, we also defined how those changes can be mapped onto NCL 3.0, the concrete syntax of the model.

NCL 3.0 [17] defines the <media> element for specifying audiovisual media in a multimedia document. It has the advantage of being media-type agnostic. Following the same principle, we propose to use <media> elements to support not only audiovisual media, but also any type of synthesized description, such as SSML and SEDL. Therefore, there are no changes in the <media> element.

To allow the integration of *Recognizer* in NCL, we propose a new element for the language, named <input>, because NCL 3.0 only considers GUI-based interactions. Analogous to <media>, <input> defines the *Recognizer* content location (src), its properties (<property>), and its anchors (<area>). The src attribute refers to the Recognizer content, described in languages such as SRGS and GestureML. To represent that a content was recognized, we define a new event type, named "recognition" event, which can only be associated to <input> elements. Similar to other types of events, it introduces reserved words for the start of the recognition ("onBeginRecognition") and for its end ("onEndRecognition" or "onRecognize").

To illustrate the usage of the these multimodal specializations, we created an extended version of "Sightseeing of Today" [59], an interactive non-linear video [60] about sightseeing in a city. In its original version, the user interacts via key/mouse to navigate between videos; in some opportunity time windows, the user can choose which touristic place the user will be guided next. The choice is: if the user presses the RED button, the "downtown" video is started; and if the user presses the GREEN button, then the "beach" video is started. Our extended version, named "Multimodal Sightseeing of Today", enables video navigation also via voice commands.

"Multimodal Sightseeing of Today" uses two multimodal descriptions, an SSML (Listing 4.1) and an SRGS (Listing 4.2) file, to enable the user to choose which video to play next.

Listing 4.3 shows the code fragment responsible for controlling the first navigation. It defines four <media> and one <input> elements. Three <media> elements ("intro", "videoDowntown" and "videoBeach", lines 16-25) define the introductory video and the two videos available for the user to choose from. The "intro" video has an anchor ("choice_moment") starting at 40 seconds in the video. The fourth <media> "audio_choice" (lines 26) refers to speech synthesis and the "asr_places" <input> (lines 79-82) supports voice commands for navigation control in this first interaction opportunity.

This <input> element defines two anchors mapping onto rules specified in the "places.sgrs" file, defining the words "downtown" and "beach".

Regarding the application behavior, it begins by starting the "intro" video (line 15) and defines three <link> elements. The first link (lines 31-37) defines that, when the "choice_moment" anchor is reached, then application asks for possible places via voice command. The other two links (lines 29-47) define that when the user says the name of a recognized place, then the corresponding video should be started.

```
1<speak xmlns="http://www.w3.org/2001/10/synthesis">
2  <s>Do you want visit the Rio de Janeiro's downtown or the
3  Copacabana beach?</s>
4  ...
5</speak>
```

Listing 4.1: downtown_or_beach_audio.ssml.

```
1<grammar xmlns="http://www.w3.org/2001/06/grammar">
2  <rule id="downtown">downtown</rule>
3  <rule id="beach">beach</rule>
4  ...
5</grammar>
```

Listing 4.2: places.sgrs.

```
1<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
2  <head>
3    <connectorBase>
4      <causalConnector id="onBeginStart">
5        <simpleCondition role="onBegin"/>
6        <simpleAction role="start"/>
7      </causalConnector>
8      <causalConnector id="onRecognizeStart">
9        <simpleCondition role="onRecognize"/>
10       <simpleAction role="start"/>
11     </causalConnector>
12   </connectorBase>
13 </head>
14 <body>
15   <port component="intro"/>
16   <media id="intro" src="intro.mp4">
17     <property name="size" value="100%, 100%"/>
18     <area label="choice_moment" begin="40s"/>
19   </media>
20   <media id="videoDowntown" src="downtown.mp4">
21     <property name="size" value="100%, 100%"/>
22   </media>
23   <media id="videoBeach" src="beach.mp4">
24     <property name="size" value="100%, 100%"/>
25   </media>
26   <media id="audio_choice" src="audio_downtown_or_beach.ssml"/>
27   <input id="asr_places" src="places.sgrs">
```

```
28      <area label="downtown" />
29      <area label="beach" />
30    </input>
31    <link xconnector="onBeginStart">
32      <bind role="onBegin" component="intro" interface="choice_moment"/>
33      <bind role="start" component="audio_choice"/>
34      <bind role="start" component="asr_places" interface="downtown" />
35      <bind role="start" component="asr_places" interface="beach"/>
36    </link>
37    <link xconnector="onRecognizeStart">
38      <bind role="onRecognize" component="asr_places" interface="beach"/>
39      <bind role="start" component="videoBeach"/>
40    </link>
41    <link xconnector="onRecognizeStart">
42      <bind role="onRecognize" component="asr_places" interface="downtown"/>
43      <bind role="start" component="videoDowntown"/>
44    </link>
45    ...
46  </body>
47</ncl>
```

Listing 4.3: "Multimodal Sightseeing of Today" NCL application.

4.1.2 Linking Multiple Modalities and Users

An important feature of MUIs is the combination of interaction modalities. According to Nigay and Coutaz [40], 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 *CompoundConditions* with *RecognitionNodes*. 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.).

To enable multiuser features in NCM, we created a new *UserClass* entity. In NCL 3.03 syntax, we add the <userClass> element defined inside <userBase>, in the document head (<head> element). It has id, min, max, and userClassDescription attributes. The userClassDescription is a URL to an external document describing the UserClass. As previously mentioned, userClassDescription is tied to how user profiles are specified.

In our proposed instantiation, the system identifies the user and defines their profiles in RDF (Resource Description Framework) [45]. Then a *userClassDescription* can use a SPARQL [61] query to choose among the users. Each user is a foaf:Person element from the FOAF [62] RDF vocabulary, used for user profiles. Additionally, this foaf:Person may define prf:name elements from UAProf [55] RDF vocabulary, used to define device profiles. The SPARQL query is responsible for selecting which users should be part of the *UserClass*.

Once having defined a *UserClass*, the developer may define <simpleCondition> elements using an event attribute named "user_id", using the scheme "user-class-id(user-id)". For instance, a <simpleCondition> with role and user_id attributes with values "onRecognize" and "BoltLikeUser(1)", respectively, is trigged only when an interaction is recognized from the first registered user from the "BoltLikeUser" *UserClass*. Moreover, we define another optional attribute in <simpleCondition>, called "excluded_user_id". In this attribute, authors can define the users who are not allowed to trigger a <simpleCondition>. Such specification of user as parameter is similar to how Soares [43] 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 developers, in <simpleCondition> elements.

In our model, the *UserClass* should enable access of runtime properties related to its users. In the extended NCL, this access occurs though an <media> element from type "x-ncl-settings" using the scheme: "user-class-id(user-id). property-name". For instance, the value "BoltLikeUser(1).canHear" access the hearing capability of the first registered user from the "BoltLikeUser" <userClass>.

Finally, it is also interesting to give authors access to the event attributes through Links. For instance, the author can store the last "user_id" from a "recognition" event or the coordinates of a pointer interaction. 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.

To illustrate the usage of these <link> multimodal and multiuser features, Appendix B presents the "Put-That-There" scenario in NCL and its two multiuser versions, namely: "I-Get-That-You-Put-It-There" and "Anyone-Get-That-Someone-Else-Put-It-There".

4.2 HTML instantiation

HTML [16] is a markup language mainly focused on supporting text-based interactive content, which has recently included support for audio and video content (HTML 5.0). Like NCL, HTML does not focus on representing modalities different from the GUI-based ones, nor about interactions aware of the users who interact with the application. To overcome these limitations, we have instantiated our proposed entities in HTML.

At some point, W3C has proposed that HTML should evolve into an XML-based equivalent, namely XHTML, focusing on XML modularization. The browser vendors, however, argue that HTML evolution should not follow this path and continue to use their own markup. Nowadays, when an HTML document, either in markup or XML syntax, is loaded into a browser engine, it becomes an object tree following the Document Object Model (DOM) standard [63]. The DOM API for HTML, simple called HTML DOM, allows programs (*e.g.* browsers) and scripts to dynamically access and update the content, structure, and style of a document, regardless of its syntax (*e.g.* HTML, XML). Figure 4.2 illustrates the current HTML DOM entities, in light blue, and our extensions, in light green. In particular, the main entities of HTML are Node and HTMLElement [64].

In HTML, everything is a *Node*, including the HTML document itself. Every *Node* element inherits *EventTarget*, which enables the scripts elements (HTMLScript) to use the DOM API to register event handlers on elements in an HTML document. Nested in the HTML document node, there are both markup elements, namely Element, and text only nodes, namely *CharacterData*. Examples of Element are HTMLElement, for HTML markup elements (*e.g.* , <img>, <p>), and *SVGElement*, for SVG markup elements. Two specializations of *HTMLElement* are *HTMLScriptElement* and *MediaElement*. The *HTMLMediaElement* is the basic entity for continuous media, such as *HTMLVideoElement* and *HTMLAudioElement*.

We propose to instantiate our entities in the HTML DOM using a browser vendor standard called CustomElements, which provides a JavaScript API to extend the HTML markup. In other words, it enables developers to create their own reusable HTMLElements in JavaScript. In our case, we propose to implement our model entities (*i.e.* *Media*, *Recognizer*, *Relationship* and *UserClass*) in JavaScript at runtime. In fact, a similar approach was followed by Soares Neto *et al.* [65], which at preprocessing time implemented their template language entities (TAL) using JavaScript.

To implement our *Media* concept, we propose to reuse the existing HTML

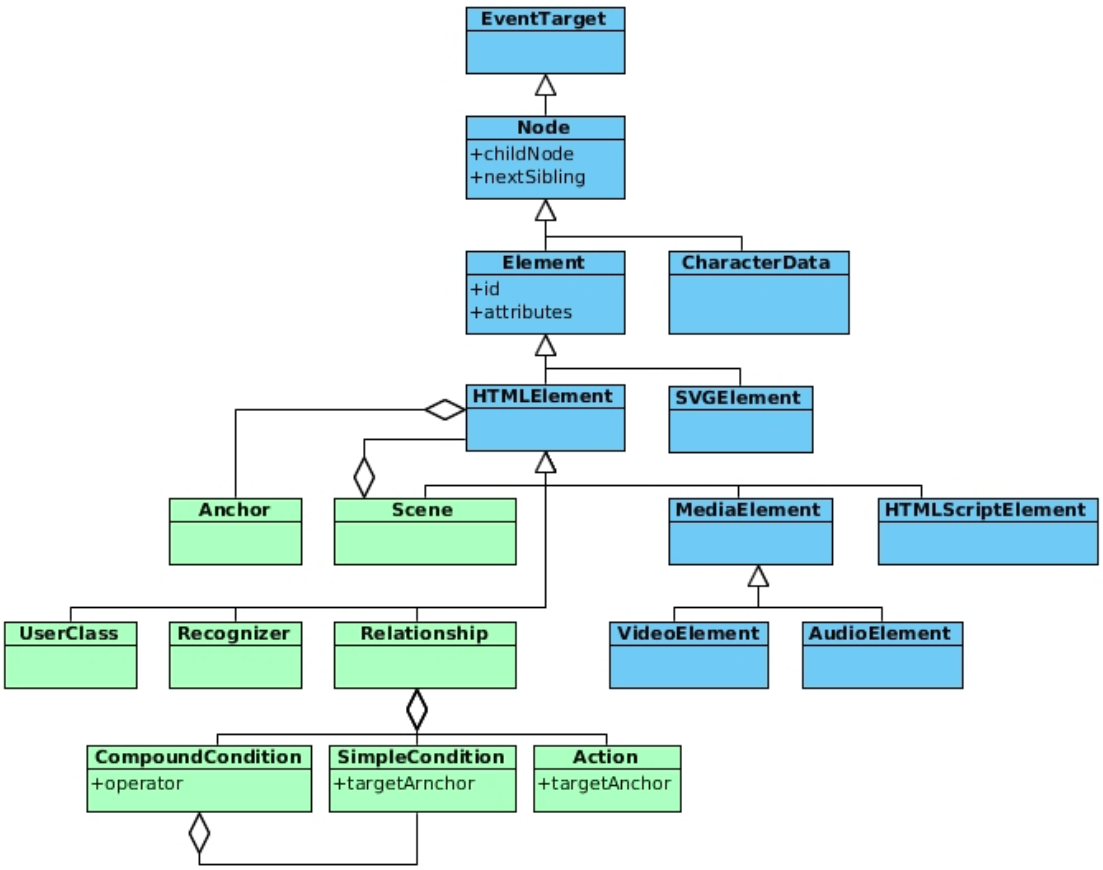


Figure 4.2: HTML DOM and proposed extensions

audiovisual modalities elements, such as <img>, <audio>, <video>, and to use a new <mm-media> element to provide synthesized modalities that inherit from HTMLElement. To implement the Recognizer concept in HTML, we propose a new element <mm-input>. All those elements may use a new <mm-area> to support our ContentAnchor and RecognitionAnchor elements.

Regarding the *Relationship* entity, we propose the <mm-link>. The <mm-link> behaves like the NCL <link>, but simplified. All common connectors do not need to be defined. Simple condition elements may be directly defined by elements <mm-onBegin>, <mm-onEnd>, and <mm-onRecognize> and action by <mm-start> and <mm-stop> elements. More than one condition can be grouped in a <mm-compoundCondtion> element inside a <mm-link>. The <mm-compoundCondition> should use an operator.

All the proposed elements, may be grouped inside a <mm-scene> element. This element enables the correct semantics of the <mm-link>. In HTML, <img> or <video> elements inside the <body> are visible by default. The elements inside an <mm-scene> are presented only be actions defined in the <mm-link> elements.

To illustrate the usage of this HTML syntax, we implemented the

"Multimodal Sightseeing of Today". To do that, we use the same SSML (Listing 4.1) and SRGS (Listing 4.2) multimodal descriptions as in the NCL version.

Listing 4.4 shows the code fragment responsible for controlling the first navigation. It defines four Media and one <mm-input> element. Three Media elements ("intro", "videoDowntown" and "videoBeach", lines 12-19) define the introductory video and the two videos available for the user to choose from. The "intro" video has an anchor ("choice_moment") starting at 40 seconds in the video. The fourth <mm-media> "audio_choice" (lines 26) is speech synthesis and "asr_places" <mm-input> (lines 79-82) supports voice commands for navigation control in this first interaction opportunity. This <mm-input> defines two anchors mapping onto rules specified in the "places.sgrs" file defining the words "downtown" and "beach".

Regarding the application behavior, it begins by starting the "intro" video (line 15) and defines three <mm-link> elements. The first link (lines 28-32) define that, when the "choice_moment" anchor is reached, then the application should ask for possible places via voice command. The other two <mm-links> (lines 33-40) define that, when the user says the name of a recognized place, then the corresponding video should be started.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="js/elements/mm.js"></script>
5 </head>
6 <body>
7   <mm-scene id="scene">
8     <mm-link>
9       <mm-onBegin interface="scene" />
10      <mm-start interface="intro" />
11    </mm-link>
12    <video id="intro" src="intro.mp4"
13      style="position: absolute; height 100%; width: 100%;">
14      <mm-area id="choice_moment" begin="40s" />
15    </video>
16    <video id="videoDowntown" src="downtown.mp4"
17      style="position: absolute; height 100%; width: 100%;">
18    <video id="videoBeach" src="beach.mp4"
19      style="position: absolute; height 100%; width: 100%;">
20    <mm-media id="audio_choice"
21      src="audio_downtown_or_beach.ssml" />
22    <mm-input id="asr_places"
23      src="places.sgrs">
24      <mm-area label="downtown" />
25      <mm-area label="beach" />
26    </mm-input>
27    <mm-link>
28      <mm-onBegin interface="intro.choice_moment" />
29      <mm-start interface="audio_choice" />
```

