UNIVERSIDADE FEDERAL FLUMINENSE

FÁBIO BARRETO

Uma Proposta de Extensão do Middleware Ginga-NCL para Interação Multimodal e Suporte Multiusuário em Ambientes Hipermídia

NITERÓI

2021

UNIVERSIDADE FEDERAL FLUMINENSE

FÁBIO BARRETO

Uma Proposta de Extensão do Middleware Ginga-NCL para Interação Multimodal e Suporte Multiusuário em Ambientes Hipermídia

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação.

Orientador:
Débora Christina Muchaluat Saade, D.Sc.

NITERÓI

2021

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

B273p Barreto, Fábio
Uma Proposta de Extensão do Middleware Ginga-NCL para Interação Multimodal e Suporte Multiusuário em Ambientes Hipermídia / Fábio Barreto ; Débora Christina Muchaluat Saade, orientadora. Niterói, 2021.
122 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2021.

DOI: http://dx.doi.org/10.22409/PGC.2021.d.02115812751

1. Sistemas Multimídia. 2. Multimídia Interativa. 3. TV Digital. 4. NCL. 5. Produção intelectual. I. Muchaluat Saade, Débora Christina, orientadora. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

FÁBIO BARRETO

Uma Proposta de Extensão do Middleware Ginga-NCL
para Interação Multimodal e Suporte Multiusuário em Ambientes Hipermídia

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação.

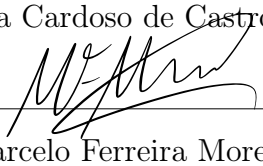
Aprovada em setembro de 2021.

BANCA EXAMINADORA

Profa. Débora Christina Muchaluat Saade - Orientadora, UFF

Prof. Célio Vinicius Neves de Albuquerque, UFF

Profa. Luciana Cardoso de Castro Salgado, UFF



Prof. Marcelo Ferreira Moreno, UFJF

Profa. Maria da Graça Campos Pimentel, USP

Niterói
2021

Dedico este trabalho a minha esposa, Marina, e aos meus filhos, Artur e Daniel, que representam a imensidão do meu amor me impulsionando, sempre, na minha trajetória.

Agradecimentos

Agradeço a Deus por iluminar meu espírito durante toda minha vida. Agradeço a minha mãe por me incentivar sempre e por suas orações.

A minha mãe de coração Imaculada pelas orações e torcida. Ao meu pai de coração Mauro pela preocupação e apoio.

Agradeço muito a minha orientadora Débora, seria injusto chamá-la apenas de orientadora. Débora transforma vidas, seu jeito radiante de conduzir tudo que faz é comovente e impossível não ser afetado.

Não poderia deixar de agradecer aos meus amigos Marina Ivanov, Eyre e Raphael Abreu pela grande ajuda e incentivo desde o início, principalmente nas tempestades de ideias e nos momentos de desânimo. Ao meu amigo do trabalho Mário João sempre com palavras de estímulo.

À minha família, agradeço a compreensão pelas minhas ausências e minhas desculpas nos momentos de desânimo e irritabilidade em que não consegui lidar de maneira amável.

Resumo

Avanços nas tecnologias de reconhecimento (por exemplo, reconhecimento de voz ou gesto) permitem novas interfaces de usuário para sistemas multimídia. É possível processar dois ou mais modos combinados de interação do usuário capturados por dispositivos de entrada ou sensores em tais sistemas. Além disso, os visualizadores podem usar diferentes dispositivos de entrada adaptados às suas necessidades especiais ou intenção particular. Outra característica interessante desses sistemas é a capacidade de perceber e identificar diferentes usuários que estão participando da experiência multimídia. Existem tecnologias capazes de relacionar a interatividade do usuário de maneira única. No entanto, as plataformas padrão de TV Digital Interativa(TVDI) não suportam totalmente interações multimodais nativamente muito menos a interação multiusuário. Portanto, este trabalho propõe uma extensão ao middleware TVDI brasileiro Ginga-NCL para fornecer interação multimodal e multiusuário. A linguagem Nested Context Language (NCL) é estendida (denominada NCL 4.0) para suportar novos eventos de interação e múltiplos usuários, permitindo-lhes identificar qual usuário interagiu com um aplicativo de TVDI. Como prova de conceito, foram desenvolvidos dois novos módulos Ginga-NCL que aderem à versão estendida proposta, um que suporta interação por voz e outro que suporta interação por olhar. Foram desenvolvidos vários aplicativos de TVDI para avaliar a proposta e foi confirmado que a versão estendida causa overhead muito baixo no processamento de eventos Ginga-NCL. Finalmente, esta tese também apresenta uma comparação entre a versão atual de NCL e a proposta de NCL 4.0 considerando o desempenho e o número de linhas de código da aplicação.

Palavras-chave: NCM, NCL, Ginga-NCL, NCL 4.0, Modelo Conceitual, Autoria Multimídia, Linguagem de Autoria, Interação Multimodal, Interação multiusuário, Aplicações multiusuário.

Abstract

Advances in recognition technologies (e.g., speech or gesture recognition) enable new user interfaces for multimedia systems. It is possible to process two or more combined modes of user interaction captured by input devices or sensors in such systems. Moreover, viewers can use different input devices tailored to their special needs or particular intention. Another interesting feature in these systems is perceiving and identifying different users who are participating in the multimedia experience. There are technologies capable of relating user interactivity uniquely. However, Digital TV (DTV) standard platforms do not fully support multimodal and multiuser interactions natively. Therefore, this work proposes an extension to the Brazilian Ginga-NCL DTV middleware to provide multimodal interaction. The Nested Context Language (NCL) is extended (called NCL 4.0) to support new interaction events and multiple users, allowing them to identify which user has interacted with a DTV application. As a proof of concept, we developed two new Ginga-NCL modules that adhere to our extended version, one that supports voice interaction and another that supports gaze interaction. We developed several DTV applications to evaluate our proposal and confirmed that our extended version cause low overhead on Ginga-NCL event processing. Finally, this thesis also presents a comparison between the current NCL version and our NCL 4.0 proposal considering performance and number of application code lines.

Keywords: NCM, NCL, Ginga-NCL, NCL 4.0, Conceptual Model, Multimedia Authoring, Authoring Language, Multimodal Interaction, Multiuser Interaction, Multiuser Applications.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AHM	:	Amsterdam Hypermedia Model;
CIDL	:	Control Information Description Language;
GUI	:	Graphical User Interface;
IIDL	:	Interaction Interface Description Language;
IoT	:	Internet Of Things;
KMS	:	Knowledge Management System;
MQTT	:	Message Queuing Telemetry Transport;
MUI	:	Multimodal User Interface;
MultiSEM	:	Multimedia Sensory Effect Model;
NCL	:	Nested Context Language;
NCM	:	Nested Context Model;
PROMIS	:	Problem Oriented Medical Information System;
RDF	:	Resource Description Framework;
RoSE	:	Representation of Sensory Effects;
SBTVD	:	Sistema Brasileiro de TV Digital;
SEDL	:	Sensory Effect Description Language;
SEM	:	Sensory Effect Metadata;
SEVino	:	Sensory Effect Video Annotation;
SMIL	:	Synchronized Multimedia Integration Language;
SMURF	:	Sensible Media aUthoRing Factory;
SRGS	:	Speech Recognition Grammar Specification;
TVD	:	TV Digital;
TVDI	:	TV Digital Interativa.

Sumário

Capítulo 1

Introdução

A difusão de informações em dispositivos digitais é feita por meio de conteúdos audiovisuais com objetos de mídia, que são classificados em: mídias discretas e mídias contínuas. Mídias discretas são compostas por itens de informação independentes do tempo e com sua dimensão unicamente espacial, como por exemplo textos, imagens e gráficos. Já as mídias contínuas são caracterizadas por sua dependência temporal entre os itens de informação, ou seja, o tempo faz parte da semântica da informação, como por exemplo áudios, vídeos e animações [34].

Sistemas hipermídia são subdivididos em três componentes principais: armazenamento, especificação (autoria) e execução (formatação) [69]. As mídias digitais, sendo elas contínuas e/ou discretas, são orquestradas por esses sistemas usando os três componentes. O componente de armazenamento define como os conteúdos das mídias serão armazenados. Já o componente de especificação diz respeito à descrição formalizada de como a aplicação hipermídia será executada. Ao se descrever essa aplicação é produzido um documento hipermídia. A execução trabalha para que as especificações feitas pelo componente de autoria possa ser executada, como por exemplo, iniciar os aplicativos necessários para a renderização das mídias, captura de eventos que ocorrem no sistema operacional do *middleware*, sincronização temporal das apresentações, etc. Além disso, as aplicações hipermídia podem permitir interação do usuário, ou seja, permitem definir apresentações interativas utilizando múltiplas mídias.

Aplicações hipermídia estão disponíveis em várias plataformas como *smartphones*, computadores, *tablets* e TVs digitais. Essas aplicações são desenvolvidas por diversos tipos de profissionais que variam desde designers gráficos, editores audiovisuais e autores que trabalham na construção de documentos hipermídia. Enquanto designers trabalham buscando definir gráficos para comunicação visual, os editores audiovisuais com a pro-

dução de determinados itens de mídia específicos da aplicação, o autor de documentos hipermídia trabalha com a integração dos módulos de modo que a aplicação atenda às especificações desejadas [69].

Os documentos hipermídia podem ser criados por meio de uma linguagem de autoria hipermídia, que permite especificar tanto propriedades temporais quanto espaciais das mídias que participam da aplicação. Essas características podem se apresentar de diversas maneiras tais como: o relacionamento entre duas ou mais mídias; o tempo de duração de cada uma; a posição da tela em que cada uma é apresentada; as interações com o usuário final da aplicação, a adaptação de conteúdo e de leiaute, entre outras [69].

Diferentes linguagens hipermídia são usadas para especificar comportamentos temporais e espaciais de um documento hipermídia. Dentre as principais, estão as linguagens de autoria HTML5 (*HyperText Markup Language*) [76], SMIL (*Synchronized Multimedia Integration Language*) [7] e NCL (*Nested Context Language*) [40]. Esta última é a linguagem adotada como padrão para desenvolvimento de serviços interativos no Sistema Brasileiro de TV Digital [2] e foco deste trabalho.

No ambiente de TV Digital, considerado um sistema hipermídia, o mesmo conteúdo é transmitido para vários telespectadores que estão sintonizados em um canal. Esses telespectadores podem ter opiniões distintas sobre interação com o conteúdo apresentado, por exemplo, alguns gostam de interagir ativamente com o conteúdo, outros preferem apenas assistir sem nenhuma interação, enquanto outros simplesmente não podem interagir por possuírem necessidades especiais. Neste sentido, é interessante que o ambiente de TV ofereça diferentes possibilidades de interação aos usuários, além de poder reagir de maneira diferenciada de acordo com o usuário responsável pela interação. Assim sendo, é importante fornecer novos mecanismos nas linguagens de autoria para que os autores de conteúdo multimídia possam desenvolver aplicações que considerem novas modalidades de interação. Para a aplicação reagir de forma diferente a cada usuário, será necessário armazenar informações sobre os usuários de forma individualizada.

Considerando o avanço dos dispositivos inteligentes e sensores conectados à IoT (*Internet of Things*), a incorporação desses dispositivos em plataformas de mídia e entretenimento é uma forma interessante para expandir as possibilidades de interação do usuário. Por exemplo, alguns televisores estão sendo desenvolvidos com interface de voz para acessar funcionalidades básicas, como pesquisa por conteúdo e controle de execução das mídias[1].

[1]https://assistant.google.com/platforms/tv/

Integração de novos tipos de interação em sistemas multimídia, além das modalidades tradicionais de interação utilizando mouse e teclado, é um tema abordado em diferentes propostas na literatura [23, 29]. De acordo com Lima et al. [23], o uso da linguagem natural juntamente com o gesto pode superar as limitações da interação de apenas uma modalidade. Isto porque a combinação de fala e gesto fornece um comportamento comunicativo altamente eficiente para interagir com aplicativos em uma experiência mais transparente que as interfaces tradicionais. Assim, sistemas de interação multimodal representam uma interessante ferramenta pra prover uma melhor qualidade na interação à medida que se possibilita o uso de modalidades mais naturais para o usuário. Porém, há de se pensar sobre as características desse sistema de interação que pode ser classificado de várias formas, dependendo se há fusão dos dados de mais de uma modalidade e se o uso de modalidades é em sequencial ou em paralelo. Em Turk et al. [72], há uma discussão aprofundada sobre a classificação de sistemas multimodais. Em um sistema multimodal exclusivo, as modalidades são usadas sequencialmente e estão disponíveis separadamente, mas não integradas pelo sistema. Em um sistema multimodal alternativo, as modalidades são usadas sequencialmente, mas são integradas em algum grau. Em um sistema multimodal concorrente, as informações modais estão disponíveis em paralelo, mas separadamente (não integradas). Finalmente, em um sistema multimodal sinérgico, os modos estão disponíveis em paralelo e totalmente integrados [72].

Em geral, os sistemas multimodais disponíveis comercialmente não incluem o processamento paralelo de múltiplos modos de entrada e não utilizam uma fusão destes dados, mas, em vez disso, processam apenas uma alternativa de modo por vez [26]. Neste trabalho, também consideramos as modalidades sendo disparadas uma da cada vez, sendo assim chamado de sistema multimodal exclusivo, de acordo com a classificação apresentada em Turk et al. [72].

Além de múltiplas modalidades de interação, aplicações multiusuário têm levado a diversas evoluções na forma de desenvolvimento de aplicações multimídia. Em ambientes interativos, a adição de múltiplos usuários implica a necessidade de identificação de tais usuários para poderem interagir com o sistema. Assim como conter estruturas que armazenem informações desses usuários. Para permitir autoria dessas aplicações, a linguagem deve permitir que o autor identifique cada usuário individualmente. Porém, identificar um usuário não é simples do ponto de vista de autoria. Visto que o código gerado seria altamente acoplado ao identificador do usuário do sistema. Outro problema é que devem ser informados especificamente quais usuários e quantos usuários existem. Por outro lado, perfis de usuários podem ser definidos mais facilmente pelo autor como proposto no

trabalho de Guedes et al. [29].

Como foi apresentado anteriormente, a linguagem de autoria NCL (*Nested Context Language*) [65] faz parte dos padrões do sistema brasileiro de TV Digital e é interpretada pelo *middleware* Ginga-NCL [2]. A linguagem NCL, em especial, oferece suporte a interatividade e é utilizada não somente para especificação de conteúdo interativo para TV Digital [67], mas também para Sistemas IPTV (*Internet Protocol TeleVision*) de acordo com a recomendação ITU-T H.761 [40]. Em televisores, a principal modalidade de interação com o conteúdo é através do controle remoto, no qual um e somente um usuário de cada vez pode navegar por menus ou selecionar botões virtuais pressionando botões do controle físico.

A adição de novas modalidades de interação abre várias possibilidades para criação de aplicações capazes de atender um grupo de usuários e cenários que até então seriam impossíveis de serem criados, tais como usuários com necessidades especiais incapazes de manusear o controle, ou ambientes em que os usuários estão com as mãos ocupadas e precisam interagir de outra forma para direcionar a execução da aplicação. No setor de saúde, podemos imaginar aplicações multimídia sendo utilizadas para terapias alternativas nas quais o indivíduo que está tendo a experiência com a aplicação possa utilizar outros modos de interação que sejam mais confortáveis para ele. Como exemplo, Faria et al [24] propõem um novo exercício cognitivo chamado, Memo-VR, que usa a tecnologia de Realidade Virtual (do inglês *Virtual Reality* - VR) combinada a técnicas de interação como o escaneamento e virtualização das mãos do usuário, em uma versão do clássico jogo da memória. Na área de educação, pode-se utilizar modalidades de interação mais próximas dos alunos a fim de apresentar o conteúdo planejado. Fua e Zhoub em [25] estudam a interação docente do curso aprendizagem *maker* no cenário de ensino à distância, principalmente observando a eficácia da interação multimodal e as mudanças nas características corporais (linguagem, gestos e *feedback*). Os resultados mostram que a interação multimodal na cena remota é benéfica para melhorar a eficiência do aprendizado conforme apresentado pelos autores.

A TV tem sido o meio dominante de consumo de mídia audiovisual em casa há décadas, apoiando experiências compartilhadas e atraindo o olhar e a atenção das pessoas próximas. No entanto, nos últimos anos, esse domínio foi corroído pelo advento da "exibição múltipla", em que os espectadores utilizam simultaneamente duas ou mais telas ou dispositivos. Por exemplo, na Austrália, 74% da população com conectividade com a Internet tem duas telas (o que significa que eles usaram duas telas simultaneamente, por

exemplo, usando uma TV e um telefone juntos). Em comparação, 26% fizeram a triagem tripla (o que significa que normalmente utilizaram uma combinação de TV, telefone e *tablet*/*laptop*). Essa transição para o uso de várias telas ocorreu porque a tecnologia e a interface da TV não conseguiram acompanhar as demandas dos usuários [51]. Isso não é diferente no Brasil, aplicações multimídia para TV digital aberta no sistema brasileiro não possuem interação multimodal por meio do equipamentos presentes em todas TVs. O Ginga-NCL, *middleware* que interpreta e executa os documentos hipermídia, oferece como principal modalidade de interação, o controle remoto da TV. Pessoas com dificuldades de operar o teclado, ou que simplesmente não preferem esta modalidade, têm poucas opções já que outras modalidades como múltiplas telas, por exemplo, não são tão exploradas pelos autores.

Já a interface do usuário com o computador permite manipular diretamente as representações audiovisuais por meio de ações de usuários. Chamadas de GUI (*Graphical User Interface*), tais interfaces seguem o paradigma WIMP (Windows, Ícones, Menus e Dispositivos apontadores) [8, 31, 63]. Pesquisas em interação humano-computador, no entanto, vão além do WIMP e propõem outros tipos de interfaces de usuário, interações multimodais, geralmente chamadas de Post-WIMP. Essas vêm ganhado um espaço cada vez maior na interface homem-máquina [41, 77]. Em particular, os avanços nas tecnologias de reconhecimento, como reconhecimento de fala ou gesto, deram origem a um tipo de interface de usuário pós-WIMP denominada MUI (*Multimodal User Interface*, interface de usuário multimodal). Segundo [47, 72], as MUIs processam duas ou mais modalidades combinadas de entrada do usuário (por exemplo, fala, toque, gesto, olhar e movimentos da cabeça e do corpo) de maneira coordenada com as modalidades de saída. Uma modalidade de entrada corresponde a uma informação gerada pelo usuário capturada por dispositivos de entrada (por exemplo, microfone) ou sensores (por exemplo, sensor de movimento). Uma modalidade de saída corresponde a um estímulo para os sentidos humanos (audição, olfato, tato, paladar ou visão) usando dispositivos audiovisuais ou atuadores (por exemplo, emissor de cheiro). Desta forma as interações podem ser estabelecidas utilizando outros dispositivos capazes de capturar formas diferentes de comunicação com a aplicação como, por exemplo, reconhecimento de voz, gestos e face atendendo inclusive a vários tipos de necessidades especiais.

Aplicações multimídia são geralmente definidas com uma linguagem específica, chamada de *linguagem de autoria multimídia*. Tais linguagens concentram-se na definição das mídias que farão parte da aplicação, a forma como serão apresentadas e sua sincronização ao longo do tempo [16, 35]. Um subconjunto dessas linguagens, geralmente

baseado em XML (eXtensible Markup Language), usa uma abordagem declarativa fornecendo construções de alto nível de abstração para definição de aplicações multimídia. Um dos princípios que norteiam tais linguagens é proporcionar uma clara separação entre a descrição da aplicação e sua implementação, *i.e.*, sua execução [35]. Assim as evoluções futuras da forma de como uma aplicação é apresentada não exigem uma redefinição, ou alteração, de toda a base das aplicações previamente especificadas. Um outro fator positivo das linguagens de autoria multimídia declarativas é que elas permitem maior facilidade para criação de aplicações por autores que não são programadores [64].

O aumento do número de dispositivos inteligentes e sensores conectados à IoT tem o potencial de mudar a forma como os consumidores interagem com a tecnologia em rede, incluindo plataformas de mídia, entretenimento e salas terapêuticas. Isso representa uma oportunidade interessante para sistemas multimídia a fim de expandir possibilidades de interação com o usuário. Neste cenário, é possível criar aplicações responsivas e interativas, redefinindo o nível de interação entre os ambientes das aplicações e seus usuários [41, 55], criando novas demandas. Assim, torna-se interessante prover para o autor de conteúdo novas construções nas linguagens de autoria declarativas para que ele possa desenvolver aplicações multimídia. Portanto considera-se que a construção de uma linguagem declarativa capaz de expressar interações multiusuário e multimodais possa avançar o estado da arte.

1.1 Definição do Problema

Esta tese de doutorado define as seguintes questões de pesquisa a serem estudadas e solucionadas por este trabalho.

Problema 1. *Como as aplicações multimídia declarativas para TV digital podem oferecer diferentes modalidades de interação, além de seleção usando o controle remoto, considerando diferentes perfis de usuários e suas necessidades?*

Problema 2. *Como as aplicações multimídia declarativas para TV digital podem identificar diferentes usuários que interagem com a TV e reagir de forma personalizada conforme quem tenha interagido?*

Nas plataformas de TV atuais, não existe um arquitetura nativa capaz de gerenciar diversos usuários e suas propriedades de forma a serem utilizados nas aplicações multimídia. Quando há vários usuários participando de uma experiência pela TV, é desejável que

a aplicação possa responder somente a um grupo específico de usuários com determinadas propriedades, ou seja, um determinado perfil. Para isso, o *middleware* precisa identificar os usuários e suas interações configurando um ambiente multiusuário.

Considerando uma interação multiusuário, um fator importante é a contextualização das informações de cada usuário que participa da experiência. Contexto esse que pode ser estendido para o ambiente em que se executa a aplicação. Alguma característica do ambiente pode ser usada para direcionar a aplicação de acordo com as preferências ou limitações do usuário. Para isso essas informações devem estar disponíveis no momento da apresentação do documento. Atualmente o *middleware* Ginga não possui a funcionalidade de armazenar informações dos usuários de maneira individualizada. Desta forma, é interessante a criação de grupos de dados que tenham significados relacionados. Por exemplo, em aplicações na área de saúde, torna-se interessante agrupar informações dos pacientes em um contexto e as do terapeuta em outro, já que há mais de um usuário de perfis diferentes. Possibilitar carregar informações sobre o perfil dos usuários que estão participando da experiência é uma funcionalidade desejável para aplicações multimídia declarativas.

1.2 Objetivos

Esta tese tem o seguinte objetivo geral:

Contribuir para autoria declarativa de aplicações hipermídia com novas facilidades que permitam vários usuários interagirem com a aplicação, de maneira individualizada, e forneçam suporte à interação multimodal em um ambiente multimídia.

Como objetivos específicos, este trabalho propõe uma extensão ao modelo NCM (*Nested Context Model*) [66] e à linguagem de autoria NCL, baseada no modelo, a fim de possibilitar a descrição de novas aplicações multimídia, com suporte a multiusuário e interação multimodal. Com a extensão proposta neste trabalho, a linguagem NCL tem potencial para ser usada para a criação de aplicações que possam ser executadas em vários ambientes multimídia interativos, como por exemplo museus interativos ou salas de terapia multimídia. NCM e NCL foram escolhidos como modelo e linguagem base para este trabalho, pois NCM utiliza um modelo de sincronização baseado em eventos, sendo possível contemplar eventos assíncronos como por exemplo de interação com usuário. Além disso, o modelo permite extensão do conjunto de eventos de interação e o armazenamento de variáveis do documento. A linguagem NCL, além de implementar as entidades oferecidas

pelo NCM, segue o paradigma declarativo, cuja especificação de documentos hipermídia se torna mais acessível para autores não programadores.

A extensão da linguagem é feita em uma nova versão 4.0 de NCL, propondo novos módulos de linguagem e alterando vários existentes. A funcionalidade de interação multimodal é proposta estendendo os tipos de eventos NCM e NCL. A implementação de suporte multiusuário é proposta com a criação de uma nova área funcional *Users*, um novo módulo chamado *User*, uma extensão do módulo *CausalConnectorFunctionality* e uma extensão do módulo *media*.

A proposta é avaliada de forma quantitativa. A proposta foi implementada no *middleware* Ginga-NCL, propondo uma arquitetura que permite a integração de diferentes dispositivos de interação ao *middleware* em um ambiente multiusuário. O desempenho da implementação é avaliado para interação por voz, sendo comparado à versão padrão do Ginga-NCL [3] com uso de código procedural Lua. Além do reconhecimento de voz, também foi implementado o módulo de interação por meio da fixação do olhar, reconhecimento de expressões faciais e gestos. Foram criadas aplicações de teste usando dois modos de interação complementares, como uma prova de conceito e para avaliar o novo sistema multimodal. Além disso, vários testes foram conduzidos para avaliar a capacidade de o *middleware* tratar eventos de interação em cenários distintos e para avaliar como os novos módulos impactam o *middleware*, medindo o atraso adicionado ao integrar os novos modos de interação. Para avaliar a arquitetura multiusuário, vários experimentos foram realizados medindo o tempo de verificação dos usuários de um determinado perfil especificado na aplicação NCL, além da medição do atraso gasto na criação de *links* de interação individualizados para cada usuário.

1.3 Contribuições da Tese

Esta tese avança o estado da arte em modelos conceituais e linguagens de autoria declarativa, pois propõe a utilização de novos tipos de evento para representar interação multimodal e suporte multiusuário. A proposta foi implementada no modelo NCM e na linguagem NCL para que possam especificar vários tipos de interação, considerando diferentes usuários e perfis. Foi especificada a versão da linguagem NCL 4.0, com uso de *XML Schema* para definição das extensões propostas. NCL 4.0 também inclui a especificação de efeitos sensoriais, especificados em [44]. Apesar de efeitos sensoriais não serem foco desta tese, eles podem ser usados em conjunto com suporte multimodal e multiusuário

proposto por esta tese.

Outra contribuição importante é a modificação do *middleware* Ginga-NCL para prover as novas funcionalidades de NCL, viabilizando o tratamento de várias modalidades de interação com o usuário. A arquitetura do Ginga-NCL foi estendida, através de um novo módulo chamado *InteractionManager*, que gerencia as várias modalidades de interação utilizadas pela aplicação multimídia.

A possibilidade de permitir a participação de vários usuários na experiência multimídia também é uma contribuição deste trabalho. O resultado disso foi a especificação da nova área funcional de NCL chamada *User* e do módulo *User* contendo os elementos *userAgent* e *userProfile* no *schema* NCL. Também foi implementada a modificação do Ginga-NCL para que o *middleware* possa validar e tomar ações diferentes quando a interação de uma determinada modalidade vier identificada com o usuário que interagiu. Para permitir manipular as propriedades de usuários em elos NCL, foi criado um novo tipo para o elemento *<media>* tanto no esquema quanto na implementação do *middleware* alterando o módulo *<Media>*. Esse tipo é o *userSettingsNode*. No modelo NCM, ele foi especificado como a entidade *UserSettingsNode* que estende a entidade *SettingsNode*. Isso foi feito para possibilitar às aplicações multimídia armazenarem informações dos usuários em contextos diferentes e em nós diferentes já que nessa nova versão é possível existir várias instâncias de *UserSettingsNode* diferente da versão 3.0 de NCL que permite somente uma instância do nó de informações de contexto por aplicação. Foi implementado um parser para carregar as informações dos usuários nesses nós de conteúdo do tipo *UserSettingsNode*. Como prova de conceito, o parser implementado carrega informações armazenadas em um arquivo XML de perfil do usuário, seguindo o padrão MPEG-21 [39].

A proposta de NCL 4.0 foi submetida à chamada de propostas para a TV 3.0 – *TV 3.0 Call for Proposals* – do Fórum do Sistema Brasileiro de TV Digital (SBTVD)[2] como extensão do *middleware* Ginga para a futura geração do sistema de TV digital no país. O Fórum SBTVD está avaliando as propostas submetidas durante o segundo semestre de 2021.

1.4 Estrutura da Tese

O restante do texto está estruturado da seguinte forma. O Capítulo 2 comenta os principais conceitos de ambientes hipermídia, apresentando várias abordagens desenvolvidas no

[2]https://forumsbtvd.org.br/tv3_0/

decorrer da história e apresentando e descrevendo os diversos modelos propostos, incluindo o modelo NCM, que foi detalhado descrevendo suas entidades e sua expressividade. Esse capítulo também apresenta os módulos e os principais elementos da linguagem NCL com suas funcionalidades, além de exemplificar aplicações NCL.

O Capítulo 3 contém os trabalhos relacionados a esta tese assim como o posicionamento deste trabalho frente aos demais encontrados na literatura.

No Capítulo 4, é apresentada a proposta de extensão ao modelo NCM e à linguagem NCL. O capítulo apresenta as entidades alteradas ou criadas para atender a cada uma das funcionalidades apresentadas neste trabalho considerando interação multimodal e multiusuário. Todos os módulos e elementos de NCL 4.0 alterados ou criados foram detalhados no capítulo.

O Capítulo 5 apresenta a implementação da proposta feita no Ginga-NCL, mostrando inicialmente a arquitetura idealizada. Em seguida, o capítulo comenta os detalhes de implementação da arquitetura na nova versão do *middleware* Ginga-NCL.

O Capítulo 6 apresenta uma avaliação das propostas desta tese. O capítulo apresenta as três etapas de avaliação da proposta de multimodalidade: a comparação usando um aplicação NCL feita para o Ginga atual e uma aplicação para o Ginga estendido; teste de carga da capacidade de captura de eventos multimodais; teste de carga para avaliar se o tratamento dos eventos multimodais afeta a execução de outras ações do Ginga estendido. Além disso, o capítulo apresenta uma avaliação da proposta de multiusuário em duas partes, a primeira com uma medição do tempo gasto para carregar e avaliar o perfil do usuário em uma aplicação no Ginga-NCL estendido e a segunda mede o tempo gasto com a criação dos links personalizados por usuário.

O Capítulo 7 apresenta vários casos de uso de NCL 4.0 em TV Digital em vários cenários multimodais e multiusuário.

Por fim, o Capítulo 8 conclui o trabalho, apresentando as conclusões, respondendo às questões de pesquisa desta tese, realçando suas contribuições e apontando os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

A partir da primeira metade do século XX, o progresso de sistemas computacionais trouxe também uma grande quantidade de documentos digitais armazenados nestes sistemas. Esta quantidade pode dificultar o entendimento do conteúdo, à medida que a navegabilidade entre os documentos se torna naturalmente difícil. Portanto, a disponibilidade do conteúdo de forma mais clara e interativa se faz necessária. Além disso, o autor do documento deve se certificar que seu produto terá seu entendimento respeitado seguindo uma semântica proposta. Desta forma o advento dos documentos hipertexto veio suprir tal necessidade. Com os hipertextos, o autor consegue disponibilizar para o leitor referências de parte do texto para um detalhamento maior quando necessário, sem que ele precise abandonar o texto principal. Então um documento representado em forma de hipertexto é composto de itens de texto que estão interligados para que possibilite uma navegabilidade mais assertiva, como apresentado pelo Halasz et al. [33].

A abordagem de *links* entre textos já vem sendo trabalhada há muito tempo. McCracken et al. [50] em 1984 apresentaram o ZOG, um sistema de seleção de menus onde as opções faziam parte de uma grande rede para prover a comunicação, ou seja, disponibilizar a informação através da máquina. A filosofia por trás deste estilo de comunicação foi inicialmente desenvolvida por Gilroy et al. [27] com PROMIS (*Problem Oriented Medical Information System*) . ZOG disponibilizou um conjunto de quadros ligados, cada quadro é composto de texto com um menu de opções que permite navegar entre outros quadros. O KMS (*Knowledge Management System*) desenvolvido por Akscyn et al. [4], Sistema de Gestão do Conhecimento, sucessor de ZOG, cria um ambiente de software de uso geral que ajude uma organização a gerenciar seu conhecimento. Um banco de dados KMS consiste em um conjunto de áreas de trabalho interligadas, do tamanho da tela. Essas áreas trabalho, também chamadas de *frames*, podem conter qualquer disposição de texto, gráficos

e itens de imagem. Cada item de texto dentro de um *frame* pode ser vinculado a qualquer outro *frame*. Assim como no ZOG, os itens de texto também podem ativar programas. Esses programas podem variar desde operações KMS atômicas a longas animações KMS (escritas na KMS *Action Language*), além de programas convencionais que normalmente são executados no sistema operacional. A Figura 2.1 mostra um exemplo de um *frame* no sistema KMS.