```
30      <mm-start interface="asr_places.downtown" />
31      <mm-start interface="asr_places.beach" />
32    </mm-link>
33    <mm-link>
34      <mm-onRecognize interface="asr_places.beach" />
35      <mm-start interface="videoBeach" />
36    </mm-link>
37    <mm-link>
38      <mm-onRecognize interface="asr_places.downtown" />
39      <mm-start interface="videoDowntown" />
40    </mm-link>
41  </mm-scene>
42</body>
43</html>
```

Listing 4.4: "Multimodal Sightseeing of Today" HTML application.

Finally, we create a new <mm-userClass> with id, min, max, and *userClassDescription* attributes. The *userClassDescription* is a URL to a SPARQL document [61] defining the required characteristics of the users. Once having defined an <mm-userClass>, the developer may define <mm-Selection> and <mm-onRecognize> elements using an event attribute named "user_id".

5
Evaluation

We address RQ1 by proposing entities (discussed in Chapter 3) to be instantiated in multimedia languages. After doing so, we have also proposed syntax instantiations for NCL and HTML (discussed in Chapter 4).

Nowadays, the development of NCL and HTML applications can be done using different approaches and representations. For instance, today NCL provide alternative representations to its XML syntax, such as JSON objects [66] and Lua tables [67]. Moreover, it is possible to develop in HTML using some alternative representations, such as in XML (*i.e.* XHTML) and YAML, or even to use only JavaScript to create the entire HTML DOM.

Given the above context, to evaluate our answer, we confront an additional question about *"how to evaluate the usage of the proposed entities despite the different ways of instantiating them in a multimedia language?"*. In other words, if we only evaluate the usage of our proposed syntax, the evaluation results will be tied to the syntax development characteristics. For the NCL syntax, for instance, Soares Neto *et al.* [68] performed a usability analysis and highlighted NCL verbosity and error-prone characteristics. To answer such question, we performed an evaluation organized in two parts. The first part focuses on the conceptual entities, whereas the second part focuses on our proposed syntaxes for HTML and NCL.

Planning the analysis for the conceptual entities was a challenge, because they were created to be independent from representation syntax. To do so, we used a block-based programing paradigm to enable users to develop applications using only the concept entities, at a level of abstraction higher than that of either NCL or HTML. This paradigm is commonly used for teaching programming or in code generation tools. In particular, this type of development has been popularized by tools such as MIT Scratch and MIT App Inventor. Although our block representation also contains its own syntax, a block syntax helps users to abstract away from specificities and lower-level textual syntax of the languages [69], helping developers to focus on the concepts we wish to evaluate.

The analyses were performed with NCL and HTML developers through a web-based evaluation form. This form not carry an execution runtime for the

application, i.e., the developers not visualize the multimedia application results; it focus only in presents model entities in the block-based and in extended language representations to capture their understanding and acceptance by the developers. In the next sections, we briefly present our block-based representation (subsection 5.1), detail our evaluation form (subsection 5.2), and discuss the results (subsection 5.3).

5.1 Block-based representation

Our block representation was developed using the web-based version of the Blockly framework. In this framework, the blocks are defined as JavaScript objects and then instantiated as SVG elements in HTML DOM. Then, those blocks as SVG elements are showed in a workspace , where the user can drag and drop elements, fill in some fields, and join elements together. We have created four group blocks, each one related with one entity.

In our block representation, a *Media* entity is defined by joining a media block, with the id field filled in, and a media content block, which can be an image, audio, video or speech synthesis block. Similar, a Recognizer entity is defined by joining a recognizer block, with the id field filled in, and a recognized content block, which can be a speech or hand gesture block. Figure 5.1 illustrates the block groups related with the *Media*, *Recognizer* and *UserClass* entities.

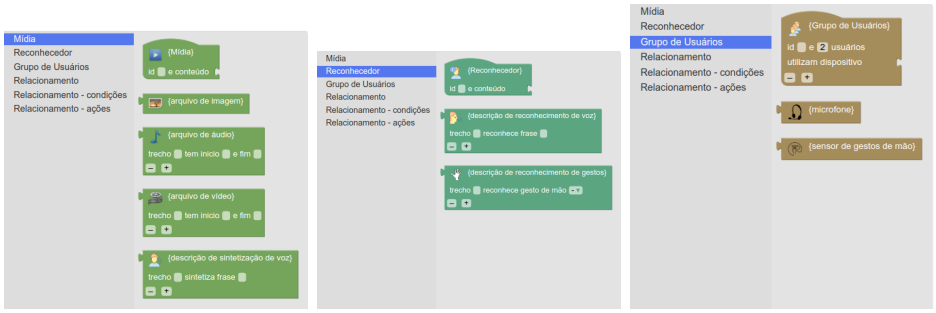


Figure 5.1: Blocks groups related to Media, Recognizer and UserClass.

A *Relationship* is defined by joining one *Relationship* block with conditions and action blocks. Condition blocks can be simple or compound. Simple condition blocks can define the trigger for: begin or end of media/recognizer anchor or block id; selection of media block; recognition of recognizer anchor or block id. A compound condition block allows combining other condition blocks and use a combination operator ("OR", "AND", "SEQ"). Finally, the action blocks can be start or stop a media/recognizer anchor or block id. Figure 5.2 illustrates the blocks related with the *Relationship* entity. To prevent users

from having the fill in all id values, the fields in these Relationship blocks are dropdowns, which list the existing media/recognizer anchor or block ids.

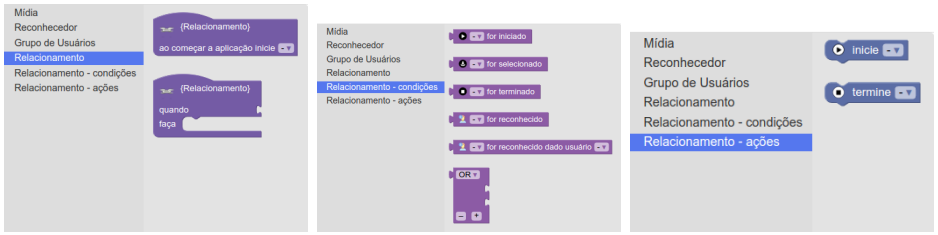


Figure 5.2: Blocks groups related to Relationship related.

To illustrate the usage of our block representation, Figure 5.3 presents the "Multimodal Sightseeing of Today" application. It defines four *Media* and one *Recognizer* (left part of Figure 5.3). The first three Media elements ("video_principal", "video_praia" and "video_centro") define the introductory video and the two videos available for the user to choose from. The "intro" video has an anchor ("creditos") starting at 300 seconds in the video. The last *Media* ("sinte_voz") defines a speech synthesis media asking about the navigation control. Finally, the *Recognizer* ("asr_places") defines the speech recognition for navigation control. This Recognizer defines two anchors, mapping onto the words "centro" and "praia".

Regarding the application behavior, we define four *Relationship*s (right part of the Figure 5.3): The first one defines that the application begins by starting the "video_principal" video. The second one defines that, when the "creditos" anchor is reached, then the application asks for possible places via voice commands. The other one defines that, when the user says the name of a recognized place, then the corresponding video should be started.

5.2 Evaluation form

Our evaluation form aimed at capturing from NCL and HTML developers' indications of their acceptance of our proposal. More precisely, the form presents questionnaires about both representations of our entities using the block-based and language syntax representations. Our questionnaires were based in the Technology Acceptance Model (TAM) [70].

TAM is based on empirical studies and argues that users' acceptance of a technology is influenced mainly by the perceived usefulness (PU) and the perceived ease of use (PEOU) of the technology. In our evaluation form, we defined PU and PEOU questions guided by Gefen's and Keil's [71] examples for both block-based and language representations. However, TAM only captures the users' perception and not their actual understanding of a

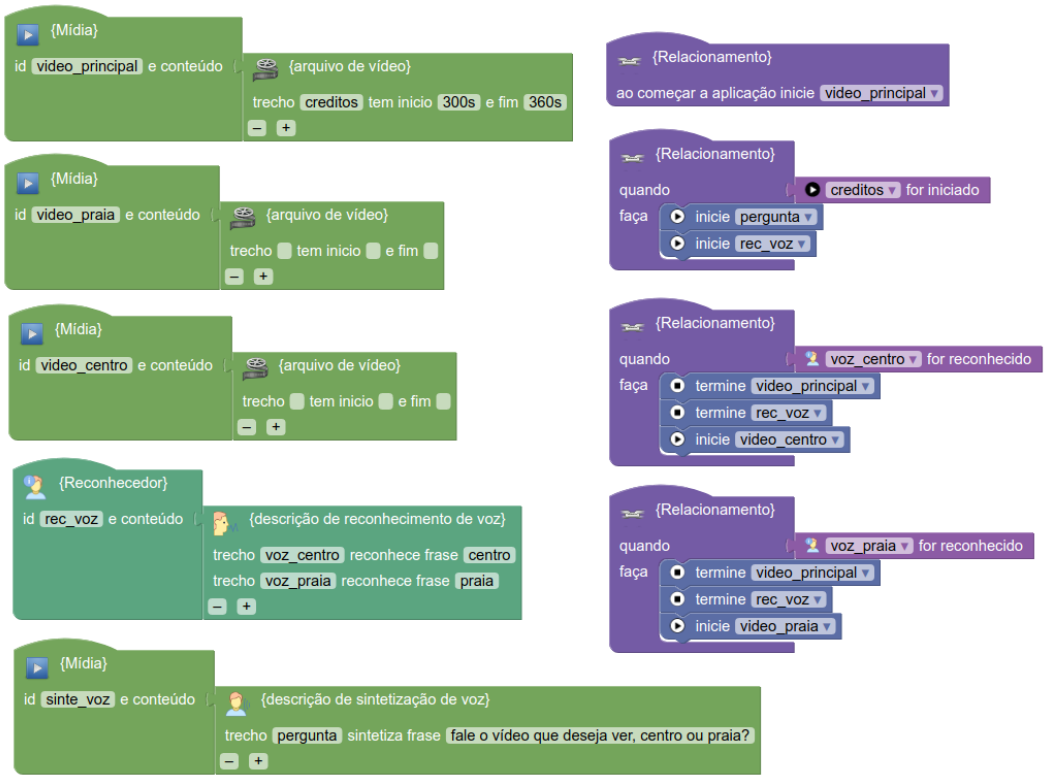


Figure 5.3: Block-based representation of "Multimodal Sightseeing of Today".

technology. To overcome this issue, we not only presented the representations but we asked users to perform tasks using them.

The evaluation study comprised 37 participants. In our analyses, we organize them into two groups based on their self-assessment of their knowledge of NCL and HTML. As we expected to have more volunteers knowledgeable in HTML, if a participant answered had the same degree of knowledge in both languages, we allocated him/her to the NCL group. We ended up with 21 participants in the HTML group and 16 in the NCL group. Because of the different sizes of the groups, all charts presented in this section use percentages to inform the proportion of participants inside each group gave a certain answer. The evaluation form is organized in seven pages. The first five pages target all participants, because they introduce concepts, ask profile questions and use the block-based representation. The last two pages are adapted given the participant main language. The participant answers questions about the extended NCL syntax if he/she belongs to the NCL group. Conversely, he/she answers questions about the extended HTML syntax if he/she belongs to the HTML group. The evaluation form pages are listed in what follows. For a complete detail of the form and its questions, we refer the reader to Appendix C, which presents screenshots of all pages.

– Page 1 for all participants introduces the evaluation form and ask for the

participants' consent to participate in the study;

- Page 2 for all participants introduces multimedia languages with multimodal and multiuser features. In particular, it shows Figure 1.4 and Figure 1.5 to distinguish languages with and without such features;
- Page 3 for all participant presents profile questions;
- Page 4 for all participants presents the block-based representation and its tasks;
- Page 5 for all participants presents TAM questions about the entities of the conceptual model;
- Page 6 for NCL participants presents the extended NCL syntax and its tasks. We also organize this page like Page 4 (same entities and tasks) but using our extended NCL syntax.
- Page 7 for NCL participants presents TAM questions about the extended NCL syntax;
- Page 6 for HTML participants presents the extended HTML syntax and its tasks. We also organize this page like the Page 4 (same entities and tasks) but using our extended HTML syntax;
- Page 7 for HTML participants presents TAM questions about the extended HTML syntax.

In the next subsections, we detail and discuss the evaluation results. First, we present the participants' profiles (subsection 5.2.1). Then, we discuss their tasks and TAM answers related to the block-based (subsection 5.2.2) and to the syntax-based representation (subsection 5.2.3).

5.2.1 Participants' profiles

The results of the profile questions enable us to characterize that most of the participants are developers with postgraduate degrees and skilled in their group language (NCL or HTML). More precisely, Figure 5.4 16 shows that most participants have a background in Computer Science, whereas as Figure 5.5 shows that they mainly consider themselves as skilled (moderate to expert answers) in their group language. To understand their skill, we also asked how many applications they developed in their language group and most of them had developed more than eight applications (illustrated in Figure 5.6).

Regarding their multimodal development skill, few participants had developed multimodal applications at the time of the study (Figure 5.7). Among those that had developed, we ask which kind application and most of them said that had created some application for the Microsoft Kinect sensor.

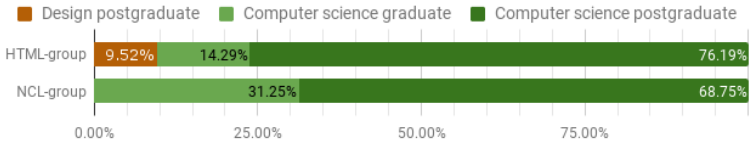


Figure 5.4: Participants' answers about their educational background.

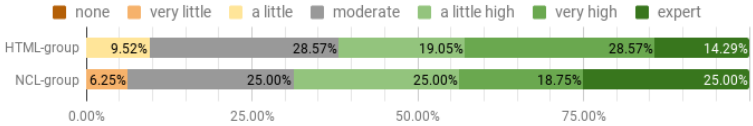


Figure 5.5: Participants' answers about their skill in their group language.

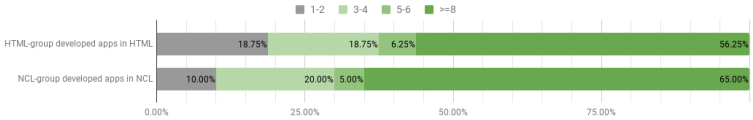


Figure 5.6: Participants' answers about the number of development applications in their main language.

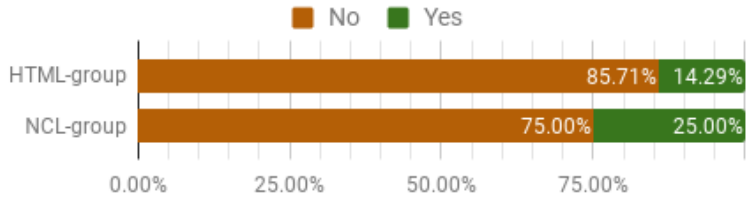


Figure 5.7: Participants' answers about whether they had developed multimodal applications.

5.2.2 Results about block-based representations

Page 4 of the evaluation form presents the block-based representation and is organized in four sections; each one presents a few concepts, followed by a task.

- Concepts 1.1: Presents the *Media* and *Relationship* blocks and a usage example. The example is an application that presents a video, shows an image when the video reaches its credits, and the video may be repeated if a user selects the image.
- Task 1.1: Asks the participant to describe a hypervideo application presented in block representation, using *Media* and *Relationship* elements. The application presents a video, shows two images when the video reaches its credits, and navigates to a video if user select the one of the images.
- Concepts 1.2: Presents the *Recognizer* block and a usage example. The example is a new version of Concepts 1.1, but the video will be repeated if a user uses a voice command.

– Task 1.2: Asks the participant to describe a new version of the hypervideo application from Task 1.1, which uses a *Recognizer* block to enable video navigation by a voice command.
– Concepts 1.3: Presents the *CompoundCondition* block and a usage example. The example is a new version of Concepts 1.2, but the video will be repeated if a user uses a voice or a gesture command.
– Task 1.3: Asks the participant to develop a new version of the hypervideo application from Task 1.2, which uses a *CompoundCondition* block to enable video navigation by a voice or a gesture command.
– Concepts 1.4: Presents the *UserClass* block and a usage example. The example is a new version of Concepts 1.2, but the video will be repeated only if a specific user gives a voice command.
– Task 1.4: Asks the participant to develop a new version of the hypervideo application from Task 1.3, which uses *UserClass* block to enable to video navigation by voice command from a specific user.

Regarding the aforementioned tasks, participants in both groups made few mistakes in both description (Figure 5.8) and creation tasks (Figure 5.9).

The answers for description tasks were classified as: “correct description”, when the participant provided a correct description with some level of detail (four or more sentences); “correct description with minor errors” when the participant provided a correct description with some level of detail, but missing some information, such as describing the end (stop) of an image or a recognizer; and “generic description”, when the participant provided a correct description but in general form (one or two sentences), which prevents us from capturing any error.

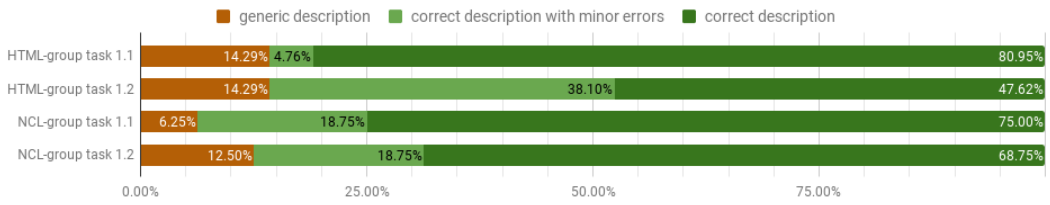


Figure 5.8: Participants’ answers in the blocks-based tasks 1.1 and 1.2.

The answers for the creation tasks were classified as: “correct”, when the participant correctly made the required block changes; “correct block but with mirror errors”, when the participant made errors such as forgetting to stop or start some recognizer, or kept using the selection interaction; and “fail”, when the changes did not solve what was asked.

Page 5 asks the participants’ opinion on TAM statements, as follows. Figure 5.10 illustrates the participants’ answers.

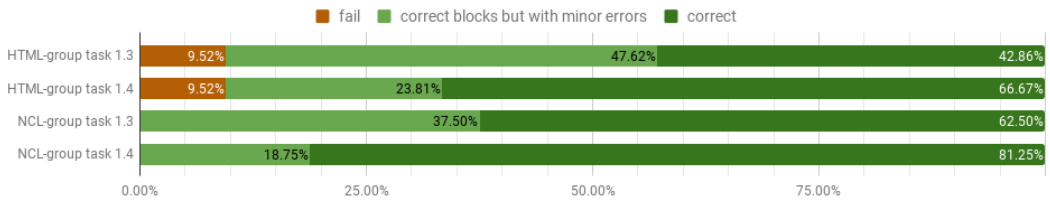


Figure 5.9: Participants' answers in the blocks-based tasks 1.3 and 1.4.

- PU 1: "The concepts presented allow to quickly specify multimodal applications."
- PU 2: "The concepts presented allow you to specify multimodal applications with quality."
- PU 3: "In general, the concepts presented are useful for specifying multimodal applications."
- PEU 1: "The concepts presented are simple and understandable."
- PEU 2: "The concepts presented are easy to learn."
- PEU 3: "In general, the concepts presented are easy to use."

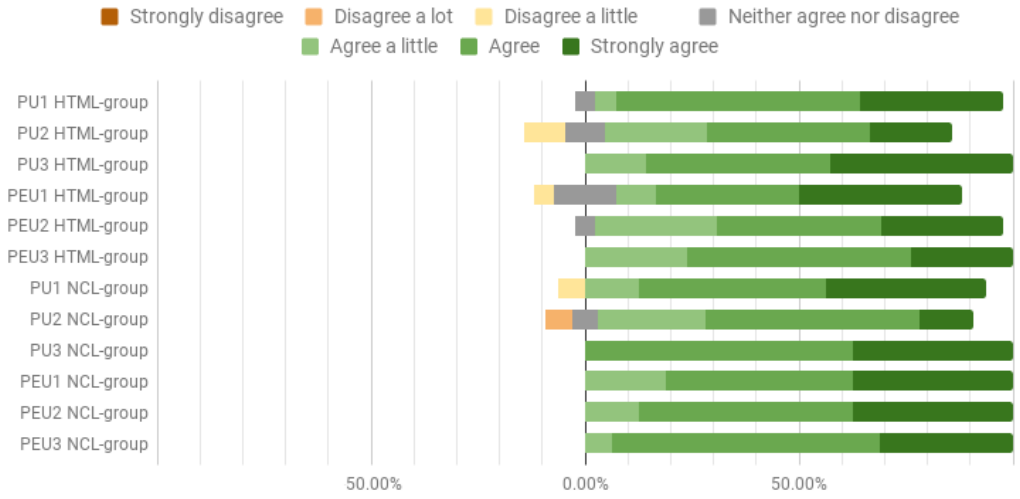


Figure 5.10: Participants' TAM answers about the block-based representation.

5.2.3 Results about extended language

Page 6 presents the syntax representations for the participant's group language. It is organized in four sections; each one presents concepts, followed by a task.

- Concepts 2.1: Presents *Media* and *Relationship* elements in the extended language syntax and a usage example. The usage example is the same as the one in Concepts 1.1.

- Task 2.1: Asks the participant to describe a hypervideo application presented in the extended language syntax, using *Media* and *Relationship* elements. The application is the same as the one in Task 1.1.
- Concepts 2.2: Presents *Recognizer* in the extended language syntax and usage example. The usage example is the same as the one in Concepts 1.2.
- Task 2.2: Asks the participant to describe a simple hypervideo application, now using a Recognizer in the extended language syntax. The application is the same as the one in Task 1.2.
- Concepts 2.3: Presents the *CompoundCondition* in the extended language syntax and a usage example. The usage example is the same as the one in Concepts 1.3.
- Tasks 2.3: Asks the participant to develop a new hypervideo application in the extended language syntax, now using a *Recognizer* and a *CompoundCondition*. The application is the same as the one in Task 1.3.
- Concepts 2.4: Presents *UserClass* in the extended language syntax and a usage example. The usage example is the same as the one in Concepts 1.4.
- Tasks 2.4: Asks the participant to develop a new hypervideo application, now using a *UserClass* in the extended language syntax. The application is the same as the one in Task 1.4.

Regarding the tasks, participants in both groups made few mistakes in both description (Figure 5.11) and creation tasks (Figure 5.12).

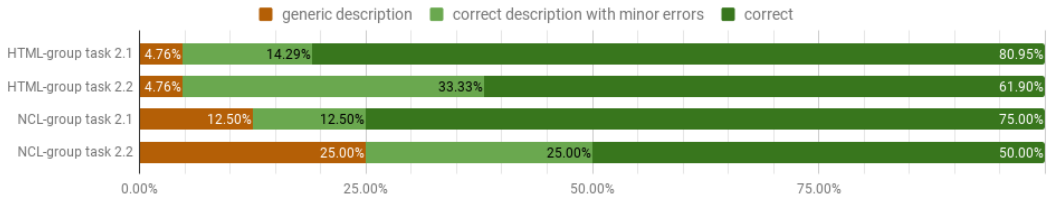


Figure 5.11: Participants' answers in the extended language tasks 2.1 and 2.2

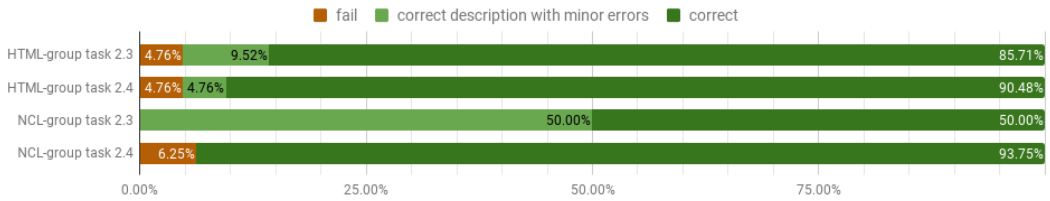


Figure 5.12: Participants' answers in the extended language tasks 2.3 and 2.4.

Page 7 asks the participants' opinion on TAM statements, as follows. Figure 5.13 illustrates the participants' answers.

– PU 1: “The extended language allows rapid development of multimodal applications.”

– PU 2: “The extended language allows the development of multimodal applications with quality.”

– PU 3: “In general, the extended language is useful for the development of multimodal applications.”

– PEU 1: “The extended language is simple and understandable.”

– PEU 2: “The extended language is easy to learn.”

– PEU 3: “Overall, the extended language is easy to use.”

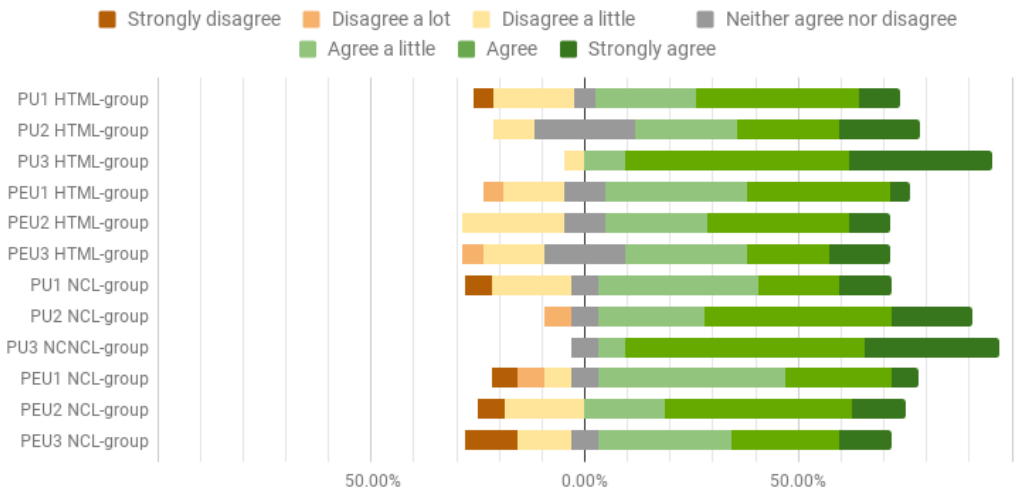


Figure 5.13: Participants’ TAM answers about the extended language.

Finally, we asked their opinion regarding the quality of our instantiation with the statement “The concepts presented in the previous section are clearly instantiated in the extended language”. Figure 5.14 illustrates the participants’ answers.

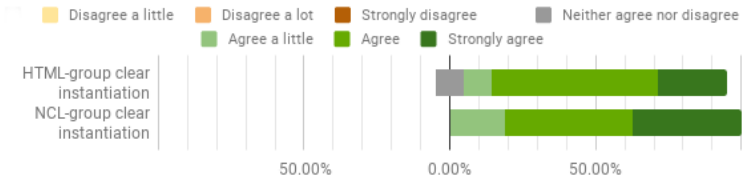


Figure 5.14: Participants’ answers about the concepts instantiation.

5.3 Discussion

First, we must highlight that we have not found a relation between the users who had not developed multimodal applications and the ones who performed the tasks with errors. Also, we have not found relation between users who self-reported to have lower knowledge of their group language and the ones who performed the extended language tasks with errors.

If participants had performed the tasks with more errors, it might indicate that they had not clearly understood the proposed entities and their TAM answer would not be very informative. However, in general, both NCL and HTML participants performed the tasks, on both block-based and extended language representations, with few errors, so their TAM answers can be considered valuable.

Regarding the task identification errors, the most common ones were related with missing the stop or start of some recognizer. This issue may be related with the fact that NCL and HTML currently support mouse/key interactions through some callback function (*e.g.* onclick and onSelection), which do not need to be activated or deactivated, unlike a recognizer.

Although most users said that the entities were clearly instantiated (Figure 5.14), their answers to the TAM questions showed a little difference between the block-based and extended languages representations. The participants gave some slightly more positive answers for the block-based ones. To understand why this was the case we need to conduct further studies.

6
Final Remarks

On the one hand, the studies performed by the multimedia research community has resulted in multimedia-oriented programming languages, such as HTML, SMIL, and NCL, which focus on the synchronization of audiovisual modalities (*e.g.* text, graphics, and videos) and GUI-based (keyboard and mouse) input modalities. On the other hand, the studies performed by the multimodal interaction research community have resulted in programming languages and frameworks that support the development of MUIs. In general, however, the languages and frameworks proposed by either community suffer some relevant drawbacks (discussed in Section 2.3). In this thesis, we propose to extend multimedia languages to support both multimodal and multiuser interactions. We believe that these extensions can contribute to the state of the art by overcoming the drawbacks and same time increasing multimodal specification expressiveness.

A multimedia language that follows our model should instantiate as first-class citizens our proposed entities, i.e. *Media*, *Recognizer*, *Relationship* and *UserClass*. By so doing, these enable their developers also handle both fusion and fission processes, thus, prevents *the strong encapsulation between fusion and fission*. Moreover, the multimedia language will also support modality selection based on the user's sensory capabilities and specification of interacting users, and the association of a user with recognition events. We discuss in Section 4 the instantiation of the model entities into the NCL and HTML languages though new elements their syntax.

Besides overcoming the above drawbacks, our proposal also achieves more expressiveness when combining both output and input modalities. It does so by using the causal relationship and anchor concepts from NCM. Those concepts enable NCM express Allen's temporal relations among output modalities. In our approach, we enable such concepts for input modalities which enable it express Allen's temporal relations among both output modalities and input modalities.

To evaluate our approach, we perform an evaluation study comprised 37 participants. We organize them into two groups based on their self-assessment of their knowledgeof NCL and HTML. We organize them as 21 participants

in the HTML group and 16 in the NCL group. This study was performed made it in two parts. The first part focuses on the conceptual entities, whereas the second part focuses on our proposed syntaxes for HTML and NCL. The first part about the conceptual entities was a challenge, because those entities were created to be independent from representation syntax. To do so, we used a block-based programing paradigm to enable users to develop applications using only the concept entities, at a level of abstraction higher than that of either NCL or HTML.

In both parts of the evaluation study, we presented the concepts entities and aiming at capture evidences of understanding and acceptance. To capture evidences of understanding, we ask developers answers coding tasks. To captures evidences of acceptance we use TAM based questionnaires. Both NCL and HTML participants, in general, performed the tasks with feel errors, which may indicate that they had reasonably understood, and they presents good acceptance in their answers to TAM questions.

6.1 Publications

Until the present moment, we archive the following publications. In particular, those publications discuss the development of envisaged scenarios in NCL, varying the modalities and interacting users.

- A. L. V. Guedes, R. G. de A. Azevedo, M. F. Moreno, and L. F. G. Soares, "Specification of Multimodal Interactions in NCL," in Proceedings of the 21st Brazilian Symposium on Multimedia and the Web, New York, NY, USA, 2015, pp. 181–187 [72].
- A. L. V. Guedes, "Towards Supporting Multimodal and Multiuser Interactions in Multimedia Languages," in In: Doctoral Consortium. Proceedings of the 2016 ACM Symposium on Document Engineering, New York, NY, USA, 2016 [73].
- A. L. V. Guedes, R. G. de A. Azevedo, and Simone Diniz Junqueira Barbosa, "Extending multimedia languages to support multimodal user interactions," Multimed Tools Appl, pp. 1–30, Oct. 2016 [74].
- A. L. V. Guedes, R. G. de Albuquerque Azevedo, S. Colcher, and S. D. J. Barbosa, "Extending NCL to Support Multiuser and Multimodal Interactions," in Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web, New York, NY, USA, 2016, pp. 39–46 [75].
- A. L. V. Guedes, Marcio Cunha, Hugo Fuks, Sérgio Colcher, and Simone Diniz Junqueira Barbosa, "Using NCL to Synchronize Media Objects,

Sensors and Actuators," in In: Workshop Internacional de Sincronismo das Coisas (WSoT), 1, 2016, Teresina. Anais do XXII Simpósio Brasileiro de Sistemas Multimídia e Web. Porto Alegre: Sociedade Brasileira de Computação, 2016. v. 2, New York, NY, USA, 2016 [76].

6.2 Future Works

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

First, we aim to investigate how to integrate higher-level constructions for dialog management into our proposal. For instance, this could be achieved through mappings from the form-based or state machine-based constructions onto our conditions and actions. In particular, the mapping of these higher-level constructs would generate stop/start of Recognizer elements, which participants often missed in our evaluation.

Second, we aim to implement a system that fulfill the execution requirements (discussed in Section 1.1) and develop more usage scenarios. In particular, since NCL is an international standard for iDTV, an NCL-based system may be used to exploit new kinds of applications in the iDTV domain. It allows to go beyond the limited interaction (*e.g.* remote control) and audiovisual media currently supported in this domain. Example of applications that can take advantages of our proposal include: (1) accessibility applications for disabled people or for the elderly; (2) educational applications for kids, like interactive classes of language or math; (3) immersive applications using different sensors and actuators.

Following the above paths, we also aim at investigating the development of multimedia applications with multimodal interaction features for mobile and ubiquitous environments. Most mobile devices have sensor technologies —such as accelerometer, compass, and geographic location— and actuators —such as vibration motors. These devices can be useful for many kinds of multimodal user interactions in multimedia applications.

Moreover, the development of graphical abstractions for multimedia authoring with multimodal interactions is also necessary. Graphical tools offer an alternative to editing XML code, which is often tedious and error prone; for instance, a graphical editor could help in expressing complex temporal relations between modalities. In particular, we can also improve our block-based representation. It can be only both a standalone tool that generate NCL/HTML code or be integrated some in authoring tool for one

those languages. For instance, NCL Composer [77] can use the block-based presentation as an authoring view.

Bibliography

[1] A. Jaimes and N. Sebe, "Multimodal human–computer interaction: A survey," *Computer Vision and Image Understanding*, vol. 108, no. 1, pp. 116–134, Oct. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314206002335

[2] M. Turk, "Multimodal interaction: A review," *Pattern Recognition Letters*, vol. 36, pp. 189–195, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865513002584

[3] S. Oviatt, "Multimodal Interfaces," in *The Human-Computer Interaction Handbook*, ser. Human Factors and Ergonomics. CRC Press, Sep. 2007, pp. 413–432. [Online]. Available: http://dx.doi.org/10.1201/9781410615862.ch21

[4] G. Ghinea, C. Timmerer, W. Lin, and S. R. Gulliver, "Mulsemedia: State of the Art, Perspectives, and Challenges," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 11, no. 1s, pp. 17:1–17:23, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2617994

[5] R. A. Bolt, "Put-That-There: Voice and Gesture at the Graphics Interface," in *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1980, pp. 262–270. [Online]. Available: http://doi.acm.org/10.1145/800250.807503

[6] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar, "WYSIWIS Revised: Early Experiences with Multiuser Interfaces," *ACM Trans. Inf. Syst.*, vol. 5, no. 2, pp. 147–167, Apr. 1987. [Online]. Available: http://doi.acm.org/10.1145/27636.28056

[7] C. Müller-Tomfelde and M. Fjeld, "Introduction: A Short History of Tabletop Research, Technologies, and Products," in *Tabletops - Horizontal Interactive Displays*, C. Müller-Tomfelde, Ed. Springer London, 2010, pp. 1–24. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-84996-113-4_1

[8] N. Elmqvist, "Distributed User Interfaces: State of the Art," in *Distributed User Interfaces*, J. A. Gallud, R. Tesoriero, and V. M. R. Penichet, Eds. Springer London, 2011, pp. 1–12. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4471-2271-5_1

[9] P. Dietz and D. Leigh, "DiamondTouch: a multi-user touch technology," in *Proceedings of the 14th annual ACM symposium on User interface software and technology*. ACM, 2001, pp. 219–226. [Online]. Available: http://dl.acm.org/citation.cfm?id=502389

[10] C. Haber, "Modeling Multiuser Interactions," in *Proceedings at the First European Computer Supported Collaborative Learning Conference, Maastricht, Germany*, 2001, pp. 22–24. [Online]. Available: http://eculturenet.org/mmi/euro-cscl/Papers/63.pdf

[11] Y. Laurillau, "IOWAState: implementation models and design patterns for identity-aware user interfaces based on state machines," in *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 2013, pp. 59–68. [Online]. Available: http://dl.acm.org/citation.cfm?id=2480299

[12] B. Dumas, D. Lalanne, and R. Ingold, "Description languages for multimodal interaction: a set of guidelines and its illustration with SMUIML," *Journal on Multimodal User Interfaces*, vol. 3, no. 3, pp. 237–247, Apr. 2010. [Online]. Available: http://link.springer.com/article/10.1007/s12193-010-0043-3

[13] K. Katsurada, H. Yamada, Y. Nakamura, S. Kobayashi, and T. Nitta, "XISL: A Modality-Independent MMI Description Language," in *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, W. Minker, D. Bühler, and L. Dybkjær, Eds. Springer Netherlands, Jan. 2005, pp. 133–148. [Online]. Available: http://link.springer.com/chapter/10.1007/1-4020-3075-4_8

[14] "Multimodal Interaction Framework," 2003. [Online]. Available: www.w3.org/TR/mmi-framework/

[15] L. A. Rowe, "Looking Forward 10 Years to Multimedia Successes," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 9, no. 1, pp. 37:1–37:7, 2013. [Online]. Available: http://doi.acm.org/10.1145/2490825

[16] "HTML 5," 2014. [Online]. Available: https://www.w3.org/TR/html5/

[17] ABNT, "ABNT NBR 15606-2:2016 Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital Parte 2: Ginga-NCL para receptores fixos e móveis - Linguagem de aplicação XML para codificação de aplicações," 2016. [Online]. Available: http://www.abntcatalogo.com.br/norma.aspx?ID=351837

[18] D. C. Bulterman and L. W. Rutledge, *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*, 2nd ed. Springer Publishing Company, Incorporated, 00084.

[19] J. L. Beckham, G. D. Fabbrizio, and N. Klarlund, "Towards SMIL as a foundation for multimodal, multimedia applications," in *EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology*, P. Dalsgaard, B. Lindberg, H. Benner, and Z.-H. Tan, Eds. ISCA, 2001, pp. 1363–1366. [Online]. Available: http://www.isca-speech.org/archive/eurospeech_2001/e01_1363.html

[20] L. A. M. C. Carvalho, A. P. Guimarães, and H. T. Macêdo, "Architectures for Interactive Vocal Environment to Brazilian Digital TV Middleware," in *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*. New York, NY, USA: ACM, 2008, pp. 22:1–22:8. [Online]. Available: http://doi.acm.org/10.1145/1621087.1621109

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

[22] "XHTML+Voice Profile 1.0," 2001. [Online]. Available: http://www.w3.org/TR/xhtml+voice/

[23] K. Wang, "SALT: A Spoken Language Interface for Web-based Multimodal Dialog Systems," in *Proc. Int. Conf. on Spoken Language Processing*, 2002. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=77497

[24] B. Dumas, D. Lalanne, and S. Oviatt, "Multimodal Interfaces: A Survey of Principles, Models and Frameworks," in *Human Machine Interaction*, D. Lalanne and J. Kohlas, Eds. Springer Berlin Heidelberg, 2009, pp. 3–26. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-00437-7_1

[25] "Speech Recognition Grammar Specification Version 1.0," 2004. [Online]. Available: http://www.w3.org/TR/speech-grammar/

[26] "Ink Markup Language (InkML)," 2011. [Online]. Available: http://www.w3.org/TR/2011/REC-InkML-20110920/

[27] T. Hachaj and M. R. Ogiela, "Semantic Description and Recognition of Human Body Poses and Movement Sequences with Gesture Description Language," in *Computer Applications for Bio-technology, Multimedia, and Ubiquitous City*, ser. Communications in Computer and Information Science, T.-h. Kim, J.-J. Kang, W. I. Grosky, T. Arslan, and N. Pissinou, Eds. Springer Berlin Heidelberg, 2012, no. 353, pp. 1–8. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-35521-9_1

[28] Ideum Inc, "Gesture Markup Language," 2016. [Online]. Available: http://www.gestureml.org/

[29] "Speech Synthesis Markup Language (SSML) Version 1.1," 2010. [Online]. Available: http://www.w3.org/TR/speech-synthesis11/

[30] H. Vilhjálmsson, N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. N. Marshall, C. Pelachaud, Z. Ruttkay, K. R. Thórisson, H. v. Welbergen, and R. J. v. d. Werf, "The Behavior Markup Language: Recent Developments and Challenges," in *Intelligent Virtual Agents*, C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis, and D. Pelé, Eds. Springer Berlin Heidelberg, 2007, pp. 99–111. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-74997-4_10

[31] ISO/IEC, "ISO/IEC 23005-3:2013 Information Technology - Media Context and Control - Part 3: Sensory Information," 2013. [Online]. Available: www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=60391

[32] "Voice Extensible Markup Language (VoiceXML) 2.1," 2007. [Online]. Available: http://www.w3.org/TR/voicexml21/

[33] Microsoft, "Speech Application Language Tags (SALT)," 2003. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms994629.aspx

[34] ISO/IEC, "ISO/IEC 23005-1:2014 - Information technology – Media context and control – Part 1: Architecture," 2014. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=60359

[35] "Multimodal Architecture and Interfaces," 2012. [Online]. Available: http://www.w3.org/TR/mmi-arch/

[36] "State Chart XML (SCXML): State Machine Notation for Control Abstraction," 2012. [Online]. Available: http://www.w3.org/TR/scxml/

[37] "EMMA: Extensible MultiModal Annotation markup language," 2009. [Online]. Available: http://www.w3.org/TR/2009/REC-emma-20090210/

[38] D. A. Dahl, "Standards for Multimodal Interaction," in *Multimodal Human Computer Interaction and Pervasive Services*, 2009, p. 409. [Online]. Available: http://www.igi-global.com/book/multimodal-human-computer-interaction-pervasive/

[39] B. Dumas, D. Lalanne, and R. Ingold, "Prototyping multimodal interfaces with smuiml modeling language," in *CHI 2008 Workshop on User Interface Description Languages for Next Generation User Interfaces, CHI*, 2008, pp. 63–66.

[40] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The Care Properties," in *Human—Computer Interaction*, ser. IFIP Advances in Information and Communication Technology. Springer, Boston, MA, 1995, pp. 115–120. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-5041-2896-4_19

[41] B. Dumas, "Frameworks, description languages and fusion engines for multimodal interactive systems," Ph.D. dissertation, Faculty of Science, University of Fribourg (Switzerland, 2010. [Online]. Available: https://doc.rero.ch/record/21372/files/DumasB.pdf

[42] "XHTML 1.0: The Extensible HyperText Markup Language," 2000. [Online]. Available: https://www.w3.org/TR/2000/REC-xhtml1-20000126/

[43] L. F. G. Soares, "Nested Context Model 3.0: Part 1 – NCM Core," *Monographs in Computer Science PUC-Rio Inf MCC18/05*, 2009. [Online]. Available: ftp://obaluae.inf.puc-rio.br/pub/docs/techreports/05_18_soares.pdf

[44] M. F. Moreno, R. Brandao, and R. Cerqueira, "Extending Hypermedia Conceptual Models to Support Hyperknowledge Specifications," *International Journal of Semantic Computing*, vol. 11, no. 01, pp. 43–64, Mar. 2017. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S1793351X17400037

[45] "RDF/XML Syntax Specification," 2014. [Online]. Available: https://www.w3.org/TR/REC-rdf-syntax/

[46] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983. [Online]. Available: http://doi.acm.org/10.1145/182.358434

[47] C.-M. Huang and C. Wang, "Synchronization for interactive multimedia presentations," *IEEE MultiMedia*, vol. 5, no. 4, pp. 44–62, Oct. 1998.

[48] Microsoft, "Getting Started With XInput." [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ee417001#multiple_controllers

[49] Google, "Supporting Multiple Game Controllers | Android Developers." [Online]. Available: https://developer.android.com/intl/pt-br/training/game-controllers/multiple-controllers.html

[50] J. Guerrero Garcia, J. Vanderdonckt, and others, "Designing workflow user interfaces with UsiXML," in *1st Int. Workshop on User Interface eXtensible Markup Language UsiXML'2010*, 2010. [Online]. Available: http://dial.uclouvain.be/downloader/downloader.php?pid=boreal:118234&datastream=PDF_01

[51] C. E. C. F. Batista, L. F. G. Soares, and G. L. de Souza Filho, "Estendendo o uso das classes de dispositivos Ginga-NCL," in *WebMedia '10: Proceedings of the 16th Brazilian Symposium on Multimedia and the Web*. XVI Brazilian Symposium on Multimedia and the web, WebMedia '10, 2010. [Online]. Available: http://www.lbd.dcc.ufmg.br/colecoes/webmedia/2010/04_webmi_c.pdf

[52] C. E. C. F. Batista, "GINGA-MD: Uma Plataforma para Suporte à Execução de Aplicações Hipermídia Multi-Dispositivo Baseada em NCL," Ph.D. dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 2013. [Online]. Available: https://doi.org/10.17771/PUCRio.acad.21956

[53] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero, "USIXML: A language supporting multi-path development of user interfaces," in *Proceedings of the 2004 International Conference on Engineering Human Computer Interaction and Interactive Systems*, ser. EHCI-DSVIS'04. Springer-Verlag, pp. 200–220. [Online]. Available: http://dx.doi.org/10.1007/11431879_12

[54] L. F. G. Soares, R. M. Costa, M. F. Moreno, and M. F. Moreno, "Multiple Exhibition Devices in DTV Systems," in *Proceedings of the 17th ACM International Conference on Multimedia*. New York, NY, USA: ACM, 2009, pp. 281–290. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631312

[55] OpenMobileAlliance. WAG UAProf. [Online]. Available: http://www.openmobilealliance.org/Technical/wapindex.aspx

[56] D. Costa and C. Duarte, "Adapting Multimodal Fission to User's Abilities," in *Universal Access in Human-Computer Interaction. Design for All and eInclusion*, C. Stephanidis, Ed. Springer Berlin Heidelberg, Jan. 2011, pp. 347–356. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-21672-5_38

[57] D. Schnelle-Walka, S. Radomski, and M. Mühlhäuser, "JVoiceXML as a modality component in the W3c multimodal architecture," *Journal on Multimodal User Interfaces*, vol. 7, no. 3, pp. 183–194, Nov. 2013. [Online]. Available: http://link.springer.com/article/10.1007/s12193-013-0119-y

[58] D. C. A. Bulterman and L. Hardman, "Structured multimedia authoring," vol. 1, no. 1, pp. 89–109, 00153. [Online]. Available: http://doi.acm.org/10.1145/1047936.1047943

[59] L. Fernando and G. Soares, "O uso da linguagem declarativa do ginga-ncl na construção de conteúdos audiovisuais interativos: a experiência do "roteiros do dia"," 2009.

[60] B. Meixner and H. Kosch, "Interactive Non-linear Video: Definition and XML Structure," in *Proceedings of the 2012 ACM Symposium on Document Engineering*, ser. DocEng '12. New York, NY, USA: ACM, 2012, pp. 49–58. [Online]. Available: http://doi.acm.org/10.1145/2361354.2361367

[61] "SPARQL Query Language for RDF," 2008. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[62] D. Brickley and L. Miller, "FOAF Vocabulary Specification," 2014. [Online]. Available: http://xmlns.com/foaf/spec/

[63] "DOM4," 2015. [Online]. Available: https://www.w3.org/TR/dom/

[64] Mozilla. DOM API: Node. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Node

[65] C. S. S. Neto, H. F. Pinto, and L. F. G. Soares, "TAL processor for hypermedia applications," in *Proceedings of the 2012 ACM Symposium on Document Engineering*. ACM, pp. 69–78. [Online]. Available: http://doi.acm.org/10.1145/2361354.2361369

[66] E. C. O. Silva, J. A. F. d. Santos, and D. C. Muchaluat-Saade, "JNS: An alternative authoring language for specifying NCL multimedia documents," in *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, Jul. 2013, pp. 1–6.

[67] D. d. S. Moraes, A. L. d. B. Damasceno, A. J. G. Busson, and C. d. S. Soares Neto, "Lua2ncl: Framework for Textual Authoring of NCL Applications using Lua," in *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*. ACM, 2016, pp. 47–54. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976851

[68] C. de Salles Soares Neto, C. S. de Souza, and L. F. G. Soares, "Linguagens Computacionais Como Interfaces: Um Estudo Com Nested Context Language," in *Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems*, ser. IHC '08. Porto Alegre, Brazil, Brazil: Sociedade Brasileira de Computação, 2008, pp. 166–175. [Online]. Available: http://dl.acm.org/citation.cfm?id=1497470.1497489

[69] R. B. Shapiro and M. Ahrens, "Beyond Blocks: Syntax and Semantics," *Commun. ACM*, vol. 59, no. 5, pp. 39–41, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2903751

[70] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989. [Online]. Available: http://www.jstor.org/stable/249008

[71] D. Gefen and M. Keil, "The Impact of Developer Responsiveness on Perceptions of Usefulness and Ease of Use: An Extension of the Technology Acceptance Model," *SIGMIS Database*, vol. 29, no. 2, pp. 35–49, Apr. 1998. [Online]. Available: http://doi.acm.org/10.1145/298752.298757

[72] A. L. V. Guedes, R. G. d. A. Azevedo, M. F. Moreno, and L. F. G. Soares, "Specification of Multimodal Interactions in NCL," in *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*, ser. WebMedia '15. New York, NY, USA: ACM, 2015, pp. 181–187. [Online]. Available: http://doi.acm.org/10.1145/2820426.2820436

[73] A. L. V. Guedes, "Towards Supporting Multimodal and Multiuser Interactions in Multimedia Languages," in *In: Doctoral Consortium. Proceedings of the 2016 ACM Symposium on Document Engineering*, ser. DocEng '16. New York, NY, USA: ACM, 2016.

[74] A. L. V. Guedes, R. G. d. A. Azevedo, and Simone Diniz Junqueira Barbosa, "Extending multimedia languages to support multimodal user interactions," *Multimedia Tools and Applications*, pp. 1–30, Oct. 2016. [Online]. Available: http://link.springer.com/article/10.1007/s11042-016-3846-8

[75] A. L. V. Guedes, R. G. de Albuquerque Azevedo, S. Colcher, and S. D. Barbosa, "Extending NCL to Support Multiuser and Multimodal Interactions," in *Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web*, ser. Webmedia '16. New York, NY, USA: ACM, 2016, pp. 39–46. [Online]. Available: http://doi.acm.org/10.1145/2976796.2976869

[76] A. L. V. Guedes, Marcio Cunha, Hugo Fuks, Sérgio Colcher, and Simone Diniz Junqueira Barbosa, "Using NCL to Synchronize Media Objects, Sensors and Actuators," in *In: Workshop Internacional de Sincronismo das Coisas (WSoT), 1, 2016, Teresina. Anais do XXII Simpósio Brasileiro de Sistemas Multimídia e Web. Porto Alegre: Sociedade Brasileira de Computação, 2016. v. 2*, ser. WebMedia '16. New York, NY, USA: ACM, 2016. [Online]. Available: http://www.lbd.dcc.ufmg.br/colecoes/wsotwebmedia/2016/003.pdf

[77] R. G. A. Azevedo, E. C. Araújo, B. Lima, L. F. G. Soares, and M. F. Moreno, "Composer: meeting non-functional aspects of hypermedia authoring environment," *Multimedia*

Tools and Applications, vol. 70, no. 2, pp. 1199–1228, May 2014. [Online]. Available: http://link.springer.com/article/10.1007/s11042-012-1216-8

A
NCL Schemas

In this appendix, we present five XML schemas of our proposed NCL 3.0 extensions. The first one, illustrated in Listing A.1, is a new schema called NCL30Input.xsd, which includes our <input> element. The second one, illustrated in Listing A.2, is a new schema called NCL30UserClass.xsd, which includes our <userClass> element. The renaming schemas are modified versions of existing ones from the NCL 3.0 (changes are highlighted in light green), to include our extensions to link-related elements. The third schema, illustrated in Listing A.3, is a modified version of the NCL30ConnectorCommonPart.xsd. The fourth schema, illustrated in Listing A.4, is an extended version of the NCL30ConnectorCausalExpression.xsd. Finally, the fifth schema, illustrated in Listing A.5, is a modified version of the NCL30Linking.xsd.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:input="http://www.ncl.org.br/NCL3.0/Input"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Input"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <complexType name="inputPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="src" type="anyURI" use="optional"/>
  </complexType>
  <element name="input" type="input:inputPrototype"/>
</schema>
```