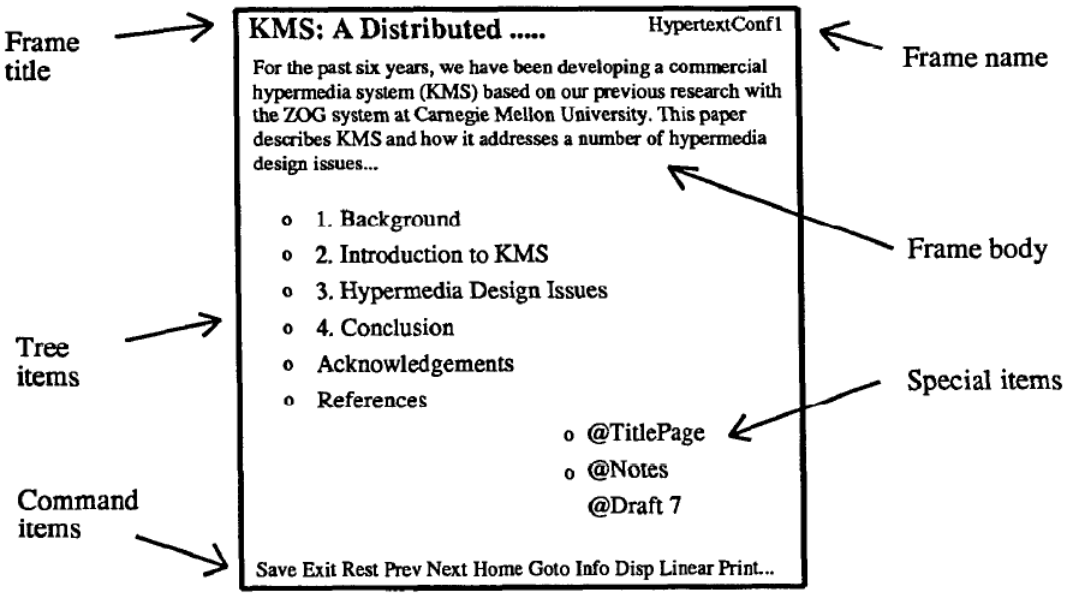


Figura 2.1: Exemplo de *frame* no sitema KMS [4]

Todos os *frames* tem nomes e esse nome consiste em duas partes: uma parte alfabética e uma parte numérica. A parte alfabética é o nome do conjunto de *frames* ao qual pertence. O conjunto esta relacionado por um tópico específico, conforme definido pelo autor documento. Os autores têm liberdade para criar um novo conjunto de *frames* sempre que criam um novo *frame*. O *frame* é composto de mais cinco objetos: título, corpo, itens especiais, links e itens de comando. O título (*Frame title*) é utilizado nos *links* quando o *frame* é referenciado. O corpo (*Frame body*) onde esta conteúdo textual que se deseja exibir. Os itens especiais (*Special items*) são utilizados para outras informações tais com comentários, notas ou formatação das palavras. Os *links* (*Tree items*) são referências para outros *frames*. E finalmente os itens de comandos (*Command items*) são utilizados para navegabilidade entre os *frames*.

Em 1987, Campbell e Goodman apresentam o HAM [18], um servidor baseado em transações para um sistema de armazenamento de hipertexto. Em vez de apresentar HAM como modelo de referência, os autores apresentaram como uma máquina abstrata. O servidor é projetado para manipular vários usuários em um ambiente de rede. O

sistema de armazenamento consiste em uma coleção de contextos, nós, elos e atributos que compõem um grafo de hipertexto. Desta forma, suas propriedades fornecem uma maneira de comparar recursos de diferentes sistemas de hipertexto. O modelo de armazenamento HAM aumentou a abrangência do conceito de *frame* do KMS com a sua maneira de armazenamento. Um grafo é o objeto HAM de nível mais alto. Ele normalmente contém todas as informações relacionadas a um tópico geral, como as informações de um projeto de software. Um gráfico contém um ou mais contextos. Os contextos podem dar suporte a configuração, histórico de versão, etc. Cada contexto tem um contexto pai e zero ou mais contextos filhos. Quando um grafo é criado, um contexto raiz inicia a árvore. O contexto que contém zero ou mais nós e elos não depende de informações contidas em seu contexto pai, resultando em uma abordagem mais organizada. Um nó contém dados do tipo texto ou códigos binários. Um elo define o relacionamento entre os nós. Enquanto o KMS agrupa informações relacionadas através de *frames* com parte dos nomes iguais, o HAM possui o conceito de contexto representado por uma entidade que contém nós, links e outros contextos.

O modelo Dexter proposto por Halasz et al. [33] origina-se do esforço para fornecer uma terminologia uniforme para representar as diferentes premissas de estruturação de hipertexto oferecidas pelos sistemas de construção de hipertexto. O núcleo do modelo é a representação ordenada de um aplicativo hipertextual em três camadas: armazenamento, parte interna do componente e de tempo de execução. A camada de armazenamento descreve a rede de nós e elos do hipertexto, sem detalhes sobre a estrutura interna e o conteúdo do nó, que é o foco da parte interna do componente. A camada de tempo de execução lida com a dinâmica e apresentação do hipertexto. Os conceitos de modelagem disponíveis na camada de armazenamento são muito básicos: os componentes descrevem informações que constituem o hipertexto e podem ser atômicas ou compostas. Links são um tipo especial de componente usado para representar caminhos navegáveis. O modelo Dexter defende muitos conceitos, como as distinções entre estrutura, navegação e apresentação de um hipertexto, cuja influência tem sido duradoura em outras áreas além do hipertexto. Assim, o modelo Dexter serve como um padrão para comparar as características e funcionalidade de vários sistemas hipertexto (e não-hipertexto).

Usando componentes que contêm fragmentos de texto, gráficos, imagens, animações, etc., que formam o conteúdo básico de uma rede hipertexto, o modelo Dexter também serve como uma base de princípios para desenvolver padrões de interoperabilidade e intercâmbio entre sistemas hipertexto. Várias contribuições subsequentes surgiram da crítica ao modelo Dexter e adicionaram formas mais complexas de organização de hipertexto,

como por exemplo sincronização temporal entre componentes introduzida pelo modelo AHM (*Amsterdam Hypermedia Model*) proposto por Hardman et al. [36] estendendo o modelo Dexter.

AHM adiciona as noções de tempo, atributos de apresentação e contexto de link ao Modelo Dexter. Foi desenvolvido por Hardman, Bulterman e van Rossum (1994) para suportar o design de sistemas hipermídia que usam meios dinâmicos como áudio, vídeo e animações. Esses tipos de mídia exigem um modelo que suporte a especificação de relações temporais entre itens de dados. Desta forma, as melhorias mais importantes em relação ao Modelo Dexter são:

- A especificação de apresentação de um componente atômico que inclui a especificação do canal e duração dos atributos.
- A especificação de apresentação inclui layout temporal, layout espacial e informações de estilo. A especificação de apresentação de um componente composto é estendida com uma lista de links de sincronização, que inclui o ID de componentes relacionados e uma relação de tempo.
- Componente composto não inclui conteúdo. Eles só agem como contêineres, ou seja, eles não contêm nenhum dado diretamente.
- Valores das âncora em componentes compostos são substituídos por uma lista de endereços indiretos (ID do componente e ID da âncora).
- Para cada componente filho, o AHM indica um par de ID do componente e uma hora de início.
- O tipo composto é especificado como paralelo ou escolha. O paralelo exibe todas as suas partes, enquanto o de escolha exibe um ou mais de seus filhos. A camada de tempo de execução implementa o mecanismo de seleção.

A Figura 2.2 fornece uma visão de alto nível dos aspectos essenciais dos modelos hipertexto, multimídia e hipermídia. Na Figura 2.2 (a), pode ser visto o hipertexto modelado como uma rede de componentes que são relacionados usando um conjunto de *links* ancorados nos componentes de origem e destino. Embora várias implementações de hipertexto possam impor restrições diferentes à estrutura interna de um componente ou à natureza exata de um conjunto de *links*/âncoras, todos os sistemas suportam a noção de "visitar" um componente por um período de tempo determinado pelo usuário, sendo essa visita

terminada no final da aplicação ou interrompida/substituída/aumentada ao seguir um *link* para um ou mais componentes. Observe que o significado de visitar um componente, ou seja, exibir os efeitos visuais ao usuário, geralmente é considerado uma propriedade interna dos dados. A Figura 2.2 (b) ilustra uma apresentação multimídia genérica. Como na Figura 2.2 (a), a apresentação é composta por uma coleção de componentes. Ao contrário da Figura 2.2 (a), os componentes devem ser apresentados em alguma ordem temporal definida pelo autor. A existência de tal relacionamento de ordenação depende de uma noção explícita de tempo no modelo. Enquanto o usuário ainda pode ter controle sobre a seleção de componentes a serem visitados, os componentes selecionados e apresentados podem mudar sem a intervenção direta do usuário devido a essa noção de tempo. Ou seja, o modelo não apenas define os componentes da apresentação, mas também define um relacionamento que especifica quando os componentes são apresentados um em relação ao outro. Os sistemas multimídia normalmente suportam dois tipos de recursos de navegação que fornecem ao usuário controle sobre a apresentação. O método ajusta a referência de tempo atual em uma apresentação, indicada pela linha horizontal mais forte na Figura 2.2 (b). Usando uma interface de controle semelhante à de uma fita cassete de áudio ou de um *compact disk player*, o usuário pode parar/iniciar/ avançar rapidamente/retroceder (e às vezes pesquisar) a apresentação. O segundo tipo de navegação - e menos comum - é semelhante a um *link* de hipertexto, indicado pela seta de ligação na Figura 2.2 (b). Na figura, o ponto de partida desta seta pode ser chamado de âncora, que é um pedaço do conteúdo que é associado a um *link*. Seguir o *link* leva o usuário ao ponto de tempo indicado pela linha pontilhada na ilustração. Isso é essencialmente equivalente a uma operação de avanço rápido, na qual, diferente deste tipo de operação, o "ponto de partida" e "ponto de parada" são definidos pelo autor do documento e não pelo usuário.

A Figura 2.2 (c) fornece uma descrição de alto nível de uma maneira de combinar hipertexto e multimídia: fazendo com que cada componente do modelo de hipertexto seja uma apresentação multimídia independente. Esse modelo tem dois conjuntos de interesses: aqueles relacionados à navegação hiperestruturada através do documento e aqueles relacionados à apresentação multimídia da informação. Para muitas formas simples de suporte hipermídia, o esboço da Figura 2.2 (c) apresenta um modelo adequado de comportamento do sistema.

Em geral, um modelo hipermídia deve ser capaz de especificar como partes individuais da informação se relacionam umas com as outras em qualquer nível que um autor de documentos ache que faria sentido. Esse nível dependeria da maneira como os dados seriam armazenados e das habilidades de apresentação/navegação dos sistemas de tempo

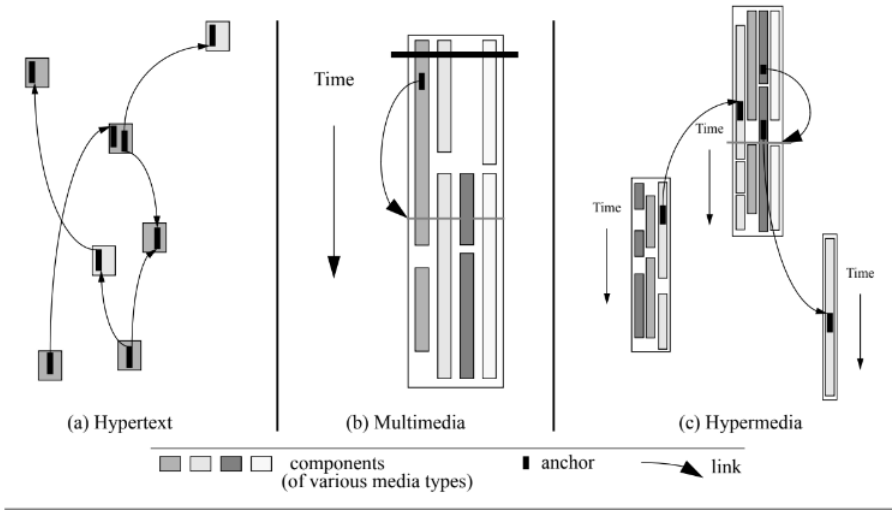


Figura 2.2: Tipos de documentos no contexto do AHM [36]

de execução disponíveis para os usuários. A especificação de restrições de tempo dentro de um documento depende, portanto, da natureza dos elementos de dados subjacentes e da forma como estes podem ser combinados para apresentação. A estrutura interna dos dados estará além do escopo do modelo hipermídia, mas a composição dos componentes permanece central para o modelo NCM (Nested Context Model) desenvolvido por Soares et al. [66]. A relação entre os objetos de um documento hipermídia pode ser estabelecida com o conceito de conector abordado por Muchaluat-Saade et al. em [54, 56] e que apresenta requisitos fundamentais, para especificação de documentos hipermídia. Dentre eles, estão: especificação de documentos de forma estruturada, a representação de tipos diferentes de objetos de mídia, especificação de relacionamentos espaço-temporais complexos entre componentes, a definição das características de apresentação separada da definição dos componentes, a especificação do comportamento temporal do documento de uma forma flexível e a possibilidade de adaptação do documento. Tais características são influenciadas pelo paradigma temporal adotado pelo modelo.

Existem vários tipos de abordagens para a implementação temporal em um sistema hipermídia como por exemplo o desenvolvido por Mattos em [49], dentre eles estão a linha do tempo e a baseada de eventos. Na primeira abordagem, todos os objetos do documento se relacionam com a linha do tempo, ou seja, todas as ações em cima dos objetos acontecem em determinados instantes na linha do tempo. Já na abordagem baseada em eventos, os objetos se relacionam de acordo com determinados eventos que acontecem durante a execução. Desta forma, ações iniciarão se determinado evento acontecer independente de seu instante na linha do tempo. A abordagem baseada em eventos permite que o autor defina documentos hipermídia que contenham ações associadas tanto a relações temporais

definidas por Allen em [5] e espaciais quanto a eventos assíncronos, como por exemplo interação do usuário como definido por Soares et al. [69] . O modelo NCM (*Nested Context Model*), detalhado na próxima seção, é baseado em eventos e por isso é usado como base para o desenvolvimento desta tese.

2.1 Modelo NCM

Modelos conceituais devem ser ricos em seus recursos semânticos para representar objetos multimídia complexos e também para expressar seus requisitos de relacionamentos. Os modelos hipermídia também são o eixo das ferramentas de autoria e formatação. Esta seção é dedicada à apresentação do modelo NCM – *Nested Context Model* criado por Soares et al. [66], fornecendo a base para a discussão sobre as facilidades de autoria e formatação realizadas nas seções a seguir.

O NCM, chamado de Modelo de Contextos Aninhados em português, possui várias entidades que representam objetos que estarão envolvidos em um conteúdo hipermídia. Dentre elas, o modelo se baseia em nós e elos (*links*). Nós representam um conjunto de informação a ser apresentada enquanto que os *links* representam os relacionamentos entre os nós.

Os nós são classificados em nós de composição e nós de conteúdo. Os nós de composição são nós que funcionam como repositório de outros nós, ou seja, seu conteúdo é uma lista de nós. Existe também uma especialização dos nós de composição que são os nós de contexto, estes contêm além de nós de conteúdo ou de contexto, uma lista de *links*. Estas estruturas possibilitam a organização de elementos de mídia que se relacionam conceitualmente. Possibilitando por exemplo, definir um comportamento para todo o contexto. Para acessar um nó dentro de um nó de composição, é necessário usar a interface para um nó de composição chamada de porta. A Figura 2.3 mostra uma representação de nós de contexto. As portas permitem relacionamentos de nós internos com nós externos ao contexto.

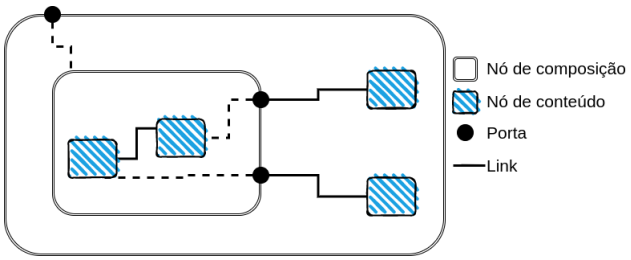


Figura 2.3: Representação de nós de contexto [66]

Os nós de conteúdo possuem um conteúdo a ser apresentado para o usuário. Este conteúdo pode variar entre: texto, imagem, áudio, vídeo ou código de execução (scripts desenvolvidos em uma linguagem procedural). Um nó também pode conter variáveis de documento (conjunto de propriedades do documento). Os nós de conteúdo são acessados através de suas âncoras que podem ser de conteúdo ou de atributo. Uma âncora de conteúdo representa um subconjunto de informação do nó de conteúdo. Portanto ela pode variar de acordo com o tipo de nó ao qual ela pertence (e.g. conjunto de quadros de um vídeo, conjunto de caracteres de um texto, etc). Já a âncora de atributo representa uma propriedade do nó. A Figura 2.4 mostra um visão geral simplificada do modelo NCM do qual foram omitidas várias entidades e propriedades para simplificar a exposição do modelo. No decorrer deste capítulo, serão explicadas as entidades pertinentes a cada tópico.

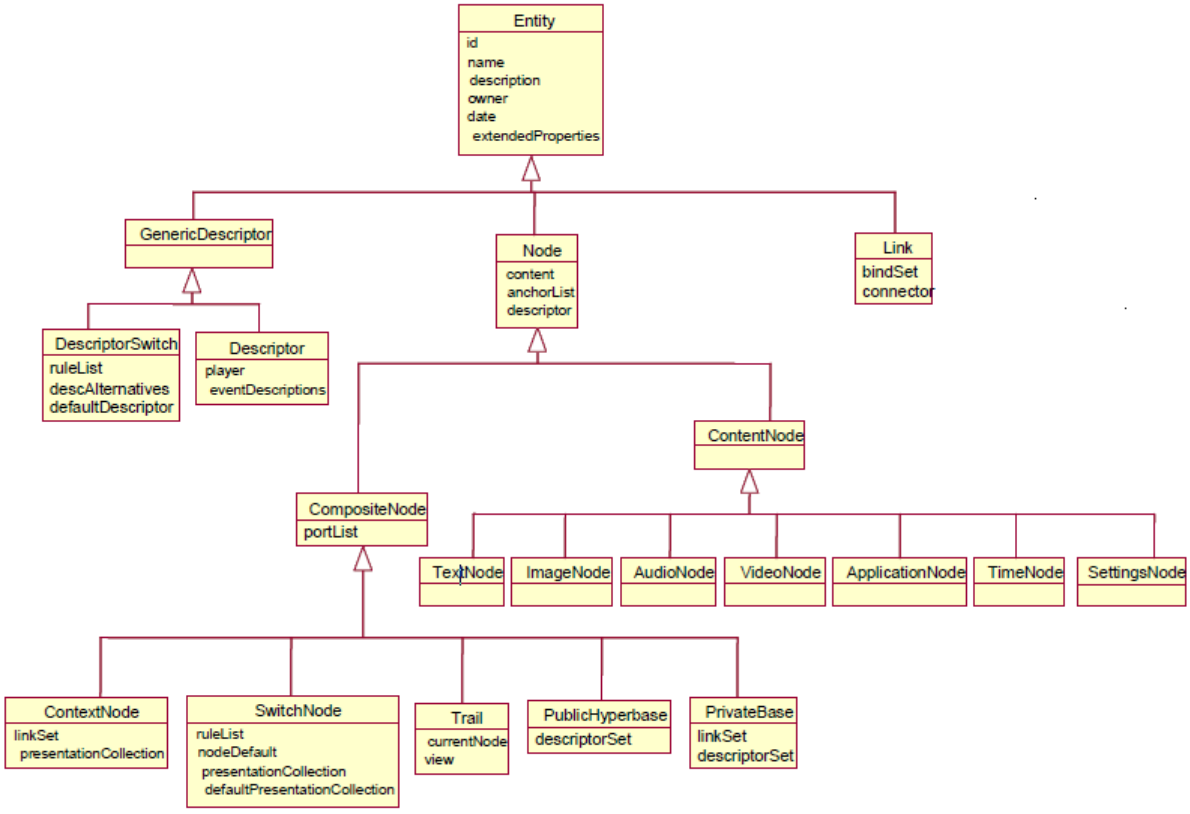


Figura 2.4: Visão geral do modelo NCM [66]

O *link* é uma entidade que materializa conexões entre nós. E essa conexão é estabelecida com duas propriedades importantes, o conector e um conjunto de associações a esse conector. O conector especifica qual tipo de relação será estabelecida. Desta forma, quando o *link* indicar um conector, a ligação vai seguir a especificação deste conector. Vários *links* podem utilizar o mesmo conector mudando os participantes da relação, havendo

assim a reutilização de elementos. O NCM provê dois tipos de conectores, os causais e os de restrição. Somente os conectores causais serão abordados neste texto, eles representam uma relação de causalidade, ou seja, quando as condições (representadas por um conjunto de papéis de condição) forem satisfeitas, um conjunto de ações deve ser executado (representado por um conjunto de papéis de ação).

Modelos multimídia são geralmente comparados em termos de seu poder de expressão, i.e., a amplitude de ideias que podem ser representadas e comunicadas nesse modelo. O arranjo temporal entre objetos de mídia em uma apresentação multimídia é particularmente importante, portanto modelos buscam atender a um conjunto de relações temporais, que usualmente são descritas como relações temporais de Allen [6]. O modelo NCM pode expressar todas as relações de sincronização definidas nas relações temporais de Allen utilizando conectores. Os conectores em NCM seguem uma abordagem baseada em eventos. Desse modo, conectores permitem que NCM expresse as relações de sincronismo temporal com eventos. A Figura 2.5 mostra os conectores para cada relação de Allen. Adicionalmente, conectores em NCM podem também representar relacionamentos assíncronos, isto é, aqueles que não se sabe quando vão acontecer. Por exemplo, relações que dependam da interação do usuário. Para isso, os conectores causais são utilizados com seu papel de condição associado a eventos de interação como por exemplo, o de seleção. O papel de ação pode estar associado ao início de uma mídia. Assim quando o usuário selecionar (evento *selection*) uma mídia do documento, pode iniciar a apresentação de uma âncora (ação *start*).

Os links conectam os vários tipos de nós existentes no documento, representando relações entre eles. Para que se estabeleça essa relação será necessário que o *link* referencie uma interface que pode ser âncora, porta ou atributo de um determinado nó. A Figura 2.6 mostra os tipos de interface que o NCM contempla.

A Figura 2.7, retirada do trabalho de Muchaluat-Saade [53], mostra um exemplo de conector e dois elos que o utilizam. O link l_1 representa o relacionamento entre os nós A, B e C. O conector R possui seus papeis onde pode representar condição ou ação. Na figura, se os nós A e B fazem o papel de condição, quando suas âncoras tornarem a condição satisfeita, uma ação vai acontecer na âncora do nó C.

O modelo NCM segue o paradigma baseado em eventos. Desta forma todo o documento hipermídia é orientado a um conjunto de eventos predefinidos e extensível. Então toda condição de elo mapeada no documento está associada a um evento. A Tabela 2.1 apresenta os eventos presentes no modelo NCM.

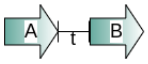
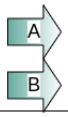
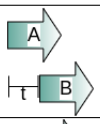
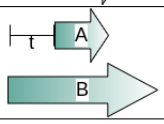
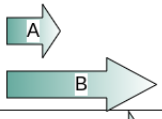
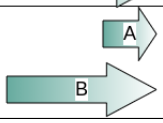
Relação de Allen	Modelo	Conector Hipermídia
A before B		onEndStartDelay
A equal B		onBeginStart onEndStop
A meets B		onEndStart
A overlaps B		onBeginStartDelay
A during B		onBeginStartDelay onEndStopDelay
A starts B		onBeginStart
A finishes B		onEndStop

Figura 2.5: Relações de Allen [6] e os respectivos conectores hipermídia em NCM que as representam

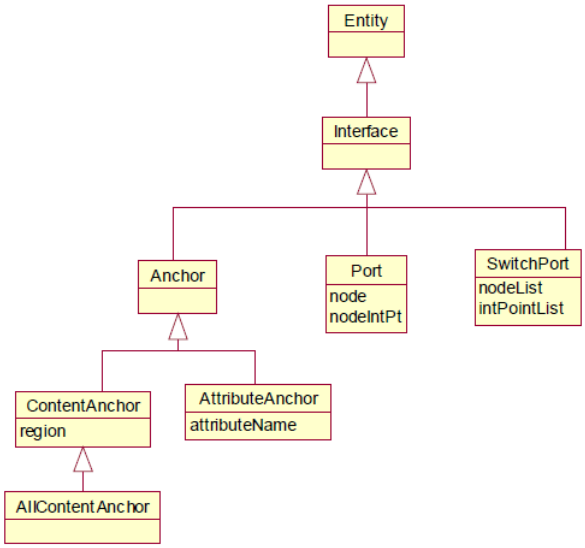


Figura 2.6: Tipos de interfaces do modelo NCM [66]

À medida que os eventos vão acontecendo, eles vão mudando de estado obedecendo a uma máquina de estados apresentada na Figura 2.8. As mudanças de estado acontecem de acordo com a execução da aplicação controlada pelo formatador que gerencia seu

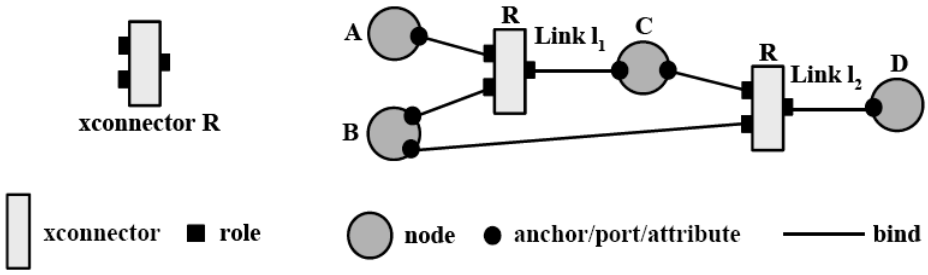


Figura 2.7: Estrutura de Link [53]

Nome	Descrição
Evento de apresentação	Representa a exibição de uma âncora de conteúdo (segmento de mídia)
Evento de composição	Representa a exibição da estrutura de um nó de composição
Evento de seleção	Representa a seleção de uma âncora de conteúdo de um nó (através de mouse, teclado ou controle remoto)
Evento de atribuição	Refere-se a mudança de valor de uma âncora de atributo de um nó
Evento de superposição do dispositivo apontador	Representa a superposição do mouse sobre uma âncora de conteúdo de um nó
Evento de arraste	Representa o arraste sobre uma âncora de conteúdo de um nó
Evento de foco	Representa o ganho de foco de uma âncora de conteúdo de um nó
Evento de preparação [43]	Representa a preparação da exibição de uma âncora de conteúdo de um nó

Tabela 2.1: Eventos presentes no modelo NCM

comportamento temporal, conforme desenvolvido por Rodrigues em [61]. Mudanças de estado de eventos eventualmente satisfazem condições associadas aos conectores e elos, causando as respectivas ações também especificadas pelos conectores e elos.

Um evento NCM pode estar em um dos seguintes estados: dormindo (*sleeping*), ocorrendo (*occurring*) ou pausado (*paused*). Todo evento possui um atributo denominado ocorrências (*occurrences*), que armazena a quantidade de vezes que o evento muda do estado ocorrendo para o estado dormindo durante a apresentação de um documento. Os eventos de apresentação e de atribuição também possuem um atributo denominado repetições (*repetitions*), que apresenta a quantidade de vezes seguidas que o mesmo deve ocorrer. Acompanhando a Figura 2.8 e tomando como exemplo um evento de apresentação, o mesmo inicia no estado dormindo. Quando há a exibição de suas unidades de

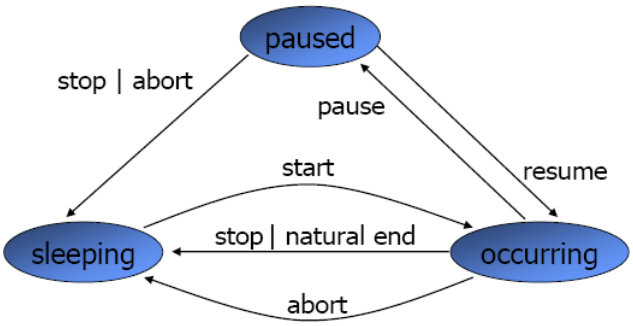


Figura 2.8: Máquina de estados dos eventos no NCM [66]

informação, o evento passa para o estado ocorrendo. Caso a apresentação seja suspensa por um tempo, o evento vai para o estado pausado e permanece enquanto a situação durar. Ao final da apresentação, o evento volta para o estado dormindo, seu atributo *occurrences* é incrementado de uma unidade, e o atributo *repetitions* é decrementado de uma unidade, caso seja maior que zero. Se, após ser decrementado, o atributo *repetitions* possuir um valor maior que zero, a apresentação do evento será reiniciada automaticamente. Quando uma apresentação de um evento é interrompida abruptamente, através de um comando de aborto da apresentação, o evento passa para o estado dormindo, sem que o atributo ocorrências seja incrementado e tornando zero o valor do atributo *repetitions*. Eventos de seleção permanecem no estado ocorrendo enquanto a âncora correspondente estiver sendo selecionada. De modo similar, eventos de arraste, foco e superposição permanecem no estado ocorrendo enquanto a respectiva operação sobre a âncora durar. Já os eventos de atribuição permanecem no estado ocorrendo enquanto os valores dos atributos estiverem sendo modificados. Evidentemente, eventos instantâneos, como uma simples atribuição de valor, podem permanecer por um tempo infinitesimal no estado ocorrendo.

Um evento de apresentação pode mudar do estado ocorrendo para dormindo em duas situações: quando termina a exibição da unidades de informação ou quando há uma ação que force o término do evento. A duração de um evento é o tempo que ele permanece no estado ocorrendo. No caso de um evento de apresentação, essa duração pode depender do objeto de mídia ou ser especificada pelo descritor do evento. A duração de um evento de apresentação será escolhida pelo formatador de documentos considerando detalhes do conteúdo, parâmetros do descritor, relacionamentos do documento (principalmente os elos) e outras informações externas, como características da plataforma de exibição. Um evento de apresentação associado a um nó de composição permanece no estado ocorrendo enquanto pelo menos um evento de apresentação associado a qualquer um dos nós filhos dessa composição estiver no estado ocorrendo ou enquanto pelo menos um elo filho do nó

de composição estiver sendo avaliado. Um evento de apresentação associado a um nó de composição está no estado pausado se pelo menos um evento de apresentação associado a qualquer um dos nós filhos da composição estiver no estado pausado e todos os outros eventos de apresentação associados aos nós filhos da composição estiverem no estado dormindo ou pausado. Do contrário, o evento de apresentação está no estado dormindo.

2.2 Linguagem NCL

A Linguagem NCL (*Nested Context Language*), desenvolvida por Soares et al. [68] é uma linguagem declarativa usada para especificar documentos hipermídia para TV digital interativa compostos de objetos de mídia (e.g., texto, imagem, gráfico, áudio, vídeo, animação, *scripts* Lua) e relacionamentos entre os objetos. As linguagens declarativas têm como principal característica a semântica declarativa cujo conceito básico é a maneira simples de determinar significado de cada sentença e não depender de como será usada, ou seja, não considera os detalhes inerentes da semântica de execução. Para construção de uma linguagem de especificação de documentos hipermídia, é necessário um modelo conceitual hipermídia que expresse todas as entidades. Na literatura existem vários modelos, alguns citados no início deste capítulo, e o NCM foi utilizado pela linguagem NCL como base. Portanto, os elementos criados na linguagem NCL foram baseados nas entidades no modelo NCM. A base sintática é o XML [75]. A versão atual da linguagem é versão 3.0, que é suportada pelo *middleware* declarativo Ginga-NCL, o formatador que executa um documento escrito em NCL. A linguagem e o formatador se tornaram padrão brasileiro de TV Digital em 2007 [2] e padrão internacional para IPTV em 2009 [40]. Para que o autor possa construir código imperativo, poderá utilizar *scripts* Lua, linguagem desenvolvida por Ierusalimschy et al. [38], que são referenciados por um tipo de mídia em NCL. O módulo NCLua permite que um código procedural em Lua consiga interagir com um documento hipermídia NCL [62].