Listing A.1: New NCL30Input.xsd.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:userclass="http://www.ncl.org.br/NCL3.0/UserClass"
  targetNamespace="http://www.ncl.org.br/NCL3.0/UserClass "
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <complexType name="userClassPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="min" type="positiveInteger" use="required"/>
    <attribute name="max" type="positiveInteger" use="required"/>
    <attribute name="src" type="anyURI" use="required"/>
  </complexType>
  <element name="userClass" type="userClass:userClassPrototype"/>
</schema>
```

Listing A.2: New NCL30UserClass.xsd.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <complexType name="parameterPrototype">
    <attribute name="name" type="string" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="component" type="IDREF" use="optional"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>
  <simpleType name="eventPrototype">
    <restriction base="string">
      <enumeration value="presentation" />
      <enumeration value="selection" />
      <enumeration value="attribution" />
      <enumeration value="composition" />
      <enumeration value="recognition" />
    </restriction>
  </simpleType>
  <simpleType name="logicalOperatorPrototype">
    <restriction base="string">
      <enumeration value="and" />
      <enumeration value="or" />
      <enumeration value="seq" />
    </restriction>
  </simpleType>
  <simpleType name="transitionPrototype">
    <restriction base="string">
      <enumeration value="starts" />
      <enumeration value="stops" />
      <enumeration value="pauses" />
      <enumeration value="resumes" />
      <enumeration value="aborts" />
    </restriction>
  </simpleType>
</schema>
```