Usando NCL, os autores podem descrever o comportamento temporal de uma apresentação multimídia, associando *hyperlinks* (interação do usuário) com objetos de mídia, definindo alternativas de apresentação (adaptação) e descrevendo o layout da apresentação em múltiplos dispositivos. NCL também permite para edição de comandos do usuário, o uso de comando externos, incluindo comando para geração de aplicação em tempo de execução.

A linguagem NCL foi desenvolvida em módulos funcionais, seguindo a abordagem

utilizada em várias linguagens [21] recomendadas pelo W3C (*World Wide Web Consortium*). Os módulos são conjuntos de elementos que estão reunidos a partir de conceitos relacionados e seguem listados na Tabela 2.2.

Módulos	Identificadores
Animation	http://www.ncl.org.br/NCL3.0/Animation
CompositeNodeInterface	http://www.ncl.org.br/NCL3.0/CompositeNodeInterface
CausalConnector	http://www.ncl.org.br/NCL3.0/CausalConnector
CausalConnectorFunctionality	http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality
ConnectorCausalExpression	http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression
ConnectorAssessmentExpression	http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression
ConnectorBase	http://www.ncl.org.br/NCL3.0/ConnectorBase
ConnectorCommonPart	http://www.ncl.org.br/NCL3.0/ConnectorCommonPart
ContentControl	http://www.ncl.org.br/NCL3.0/ContentControl
Context	http://www.ncl.org.br/NCL3.0/Context
Descriptor	http://www.ncl.org.br/NCL3.0/Descriptor
DescriptorControl	http://www.ncl.org.br/NCL3.0/DescriptorControl
EntityReuse	http://www.ncl.org.br/NCL3.0/EntityReuse
ExtendedEntityReuse	http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse
Import	http://www.ncl.org.br/NCL3.0/Import
Layout	http://www.ncl.org.br/NCL3.0/Layout
Linking	http://www.ncl.org.br/NCL3.0/Linking
Media	http://www.ncl.org.br/NCL3.0/Media
MediaContentAnchor	http://www.ncl.org.br/NCL3.0/MediaContentAnchor
KeyNavigation	http://www.ncl.org.br/NCL3.0/KeyNavigation
PropertyAnchor	http://www.ncl.org.br/NCL3.0/PropertyAnchor
Structure	http://www.ncl.org.br/NCL3.0/Structure
SwitchInterface	http://www.ncl.org.br/NCL3.0/SwitchInterface
TestRule	http://www.ncl.org.br/NCL3.0/TestRule
TestRuleUse	http://www.ncl.org.br/NCL3.0/TestRuleUse
Timing	http://www.ncl.org.br/NCL3.0/Timing
TransitionBase	http://www.ncl.org.br/NCL3.0/TransitionBase
Transition	http://www.ncl.org.br/NCL3.0/Transition
Metainformation	http://www.ncl.org.br/NCL3.0/MetaInformation

Tabela 2.2: Módulos da Linguagem NCL 3.0 [64]

As principais entidades da linguagem NCL e que estão relacionadas com o escopo deste trabalho são: nós, elos e conectores. Cada uma delas tem um conceito a ser seguido a fim de ter a especificação de um documento com toda sua lógica de exibição.

Como apresentado na Seção 2.1, o nó na grande maioria das vezes vai conter o conteúdo a ser exibido para um usuário da aplicação multimídia. Em NCL, o nó de conteúdo é expresso pelo elemento *<media>* que tem, entre vários atributos, o atributo src indicando o conteúdo a ser exibido. O elemento *<media>* define então o que deve ser exibido. As regiões representadas pela *tag* *<region>* definem onde será exibido o conteúdo do nó. Já os descritores informam como exibir o conteúdo do nó e são representados pela *tag* *<descriptor>*. O momento da exibição do conteúdo é representado pelo relacionamento entre nós, que é feito por elos e conectores, com os elementos *<link>* e *<xconnector>* respectivamente. A Tabela 2.3 mostra onde esses elementos estão distribuídos nos módulos e áreas funcionais.

O módulo *Media* define os tipos básicos de objetos de mídia. Para definir esses objetos,

Áreas Funcionais	Módulos	Elementos
Structure	*Structure*	ncl
		head
		body
Layout	*Layout*	*regionBase*
		region
Components	*Media*	*media*
	Context	*context*
Interfaces	*MediaContentAnchor*	*area*
	PropertyAnchor	*property*
	CompositeNodeInterface	*port*
	SwitchInterface	*switchPort*
		mapping
Presentation Specification	*Descriptor*	*descriptor*
		descriptorParam
		descriptorBase
Linking	*Linking*	*bind*
		bindParam
		linkParam
		link
Connectors	*CausalConnectorFunctionality*	*causalConnector*
		connectorParam
		simpleCondition
		compoundCondition
		simpleAction
		compoundAction
		assessmentStatement
		attributeAssessment
		valueAssessment
		compoundStatement
	ConnectorBase	*connectorBase*
...	...	...

Tabela 2.3: Algumas Áreas Funcionais da Linguagem NCL 3.0 [64]

existe o elemento <*media*>. Além do atributo id, cada objeto de mídia pode definir dois atributos principais: src, que define o URI (*Uniform Resource Identifier*) do conteúdo e tipo do objeto. A apresentação de um documento NCL requer a sincronização de vários objetos de mídia, que são especificados por meio de elementos do tipo <*media*>. Para cada objeto de mídia, um reprodutor de mídia controla a apresentação de seu conteúdo e seus eventos NCL.

Além do conteúdo a ser transmitido, o nó de conteúdo pode armazenar variáveis de ambiente com o nó do tipo *settings node*. Esse nó representa todas as variáveis controladas diretamente pelo formatador (ferramenta responsável pela apresentação de um documento NCL). As variáveis são representadas pelas propriedades do nó. Alguns atributos podem ter seus valores modificados por ações dos elos, outros podem ter seus valores testados inclusive pelas regras de nós *switch* e de *switch* de descritores, a fim de realizar a escolha entre alternativas.

O uso do nó de variáveis de ambiente é feito por meio do elemento *<media>* e colocado no atributo *type* a string "application/x-ncl-settings". Só pode haver apenas um nó deste tipo em um documento NCL, cujas propriedades são variáveis globais definidas pelo autor do documento ou variáveis de ambiente reservadas (variáveis de sistema) que podem ser manipuladas pelo formatador NCL. A Tabela 2.4 indica as variáveis de sistema já definidas, sua semântica e seus valores possíveis. Esse conjunto de variáveis é gerenciado pelo sistema receptor. Essas variáveis podem ser lidas, mas não podem ter seus valores alterados por uma aplicação NCL, um procedimento Lua ou qualquer outro procedimento imperativo ou declarativo. As aplicações nativas do receptor podem alterar os valores das variáveis e devem persistir durante o ciclo de vida do receptor.

A Tabela 2.5 descreve o significado e os valores possíveis das variáveis de ambiente do grupo *user* no middleware Ginga-NCL, um conjunto de variáveis também gerenciado pelo sistema receptor. Assim como as variáveis da Tabela 2.4, essas outras podem ser lidas, mas não podem ter seus valores alterados por uma aplicação NCL, um procedimento Lua ou qualquer outro procedimento imperativo ou declarativo. E as aplicações nativas do receptor podem alterar os valores das variáveis e devem persistir durante o ciclo de vida do receptor. A proposta de extensão desta tese possibilita que as características do usuário sejam carregadas de um arquivo de perfil dinamicamente, ou seja, no inicio da aplicação NCL, além de poderem ser alteradas durante a execução pela aplicação NCL, se o usuário permitir.

Além dos grupos de variáveis citados na Tabela 2.4 e na Tabela 2.5, existem mais 6 grupos de varáveis de ambiente que fazem parte da linguagem porém não serão detalhadas neste texto por não fazerem parte do escopo deste trabalho.

A definição do relacionamento entre as mídias é feito por meio do elemento *<xconnector>* representando os conectores do NCM e do *<link>* representando os elos. Os relacionamentos são materializados com os *links* e esses referenciam conectores também especificados pelo autor. Um conector especifica uma relação independentemente dos objetos participantes, ou seja, ele não especifica quais nós (representados por elementos *<media>, <context>, <body> e <switch>*) irão interagir através da relação. Um elemento *<link>*, por sua vez, representa uma relação, do tipo definido por seu conector, interligando diferentes nós. *Links* que representam o mesmo tipo de relação, mas interconectam nós diferentes, podem reutilizar o mesmo conector, reutilizando todas as especificações anteriores. Um conector especifica, por meio de seus elementos filhos, um conjunto de pontos de interface, chamados papéis. Um elemento *<link>* refere-se a um

Variável	Significado	Valores possíveis
system.language	Idioma do áudio	Valores definidos pela ISO 639-1
system.caption	Idioma do *closed caption*	Valores definidos pela ISO 639-1
system.subtitle	Idioma das legendas (*subtitle*)	Valores definidos pela ISO 639-1
system.returnBitRate(i)	Taxa do canal interativo (i) em Kbps	real
system.screenSize	Tamanho da tela do dispositivo de exibição, em linhas, pixels/linha, quando uma classe não é definida	(inteiro, inteiro)
system.screenGraphicSize	Resolução configurada para o plano gráfico da tela do dispositivo de exibição, em (linhas, pixels/linha), quando uma classe não é definida	(inteiro, inteiro)
system.audioType	Tipo de áudio do dispositivo de exibição, quando uma classe não é definida	"mono"|"stereo"|"5.1"
system.screenSize(i)	Tamanho da tela do dispositivo (i) em (linhas,pixels/linha)	(inteiro, inteiro)
system.screenGraphicSize(i)	Resolução definida para o plano gráfico do dispositivo(i) em (linhas, pixels/linha)	(inteiro, inteiro)
system.audioType(i)	Tipo de áudio do dispositivo(i)	"mono"|"stereo"|"5.1"
system.devNumber(i)	Número de dispositivos de exibição cadastrados na classe(i)	inteiro
system.classType(i)	Tipo da classe (i)	("passive" | "ative")
system.parentDeviceRegion(i)	Identifica o *element* <*region*> em uma outra <*regionBase*> associada ao dispositivo pai que cria o mapa de bits enviado à classe passiva (i);nesta região, o mapa de bits também deve ser exibido	Cinco números separados por vírgulas, cada um seguindo as regras de valores associados para as propriedades *left*, *top*, *width*, *height*, e *zIndex*, respectivamente
system.info(i)	Lista de exibidores de mídia da classe (i) de dispositivos de exibição	string
system.classNumber	Número de classes de dispositivos de exibição definidas	inteiro
system.CPU	Desempenho da CPU em MIPS	real
system.memory	Espaço de memória em Mbytes	inteiro
system.operatingSystem	Tipo do sistema operacional	string
system.javaConfiguration	Tipo e versão da configuração Java suportada pelo receptor JVM	string (tipo imediatamente seguido da versão, como por exemplo: "CLDC1.1")
system.javaProfile	Tipo e versão do perfil Java suportado pelo receptor JVM	string (tipo imediatamente seguido da versão, como por exemplo: "MIDP2.0")
system.luaVersion	Versão da máquina Lua suportada pelo receptor	string
system.ncl.version	Versão da linguagem NCL	string
system.GingaNCL.version	Versão do ambiente Ginga-NCL	string
system.xxx	Qualquer variável com o prefixo "system" deve ser reservada para uso futuro	

Tabela 2.4: Variáveis de ambiente de uma aplicação em Linguagem NCL 3.0 [64]

conector e define um conjunto de elementos <*bind*>, que associam cada ponto de extremidade do *link* (interface de nó) a um papel (*role*) do *conector* usado conforme pode-se ver na Figura 2.9, retirada do trabalho de Soares et al. [64].

Para definirmos um conector causal é necessário especificar pelo menos dois papeis

Variável	Significado	Valores possíveis
user.age	Idade do usuário	inteiro
user.location	Localização do usuário (código de endereçamento postal)	string
user.genre	Sexo do usuário	"m" | "f"
user.xxx	Qualquer variável com o prefixo "*user*" deve ser reservada para uso futuro	

Tabela 2.5: Variáveis de ambiente relacionadas ao usuário de uma aplicação em Linguagem NCL 3.0 [64]

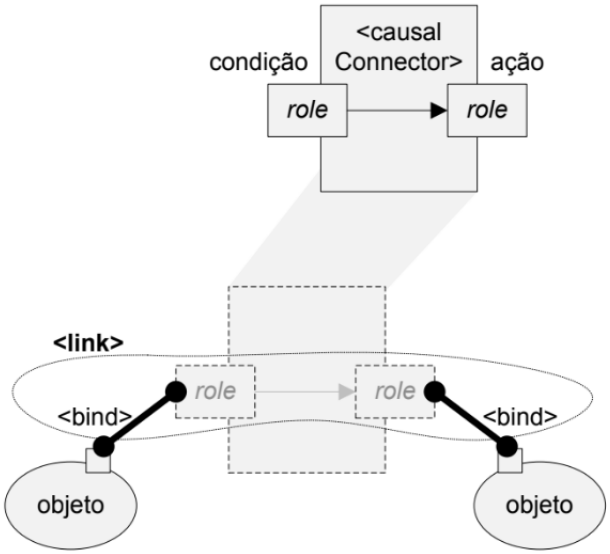


Figura 2.9: Representação do link NCL [64]

(*roles*), um que representa a condição e outro que representa a ação ambos sobre eventos NCL. Quando a condição for satisfeita, a ação será disparada. Os *links* relacionam os objetos de mídia aos papeis de um conector por meio de *binds*. Desta forma, se um objeto de mídia estiver associado ao papel que representa a condição, isso significa que o elo vai ser disparado se o evento ocorrido estiver relacionado ao objeto ligado. De maneira análoga, a ação vai acontecer sobre o objeto de mídia que estiver ligado ao papel que representa a ação.

A lista de eventos a serem monitorados por um reprodutor de mídia também deve ser computada pelo formatador NCL, levando em consideração a especificação do documento NCL. Deve obrigatoriamente verificar todos os *links* onde o objeto de mídia e o descritor resultante (se existir) participam. Ao computar os eventos a serem monitorados, o formatador NCL deve obrigatoriamente levar em consideração a perspectiva do objeto de mídia, ou seja, o caminho dos elementos $<body>$ e dos elementos $<context>$ descendentes até chegar ao elemento $<media>$. Apenas os links contidos nesses elementos $<body>$ e $<context>$ devem ser considerados para calcular os eventos monitorados.

As relações em NCL são baseadas em eventos. Um evento é uma ocorrência no tempo que pode ser instantânea ou ter uma duração mensurável. NCL 3.0 define os seguintes tipos de eventos [3]:

Apresentação definido pela apresentação de um subconjunto das unidades de informação de um objeto de mídia, especificado em NCL pelo elemento <*area*>, ou pelo próprio nó de mídia (apresentação de todo o conteúdo). Os eventos de apresentação também podem ser definidos em nós de composição (representados por um elemento <*body*>, <context> ou <switch>), representando a apresentação das unidades de informação de qualquer nó dentro de um nó de composição;

Seleção definido pela seleção de um subconjunto de unidades de informação de um objeto de mídia (que é especificado em NCL pelo elemento <*area*>, ou pela âncora de conteúdo inteiro do nó de mídia), sendo apresentado e visível;

Atribuição definido pela atribuição de um valor a uma propriedade de um nó (representado por um elemento <*media*>, <*body*>, <*context*> ou <*switch*>). A propriedade deve obrigatoriamente ser declarada em um elemento filho <*property*> do nó.

Preparação definido pela preparação do conteúdo para apresentação de uma âncora de um elemento <*media*>. A preparação realizada a busca antecipada da parte inicial deste conteúdo.

O módulo *CausalConnectorFunctionality* permite apenas a definição de relações causais, definidas pelo elemento <*causalConnector*> do módulo *CausalConnector*. Um elemento <*causalConnector*> une expressões, que definem uma condição e uma ação. Quando a expressão de condição for satisfeita, a expressão de ação deve ser executada. Um conector causal possui o atributo id, que identifica exclusivamente o elemento dentro de um documento. Uma expressão de condição pode ser simples (elemento <*simpleCondition*>) ou composta (elemento <*compoundCondition*>). Ambos os elementos são definidos pelo módulo *ConnectorCausalExpression*.