Listing A.3: Extended NCL30ConnectorCommonPart.xsd.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd"/>
  <simpleType name="conditionRoleUnion">
    <union memberTypes="string
    connectorCausalExpression:conditionRolePrototype"/>
  </simpleType>
  <simpleType name="conditionRolePrototype">
    <restriction base="string">
      <enumeration value="onBegin" />
      <enumeration value="onEnd" />
      <enumeration value="onPause" />
      <enumeration value="onResume" />
      <enumeration value="onAbort" />
      <enumeration value="onRecognize" />
    </restriction>
  </simpleType>
  <simpleType name="maxUnion">
    <union memberTypes="positiveInteger
    connectorCausalExpression:unboundedString"/>
  </simpleType>
  <simpleType name="unboundedString">
    <restriction base="string">
      <pattern value="unbounded"/>
    </restriction>
  </simpleType>
```

```
<complexType name="simpleConditionPrototype">
  <attribute name="role" type="connectorCausalExpression:conditionRoleUnion"
    use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype"
    use="optional"/>
  <attribute name="user_id" type="string" use="optional"/>
  <attribute name="excluded_user_id" type="string" use="optional"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="transition" type="connectorCommonPart:transitionPrototype"
    use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion"
    use="optional"/>
  <attribute name="qualifier"
    type="connectorCommonPart:logicalOperatorPrototype" use="optional"/>
</complexType>
<complexType name="compoundConditionPrototype">
  <attribute name="operator"
    type="connectorCommonPart:logicalOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>
<simpleType name="actionRoleUnion">
  <union memberTypes="string connectorCausalExpression:actionNamePrototype"/>
</simpleType>
<simpleType name="actionNamePrototype">
  <restriction base="string">
    <enumeration value="start" />
    <enumeration value="stop" />
    <enumeration value="pause" />
    <enumeration value="resume" />
    <enumeration value="abort" />
    <enumeration value="set" />
  </restriction>
</simpleType>
<simpleType name="actionOperatorPrototype">
  <restriction base="string">
    <enumeration value="par" />
    <enumeration value="seq" />
  </restriction>
</simpleType>
<complexType name="simpleActionPrototype">
  <attribute name="role" type="connectorCausalExpression:actionRoleUnion"
    use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype"
    use="optional"/>
  <attribute name="actionType"
    type="connectorCausalExpression:actionNamePrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="value" type="string" use="optional"/>
  <attribute name="repeat" type="positiveInteger" use="optional"/>
  <attribute name="repeatDelay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion"
    use="optional"/>
  <attribute name="qualifier"
    type="connectorCausalExpression:actionOperatorPrototype" use="optional"/>
</complexType>
```

```
<complexType name="compoundActionPrototype">
  <choice minOccurs="2" maxOccurs="unbounded">
    <element ref="connectorCausalExpression:simpleAction" />
    <element ref="connectorCausalExpression:compoundAction" />
  </choice>
  <attribute name="operator"
    type="connectorCausalExpression:actionOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>
<element name="simpleCondition"
  type="connectorCausalExpression:simpleConditionPrototype" />
<element name="compoundCondition"
  type="connectorCausalExpression:compoundConditionPrototype" />
<element name="simpleAction"
  type="connectorCausalExpression:simpleActionPrototype" />
<element name="compoundAction"
  type="connectorCausalExpression:compoundActionPrototype" />
</schema>
```

Listing A.4: Extended NCL30ConnectorCausalExpression.xsd.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
targetNamespace="http://www.ncl.org.br/NCL3.0/Linking"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<complexType name="paramPrototype">
  <attribute name="name" type="string" use="required"/>
  <attribute name="value" type="anySimpleType" use="required"/>
  <attribute name="component" type="IDREF" use="required"/>
  <attribute name="interface" type="string" use="optional"/>
</complexType>
<complexType name="bindPrototype">
  <sequence minOccurs="0" maxOccurs="unbounded">
    <element ref="linking:bindParam"/>
  </sequence>
  <attribute name="role" type="string" use="required"/>
  <attribute name="component" type="IDREF" use="required"/>
  <attribute name="interface" type="string" use="optional"/>
</complexType>
<complexType name="linkPrototype">
  <sequence>
    <element ref="linking:linkParam" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="linking:bind" minOccurs="2" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="id" type="ID" use="optional"/>
  <attribute name="xconnector" type="string" use="required"/>
</complexType>
<element name="linkParam" type="linking:paramPrototype"/>
<element name="bindParam" type="linking:paramPrototype"/>
<element name="bind" type="linking:bindPrototype" />
<element name="link" type="linking:linkPrototype" />
</schema>
```

Listing A.5: Extended NCL30Linking.xsd.

B
NCL Envisaged Scenarios

In this appendix, we present three of the envisaged scenarios using our extended NCL syntax, namely: “Put-that-there”, “I-Get-That-You-Put-It-There” and “Anyone-Get-That-Someone-Else-Put-It-There”. They enable users to move an image (the Ginga logo) using a combination of voice and gesture commands. To enable voice interactions, the scenarios use the same description files, namely “sentences.ssml” (Listing B.1) and “commands.srgs” (Listing B.2).

```
1<speak xmlns="http://www.w3.org/2001/10/synthesis">
2  <s id="repeat_question">where?</s>
3</speak>
```

Listing B.1: sentences.ssml.

```
1<grammar xmlns="http://www.w3.org/2001/06/grammar">
2  <rule id="put_that">put that</rule>
3  <rule id="there">there</rule>
4</grammar>
```

Listing B.2: commands.sgrs.

Listing B.3 presents the NCL code of <media>, <input> and <ports> elements shared among the scenarios. Those scenarios only differ by the used <link> elements (discussed next). The <media> elements (lines 4-9) are “logo”, which define the movable image, and “sentences”, which define the speech feedback using “sentences.ssml”. An <area> element is defined in the “sentences” pointing to the SSML fragment responsible for defining the word “where”. The <input> elements (lines 10-19) are “pointer” input, responsible for recognizing the point on the screen at which the user’s finger is pointing, and “asr” input, to enable voice commands using the “commands.sgrs” file. Two anchors are defined in “asr”, pointing to SRGS rules with the ids “put that” and “there”. Both scenarios begin by starting the “logo”, “pointer”, and “asr” elements, as defined in the <port> elements (lines 1-3). In particular, the “asr” media is started by the “put_that” interface, which defines the word expected in the first interaction.

```
<port component="icon"/>
<port component="pointer"/>
<port component="asr" interface="put_that"/>
  <media id="icon" src="ginga.png">
  <property name="location" value="20%, 20%"/>
</media>
<media id="sentences" src="sentences.ssml">
  <area label="where"/>
</media>
<input id="pointer" type="application/x-ncl-pointer">
  <area label="move"/>
  <property name="x"/>
  <property name="y"/>
</input>
<input id="asr" type="application/srgs+xml" src="commands.sgrs">
  <property name="userClass" name="BoltLikeUser"/>
  <area label="put_that"/>
  <area label="there"/>
</input>
```

Listing B.3: Used <media>, <input> and <port> elements in the three scenarios.

The first scenario, “Put-that-there”, requires two <link> elements, which use the new “recognition” event, both illustrated in . The first <link> defines that when the application recognizes a “put that” voice command, followed by the selection of the “logo” media, it will synthesize the word “where” and expect a “there” voice command. The second <link> defines that when the application recognize a “there” voice command, followed by a move of the pointer, the “logo” must be moved to the new position at which the user’s finger is pointing.

```
<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <connectorBase>
      <causalConnector id="onRecognizeSEQonSelectionStart">
        <compoundCondition operator="seq">
          <simpleCondition role="onSelection"/>
          <simpleCondition role="onRecognize"/>
        </compoundCondition>
        <simpleAction role="start"/>
      </causalConnector>
      <causalConnector id="onRecognizeSet">
        <connectorParam name="varSet"/>
        <simpleCondition role="onRecognize" qualifier="seq"/>
        <simpleAction role="set" value="$varSet"/>
      </causalConnector>
    </connectorBase>
  </head>
  <body>
    ... <!--### code from Listing 16 ###-->
    <link xconnector="onRecognizeSEQonSelectionStart">
      <bind role="onRecognize" component="asr" interface="put_that"/>
```

```
22      <bind role="onSelection" component="icon"/>
23      <bind role="start" component="sentences" interface="where"/>
24      <bind role="start" component="asr" interface="there"/>
25    </link>
26    <link xconnector="onRecognizeSet">
27      <bind role="onRecognize" component="asr" interface="there"/>
28      <bind role="onRecognize" component="pointer" interface="move"/>
29      <bind role="getValueX" component="pointer" interface="x"/>
30      <bind role="getValueY" component="pointer" interface="y"/>
31      <bind role="set" component="icon" interface="left">
32        <bindParam name="varSet" value="$getValueX"/>
33      </bind>
34      <bind role="set" component="icon" interface="top">
35        <bindParam name="varSet" value="$getValueY"/>
36      </bind>
37    </link>
38  </body>
39</ncl>
```

Listing B.4: Code fragment of "Put-That-There" in NCL.

The other two scenarios consider interactions of two distinct users. One user selects the icon and the other one moves it. Both scenarios require that users have a pointer and a microphone device. Such requirement is defined by the "boltlikeuser.sparql" file, illustrated in .

```
1PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2PREFIX prf: <http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430>
3SELECT ?person
4WHERE {
5 ?person foaf:mbox ?email FILTER regex(?email, "@inf.puc-rio.br$") .
6 ?person prf:component ?component1 . ?component1 prf:name ?name1 .
7 FILTER regex(?name1, "leapmotion") .
8 ?person prf:component ?component2 .  ?component2 prf:name ?name2 .
9 FILTER regex(?name2, "microphone")
10}
```

Listing B.5: boltlikeuser.sparql.

The "I-Get-That-You-Put-It-There" scenario considers interactions from two identified users. shows the code fragment of this scenario, which uses a <userClass> element (lines 20-23) and modified versions of the <link> elements from the previous scenario. The difference from previous <link> elements use is the "user_id" in <linkParam> to specify each interacting user. In the <link> elements' connectors, those "user_id" attributes are used as parameter for <simpleCondition>.

```
1  <ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
2  <head>
3    <connectorBase>
4      <causalConnector id="onRecognizeSEQonSelectionByUserStart">
5        <connectorParam name="user_id"/>
6        <compoundCondition operator="seq">
```

```
7           <simpleCondition role="onRecognize" user_id="$user_id"/>
8           <simpleCondition role="onSelection" user_id="$user_id"/>
9         </compoundCondition>
10        <simpleAction role="start"/>
11      </causalConnector>
12      <causalConnector id="onRecognizeByUserSet">
13        <connectorParam name="user_id"/>
14        <connectorParam name="varSet"/>
15        <simpleCondition role="onRecognize" qualifier="seq"
16          user_id="$user_id" />
17        <simpleAction role="set" value="$varSet"/>
18      </causalConnector>
19    </connectorBase>
20    <userBase>
21      <userClass id="BoltLikeUser" min="2" max="2"
22        userClassDescription="boltlikeuser.sparql"/>
23    </userBase>
24  </head>
25  <body>
26    ... <!--### code from Listing 16 ###-->
27    <link xconnector="onRecognizeSEQonSelectionByUserStart">
28      <linkParam name="user_id" value="BoltLikeUser(1)"/>
29      <bind role="onRecognize" component="asr" interface="put_that"/>
30      <bind role="onSelection" component="icon"/>
31      <bind role="start" component="sentences" interface="where"/>
32      <bind role="start" component="asr" interface="there"/>
33    </link>
34    <link xconnector="onRecognizeSEQByUserSet">
35      <linkParam name="user_id" value="BoltLikeUser(2)"/>
36      <bind role="onRecognize" component="asr" interface="there"/>
37      <bind role="onRecognize" component="pointer" interface="move"/>
38      <bind role="getValueX" component="pointer" interface="x"/>
39      <bind role="getValueY" component="pointer" interface="y"/>
40      <bind role="set" component="icon" interface="left">
41        <bindParam name="varSet" value="$getValueX"/>
42      </bind>
43      <bind role="set" component="icon" interface="top">
44        <bindParam name="varSet" value="$getValueY"/>
45      </bind>
46    </link>
47  </body>
48</ncl>
```

Listing B.6: Code fragment of “I-Get-That-You-Put-It-There”.

Finally, the “Anyone-Get-That-Someone-Else-Put-It-There” scenario also considers interactions of any two users, but not specified ones. More precisely, it first expects the interaction of any user and, in sequence, the interaction of another one, who must be different from the first one. To define this behavior, we split the first <link> of previous scenarios in two links. shows the code fragment of the three <link> elements, which are described in what follows.

The first <link> (lines 32-37) says that when any user perform the voice command “put_that”, we then store its “user_id”. To get the “user_id”, this

<link> uses our extended <linkParam> (lines 34-35) and <connectorParam> (lines 6-7). The <connectorParam> access the "user_id" from a "recognition" event of the given interface passed in the <linkParam>. Then, the link sets the "user_id" value to a given role, which in our case is the SettingsNode.

The second <link> (lines 38-44) says that when the user, with the stored "user_id", performs the selection of the "logo" media, the application will synthesize the word "where" and expect a "there" voice command. To use the stored "user_id" in the <simpleCondtion> for the "onSelection" role, this <link> also uses our extended <linkParam> (lines 39-40).

Finally, the third <link> (lines 45-58) says that when another user performs the voice command "there", the "logo" must be moved to the new position at which this user is pointing. 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.

```
1<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
2  <head>
3    <connectorBase>
4      <causalConnector id="onRecognizeSetUserId">
5        <simpleCondition role="onRecognize"/>
6        <connectorParam name="get_recognized_user" evenType="recognition"
7          attributeType="user_id"/>
8        <simpleAction role="set" value="$get_recognized_user"/>
9      </causalConnector>
10     <causalConnector id="onSelectionByUserStart">
11       <connectorParam name="user_id"/>
12       <simpleCondition role="onSelection" user_id="$user_id"/>
13       <simpleAction role="start"/>
14     </causalConnector>
15     <causalConnector id="onRecognizeByExcludedUserSet">
16       <connectorParam name="excluded_user_id"/>
17       <simpleCondition role="onRecognize" qualifier="seq"
18         excluded_user_id="$excluded_user_id"/>
19       <simpleAction role="set"/>
20     </causalConnector>
21   </connectorBase>
22   <userBase>
23     <userClass id="BoltLikeUser" min="2" max="2"
24       userClassDescription="boltlikeuser.sparql"/>
25   </userBase>
26 </head>
27 <body>
28   ... <!--### code from Listing 16 ###-->
29   <media id="settings" type="application/x-ncl-settings">
30     <property name="first_user"/>
31   </media>
32   <link xconnector="onRecognizeSetUserId">
33     <bind role="onRecognize" component="asr" interface="put_that"/>
34     <linkParam name="get_recognized_user" component="asr"
35       interface="put_that"/>
36     <bind role="set" component="settings" interface="first_user"/>
37   </link>
```

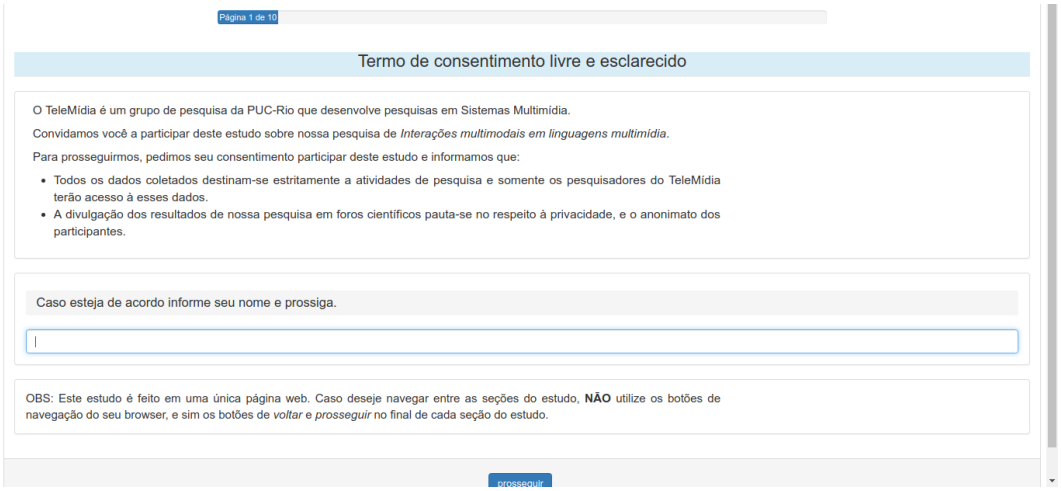
```
38    <link xconnector="onSelectionByUserStart">
39      <linkParam name="user_id" component="settings"
40         interface="first_user"/>
41      <bind role="onSelection" component="logo"/>
42      <bind role="start" component="sentences" interface="where"/>
43      <bind role="start" component="asr" interface="there"/>
44    </link>
45    <link xconnector="onRecognizeByExcludedUserSet">
46      <linkParam name="excluded_user_id" component="settings"
47        interface="first_user"/>
48      <bind role="onRecognize" component="asr" interface="there"/>
49      <bind role="onRecognize" component="pointer" interface="move"/>
50      <bind role="getValueX" component="pointer" interface="x"/>
51      <bind role="getValueY" component="pointer" interface="y"/>
52      <bind role="set" component="icon" interface="left">
53        <bindParam name="varSet" value="$getValueX"/>
54      </bind>
55      <bind role="set" component="icon" interface="top">
56        <bindParam name="varSet" value="$getValueY"/>
57      </bind>
58    </link>
59  </body>
60</ncl>
```

Listing B.7: Code fragment of “Anyone-Get-That-Someone-Else-Put-It-There”.

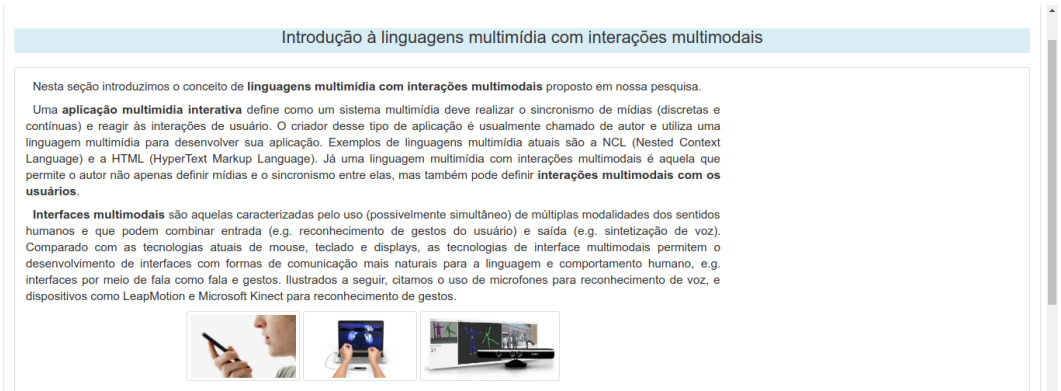
C
Screenshots

Each section of this appendix presents screenshots of each page of the website used in the evaluation study.

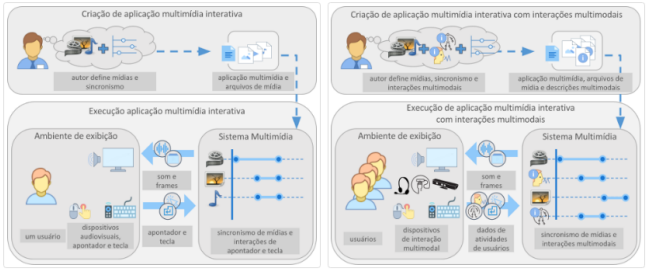
C.1
Page 1 for all participants



C.2
Page 2 for all participants



A figura a seguir ilustra a criação e execução de uma aplicação multimídia com interações multimodais. Na criação, a principal diferença é que o autor define não apenas as mídias e o sincronismos entres elas, mas também, as interações multimodais utilizando descrições multimodais. Por exemplo, ele pode utilizar uma descrição de reconhecimento de voz e um de reconhecimento de gestos. Já na execução, a principal diferença é que o sistema utiliza **dispositivos de interação multimodal** para realizar reconhecimentos, além de dispositivos audiovisuais para exibir o conteúdo das mídias (som e frames), e apontador e tecla para capturar interações dos usuários.



voltar prosseguir

C.3
Page 3 for all participants

Dados do participante

Qual sua formação ou curso atual?

- Graduação em computação
- Graduação em design
- Pós-graduação em computação
- Pós-graduação em design
- Outra graduação ou outra pós-graduação

Quantas aplicações multimídia interativas você já desenvolveu (independente de tecnologia ou linguagem de programação)?

- 0
- 1-2
- 3-4
- 5-6
- 7-8
- 8 ou mais

Você já desenvolveu alguma aplicação com interações multimodais?

- Não
- Sim

Você já utilizou alguma ferramenta programação por blocos, como Scratch ou MIT AppInventor?

- Não
- Sim

Qual seu conhecimento no desenvolvimento em NCL?

- Nenhum
- Muito pouco
- Pouco
- Razoável
- Alto
- Muito alto
- Expert

Qual seu conhecimento no desenvolvimento em HTML/JavaScript?

- Nenhum
- Muito pouco
- Pouco
- Razoável
- Alto
- Muito alto
- Expert

C.4
Page 4 for all participants

Em nossa pesquisa propomos quatro conceitos que são necessários para uma linguagem multimídia suportar interações multimodais, são eles: Mídia, Reconhecedor, Relacionamento e Grupo de Usuários.

Para evitar possíveis dificuldades com a sintaxe das linguagens, apresentaremos esses conceitos utilizando uma abordagem de **desenvolvimento de aplicações através de blocos**. Esse tipo de abordagem é bastante utilizado para o ensino de programação ou para ferramentas de geração de código. Em especial, esse tipo de desenvolvimento foi popularizado por ferramentas como o **MIT Scratch** e **MIT App** Inventor (ilustrados a seguir).

Vamos agora detalhar esses conceitos e pedir que você realize algumas tarefas. **Ressaltamos que essas tarefas NÃO se destinam a avaliar seus conhecimentos, mas sim capturar evidências de nossa pesquisa.**

Conceitos 1.1

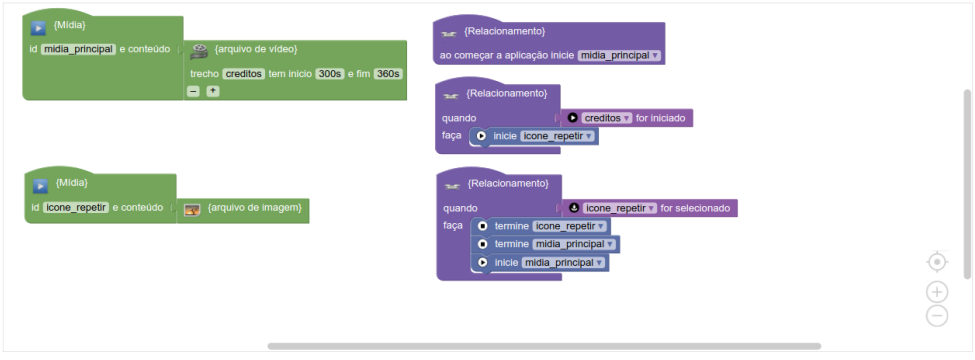
O conceito de Mídia permite definir o uso de conteúdo audiovisual. Uma Mídia é definida por **um identificador, um conteúdo e âncoras**. O conteúdo consiste em um arquivo de mídia, como imagem jpeg, vídeo mp4, áudio mp3, entre outros. As âncoras são porções do conteúdo e podem ser trechos temporais (e.g. entre 300s e 360s) ou trechos

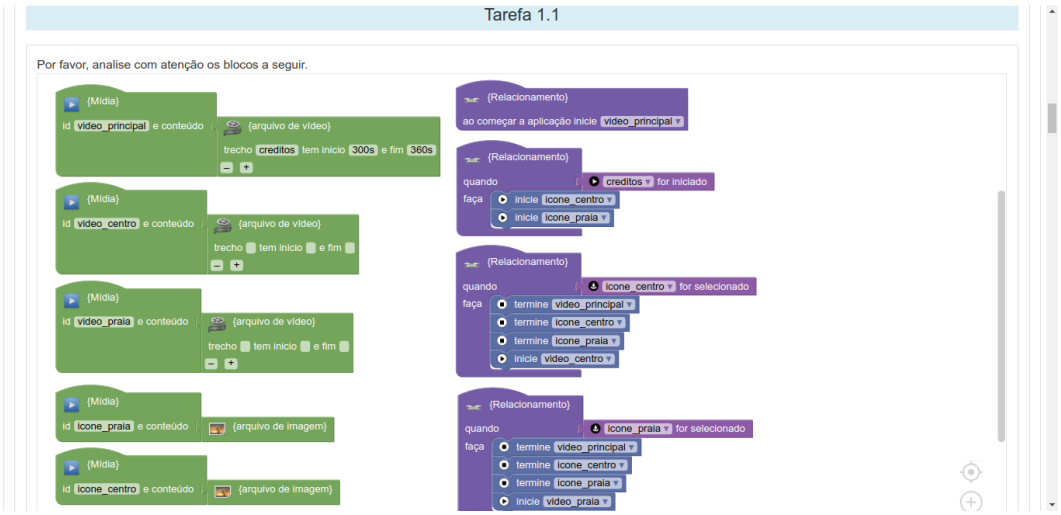
Na nossa representação de blocos, uma Mídia é definida juntando um bloco de Mídia, com o campo id preenchido, e um bloco de conteúdo, com as âncoras preenchidas. Os blocos a seguir definem duas mídias com identificadores *midia_principal* e *icone_repetir*, e conteúdos de vídeo e imagem, respectivamente. Em especial, *midia_principal* possui uma âncora chamada de *creditos* que inicia aos 300 e termina aos 360 segundos.

O conceito de Relacionamento permite definir o comportamento das aplicações por meio de relações causais. Um Relacionamento é definido por **um conjunto de condições e um conjunto de ações**. As ações são executadas em sequência quando as condições são satisfeitas.

Na nossa representação de blocos, um Relacionamento é criado juntando um bloco de Relacionamento com blocos de Condições e blocos de Ações. Blocos de Ações podem ser de iniciar ("inicie") ou parar ("pare") a execução de uma Mídia ou Reconhecedor. Já as condições podem ser simples ou compostas. Blocos de Condições simples podem ser de início ("for iniciado") ou fim ("for terminado") de uma mídia ou de sua âncora, seleção de mídia pelo usuário ("for selecionado") ou reconhecimento de uma interação multimodal ("for reconhecido")

Para ilustrar o uso desses conceitos, os blocos a seguir definem uma aplicação que apresenta um vídeo e um ícone, durante os créditos desse vídeo. Se o usuário selecionar o ícone, o vídeo é reiniciado. Essa aplicação utiliza duas Mídias (*midia_principal* e *icone_repetir*) e três Relacionamentos para definir o comportamento da aplicação. O primeiro Relacionamento define que a *midia_principal* inicia junto com a aplicação. O segundo Relacionamento define que quando a *midia_principal* alcançar o seu trecho de *creditos* a mídia *icone_repetir* deve ser iniciada. O terceiro Relacionamento define que quando a mídia *icone_repetir* for selecionada, a *midia_principal* deve ser reiniciada (terminada e iniciada)

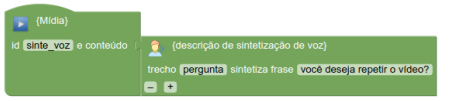




Qual é o comportamento da aplicação que corresponde aos blocos acima?

Conceitos 1.2

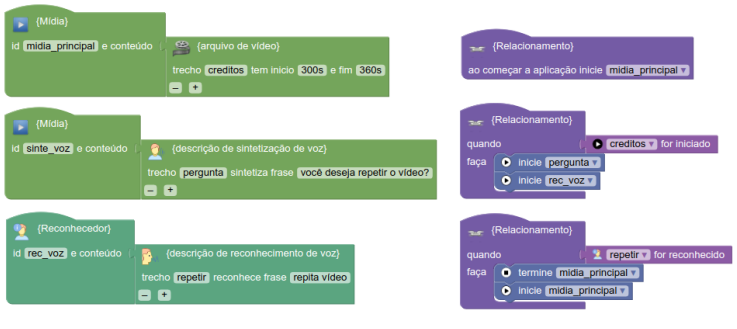
Além de mídias como imagens, áudios e vídeos, o conceito de Mídia abrange também outras modalidades de conteúdo, e.g. voz sintetizada. Os blocos a seguir definem uma Mídia com identificador *sinte_voz*, que tem como conteúdo um arquivo de texto para sintetização de voz. Esse conteúdo de sintetização possui o trecho *pergunta* que sintetiza a frase "você deseja repetir o vídeo?".



Um Reconhecedor permite o reconhecimento de interações multimodais realizadas por usuários, como voz e gestos. Ele é definido por **um identificador, seu conteúdo e âncoras**. Entretanto, diferente de Mídia, o conteúdo de um Reconhecedor deve ser uma descrição de reconhecimento. As âncoras são porções delimitadas dessa descrição. Por exemplo, uma descrição de reconhecimento de voz deve ter âncoras que definem frases a serem reconhecidas.

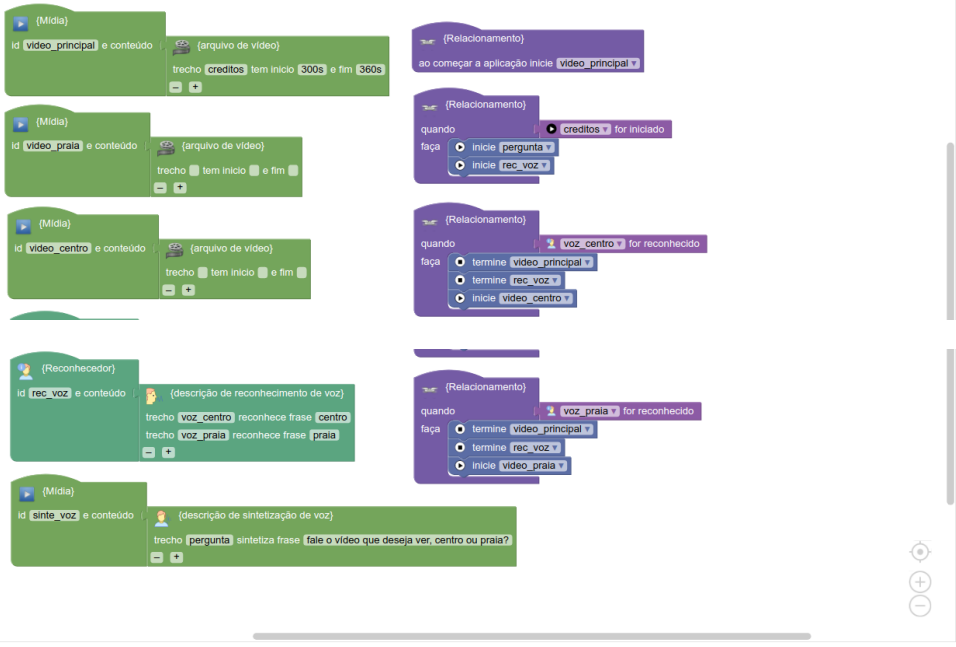
Na nossa representação de blocos, o Reconhecedor é definido juntando-se um bloco de Reconhecedor com o campo id preenchido, e um bloco de conteúdo com as âncoras preenchidas. Os blocos a seguir definem o reconhecedor *rec_voz*, que tem um reconhecimento de voz como conteúdo. Essa descrição tem o trecho *repetir* que reconhece o comando de voz "repita vídeo".

Para ilustrar o uso desses conceitos, os blocos a seguir definem uma nova versão da aplicação em Conceitos 1.1, a qual que reinicia um vídeo dada uma interação. Nessa versão, em vez de selecionar o ícone, o vídeo é reiniciado quando usuário falar "repita vídeo". Essa aplicação utiliza duas Mídia (*midia_principal* e *sinte_voz*), um Reconhecedor (*rec_voz*) e três Relacionamentos. O primeiro Relacionamento define que a *midia_principal* é iniciada quando a aplicação iniciar. O segundo Relacionamento define que quando a *midia_principal* alcançar o seu trecho de *creditos* (300s), a frase da âncora *pergunta* é sintetizada e o Reconhecedor *rec_voz* inicia o reconhecimento. O último Relacionamento define que quando for reconhecida a âncora *repetir*, a *midia_principal* deve ser reiniciada (terminada e iniciada).



Tarefa 1.2

Os blocos a seguir são uma versão modificada da Tarefa 1.1. Por favor, analise com atenção.

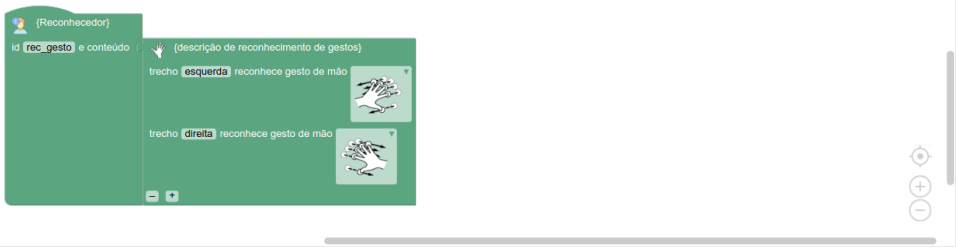


Qual é o novo comportamento da aplicação?

Conceitos 1.3

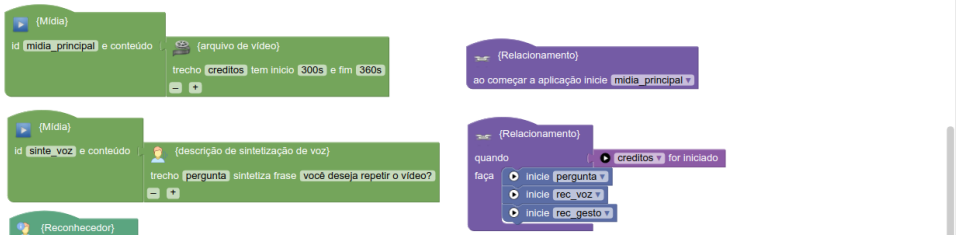
Um dos principais benefícios de interfaces multimodais é o uso de diferentes interações, ou seja, as interações de usuários podem ser realizadas por diferentes modalidades.

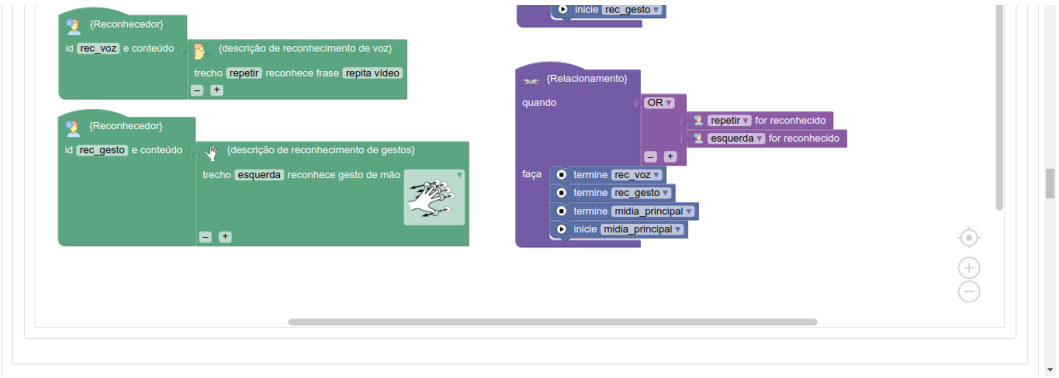
Os blocos a seguir definem o reconhecedor *rec_gestos*, que tem uma descrição de reconhecimento de gestos como conteúdo. Esse conteúdo utiliza duas âncoras (*esquerda* e *direita*) para definir gestos de deslizar mão nessas direções.



Em nossa representação de blocos, a combinação de modalidades de interação pode ser feita utilizando uma condição composta. A combinação de condições compostas podem utilizar os seguintes operadores: "or" quando apenas uma das condições é necessária; 'and' quando todas as condições são necessárias em qualquer ordem; e 'seq' quando todos as condições são necessárias e na sequência estabelecida.