O elemento <*simpleCondition*> tem um atributo *role*, cujo valor deve obrigatoriamente ser único no conjunto de *roles* do conector. O *role* é um ponto de interface do conector, que será associado às interfaces do nó por um *link* que se refere ao conector. Uma <*simpleCondition*> também define um tipo de evento (atributo *eventType*) e a qual transição ele se refere (atributo *transition*). Os atributos *eventType* e *transition* são opcionais. Eles podem ser inferidos pelo valor do *role*, se valores reservados forem usados. Caso contrário, os atributos *eventType* e *transition* precisam ser especificados.

Os valores reservados usados para definir os *roles* de <*simpleCondition*> são indicados na Tabela 2.6. Se um valor de *eventType* for "*selection*", o *role* também pode definir a qual tecla de controle remoto ela se refere, por meio de seu atributo *key*. Pelo menos os seguintes valores (diferenciando maiúsculas de minúsculas) devem ser aceitos para o atributo *key*: "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z","*", "#", "*MENU*", "*INFO*", "*GUIDE*", "*CURSOR_DOWN*", "*CURSOR_LEFT*", "*CURSOR_RIGHT*", "*CURSOR_UP*", "*CHANNEL_DOWN*", "*CHANNEL_UP*","*VOLUME_DOWN*", "*VOLUME_UP*", "*ENTER*", "*RED*", "*GREEN*", "*YELLOW*", "*BLUE*", "*BACK*", "*EXIT*", "*POWER*", "*REWIND*", "*STOP*", "*EJECT*", "*PLAY*", "*RECORD*", "*PAUSE*".

Role Value	*Transition Value*	*Event Type*
onBegin	*starts*	*presentation*
onEnd	*stops*	*presentation*
onAbort	*aborts*	*presentation*
onPause	*pauses*	*presentation*
onResume	*resumes*	*presentation*
onSelection	*starts*	*selection*
onBeginSelection	*starts*	*selection*
onEndSelection	*stops*	*selection*
onBeginAttribution	*starts*	*attribution*
onEndAttribution	*stops*	*attribution*
onAbortAttribution	*aborts*	*attribution*
onPauseAttribution	*pauses*	*attribution*
onResumeAttribution	*resumes*	*attribution*
onBeginPreparation	*starts*	*preparation*
onEndPreparation	*stops*	*preparation*
onAbortPreparation	*aborts*	*preparation*
onPausePreparation	*pauses*	*preparation*
onResumePreparation	*resumes*	*preparation*

Tabela 2.6: Valores de papel de condição reservados associados a máquinas de estado de evento [40]

A cardinalidade do *role* especifica o número mínimo (atributo min) e máximo (atributo max) de participantes que podem desempenhar a função (número de *binds*) quando o <*causalConnector*> é usado para criar um <*link*>. O valor mínimo de cardinalidade deve ser sempre um valor finito positivo, maior que zero e menor ou igual ao valor máximo de cardinalidade. Quando o valor de cardinalidade máxima é maior do que um, vários participantes podem desempenhar o mesmo *role*, ou seja, pode haver vários *binds* conectando diversos nós ao mesmo *role*. O valor "*unbounded*"(ilimitado) pode ser definido como o atributo max, se o *role* puder ter *binds* ilimitados associados a ele. Nestes dois últimos casos, um atributo qualificador deve ser especificado informando a relação lógica

entre os *binds* de condição simples. Conforme descrito na Tabela 2.7, os valores possíveis para o atributo qualificador são: "or" e "and". Se o qualificador estabelecer o operador lógico "OR", a ação do *link* será disparada sempre que ocorrer qualquer condição. Se o qualificador estabelecer o operador lógico "and", a ação do link será disparada depois que todas as condições simples ocorrerem.

Elemento *Role*	Operador	Significado
simpleCondition	*or*	Verdadeiro sempre que ocorrer qualquer condição simples associada.
simpleCondition	*and*	Verdadeiro imediatamente após todas as condições simples associadas serem verdadeiras.

Tabela 2.7: Valores de qualificador de condição simples [40]

Um atributo de atraso também pode ser definido para uma <*simpleCondition*> especificando que a condição será verdadeira após um atraso de tempo a partir do momento em que ocorre a transição. O elemento <*compoundCondition*> tem um atributo de operador booleano ("*and*"ou "*or*") relacionando seus elementos filhos: <*simpleCondition*>, <*compoundCondition*>, <*assessmentStatement*> e <*compoundStatement*>. Um atributo de atraso também pode ser definido especificando que a condição composta será verdadeira após um atraso de tempo a partir do momento em que a expressão que relaciona seus elementos filhos for verdadeira. Os elementos <*assessmentStatement*> e <*compoundStatement*> são definidos pelo módulo *ConnectorAssessmentExpression*.

O módulo *ConnectorAssessmentExpression* define quatro elementos: <*assessmentStatement*>, <*attributeAssessment*>, <*valueAssessment*> e <*compoundStatement*>. O <*attributeAssessment*> tem um atributo *role*, que deve ser exclusivo no conjunto de funções do conector. Como de costume, o *role* é um ponto de interface do conector, que está associado às interfaces do nó por um <*link*> que se refere ao conector. Um <*attributeAssessment*> também define um tipo de evento (atributo *eventType*). Se o valor de *eventType* for "*selection*", o <*attributeAssessment*> também deve definir a qual mecanismo de seleção (por exemplo, teclado ou teclas de controle remoto) ele se refere, por meio de seu atributo *key*. Se o *key* não for especificado, é possível fazer a seleção por meio de teclas de navegação e <*ENTER*> ou ainda por meio de um dispositivo apontador (mouse, tela de toque, etc.). Se o valor do *eventType* for "*presentation*"ou "*selection*", o *attributeType* é opcional e, se presente, deve obrigatoriamente especificar o estado do evento ("*state*"); se o *eventType* for "*attribution*", o *attributeType* será opcional e pode ter o valor "*nodeProperty*"(padrão) ou "*state*". No primeiro caso, o evento representa uma propriedade do nó a ser avaliada; no segundo caso, o evento representa a avaliação do estado do evento de atribuição correspondente. Um valor de deslocamento pode ser adicionado a um

<*attributeAssessment*> antes da comparação. Por exemplo, um deslocamento pode ser adicionado a uma avaliação de atributo para especificar: "a posição vertical da tela mais 50 pixels". O elemento <*valueAssessment*> tem um atributo de valor que pode assumir um valor de estado de evento ou qualquer valor a ser comparado com uma propriedade de nó. O elemento <*assessmentStatement*> tem um atributo comparador que compara os valores inferidos de seus elementos filhos (elemento <*attributeAssessment*> e elemento <*valueAssessment*>). O atributo comparador deve ter um dos seguintes valores: "*eq*", "*ne*", "*gt*", "*lt*", "*gte*"ou "*lte*". O elemento <*compoundStatement*> tem um atributo de operador booleano ("*and*" ou "*or*") relacionando seus elementos filhos: <*assessmentStatement*> ou <*compoundStatement*>. O atributo *isNegated* também pode ser definido para especificar que o elemento filho <*compoundStatement*> deve ser negado antes que a operação booleana seja avaliada. Uma expressão de ação captura ações que podem ser executadas em relações causais e podem ser compostas por um elemento <*simpleAction*> ou <*compoundAction*>, também definido pelo módulo *ConnectorCausalExpression*. O elemento <*simpleAction*> tem um atributo *role*, que deve ser único no conjunto de *roles* do conector. Como de costume, o *role* é um ponto de interface do conector, que está associado às interfaces do nó por um <link> que se refere ao conector. Uma <*simpleAction*> também define um tipo de evento (atributo *eventType*) e qual transição de estado de evento ele aciona (*actionType*). Os atributos *eventType* e *actionType* são opcionais. Eles podem ser inferidos pelo valor do *role* se valores reservados forem usados; caso contrário, *eventType* e *actionType* são obrigatórios. Os valores reservados usados para definir os papeis de <*simpleAction*> são indicados na Tabela 2.8.

valor do Role	Tipo da Ação	Tipo do Evento
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*abort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*start*	*attribution*
startAttribution	*start*	*attribution*
stopAttribution	*stop*	*attribution*
abortAttribution	*abort*	*attribution*
pauseAttribution	*pause*	*attribution*
resumeAttribution	*resume*	*attribution*
startPreparation	*start*	*preparation*
stopPreparation	*stop*	*preparation*
abortPreparation	*abort*	*preparation*
pausePreparation	*pause*	*preparation*
resumePreparation	*resume*	*preparation*

Tabela 2.8: Valores de papeis de ação reservados associados a máquinas de estado de evento [40]

Como acontece com os elementos <*simpleCondition*>, a cardinalidade do *role* especifica o número mínimo (atributo mínimo) e máximo (atributo máximo) de participantes que podem desempenhar o papel (número de ligações) quando <*causalConnector*> é usado para criar um link. Quando o valor de cardinalidade máxima é maior que um, vários participantes podem desempenhar o mesmo *role*. Quando tem o valor "*unbounded*", o número de *binds* é ilimitado. Nestes dois últimos casos, as ações devem ser executadas na mesma ordem de definição dos *binds*.

O elemento <*compoundAction*> agrupa elementos filho: <*simpleAction*> e <*compoundAction*>. A execução das ações deve ser realizada na ordem em que são especificadas. Um atributo de atraso também pode ser definido, especificando que a ação composta acionada deve ser aplicada após o retardo.

O elemento <*causalConnector*> pode ter elementos filhos <*connectorParam*>, que são usados para parametrizar os valores de atributo do conector. O módulo *ConnectorCommonPart* define o tipo do elemento <*connectorParam*>, que possui atributos de nome e tipo. Para especificar quais atributos recebem valores de parâmetro definidos pelo conector, seus valores são especificados como o nome do parâmetro precedido pelo símbolo $. Por exemplo, para parametrizar o atributo de atraso, um parâmetro chamado *actionDelay* pode ser definido (<*connectorParam name* = "*actionDelay*" *type* = "*unsignedLong*"/>) e o valor *"$actionDelay"*é usado no atributo (*delay* = *"$actionDelay"*). Os elementos do módulo *CausalConnectorFunctionality*, seus atributos e seus elementos filhos devem estar de acordo com a Tabela 2.9.

O módulo *ConnectorBase* define um elemento chamado <*connectorBase*>, que permite agrupar conectores. O elemento <*connectorBase*> deve ter o atributo id, que identifica exclusivamente o elemento em um documento. O conteúdo exato de uma base de conectores é especificado pelo conjunto de conectores utilizados no documento. Porém, como a definição de conectores não é facilmente feita por usuários inexperientes, a ideia é ter usuários experientes definindo conectores, armazenando-os em bibliotecas (bases de conectores) que podem ser importadas e disponibilizando-os a terceiros para criação de *links* como pode-se ver na seção Importação de Conectores em [64].

O elemento do módulo *ConnectorBase*, representado pelo elemento <*connectorBase*>, tem como atributo o *id* e seus elementos filhos podem ser <*importBase*> ou <*causalConnector*>, com uma ou várias ocorrências desses.

Elementos	Atributos	Conteúdo
causalConnector	id	(*connectorParam**, (*simpleCondition* | *compoundCondition*), (*simpleAction* | *compoundAction*))
connectorParam	*name*, *type*	*Empty*
simpleCondition	*role*, *delay*, *eventType*, *key*,*transition*, min, max, *qualifier*	*Empty*
compoundCondition	*operator*, *delay*	((*simpleCondition* | *compoundCondition*)+, (*assessmentStatement* | compoundStatement)*)
simpleAction	*role*, *delay*, *eventType*, *actionType*, *value*, min, max, *duration*, by	*Empty*
compoundAction	*operator*, *delay*	(*simpleAction* | *compoundAction*)+
assessmentStatement	*comparator*	(*attributeAssessment*, (*attributeAssessment* | *valueAssessment*))
attributeAssessment	*role*, *eventType*, *key*, *attributeType*, *offset*	*Empty*
valueAssessment	*value*	*Empty*
compoundStatement	*operator*, *isNegated*	(*assessmentStatement* | *compoundStatement*)+

Tabela 2.9: Módulo *CausalConnectorFunctionality* estendido [40]

2.3 Exemplo de Aplicação NCL

A Listagem 2.1 apresenta um exemplo de uma aplicação NCL na qual será exibido um vídeo de turismo e quando chegar a um determinado momento do vídeo será exibida uma imagem de anúncio. A aplicação possui dois elementos do tipo *media* um para referenciar o conteúdo do vídeo, linha 25, e outro o conteúdo de imagem, linha 28, com o anúncio de uma agência de turismo. Para especificar as características de apresentação de cada um dos nós *media*, foram criadas duas regiões e os dois descritores nas linhas 5, 6, 9 e 10.