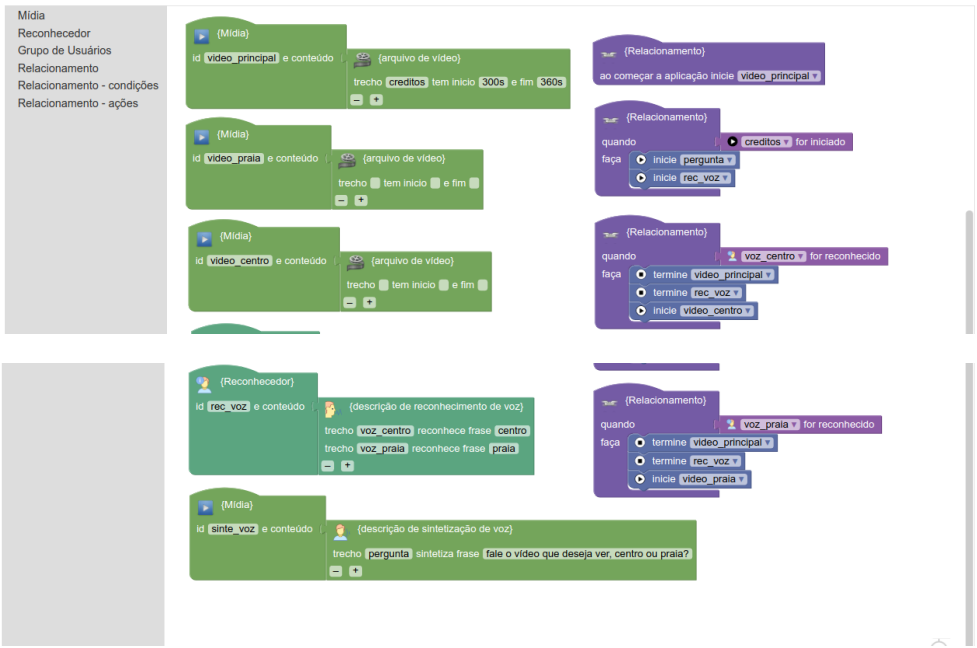
Os blocos a seguir definem uma nova versão da aplicação em Conceitos 1.2 que reinicia um vídeo dada uma interação por voz. Nessa versão falar "repita vídeo" ou (operador "or") fazer quando gesto de deslizar a mão para esquerda.





Tarefa 1.3

Agora pedimos que você edite os blocos da Tarefa 1.2 (copiados a seguir) para que seja possível a interação por voz **ou** (operador OR) interação por gestos. Na interação por gesto, considere que o gesto de deslizar a mão para esquerda indica centro e gesto de deslizar a mão para direita indica praia.

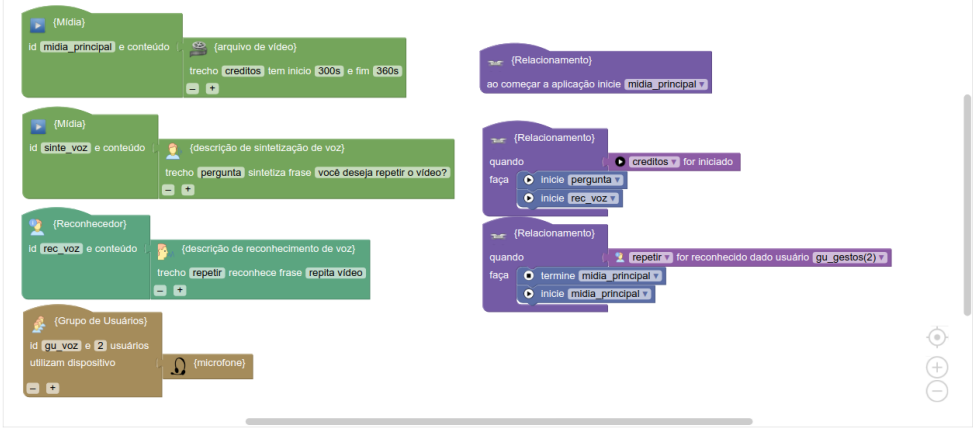


Conceitos 1.4

O conceito de Grupo de usuários permite identificar unicamente interações de cada usuário. Por exemplo, o comando de voz apenas de um determinado usuário. Ele é definido por **um identificador, número máximo de participantes e quais dispositivos estes utilizam**. Mais precisamente, esses dispositivos definem o que um usuário necessita possuir para participar do grupo.

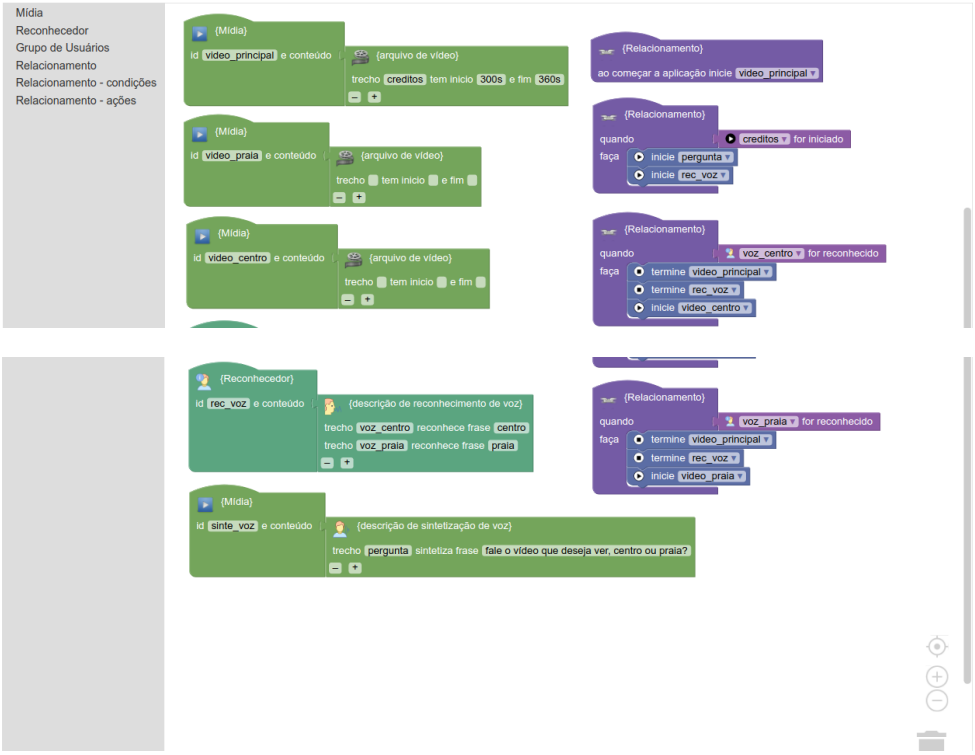
Na nossa representação de blocos, o Grupo de Usuários é definida juntando um bloco de Grupo de Usuários com campo id preenchido e blocos de dispositivos. Por exemplo, os blocos a seguir definem um grupo de usuário *gu_gestos* que pode ter até 2 membros, cada qual com seu reconhecedor de gestos (e.g. LeapMotion).

Para ilustrar o uso de um Grupo de Usuários, os blocos a seguir são uma nova versão da aplicação em Conceitos 1.2, a qual reinicia um vídeo dada uma interação por voz. Nessa versão o vídeo será reiniciado apenas quando o segundo usuário de um grupo falar "repita vídeo". Esse grupo é definido com um máximo de 2 usuários com microfone.

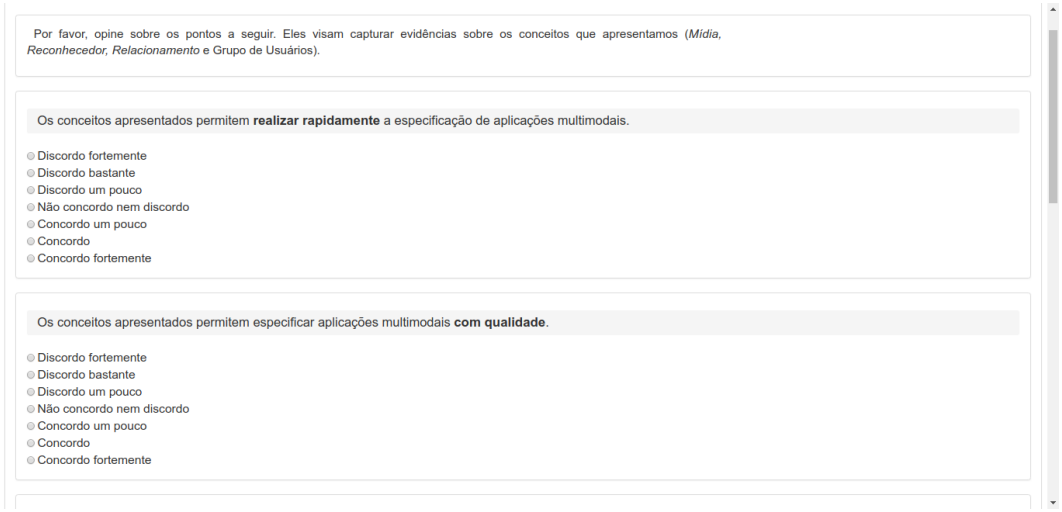


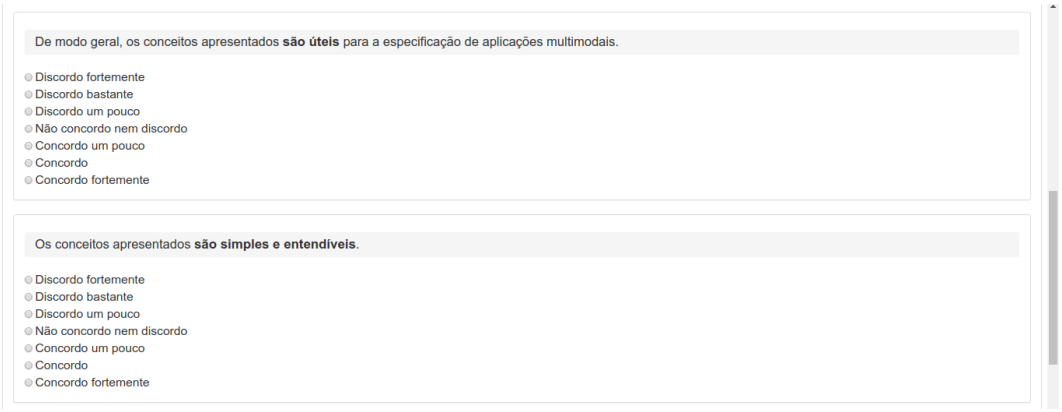
Tarefa 1.4

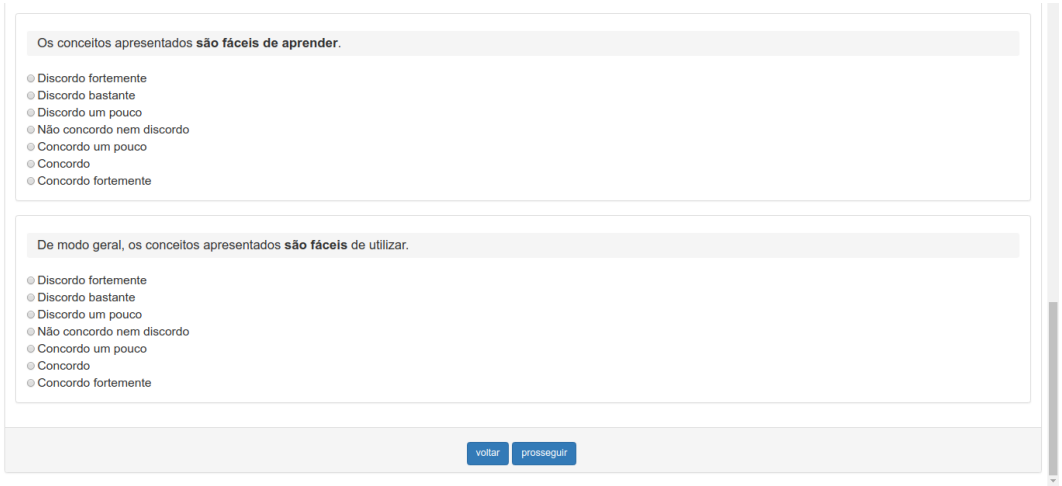
Agora pedimos que você edite novamente os blocos da Tarefa 1.2 (copiados a seguir) para que apenas o segundo usuário, de um grupo de 3 usuários com microfone, possa realizar a interação por voz.



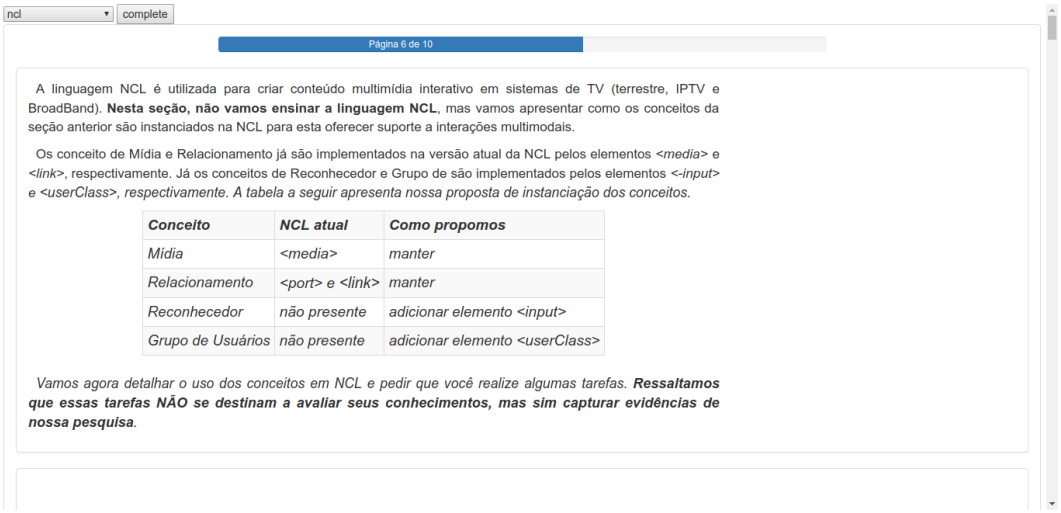
C.5
Page 5 for all participants







C.6
Page 6 for NCL participants

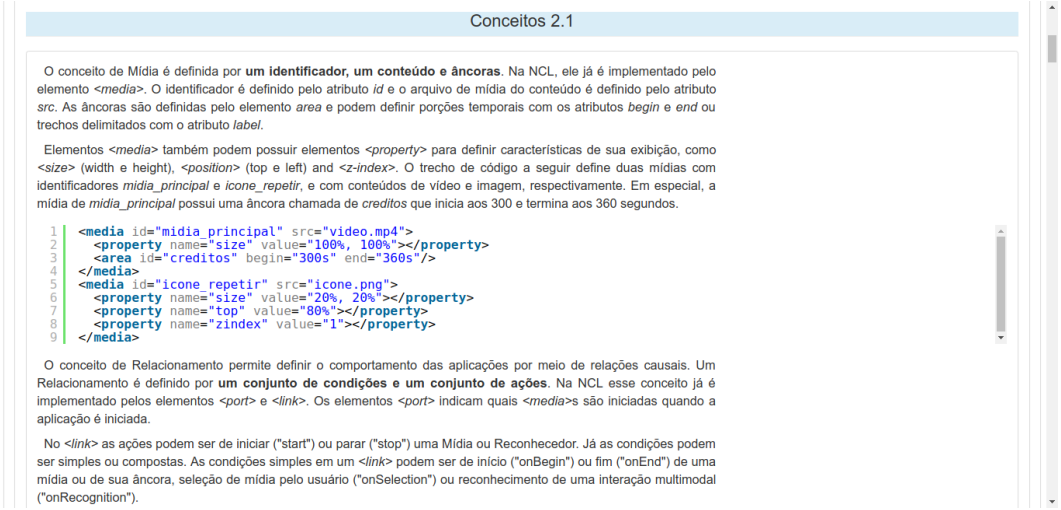


A linguagem NCL é utilizada para criar conteúdo multimídia interativo em sistemas de TV (terrestre, IPTV e BroadBand). **Nesta seção, não vamos ensinar a linguagem NCL**, mas vamos apresentar como os conceitos da seção anterior são instanciados na NCL para esta oferecer suporte a interações multimodais.

Os conceito de Mídia e Relacionamento já são implementados na versão atual da NCL pelos elementos *<media>* e *<link>*, respectivamente. Já os conceitos de Reconhecedor e Grupo de são implementados pelos elementos *<-input>* e *<userClass>*, *respectivamente. A tabela a seguir apresenta nossa proposta de instanciação dos conceitos.*

Conceito	***NCL atual***	***Como propomos***
Mídia	*<media>*	*manter*
Relacionamento	*<port> e <link>*	*manter*
Reconhecedor	*não presente*	*adicionar elemento <input>*
Grupo de Usuários	*não presente*	*adicionar elemento <userClass>*

Vamos agora detalhar o uso dos conceitos em NCL e pedir que você realize algumas tarefas. ***Ressaltamos que essas tarefas NÃO se destinam a avaliar seus conhecimentos, mas sim capturar evidências de nossa pesquisa****.*



O conceito de Mídia é definida por **um identificador, um conteúdo e âncoras**. Na NCL, ele já é implementado pelo elemento *<media>*. O identificador é definido pelo atributo *id* e o arquivo de mídia do conteúdo é definido pelo atributo *src*. As âncoras são definidas pelo elemento *area* e podem definir porções temporais com os atributos *begin* e *end* ou trechos delimitados com o atributo *label*.

Elementos *<media>* também podem possuir elementos *<property>* para definir características de sua exibição, como *<size>* (width e height), *<position>* (top e left) and *<z-index>*. O trecho de código a seguir define duas mídias com identificadores *midia_principal* e *icone_repetir*, e com conteúdos de vídeo e imagem, respectivamente. Em especial, a mídia de *midia_principal* possui uma âncora chamada de *creditos* que inicia aos 300 e termina aos 360 segundos.

```
<media id="midia_principal" src="video.mp4">
  <property name="size" value="100%, 100%"></property>
  <area id="creditos" begin="300s" end="360s"/>
</media>
<media id="icone_repetir" src="icone.png">
  <property name="size" value="20%, 20%"></property>
  <property name="top" value="80%"></property>
  <property name="zindex" value="1"></property>
</media>
```

O conceito de Relacionamento permite definir o comportamento das aplicações por meio de relações causais. Um Relacionamento é definido por **um conjunto de condições e um conjunto de ações**. Na NCL esse conceito já é implementado pelos elementos *<port>* e *<link>*. Os elementos *<port>* indicam quais *<media>*s são iniciadas quando a aplicação é iniciada.

No *<link>* as ações podem ser de iniciar ("start") ou parar ("stop") uma Mídia ou Reconhecedor. Já as condições podem ser simples ou compostas. As condições simples em um *<link>* podem ser de início ("onBegin") ou fim ("onEnd") de uma mídia ou de sua âncora, seleção de mídia pelo usuário ("onSelection") ou reconhecimento de uma interação multimodal ("onRecognition").

A relação entre condições e ações é definido pelo atribudo *xconnector* do *<link>*. Por exemplo, um *xconnector* "onBeginStart" define uma condição "onBegin" e ações de "start". Já um *xconnector onSelectionStopStart* define uma condição "onSelection" e ações de "stop" e "start". A associação de elementos *<media>* ou *<input>* com as condições e ações de um *<link>* é definida pelo elemento *<bind>* do *<link>*.

Para ilustrar o uso desses conceitos, o trecho de código a seguir define uma aplicação que apresenta um vídeo e um ícone durante os créditos do vídeo. Se o usuário selecionar o ícone, o vídeo é reiniciado. Ele utiliza dois elementos de *<media>* (*video_principal* e *icone_repetir*), um *<port>* e dois *<link>*. O *<port>* define que o elemento *midia_principal* inicia com a aplicação. O primeiro *<link>* define que quando o *video_principal* alcançar o seu trecho de *creditos a imagem icone_repetir é iniciada. O segundo <link> define que quando esse icone_repetir for selecionado o video_principal será reiniciado (stop e start).*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl>
  <connectorBase>
    <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
  </connectorBase>

  <body>
    <port component="video_principal"></port>
    <media id="video_principal" src="video.mp4">
      <property name="size" value="100%, 100%"></property>
      <area id="creditos" begin="300s" end="360s"/>
    </media>
    <media id="icone_repetir" src="icone_repetir.png">
      <property name="size" value="20%, 20%"></property>
      <property name="zindex" value="1"></property>
    </media>
    <link xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="video_principal" interface="creditos"></bind>
      <bind role="start" component="icone_repetir"></bind>
    </link>
    <link xconnector="conEx#onSelectionStopStart">
      <bind role="onSelection" component="icone_repetir"></bind>
      <bind role="stop" component="video_principal"></bind>
      <bind role="start" component="video_principal"></bind>
    </link>
  </body>
</ncl>
```

Tarefa 2.1

Por favor, analise com atenção o trecho de código NCL a seguir.

```
<?xml version= "1.0" encoding= "ISO-8859-1" ?>
<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <connectorBase>
      <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
    </connectorBase>
  </head>
  <body>
    <port component="video_inicial"></port>
    <media id="video_inicial" src="video_inicial.mp4">
      <property name="size" value="100%, 100%"></property>
      <area id="creditos" begin="300s" end="360s"/>
    </media>
    <media id="video_centro" src="video_centro.mp4">
      <property name="size" value="100%, 100%"></property>
    </media>
    <media id="video_praia" src="video_praia.mp4">
      <property name="size" value="100%, 100%"></property>
    </media>
    <media id="icone_centro" src="icone_centro.png">
      <property name="top" value="80%"></property>
      <property name="size" value="20%, 20%"></property>
      <property name="zindex" value="1"></property>
    </media>
    <media id="icone_praia" src="icone_praia.png">
      <property name="location" value="80%, 80%"></property>
      <property name="size" value="20%, 20%"></property>
      <property name="zindex" value="1"></property>
    </media>

    <link xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="video_inicial" interface="creditos">
      <bind role="start" component="icone_centro"></bind>
      <bind role="start" component="icone_praia"></bind>
    </link>
    <link xconnector="onSelectionStopStart">
      <bind role="onSelection" component="icone_centro"></bind>
      <bind role="stop" component="icone_centro"></bind>
      <bind role="stop" component="icone_praia"></bind>
      <bind role="stop" component="video_inicial"></bind>
      <bind role="start" component="video_centro"></bind>
    </link>
    <link xconnector="onSelectionStopStart">
      <bind role="onSelection" component="icone_praia"></bind>
      <bind role="stop" component="icone_centro"></bind>
      <bind role="stop" component="icone_praia"></bind>
      <bind role="stop" component="video_inicial"></bind>
      <bind role="start" component="video_praia"></bind>
    </link>
  </body>
</ncl>
```

Qual é o comportamento da aplicação?

Conceitos 2.2

Além de mídias como imagens, áudios e vídeos, o elemento *<media>* da NCL estendida também deve suportar outras modalidades de conteúdo, como interações por voz.

Os trechos de código a seguir ilustram o uso de sintetização de voz na NCL estendida. O primeiro trecho de código apresenta o arquivo *sinte_voz.ssml* que segue o formato SSML (Speech Synthesis Markup Language) para sintetização de voz. Ele possui um elemento <s> com o identificador "pergunta" que sintetizada a frase: "você deseja repetir o vídeo?". O segundo trecho de código define um elemento *<media>* com identificador *sinte_voz*, que tem como conteúdo o arquivo *sinte_voz.ssml*. Essa *<media>* possui uma âncora que indica a frase a ser sintetizada.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="repete">repita vídeo</rule>
</grammar>
```

```
<input id="rec_voz" src="rec_voz.srgs">
  <area label="repete"></area>  <!-- referência id repete de rec_voz.srgs-->
</input>
```

Para ilustrar o uso de interação por voz, o trecho de código a seguir apresenta uma nova versão (modificações em destaque) da aplicação da parte Conceitos 2.1 que reinicia um vídeo dada uma interação por seleção. Nessa versão, em vez de selecionar o ícone, o vídeo é reiniciado quando usuário falar "repita vídeo". A primeira diferença está no uso dos elementos *sinte_voz* e *rec_voz* para interação por voz, que utilizam os arquivos *sinte_voz.srgs* e *rec_voz.srgs* (apresentados acima).

Dois elementos *<link>* também foram modificados. O primeiro *<link>* modificado define que quando o *video_principal* alcançar o trecho de *creditos* (300s), a frase da âncora *pergunta* é sintetizada e o reconhecedor *rec_voz* é ativado (a partir desse momento, ele poderá reconhecer interações de voz). Já o segundo *<link>* modificado define que quando for reconhecida a âncora *repete*, a *midia_principal* será reiniciada (terminada e iniciada).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl>
  <head>
    <connectorBase>
      <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
    </connectorBase>
  </head>
  <body>
    <port component="video_principal"></port>
    <media id="video_principal" src="video.mp4">
      <property name="size" value="100%, 100%"></property>
      <area id="creditos" begin="300s" end="360s"/>
    </media>
    <media id="sinte_voz" src="sinte_voz.ssml">
      <area label="pergunta"/> <!-- referência id pergunta de sinte_voz.ssml-->
    </media>
    <input id="rec_voz" src="rec_voz.srgs">
      <area label="repete"/>  <!-- referência id repete de rec_voz.srgs-->
    </input>
    <link xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="video_principal" interface="creditos"></bind>
      <bind role="start" component="sinte_voz" interface="pergunta"></bind>
      <bind role="start" component="rec_voz"></bind>
    </link>
    <link xconnector="conEx#onRecognizeStopStart">
      <bind role="onRecognize" component="rec_voz" interface="repete"></bind>
      <bind role="stop" component="rec_voz"></bind>
      <bind role="stop" component="video_principal">
      <bind role="start" component="video_principal">
    </link>
  </body>
</ncl>
```

Tarefa 2.2

Considere as descrições sinte_voz_videos.ssml e rec_voz_videos.srgs a seguir.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<speak xmlns="http://www.w3.org/2001/10/synthesis">
  <s id="repetir">fale o vídeo que deseja ver, centro ou praia?</s>
</speak>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="voz_centro">centro</rule>
  <rule id="voz_praia">praia</rule>
</grammar>
```

O código NCL a seguir é uma versão modificada da Tarefa 2.1 para permitir interações multimodais utilizando as duas descrições acima. Por favor, analise com atenção.

```
<?xml version= "1.0" encoding= "ISO-8859-1" ?>
<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
   <connectorBase>
     <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
   </connectorBase>
 </head>
 <body>
   <port component="video_inicial"></port>
   <media id="video_inicial" src="video_inicial.mp4">
     <property name="size" value="100%, 100%"></property>
     <area id="creditos" begin="300s" end="360s"/>
   </media>
   <media id="video_centro" src="centro.mp4">
     <property name="size" value="100%, 100%"></property>
   </media>
   <media id="video_praia" src="praia.mp4">
     <property name="size" value="100%, 100%"></property>
   </media>
   <media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
     <area label="pergunta"/>
   </media>
   <input id="rec_voz_videos" src="rec_voz_videos.sgrs">
     <area label="centro"/>
     <area label="praia"/>
   </input>
   <link xconnector="conEx#onBeginStart">
     <bind role="onBegin" component="video_inicial" interface="creditos"></bind>
     <bind role="start" component="sinte_voz_videos" interface="pergunta"></bind>
     <bind role="start" component="rec_voz_videos"></bind>
   </link>
   <link xconnector="conEx#onRecognizeStart">
     <bind role="onRecognize" component="rec_voz_videos" interface="centro"></bind>
     <bind role="stop" component="rec_voz_videos"></bind>
     <bind role="stop" component="video_inicial"></bind>
     <bind role="start" component="video_centro"></bind>
   </link>
```

```
   <link xconnector="conEx#onRecognizeStart">
     <bind role="onRecognize" component="rec_voz_videos" interface="praia"></bind>
     <bind role="stop" component="rec_voz_videos"></bind>
     <bind role="stop" component="video_inicial"></bind>
     <bind role="start" component="video_praia"></bind>
   </link>
 </body>
```

Qual é o novo comportamento da aplicação?

Conceitos 2.3

Um dos principais benefícios de interfaces multimodais é permitir que as interações dos usuários possam ser realizadas por diferentes modalidades.

O trecho de código a seguir ilustra o arquivo *rec_gestos.gml* que segue o formato GML (Gesture Markup Language) para reconhecimento de gestos de mão. Ele possui dois gestos com os identificadores *esquerda* e *direita*, que definem gestos de mão para esquerda e para direita, respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<GestureMarkupLanguage>
  <Gesture id="esquerda" type="swipe">
    ...
  </Gesture>
  <Gesture id="direta" type="swipe">
    ...
  </Gesture>
</GestureMarkupLanguage>
```

```
<input id="rec_gesto" src="rec_gesto.gml">
  <area label="esquerda"></area> <!-- referência id esquerda de rec_gesto.gml-->
  <area label="direita"></area> <!-- referência id direita de rec_gesto.gml-->
</input>
```

Na NCL estendida, a combinação de modalidades de interação pode ser definida utilizando uma condição composta. Uma condição composta em um *<link>* combina condições simples utilizando um dos seguintes operadores: 'or' quando apenas uma das condições é necessária; 'and' quando todas condições são necessárias em qualquer ordem; e 'seq' quando todas as condições são necessárias e devem acontecer na sequência estabelecida.

Para ilustrar o uso de combinação de modalidades, o trecho de código a seguir apresenta uma uma nova versão (modificações em destaque) da aplicação em Conceitos 2.2, a qual reinicia um vídeo dada uma interação por voz. Nessa versão, o vídeo é reiniciado quando usuário falar "repita vídeo" ou (operador "or") quando fizer um gesto de deslizar mão para esquerda.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl>
  <head>
    <connectorBase>
      <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
    </connectorBase>
  </head>
  <body>
    <port component="video_principal"></port>
    <media id="video_principal" src="video.mp4">
      <property name="size" value="100%, 100%"></property>
      <area id="creditos" begin="300s" end="360s"/>
    </media>
    <media id="sinte_voz" src="sinte_voz.ssml">
      <area label="pergunta"/> <!-- referência id pergunta de sinte_voz.ssml-->
    </media>
    <input id="rec_voz" src="rec_voz.srgs">
      <area label="repete"></area> <!-- referência id repete de rec_voz.srgs-->
    </input>
    <input id="rec_gesto" src="rec_gesto.gml">
      <area label="esquerda"></area> <!-- referência id esquerda de rec_gesto.gml-->
    </input>
    <link xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="video_principal" interface="creditos"></bind>
      <bind role="start" component="sinte_voz" interface="pergunta"></bind>
      <bind role="start" component="rec_voz"></bind>
      <bind role="start" component="rec_gesto"></bind>
    </link>
    <link xconnector="conEx#onOrRecognizeStopStart">
      <bind role="onRecognize" component="rec_voz" interface="repete">
      <bind role="onRecognize" component="rec_gesto" interface="esquerda">
      <bind role="stop" component="rec_voz"></bind>
      <bind role="stop" component="rec_gesto"></bind>
      <bind role="stop" component="video_principal">
      <bind role="start" component="video_principal">
    </link>
  </body>
</ncl>
```

Tarefa 2.3

Agora pedimos que você edite o código NCL da Tarefa 2.2 (copiado a seguir) para que seja possível a interação por voz **ou** (operador OR) interação por gestos. Na interação por gesto, você pode referenciar o arquivo de descrição *rec_gestos.gml*, apresentado em Conceitos 2.3, e considere que gesto de deslizar a mão para esquerda indica centro e gesto de deslizar a mão para direita indica praia.

```
<?xml version= "1.0" encoding= "ISO-8859-1" ?>
<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
  <connectorBase>
   <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
  </connectorBase>
 </head>
 <body>
  <port component="video_inicial"/>
  <media id="video_inicial" src="video_inicial.mp4">
   <property name="size" value="100%, 100%"/>
   <area id="creditos" begin="300s" end="360s"/>
  </media>
  <media id="video_centro" src="centro.mp4">
   <property name="size" value="100%, 100%"/>
  </media>
  <media id="video_praia" src="praia.mp4">
   <property name="size" value="100%, 100%"/>
  </media>
  <media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
   <area label="pergunta"/>
  </media>
  <input id="rec_voz_videos" src="rec_voz_videos.sgrs">
   <area label="centro"/>
   <area label="praia"/>
  </input>
```

```
  <link xconnector="conEx#onBeginStart">
   <bind role="onBegin" component="video_inicial" interface="creditos"/>
   <bind role="start" component="sinte_voz_videos" interface="pergunta"/>
   <bind role="start" component="rec_voz_videos"/>
  </link>
  <link xconnector="conEx#onRecognizeStart">
   <bind role="onRecognize" component="rec_voz_videos" interface="centro"/>
   <bind role="stop" component="rec_voz_videos"/>
   <bind role="stop" component="video_inicial"/>
   <bind role="start" component="video_centro"/>
  </link>
  <link xconnector="conEx#onRecognizeStart">
   <bind role="onRecognize" component="rec_voz_videos" interface="praia"/>
   <bind role="stop" component="rec_voz_videos"/>
   <bind role="stop" component="video_inicial"/>
   <bind role="start" component="video_praia"/>
  </link>
 </body>
</ncl>
```

Conceitos 2.4

O conceito de Grupo de Usuários é definido **por um identificador, número máximo de participantes e quais dispositivos estes utilizam**. Na NCL, o Grupo de Usuários é implementado pelo elemento <*userClass*>, filho do elemento <*head*>. O identificador é definido pelo atributo *id* e os dispositivos são definidos por um arquivo de descrição no formato SPARQL. O trecho de código a seguir apresenta o arquivo de descrição de usuários *gu_leap_microphone.sparql*. Essa descrição define que cada usuário do grupo deve ter um microfone e um lLeapMotion.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
PREFIX prf: <http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430>
SELECT ?person
WHERE {
?person prf:component ?component.
?component prf: name ?name FILTER regex(?name, "Leap Motion")
  ?name FILTER regex(?name, "microfone")
}
```

Para ilustrar o uso de um Grupo de Usuários em NCL, o trecho de código a seguir apresenta uma nova versão (modificações em destaque) da aplicação em Conceitos 2.2, a qual reinicia um vídeo dada uma interação por voz. Nessa versão, o vídeo será reiniciado apenas quando o segundo usuário falar "repita vídeo". Esse grupo é definido com um máximo de 2 usuários e com os dispositivos descritos no arquivo *gu_leap_microphone.sparql* (apresentado acima).

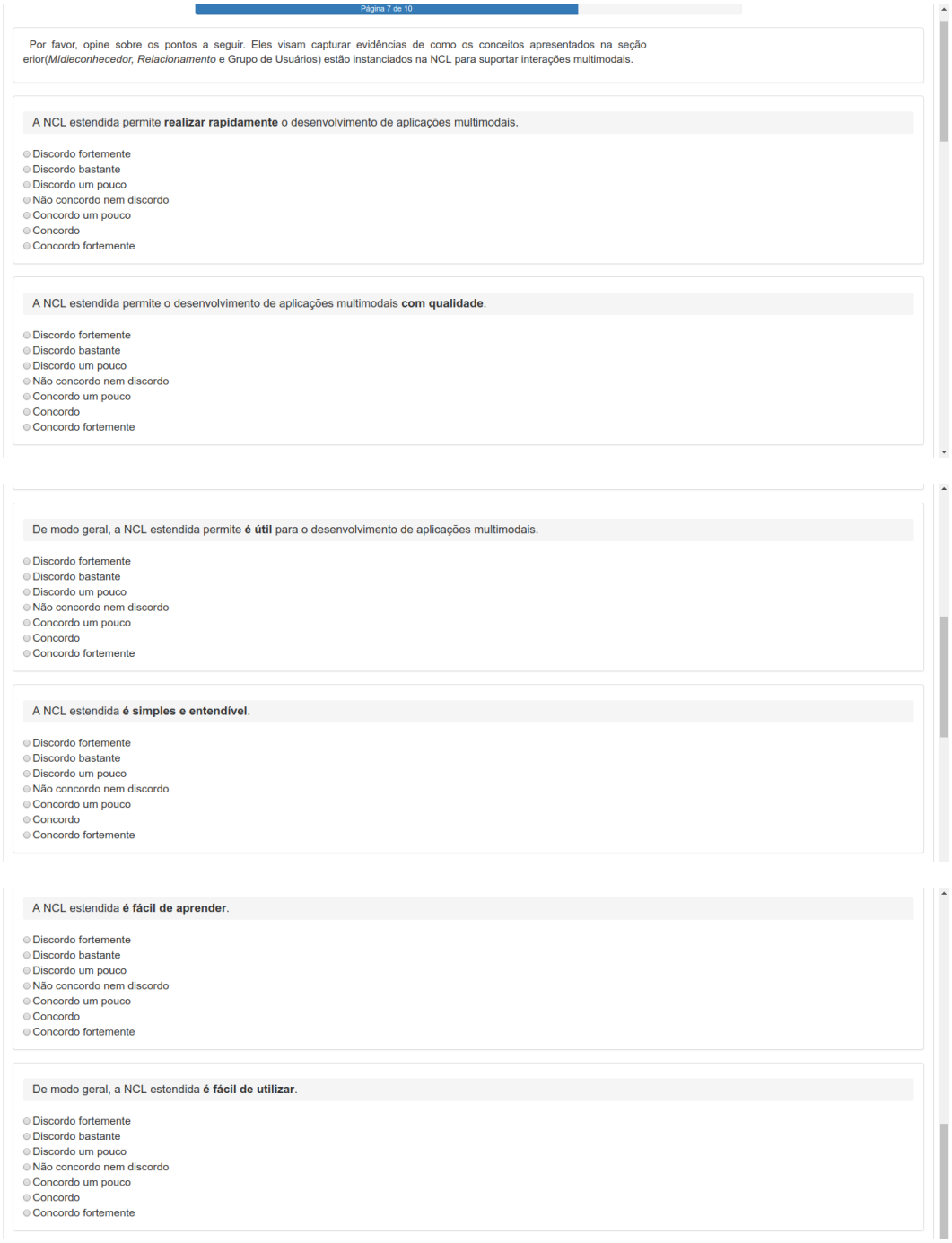
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl>
  <head>
    <connectorBase>
      <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
    </connectorBase>
    <userBase>
      <userClass id="gu_leap_microphone" max="2"
        userClassDescription="gu_leap_microphone.sparql"></userClass>
    </userBase>
  </head>
  <body>
    <port component="video_principal"></port>
    <media id="video_principal" src="video.mp4">
      <property name="size" value="100%, 100%"></property>
      <area id="creditos" begin="300s" end="360s"/>
    </media>
    <media id="sinte_voz" src="sinte_voz.srgs">
      <area label="pergunta"/> <!-- referência id pergunta de sinte_voz.srgs-->
    </media>
    <input id="rec_voz" src="rec_voz.srgs">
      <area label="repete"/> <!-- referência id repete de rec_voz.srgs-->
    </input>
    <link xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="video_principal" interface="creditos"></bind>
      <bind role="start" component="sinte_voz" interface="pergunta"></bind>
      <bind role="start" component="rec_voz"></bind>
    </link>
    <link xconnector="conEx#onRecognizeStart">
      <bind role="onRecognize" component="rec_voz" interface="repete">
        <bindParam name="user_id" value="gu_leap_microphone(2)"></bindParam>
      </bind>
      <bind role="stop" component="video_principal"></bind>
      <bind role="start" component="video_principal"></bind>
    </link>
  </body>
</ncl>
```

Tarefa 2.4

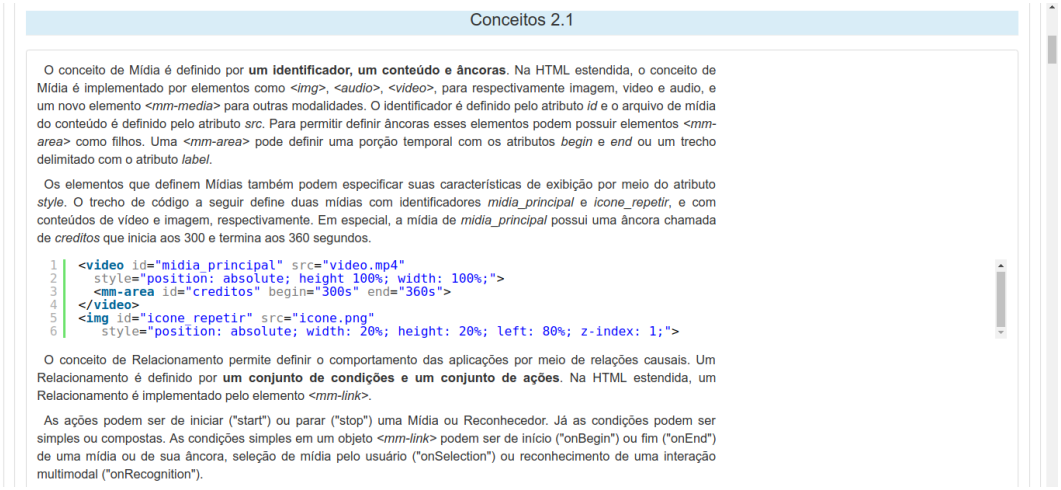
Agora pedimos que você edite novamente o trecho de código NCL da Tarefa 2.2 (copiado a seguir) para que apenas o segundo usuário, de um grupo de 3 usuários com microfone, possa realizar a interação por voz.