Os relacionamentos entre os nós são representados pelo *links* das linhas 29 e 33. O primeiro link representa a relação que promove a exibição da imagem do anúncio em um determinado momento do vídeo. Para isso foi criado um conector causal *onBeginStart* (linha 15), ou seja, quando o nó que faz o papel de *onBegin* iniciar, o elemento que faz o papel de *start* começará. Assim, o *link* da linha 35 do tipo *onBeginStart* define que quando iniciar o elemento, que neste caso é uma âncora do vídeo (ancora1, definida na linha 30), o nó de imagem (imgAgencia) começará ser exibido. O *link* da linha 40 estabelece uma relação de término, ou seja, quando terminar o vídeo, a imagem também deixará de ser exibida.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<ncl id="appNcl" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
    <regionBase>
        <region id="rgVideo" width="100%" height="70%" top="10%"/>
        <region id="rgPropaganda" width="40%" height="30%" top="0%"/>
  </regionBase>
  <descriptorBase>
    <descriptor id="dsVideo" region="rgVideo" explicitDur="100s"/>
    <descriptor id="dsPropaganda" region="rgPropaganda"/>
  </descriptorBase>
  <connectorBase>
    <causalConnector id="onBeginStart">
      <simpleCondition role="onBegin"/>
      <simpleAction role="start"/>
    </causalConnector>
    <causalConnector id="onEndStop">
      <simpleCondition role="onEnd"/>
      <simpleAction role="stop"/>
    </causalConnector>
  </connectorBase>
</head>
<body>
  <port id="pInicio" component="video"/>
  <media id="video" descriptor="dsVideo" src="Tour.mpg">
    <area id="ancora1"  begin="70s"   end="74s"/>
  </media>
  <media id="imgAgencia" descriptor="dsPropaganda" src="agencia.png"
   />
  <link xconnector="onBeginStart">
      <bind role="onBegin" component="video" interface="ancora1"/>
      <bind role="start" component="imgAgencia"/>
  </link>
  <link xconnector="onEndStop">
    <bind role="onEnd" component="video"/>
    <bind role="stop" component="imgAgencia"/>
  </link>
</body>
</ncl>
```

Listagem 2.1: Exemplo de uma aplicação NCL

2.3.1 Considerações Finais

Este capítulo apresentou uma fundamentação teórica sobre modelos hipermídia, o modelo NCM e a linguagem NCL. Iniciou apresentando os primeiros modelos construídos para representar documentos hipermídia, além da evolução histórica dessa representatividade. Logo em seguida, o capítulo apresentou o modelo NCM e suas principais entidades e que estão relacionadas com a proposta de extensão desta tese. O capítulo finaliza apresentando os conceitos que envolvem a linguagem NCL e seus principais elementos. Alguns desses elementos são estendidos por esta tese. O próximo capítulo oferece uma visão das pesquisas relacionadas ao tema desta tese e compara-as com as proposições que foram colocadas ao longo deste trabalho. Essas pesquisas tratam da modelagem de documentos hipermídia, assim como de linguagens utilizadas para a especificação dos documentos hipermídia e que oferecem suporte multimodal e multiusuário.

Capítulo 3

Trabalhos Relacionados

Os trabalhos apresentados neste capítulo vão desde abordagens para definição e classificação de multimodalidade [73], propostas como [32] de uma especificação de interação multimodal e multiusuário, apresentações de padrão como em [45] com o padrão MPEG-V, implementações que proveem integração de sistemas em [59] integrando o Ginga-NCL com M-Hub [70] até trabalhos como [51], que discute a importância e os desafios encontrados para implementação de TV multiusuário.

3.1 Multimodalidade

Normalmente, os sistemas de televisão digital permitem o controle sobre o conteúdo transmitido por meio do Guia Eletrônico de Programação (EPG - *Eletronic Program Guide*). O EPG consiste em uma grade, onde as colunas representam os canais de televisão, enquanto as linhas representam os intervalos de tempo. Nesse contexto, Turunen et al. [73] descrevem como novas modalidades de interface de usuário podem ser usadas para fornecer vários métodos de entrada e saída para interação com o EPG. As modalidades de interação apresentadas em [73] incluem gestos, entrada de fala e toque físico. A interface de gestos é usada junto com o teclado do telefone celular. Nesta modalidade de interação, diferentes orientações do celular mudam o funcionamento do teclado. Por exemplo, as teclas de seta vertical para baixo são usadas para mover a seleção no EPG e as teclas de seta horizontal executam funções de *zoom*.

Em [73], a interação por voz inclui comandos para navegação no aplicativo (por exemplo, "Vá para o guia de programação") e para assistir à mídia ("Vá para o canal de notícias"). Os autores também fornecem uma interface de toque físico por meio de uma placa de controle semelhante a um controle remoto clássico, mas em vez de botões, ela

possui etiquetas RFID atrás de ícones que se comunicam por meio de celular. Quando um usuário toca um desses objetos com um telefone celular, o comando é lido da *tag* e entregue ao sistema. De acordo com Turunen et al. [73], os ícones tornam o sistema mais fácil de usar do que um controle remoto clássico. Quando o usuário toca em qualquer ícone do controle, o comando associado é enviado ao EPG.

Na literatura, há diversas soluções propostas para trazer maior interatividade para o usuário, por meio do *middleware* Ginga. Em especial pela adição de dispositivos que funcionem como uma segunda tela de interação com a TV [57]. Dentre estas soluções, destaca-se a de Batista et al. [14]. Os autores propõem um módulo para o *middleware* Ginga que possibilita aplicações NCL utilizarem recursos de outros dispositivos secundários para controlar a exibição de conteúdo, interação e captura de eventos. Além disso, a solução permite criar *links* que são acionados de acordo com a identificação do dispositivo secundário. Um característica dessa solução é a utilização de uma API para gerenciamento e registro das classes de dispositivos que irão se comunicar com o dispositivo pai. Um caso de uso é a utilização de celulares para responder pesquisas, controlar a televisão e visualizar conteúdo em conjunto com a aplicação. É importante notar que nesse trabalho relacionado, são sugeridas novas modalidades de interação por meio de toque e foram lançadas as bases para o desenvolvimento de extensões ao Ginga.

O trabalho de Pedrosa et al. [58] especifica e implementa um componente que permite o recebimento de eventos multimodais por parte de aplicações em C++ residentes no Ginga. Na arquitetura proposta, módulos de comunicação devem enviar um documento XML contendo informações do evento, que por sua vez será traduzido por um *parser* para um objeto, informando a um gerenciador de eventos para notificar as aplicações que manifestaram o interesse pelo evento. No trabalho de Pedrosa et al. [58], a principal contribuição foi utilizar o subsistema Ginga para dar apoio à notificação de eventos de multimodalidade para aplicações C++. Em contraste, no presente trabalho, é proposta uma arquitetura que permite que o tratador de eventos do Ginga reconheça também eventos provenientes de outras modalidade de interação. Com isso, possibilita que aplicações declarativas em NCL sejam construídas com multimodalidade. Além disso, este trabalho propõe a parametrização destes eventos de multimodalidade, de forma a permitir também a interação multimodal por múltiplos usuários.

O trabalho de Carvalho et al. [20] propõe uma arquitetura de software focada na interação entre o usuário e a televisão por meio de sua voz com comandos vocais simples, com o objetivo de substituir funções do controle remoto. Para permitir tais aplicações, os

autores propõem diversos novos elementos na linguagem NCL, que são derivados da linguagem VoiceXML. A abordagem de extensão de NCL depende de modificações extensas no formatador da linguagem, porém, no trabalho não é especificado como o formatador deverá reagir. Além disso, não é especificado como o software de reconhecimento poderá ser acoplado ao Ginga-NCL.

Luque et al. [48] propõem a inserção de objetos 3D interativos na tela principal da televisão. Além disso, a proposta apresentada em [48] também permite a interação por meio de um dispositivo tátil portátil, ou seja, uma tela secundária. A segunda tela fornece informações adicionais, como estatísticas e visualizações adicionais apresentadas sob demanda. A proposta deles foi aplicada à TV interativa que é capaz de receber e reproduzir conteúdo proveniente de redes de difusão (como a televisão digital terrestre) e da Internet (ou seja, a rede de banda larga). É importante ressaltar que a interação multimodal proposta em [48] é caracterizada pela utilização de mais de um dispositivo de interação (*joystick* e *tablet*). Porém, em nossa proposta, a interação multimodal está relacionada ao uso de diferentes modos de interação, como voz, gesto ou reconhecimento do olhar.

Guedes et al. [32] propõem uma estrutura de programação de alto nível para apoiar interfaces de usuário multimodais em aplicações multimídia interativas. O framework integra diferentes tipos de modalidades de entrada e saída. Ou seja, suporta modalidades de entrada geradas pelo usuário, como gestos e reconhecedores de voz, e modalidades de saída, como conteúdo audiovisual tradicional, sintetizadores de fala e atuadores. O trabalho modela tipos de entrada multimodais que suportam novas modalidades de entrada, tais como gestos e reconhecimento de voz, e diferentes modalidades de saída, como os conteúdos audiovisuais tradicionais, sintetizadores de voz e atuadores. Para o reconhecimento de voz, os trechos a serem reconhecidos são definidos por meio de arquivos SRGS (*Speech Recognition Grammar Specification*) [1]. O *framework* proposto não foi implementado no Ginga-NCL. Essa proposta é bastante diferente da proposta desta tese, onde representam-se diferentes tipos de interação (voz, gesto, etc) através de diferentes tipos de eventos NCM. Sendo assim, não é necessário criar novos tipos de nós e âncoras NCM para reconhecer a ocorrência desses eventos, como *RecognitionNode* e *RecognitionAnchor* propostos em [32]. Além disso, fica transparente para o autor o domínio da gramática utilizada pelos reconhecedores.

Pereira et al. [60] propõem uma infraestrutura de software que tem por objetivo integrar o Ginga-NCL [2] com o M-Hub [70], um middleware voltado ao domínio da IoT

que permite a descoberta dinâmica, estabelecimento de conexão, acesso e distribuição de dados de/para objetos inteligentes. Uma limitação do trabalho é imposta pela limitação das informações que podem ser trocadas via script Lua pois é a única forma de capturar informações publicadas no *broker*. O trabalho não propõe extensões na linguagem NCL de forma que o autor da aplicação possa tratar interações vindas dos dispositivos inteligentes.

Na proposta de Farias et al. [22], o Ginga foi estendido com suporte ao MQTT, para prover integração entre IoT (*Internet of Things*) [28] e TV digital (TVD). O trabalho desenvolveu aplicativos para validar a estrutura proposta utilizando comunicação MQTT em uma rede doméstica. Informações coletadas da rede doméstica por sensores puderam ser apresentadas na TV. Assim, o trabalho adiciona novos recursos ao *middleware* Ginga, que permitiu a integração de receptores de TVD em aplicativos de IoT, por uma API desenvolvida em NCLua [62]. Porém esta abordagem mantém uma dependência com scripts NCLua onde há uma limitação na troca de informações com as aplicações NCL.

Abaixo segue a lista de funcionalidades para interação multimodal proposta por esta tese para NCL 4.0 e que foi implementada no middleware Ginga:

1. Gerenciamento nativo de multimodalidade em NCL sem utilizar scripts Lua
2. Extensível para outras modalidades
3. Início automático dos reconhecedores de modalidades de interação
4. Implementado no middleware Ginga.
5. Execução distribuída dos reconhecedores
6. Identificação do usuário no momento da interação.

A Tabela 3.1 apresenta uma comparação da proposta desta tese, chamada NCL 4.0, com outros trabalhos citados nesta seção.

3.2 Multiusuário

O trabalho de Guedes et al. apresentado em [30] se concentra em questões de especificação de interações multiusuário. E também como o autor define os requisitos de interação do usuário e usa informações de contexto. Para isso o trabalho apresenta uma nova entidade *UserClass* que representa diferentes tipos de usuários. Além disso, tem uma descrição

Funções	1	2	3	4	5	6
NCL 4.0	sim	sim	sim	sim	sim	sim
NCL 3.0 [64]	não	não	não	não	não	não
Turumen [73]	sim	sim	não	não	não	não
Batista [14]	sim	sim	não	sim	sim	não
Pedrosa [58]	sim	sim	não	sim	não	sim
Carvalho [20]	não	não	não	sim	sim	não
Luque [48]	sim	não	não	não	sim	não
Guedes [32]	sim	sim	não	não	sim	sim

Tabela 3.1: Tabela comparativa dos trabalhos relacionados a multimodalidade

de perfil por meio da classe *userClassDescription* que referencia arquivo do tipo RDF (Resource Description Framework) [46] . Suas propriedades podem ser armazenadas no *SettingsNode*. Não é abordado como o sistema multimídia deve reunir a descrição do perfil dos usuários e a recuperação de suas variáveis de contexto de tempo de execução. Além disso, todas as características de todos os usuários são armazenadas em um único nó de propriedades do documento multimídia, *SettingsNode*. Neste nó, é armazenado um vetor para cada propriedade que se deseja manter dos usuários. A proposta desta tese se aproxima da abordagem de [30], à medida que foi criada também uma entidade para representar a classe de usuários. Porém difere em alguns aspectos, pois foram criadas duas entidades para representar os usuários individualmente, uma para identificá-lo e outra para armazenar suas características, sendo representada por um nó de propriedades.

McGill et al. [51] discutem o uso da TV e multitelas, os problemas que esse uso apresenta com relação ao papel da TV em contextos sociais compartilhados e o impacto potencial que novas tecnologias podem ter sobre como usamos e interagimos com a televisão. O uso compartilhado da TV é problemático, tanto do ponto de vista da interação quanto da incapacidade de usar a TV de forma independente, sem afetar o uso de terceiros. Atualmente, os usuários superam esses problemas por meio de várias telas, mas isso também é problemático do ponto de vista social, com potencial para aumento do isolamento digital e falta de compartilhamento com relação aos presentes. McGill et al. demonstram como esses problemas podem ser resolvidos com design de interação de TV, apresentando maneiras em que o uso multiusuário pode ser facilitado por meio de interfaces de uso compartilhado e multivisão, e examinando como a TV pode permitir maior compartilhamento e, portanto, consciência da atividade do dispositivo propondo dois tipos de interface. A interação mediada e a interação concorrente. Com isso, foi demonstrado que a TV é capaz de fazer substancialmente mais do que atualmente é solicitado; ao contrário do uso existente, pode ser de relevância crescente na era multiusuário e multitelas.

Outra questão interessante, discutida em [51], foi a possibilidade de que todos os usuários pudessem estar no controle da interação. Destacando-se: *subsets* – onde diferentes membros do grupo têm o controle de diferentes funções exigindo assim cooperação; *hierarchy* – onde a interação de um membro sobrepõe a do outro; *plurality* – a decisão da seleção é baseada na maioria dos votos, mas a navegabilidade é concorrente e, finalmente, *blocking* – permite que membros possam bloquear temporariamente outro membro do controle. Em [51], os autores não apresentam uma arquitetura para prover interação multiusuário, mesmo porque o objetivo do artigo é mostrar como os comportamentos existentes para compartilhamento de uso podem ser reaproveitados e virtualizados.

Abaixo segue a lista de funcionalidades para aplicações multiusuário proposta por esta tese e que foi implementada em NCL 4.0 e no middleware Ginga:

1. Identificação do usuário.
2. Gerenciamento nativo de suporte multiusuário em NCL
3. Validação de perfil de usuário
4. Carregamento automático das propriedades do usuários, conforme seu perfil ou descrição individual
5. Uso do MPEG-21 parte 22 [39] para especificação de propriedades dos usuários
6. Implementado no middleware Ginga.

A Tabela 3.2 apresenta uma comparação de NCL 4.0 com outros trabalhos citados nesta seção.

Funções	1	2	3	4	5	6
NCL 4.0	sim	sim	sim	sim	sim	sim
NCL 3.0 [64]	não	não	não	não	não	sim
Guedes [30]	não	sim	sim	não	não	não
Mark McGill [51]	não	não	não	não	não	não

Tabela 3.2: Tabela comparativa dos trabalhos relacionados a multiusuário

Capítulo 4

Proposta de Extensões ao Modelo NCM e à Linguagem NCL

A Linguagem NCL *Nested Context Language*) é baseada no modelo conceitual NCM (*Nested Context Model*) [66]. Esta tese propõe extensões ao modelo NCM e à linguagem NCL para incluir as facilidades de interação multimodal e suporte multiusuário. As próximas seções detalham as extensões propostas.

4.1 Extensões ao Modelo NCM

A extensão ao Modelo NCM (*Nested Context Model*) proposta nesta tese se divide em duas partes. A primeira parte diz respeito a novas entidades NCM para suporte a múltiplos usuários, proposta que estende as ideias de Guedes [32], e também informações de contexto, considerando a abordagem apresentada por Josué et al. [42]. A segunda parte diz respeito à criação de novos tipos de eventos no modelo NCM para dar suporte à interação multimodal.

4.1.1 Multiusuário

Em relação à modelagem dos usuários que participam de uma experiência multimídia, este trabalho representa um usuário individualmente com suas características específicas que poderiam, por exemplo, ser carregadas de um perfil especificado em uma arquivo XML no padrão MPEG-21 parte 22 [39]. As características podem conter desde preferências de tipos de mídia até limiares de percepções como volume de som, luminosidade, etc.

Para representar os usuários individualmente de maneira específica, é proposta a en-

tidade *userAgent*. Um *userAgent* indica um único usuário e tem o atributo *src*, que indica a especificação de suas características específicas.

Para representar um perfil de usuário, foi proposta a entidade *userProfile*, que contém o atributo *id* identificando o perfil, *src* para indicar as características do perfil de usuário em questão por meio de um arquivo de propriedades e por fim, os atributos *min* e *max* definindo número mínimo e máximo de usuários participantes com determinado perfil.

Para que as características de um usuário específico ou de um perfil possam ser usadas em elos entre nós de um documento NCM, é proposta a entidade *UserSettingsNode*, que faz referência a um usuário (*userAgent*) ou perfil (*userProfile*), permitindo o uso de eventos de atribuição relacionados a propriedades desse nó. A Figura 4.1 ilustra tais entidades.

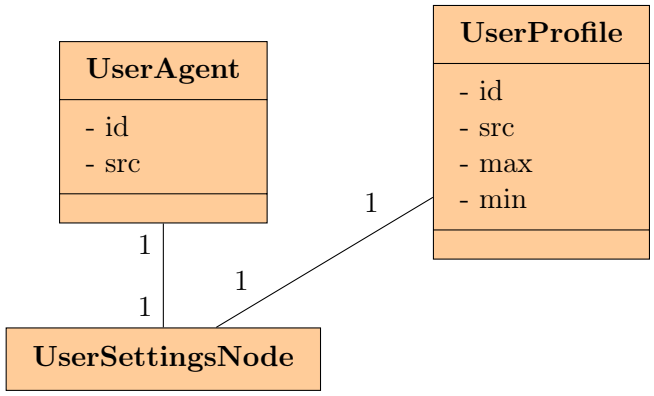


Figura 4.1: Entidades para suporte multiusuário

4.1.2 Informação Contextual

No modelo NCM, a entidade *SettingsNode* é do tipo *ContentNode*, portanto herda todas as características especificadas em nó de conteúdo como por exemplo suas propriedades. Esta tese propõe que a representação de informações contextuais tanto de ambiente como de usuário sejam modeladas como um subtipo de *SettingsNode*. A Figura 4.2 ilustra os dois tipos específicos de nó, *AmbientSettingsNode* e *UserSettingsNode*, ambos subtipos do nó *SettingsNode* de NCM. Na figura, uma lista de atributos reduzida é apresentada para exemplificar diferentes características de usuários e ambientes que podem ser modeladas. Na entidade *UserSettingsNode*, o atributo *heartbeat* contém o batimento cardíaco de um usuário específico, *temperature* a sua temperatura corporal, e o *age* sua idade. Já o atributo *user*, que relaciona esta entidade à entidade *UserAgent*, identifica o usuário diante dos outros como foi descrito na Seção 4.1.1. Desta forma, possibilita-se carregar todas as características associadas ao *UserAgent* no nó *UserSettingsNode* correspondente, assim suas informações individuais estarão acessíveis por meio de um nó de conteúdo. É

importante ressaltar que outros atributos podem ser adicionados às entidades *AmbientSettingsNode* e *UserSettingsNode*, além do fato de que um documento NCM pode conter mais de uma instância dessas entidades, diferente de como acontece com *SettingsNode*, que só pode conter uma ocorrência por documento na especificação de NCL 3.0 [68].

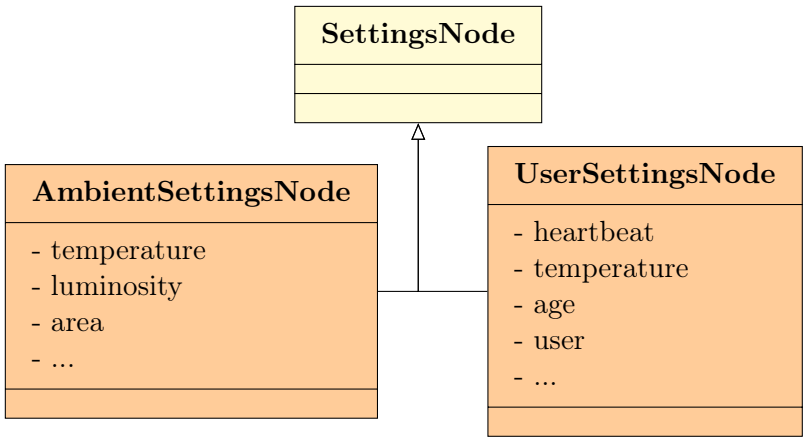


Figura 4.2: Nós para representação de características de usuários e de ambientes

É importante notar que os nós apresentados na Figura 4.2 são capazes de armazenar informações coletadas por sensores presentes na instalação física (e.g. temperatura, luminosidade, umidade, etc) ou ligados ao usuário (e.g. batimento cardíaco, pressão sanguínea, etc.). Essa é uma abordagem que permite ao autor da aplicação considerar o estado do ambiente ou do usuário para a definição do comportamento da aplicação ou sua adaptação.

4.1.3 Interação Multimodal

O suporte à interação multimodal pode trazer uma sobrecarga a mais para a autoria de aplicações multimídia. Assim, seria ideal que autores que desenvolvem essas aplicações multimídia não se preocupem com detalhes sobre a implementação da captação da interação ou a definição de outras linguagens específicas como SRGS (*Speech Recognition Grammar Specification*), como ocorre na proposta de Guedes et al. [29]. A integração de novos tipos de interação em sistemas multimídia, além das modalidades tradicionais por mouse e teclado, é um tema abordado em diferentes propostas na literatura [14, 23, 57]. De acordo com Lima et al. [23], o uso da linguagem natural juntamente com o gesto pode superar as limitações da interação de apenas uma modalidade. Isto porque a combinação de fala e gesto fornece um comportamento comunicativo altamente eficiente para interagir com aplicativos em uma experiência mais transparente que as interfaces tradicionais.

O modelo NCM já prevê diversos eventos relacionados com a interação do usuário e

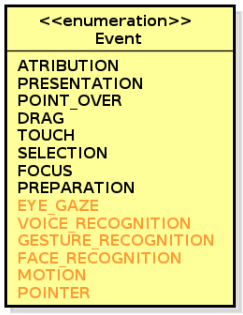


Figura 4.3: Eventos do Modelo NCM estendido.

também atributos que são alterados com a ocorrência destes eventos. Conforme Soares et al. [66] e Josué et al. [43], os eventos podem ser dos seguintes tipos: apresentação, composição, seleção, superposição, arraste, foco, atribuição e preparação. A evolução tecnológica possibilita a exploração de outras modalidades de interação como reconhecimento de gestos ou voz. Para isso, este trabalho propõe novos tipos de eventos NCM. Por exemplo, dispositivos que mapeiam o movimento dos olhos podem ser utilizados para disparar o evento *eyeGaze*. Portanto, esta tese propõe outros eventos de interação do usuário, além do evento de seleção, tais como: *touch*, *motion*, *eyeGaze*, *pointer*, *voiceRecognition*, *gestureRecognition*, *handPoseRecognition* ou *faceRecognition*. Eles representam respectivamente a interação por toque, movimento do corpo, fixação do olhar, apontamento, reconhecimento de voz, reconhecimento de gesto, reconhecimento de pose da mão e reconhecimento de expressão facial.

Tais eventos possuem atributos específicos que especializam o evento. Assim como o atributo *key* indica a tecla que foi selecionada em um evento de seleção, esse atributo é usado de forma similar pelos outros tipos de evento. Por exemplo, para um evento de *gestureRecognition*, o *key* igual a *handUp* indica a ocorrência de um gesto de levantar a mão. A definição da semântica dos valores de tal atributo, ou seja, dos tipos de gestos, comandos de voz, expressões faciais, etc., é dependente do tipo de evento em questão. A Figura 4.3 mostra os tipos de evento propostos por esta tese destacados com a fonte de cor laranja e a Tabela 4.1 descreve cada um deles.

Uma vez definidos diferentes usuários na aplicação, o autor pode definir relacionamentos com eventos que ocorram somente quando determinados perfis de usuário (ou um usuário específico) realizam uma ação. Para permitir essa nova funcionalidade, os eventos e conectores NCM foram estendidos também. Um evento de interação possui, além de seus atributos definidos por NCM, um atributo *user* que identifica o usuário que disparou o evento durante a execução da aplicação. Seguindo a abordagem de Guedes et al. [29],

Nome de Evento	**Descrição do tipo de interação**
touch	interação por toque
bodyMotionRecognition	movimento do corpo
eyeGaze	fixação dos olhos
pointer	apontamento
voiceRecognition	reconhecimento de voz
gestureRecognition	reconhecimento de gesto
faceRecognition	reconhecimento de expressão facial
handPoseRecognition	reconhecimento de poses das mãos

Tabela 4.1: Novos eventos de interação propostos para NCM

conectores, mais especificamente suas condições, também possuem um atributo adicional *user*, permitindo a identificação do perfil do usuário na condição de uma relação causal. Quando da execução da aplicação, o reconhecimento ou validação do usuário poderá ser feito de várias maneiras: usando identificação por RFID [78], reconhecimento facial, *scanner* ou até mesmo temperatura corporal.

A Figura 4.4 apresenta o modelo NCM estendido conforme as propostas desta tese. As cores da figura ajudam a identificar o escopo deste trabalho. As entidades em amarelo fazem parte do modelo NCM 3.0 já existente. As entidades de cor laranja estão sendo propostas por este trabalho. Pode-se observar as entidades *UserSettingsNode* e *AmbientSettingsNode* como subclasses da entidade *SettingsNode*, portanto herdando todas as suas características como foi dito anteriormente. Pode-se observar também que as entidades *UserAgent* e *UserProfile*, ambas subclasses de *Entity*, podem ser associadas com *UserSettingsNode*. A associação de *UserAgent* e *UserProfile* com *UserSettingsNode* é de um pra um. A Figura 4.4 inclui também as entidades (em azul) relacionadas a efeitos sensoriais, que foram propostas em Josué et al. [42].

Além da extensão do modelo NCM, apresentada na Figura 4.4, este trabalho propõe uma extensão da linguagem NCL de maneira que o autor consiga usar novas facilidades para representação de múltiplos usuários e diferentes eventos de interação em sua aplicação multimídia. A extensão da linguagem NCL é discutida na próxima seção.

4.2 NCL 4.0

Para implementar as alterações no modelo NCM descritas na Seção 4.1, esta tese e os trabalhos [9, 11–13, 42, 52, 74] propõem uma extensão da linguagem NCL (*Nested Context Language*) denominada NCL 4.0. Diversas alterações são sugeridas para expressar as

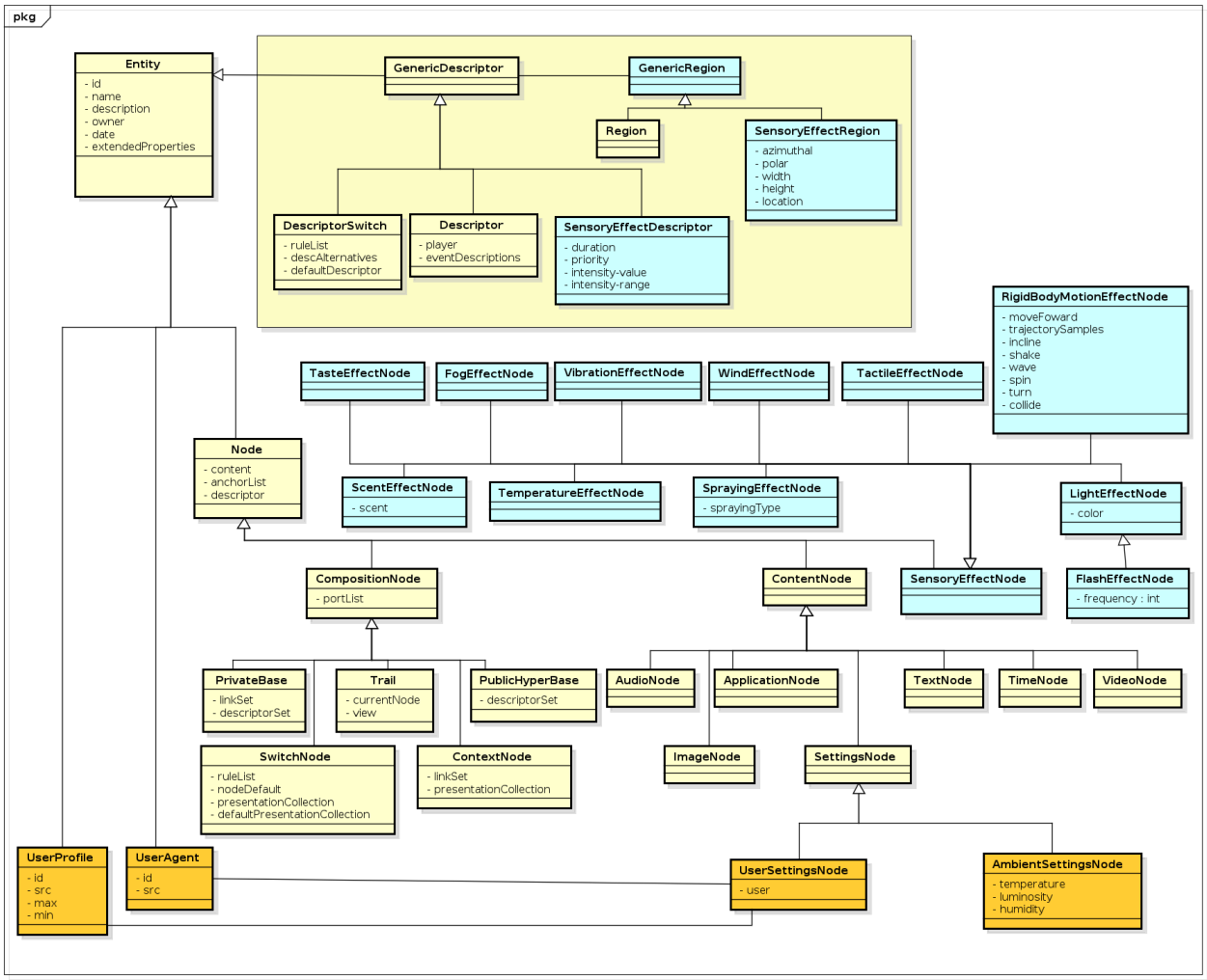


Figura 4.4: Modelo NCM estendido.

entidades idealizadas na extensão do modelo proposta neste trabalho. A proposta de NCL 4.0 é especificada utilizando XML *Schema* [71]. O esquema proposto foi estendido do esquema da linguagem NCL 3.0[1]. A extensão proposta contém os novos elementos além de alterações nos elementos existentes para contemplar o que foi modelado[2].

A linguagem NCL 4.0 é divida em módulos e estes são agrupados em área funcionais. Esta versão propõe uma nova área funcional *Users* contendo o novo módulo *User*. A proposta de Josué [44] inclui o novo módulo *Effect* dentro da área funcional *Components*.

Esta tese foca nas entidades relacionadas com interação multimodal e multiusuário em aplicações multimídia. Para isso, propõe alteração no módulo *Media* da área funcional *Components*, onde foi modificado o elemento *media* acrescentando mais dois tipos possíveis de nós *settings*, "*ambient-settings*"e "*user-settings*". Os novos eventos de interação foram contemplados com a alteração do módulo *CausalConnectorFunctionality* da área funcional *Connectors* com adição de novos tipos de evento e também novos papeis de condição predefinidos. Como dito anteriormente, foi acrescentada uma nova área funcional denominada *Users*, contendo o módulo *User*, que define os elementos *userAgent* e *userProfile*.

A representação das entidades *UserAgent* e *UserProfile* é feita pela adição dos elementos XML <*userAgent*> e <*userProfile*>. Tais elementos são definidos para representar um perfil ou um usuário individualmente. O elemento <*userAgent*> possibilita ao autor criar uma aplicação sob medida com a representação de um usuário individualmente. A Tabela 4.2 apresenta seus atributos. Em *src* é especificado o arquivo XML com as características do usuário seguindo o formato MPEG-21 parte 22 [39] e que serão carregadas para o nó *userSettings* correspondente. Desta forma, o autor poderá criar links associados a essas propriedades. Além disso, os eventos de interação pode ser condicionados a usuários especificados com a cláusula <userAgent>. A Listagem 4.1 apresenta um exemplo da definição de um usuário específico em NCL 4.0.