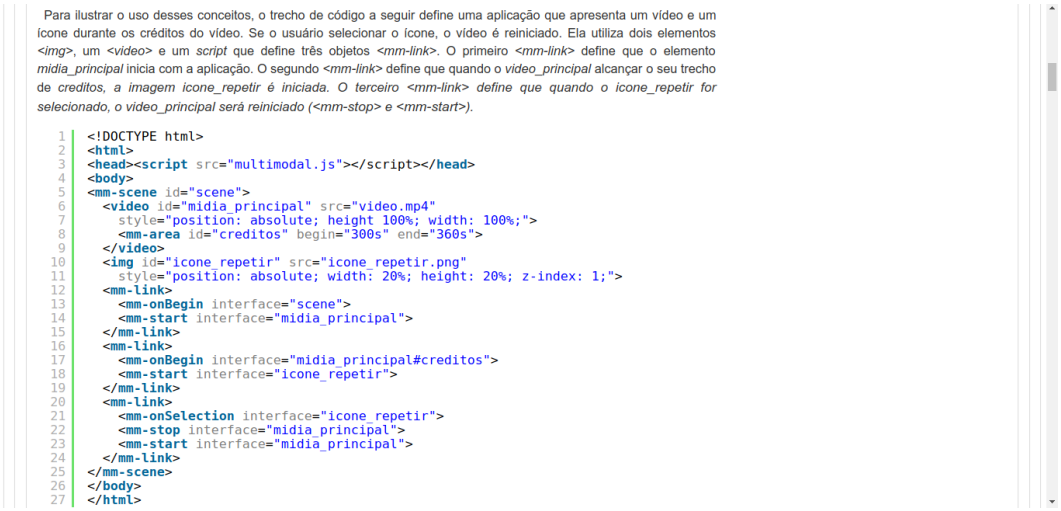
```
<?xml version= "1.0" encoding= "ISO-8859-1" ?>
<ncl xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
  <connectorBase>
   <importBase documentURI="causalConnBase.ncl" alias="conEx"></importBase>
  </connectorBase>
 </head>
 <body>
  <port component="video_inicial"/>
  <media id="video_inicial" src="video_inicial.mp4">
   <property name="size" value="100%, 100%"/>
   <area id="creditos" begin="300s" end="360s"/>
  </media>
  <media id="video_centro" src="centro.mp4">
   <property name="size" value="100%, 100%"/>
  </media>
  <media id="video_praia" src="praia.mp4">
   <property name="size" value="100%, 100%"/>
  </media>
  <media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
   <area label="pergunta"/>
  </media>
  <input id="rec_voz_videos" src="rec_voz_videos.sgrs">
   <area label="centro"/>
   <area label="praia"/>
  </input>

  <link xconnector="conEx#onBeginStart">
   <bind role="onBegin" component="video_inicial" interface="creditos"/>
   <bind role="start" component="sinte_voz_videos" interface="pergunta"/>
   <bind role="start" component="rec_voz_videos"/>
  </link>
  <link xconnector="conEx#onRecognizeStart">
   <bind role="onRecognize" component="rec_voz_videos" interface="centro"/>
   <bind role="stop" component="rec_voz_videos"/>
   <bind role="stop" component="video_inicial"/>
   <bind role="start" component="video_centro"/>
  </link>
  <link xconnector="conEx#onRecognizeStart">
   <bind role="onRecognize" component="rec_voz_videos" interface="praia"/>
   <bind role="stop" component="rec_voz_videos"/>
   <bind role="stop" component="video_inicial"/>
   <bind role="start" component="video_praia"/>
  </link>
 </body>
</ncl>
```

C.7
Page 7 for NCL participants



C.8
Page 6 for HTML participants





Tarefa 2.1

Por favor, analise com atenção o trecho de código HTML a seguir.

```
<html>
<head><script src="multimodal.js"></script></head>
<mm-scene id="scene">
  <video id="video_inicial" src="video_inicial.mp4"
    style="position: absolute; height 100%; width: 100%;">
    <mm-area id="creditos" begin="300s" end="360s">
  </video>
  <video id="video_centro" src="video_centro.mp4"
    style="position: absolute; height 100%; width: 100%;">
  </video>
  <video id="video_praia" src="video_praia.mp4"
    style="position: absolute; height 100%; width: 100%;">
  </video>
  <img id="icone_centro" src="icone_centro.png"
    style="position: absolute; width: 20%; height: 20%; z-index: 1;">
  <img id="icone_praia" src="icone_praia.png"
    style="position: absolute; width: 20%; height: 20%; left: 80%; z-index: 1;">
```

```
    style="position: absolute; width: 20%; height: 20%; left: 80%; z-index: 1;">
  <mm-link>
    <mm-onBegin interface="scene">
    <mm-start interface="video_inicial">
  </mm-link>
  <mm-link>
    <mm-onBegin interface="video_inicial#creditos">
    <mm-start interface="icone_centro">
    <mm-start interface="icone_praia">
  </mm-link>
  <mm-link>
    <mm-onSelection interface="icone_centro">
    <mm-stop interface="video_inicial">
    <mm-stop interface="icone_centro">
    <mm-stop interface="icone_praia">
    <mm-start interface="video_centro">
  </mm-link>
  <mm-link>
    <mm-onSelection interface="icone_praia">
    <mm-stop interface="video_inicial">
    <mm-stop interface="icone_centro">
    <mm-stop interface="icone_praia">
    <mm-start interface="video_praia">
  </mm-link>
</mm-scene>
</body>
</html>
```

Qual é o comportamento da aplicação?

Conceitos 2.2

Além de mídias como imagens, áudios e vídeos, a HTML estendida também deve suportar outras modalidades de conteúdo, como interações por voz, através do elemento *<mm-media>*

Os trechos de código a seguir ilustram o uso de sintetização de voz na HTML estendida. O primeiro trecho de código apresenta o arquivo *sinte_voz.ssml* que segue o formato SSML (Speech Synthesis Markup Language) para sintetização de voz. Ele possui um elemento *<s>* com identificador "pergunta" que sintetizada a frase: "você deseja repetir o vídeo?". O segundo trecho de código define uma *<mm-media>* com identificador *sinte_voz*, que tem como conteúdo o arquivo *sinte_voz.ssml*. Esse *<mm-media>* possui uma âncora que indica a frase a ser sintetizada.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<speak xmlns="http://www.w3.org/2001/10/synthesis">
  <s id="pergunta">você deseja repetir o vídeo?</s>
</speak>
```

```
<mm-media id="sinte_voz" src="sinte_voz.ssml">
  <mm-area label="pergunta"> <!-- referência id pergunta de sinte_voz.ssml-->
</mm-media>
```

O conceito de Reconhecedor é definido por **um identificador, seu conteúdo e âncoras**. Na HTML estendida, ele é implementado pelo elemento *<mm-input>*. O identificador é definido pelo atributo *id* e a descrição de reconhecimento é definida pelo atributo *src*. As âncoras são definidas pelo elemento *area* e podem definir trechos delimitados da descrição com o atributo *label*.

Para ilustrar o uso de um Reconhecedor, os trechos de código a seguir ilustram o uso de reconhecimento de voz na HTML estendida. O primeiro trecho ilustra o arquivo *rec_voz.srgs* que segue o formato SRGS (Speech Recognition Grammar Specification) reconhecimento de voz. Ele possui uma frase com o identificador *repete* que define o reconhecimento de voz "repita vídeo". O segundo define um *<mm-input>* com identificador *rec_voz*, que tem como conteúdo o arquivo *rec_voz.srgs*. Além disso esse *<mm-input>* possui uma âncora que indica a frase a ser reconhecida.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="repete">repita vídeo</rule>
</grammar>
```

```
<mm-input id="rec_voz" src="rec_voz.srgs">
  <mm-area label="repete"> <!-- referência id repete de rec_voz.srgs-->
</mm-input>
```

Para ilustrar o uso de interação por voz, o trecho de código a seguir apresenta uma nova versão (modificações em destaque) da aplicação da parte Conceitos 2.1 que reinicia um vídeo dada uma interação por seleção. Nessa versão, em vez de selecionar o ícone, o vídeo é reiniciado quando usuário falar "repita vídeo". A primeira diferença está no uso dos elementos *sinte_voz* e *rec_voz* para interação por voz, que utilizam os arquivos *sinte_voz.srgs* e *rec_voz.srgs* (apresentados acima).

Dois elementos *<mm-link>* do elemento *<script>* também foram modificados. O primeiro *<mm-link>* modificado define que quando o *video_principal* alcançar o trecho de *creditos* (300s), a frase da âncora *pergunta* é sintetizada e o reconhecedor *rec* é ativado (a partir desse momento, ele poderá reconhecer interações de voz). O segundo *<mm-link>* modificado define que quando for reconhecida a âncora *repete*, a *midia_principal* será reiniciada (terminada e iniciada).

```
<!DOCTYPE html>
<html>
<head><script src="multimodal.js"></script></head>
<body>
  <mm-scene id="scene">
    <video id="midia_principal" src="video.mp4"
      style="position: absolute; height 100%; width: 100%;">
      <mm-area id="creditos" begin="300s" end="360s">
    </video>
    <mm-media id="sinte_voz" src="sinte_voz.ssml">
      <mm-area label="pergunta"> <!-- referência id pergunta de sinte_voz.ssml-->
    </mm-media>
    <mm-input id="rec_voz" src="rec_voz.srgs">
      <mm-area label="repete"> <!-- referência id repete de rec_voz.srgs-->
    </mm-input>
    <mm-link>
      <mm-onBegin interface="scene">
      <mm-start interface="midia_principal">
    </mm-link>
    <mm-link>
      <mm-onBegin interface="midia_principal#creditos">
      <mm-start interface="sinte_voz">
      <mm-start interface="rec_voz">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz#repete">
      <mm-stop interface="rec_voz">
      <mm-stop interface="midia_principal">
      <mm-start interface="midia_principal">
    </mm-link>
  </mm-scene>
</body>
</html>
```

Tarefa 2.2

Considere as descrições sinte_voz_videos.ssml e rec_voz_videos.srgs, respectivamente, a seguir.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<speak xmlns="http://www.w3.org/2001/10/synthesis">
  <s id="repetir">fale o vídeo que deseja ver, centro ou praia?</s>
</speak>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="voz_centro">centro</rule>
  <rule id="voz_praia">praia</rule>
</grammar>
```

O código HTML a seguir é uma versão modificada da Tarefa 2.1 para permitir interações multimodais utilizando as duas descrições acima. Por favor, analise com atenção.

```
<html>
<head><script src="multimodal.js"></script></head>
<body>
  <mm-scene id="scene">
    <video id="video_inicial" src="video_inicial.mp4"
      style="position: absolute; height 100%; width: 100%;">
      <mm-area id="creditos" begin="300s" end="360s">
    </video>
      style="position: absolute; height 100%; width: 100%;">
    </video>
    <video id="video_praia" src="video_praia.mp4"
      style="position: absolute; height 100%; width: 100%;">
    </video>
    <mm-media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
      <mm-area label="pergunta">
    </mm-media>
    <mm-input id="rec_voz_videos" src="rec_voz_videos.srgs">
      <mm-area label="centro">
      <mm-area label="praia">
    </mm-input>
    <mm-link>
      <mm-onBegin interface="scene">
      <mm-start interface="video_inicial">
    </mm-link>
    <mm-link>
      <mm-onBegin interface="video_inicial#creditos">
      <mm-start interface="sinte_voz_videos#pergunta">
      <mm-start interface="rec_voz_videos">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz_videos#centro">
      <mm-stop interface="rec_voz_videos">
      <mm-stop interface="video_inicial">
      <mm-start interface="video_centro">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz_videos#praia">
      <mm-stop interface="rec_voz_videos">
      <mm-stop interface="video_inicial">
      <mm-start interface="video_praia">
    </mm-link>
  </mm-scene>
</body>
</html>
```

Qual é o novo comportamento da aplicação?

Conceitos 2.3

Um dos principais benefícios de interfaces multimodais é permitir que as interações dos usuários possam ser realizadas por diferentes modalidades.

O trecho de código a seguir ilustra o arquivo *rec_gestos.gml* que segue o formato GML (Gesture Markup Language) para reconhecimento de gestos de mão. Ele possui dois gestos com os identificadores *esquerda* e *direita*, que definem gestos de deslizar mão para esquerda e para direita, respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<GestureMarkupLanguage>
  <Gesture id="esquerda" type="swipe">
    ...
  </Gesture>
  <Gesture id="direta" type="swipe">
    ...
  </Gesture>
</GestureMarkupLanguage>
```

```
<mm-input id="rec_gesto" src="rec_gesto.gml">
  <mm-area label="esquerda"> <!-- referência id esquerda de rec_gesto.gml-->
</mm-input>
```

Na HTML estendida, a combinação de modalidades de interação pode ser feita utilizando uma condição composta. Uma condição composta é definida pelo elemento *<mm-compoundCondition>* e tem condições simples como filhos. A combinação entre as condições é definida pelo atributo *operator* utilizando um dos seguintes valores: 'or' quando apenas uma das condições é necessária; 'and' quando todas as condições são necessárias em qualquer ordem; e 'seq' quando todas as condições são necessárias e devem acontecer na sequência estabelecida.

Para ilustrar o uso de combinação de modalidades, o trecho de código a seguir apresenta uma nova versão (modificações em destaque) da aplicação em Conceitos 2.2, a qual reinicia um vídeo dada uma interação por voz. Nessa versão, o vídeo é reiniciado quando usuário falar "repita vídeo" ou (operador "or") quando fizer um gesto de deslizar mão

```
<!DOCTYPE html>
<html>
<head><script src="multimodal.js"></script></head>
<body>
  <mm-scene id="scene">
    <video id="midia_principal" src="video.mp4"
      style="position: absolute; height 100%; width: 100%;">
      <mm-area id="creditos" begin="300s" end="360s">
    </video>
    <mm-media id="sinte_voz" src="sinte_voz.ssml">
      <mm-area label="pergunta"> <!-- referência id pergunta de sinte_voz.ssml-->
    </mm-media>
    <mm-input id="rec_voz" src="rec_voz.srgs">
      <mm-area label="repete"> <!-- referência id repete de rec_voz.srgs-->
    </mm-input>
    <mm-input id="rec_gesto" src="rec_gesto.gml">
      <mm-area label="esquerda"> <!-- referência id esquerda de rec_gesto.gml-->
    </mm-input>
    <mm-link>
      <mm-onBegin interface="scene">
      <mm-start interface="midia_principal">
    </mm-link>
```

```
    <mm-link>
      <mm-onBegin interface="midia_principal#creditos">
      <mm-start interface="sinte_voz#pergunta">
      <mm-start interface="rec_voz">
      <mm-start interface="rec_gesto">
    </mm-link>
    <mm-link>
      <mm-compoundCondition operator="or">
        <mm-onRecognize interface="rec_voz#repete">
        <mm-onRecognize interface="rec_gesto#esquerda">
      </mm-compoundCondition>
      <mm-stop interface="rec_voz">
      <mm-stop interface="rec_gesto">
      <mm-stop interface="midia_principal">
      <mm-start interface="midia_principal">
    </mm-link>
  </mm-scene>
</body>
</html>
```

Tarefa 2.3

Agora pedimos que você edite o código HTML da Tarefa 2.2 (copiado a seguir) para que seja possível a interação por voz **ou** (operador OR) interação por gestos. Na interação por gesto, você pode referenciar o arquivo de descrição *rec_gestos.gml*, apresentado em Conceitos 2.3. Considere também que o gesto de deslizar a mão para esquerda indica centro e gesto de deslizar a mão para direita indica praia.

```
<html>
<head><script src="multimodal.js"></script></head>
<body>
 <mm-scene id="scene">
  <video id="video_inicial" src="video_inicial.mp4"
   style="position: absolute; height 100%; width: 100%;">
   <mm-area id="creditos" begin="300s" end="360s">
  </video>
  <video id="video_centro" src="video_centro.mp4"
   style="position: absolute; height 100%; width: 100%;">
  </video>
  <video id="video_praia" src="video_praia.mp4"
   style="position: absolute; height 100%; width: 100%;">
  </video>
  <mm-media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
   <mm-area label="pergunta">
  </mm-media>
  <mm-input id="rec_voz_videos" src="rec_voz_videos.srgs">
   <mm-area label="centro">
   <mm-area label="praia">
  </mm-input>
```

```
  <mm-link>
    <mm-onBegin interface="scene">
    <mm-start interface="video_inicial">
  </mm-link>
  <mm-link>
    <mm-onBegin interface="video_inicial#creditos">
    <mm-start interface="sinte_voz_videos#pergunta">
    <mm-start interface="rec_voz_videos">
  </mm-link>
  <mm-link>
    <mm-onRecognize interface="rec_voz_videos#centro">
    <mm-stop interface="rec_voz_videos">
    <mm-stop interface="video_inicial">
    <mm-start interface="video_centro">
  </mm-link>
  <mm-link>
    <mm-onRecognize interface="rec_voz_videos#praia">
    <mm-stop interface="rec_voz_videos">
    <mm-stop interface="video_inicial">
    <mm-start interface="video_praia">
  </mm-link>
 </mm-scene>
</body>
</html>
```

Conceitos 2.4

O conceito de Grupo de Usuários é definido **por um identificador, número máximo de participantes e quais dispositivos estes utilizam**. Na HTML estendida, um Grupo de Usuários é definido pelo elemento *<mm-userClass>*, o qual deve referenciar um arquivo SPARQL por meio de seu atributo *src*. O formato SPARQL é utilizado para definir como um determinado ususário é selecionado para participar ou não de um determinado grupo . O trecho de código a seguir apresenta o arquivo de descrição de usuários *gu_leap_microphone.sparql*, o qual que cada usuário do grupo deve ter um microfone e um leapmotion.

```
var sparql =
  `PREFIX foaf: <http://xmlns.com/foaf/0.1>
  PREFIX prf: <http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430>
  SELECT ?person
  WHERE {
    ?person prf:component ?component.
    ?component_prf: name ?name FILTER regex(?name, "Leap Motion")
    ?name FILTER regex(?name, "microfone")
  }`
var gu_leap_microphone = new UserGroup(sparql, 2)
```

Para ilustrar o uso de um Grupo de Usuários em HTML, o trecho de código a seguir apresenta uma uma nova versão (modificações em destaque) da aplicação definida em Conceitos 2.2, a qual reinicia um vídeo dada uma interação por voz. Nessa versão, o vídeo será reiniciado apenas quando o segundo usuário falar "repita vídeo". Esse grupo é definido com um máximo de 2 usuários e com os dispositivos descritos no arquivo *gu_leap_microphone.sparql* (apresentado acima).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
<head><script src="multimodal.js"></script></head>
<body>
  <mm-scene id="scene">
    <video id="midia_principal" src="video.mp4"
      style="position: absolute; height 100%; width: 100%;">
      <mm-area id="creditos" begin="300s" end="360s">
    </video>
    <mm-media id="sinte_voz" src="sinte_voz.ssml">
      <mm-area label="pergunta">  <!-- referência id sinte_voz de sinte_voz.ssml-->
    </mm-media>
    <mm-input id="rec_voz" src="rec_voz.srgs">
      <mm-area label="repete"> <!-- referência id repete de rec_voz.srgs-->
    </mm-input>
    <mm-userClass id="gu_leap_microphone" src="gu_leap_microphone.sparql" max="2">
    <mm-link>
      <mm-onBegin interface="scene">
      <mm-start interface="midia_principal">
    </mm-link>
    <mm-link>
      <mm-onBegin interface="midia_principal#creditos">
      <mm-start interface="sinte_voz">
      <mm-start interface="rec_voz">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz#repete" user_id="gu_leap_microphone(2)">
      <mm-stop interface="midia_principal">
      <mm-start interface="midia_principal">
    </mm-link>
  </mm-scene>
</body>
</html>
```

Tarefa 2.4

Agora pedimos que você edite novamente o trecho de código HTML da Tarefa 2.2 (copiado a seguir) para que apenas o segundo usuário,de um grupo de 3 usuários com microfone, possa realizar a interação por voz.

```
<html>
<head><script src="multimodal.js"></script></head>
<body>
 <mm-scene id="scene">
  <video id="video_inicial" src="video_inicial.mp4"
   style="position: absolute; height 100%; width: 100%;">
   <mm-area id="creditos" begin="300s" end="360s">
  </video>
  <video id="video_centro" src="video_centro.mp4"
   style="position: absolute; height 100%; width: 100%;">
  </video>
  <video id="video_praia" src="video_praia.mp4"
   style="position: absolute; height 100%; width: 100%;">
  </video>
  <mm-media id="sinte_voz_videos" src="sinte_voz_videos.ssml">
   <mm-area label="pergunta">
  </mm-media>
  <mm-input id="rec_voz_videos" src="rec_voz_videos.srgs">
   <mm-area label="centro">
   <mm-area label="praia">
  </mm-input>
```

```
    <mm-link>
      <mm-onBegin interface="scene">
      <mm-start interface="video_inicial">
    </mm-link>
    <mm-link>
      <mm-onBegin interface="video_inicial#creditos">
      <mm-start interface="sinte_voz_videos#pergunta">
      <mm-start interface="rec_voz_videos">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz_videos#centro">
      <mm-stop interface="rec_voz_videos">
      <mm-stop interface="video_inicial">
      <mm-start interface="video_centro">
    </mm-link>
    <mm-link>
      <mm-onRecognize interface="rec_voz_videos#praia">
      <mm-stop interface="rec_voz_videos">
      <mm-stop interface="video_inicial">
      <mm-start interface="video_praia">
    </mm-link>
  </mm-scene>
</body>
</html>
```

C.9
Page 7 for HTML participants

Por favor, opine sobre os pontos a seguir. Eles visam capturar evidências de como os conceitos apresentados na seção anterior (Mídia, Reconhecedor, Relacionamento e Grupo de Usuários) estão instanciados na HTML para suportar interações multimodais.

A HTML estendida permite **realizar rapidamente** o desenvolvimento de aplicações multimodais.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente

A HTML estendida permite o desenvolvimento de aplicações multimodais **com qualidade**.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente

De modo geral, a HTML estendida **é útil** para o desenvolvimento de aplicações multimodais.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente

A HTML estendida **é simples e entendível**.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente

A HTML estendida **é fácil de aprender**.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente

De modo geral, a HTML estendida **é fácil de utilizar**.

- Discordo fortemente
- Discordo bastante
- Discordo um pouco
- Não concordo nem discordo
- Concordo um pouco
- Concordo
- Concordo fortemente