```
1 <userBase>
2   <userAgent id="userA" src="user1.xml"/>
3 </userBase>
```

Listagem 4.1: Definição de características de um usuário participante da aplicação

Outra maneira de representar os usuários é por meio do elemento <*userProfile*>, da mesma forma que <*userAgent*>, o perfil também tem o atributo *src* que indica o cami-

[1]http://www.ncl.org.br/pt-br/schemasxml
[2]https://github.com/FabioBarr/NCL4.0.git

Atributo	Descrição	Valor	Obrigatoriedade
id	Identifica o elemento dentro da aplicação NCL.	Qualquer cadeia de caracteres que comece com uma letra ou um sublinhado ("_") e que contenha apenas letras, dígitos, "."e "_".	Obrigatório
src	Define a URI do arquivo com as propriedades do usuário que serão carregadas.	Devem obrigatoriamente estar de acordo com a Tabela 11 em [3]	Opcional

Tabela 4.2: Descrição dos atributos do elemento <userAgent>

nho para o arquivo com as propriedades do perfil, conforme definido na Tabela 4.3. A Listagem 4.2 apresenta um exemplo da definição de um perfil de usuário. Porém, neste caso as propriedades do perfil serão utilizadas para verificar se os usuários encontrados no receptor de TV digital (especificados por arquivos XML, por exemplo) correspondem às propriedades definidas no perfil. Caso haja, as propriedades dos usuários serão carregadas em nós do tipo *userSettings* correspondentes, declarados no corpo do documento NCL. Assim os usuários que atendam ao perfil terão suas propriedades consideradas pela aplicação. Além disso, para cada link criado indicando este perfil no parâmetro *user* do papel do conector usado pelo link, serão criados links dinâmicos para todos usuários que atendem o perfil. Tais links terão a identificação do usuário que foi especificada no arquivo de propriedades do usuário. Na Listagem 4.3, pode-se ver um exemplo de link com perfil. Quando esta aplicação foi executada pelo Ginga, este link que referencia um perfil de usuário gera links dinâmicos apresentados na Listagem 4.4 para um ambiente que tenha os usuários identificados por *U01* e *U02* atendendo ao perfil especificado por meio do perfil *profile1*.

```
1 <userBase>
2   <userProfile id="profile1" max="2" src="profile1.xml"/>
3 </userBase>
```

Listagem 4.2: Definição de características de um perfil de usuário participante da aplicação

```
1 ...
2     <link xconnector="onVoiceRecognitionStop">
3           <bind role="onVoiceRecognition" component="florestaVideo">
4             <bindParam name="key" value="parar"/>
5             <bindParam name="user" value="profile1"/>
6           </bind>
```

```
7           <bind role="stop" component="florestaVideo"/>
8       </link>
9   ...
```

Listagem 4.3: Definição de link para usuários que atendem o perfil “profile1”

```
1   ...
2       <link xconnector="onVoiceRecognitionStop">
3           <bind role="onVoiceRecognition" component="florestaVideo">
4             <bindParam name="key" value="parar"/>
5             <bindParam name="user" value="U01"/>
6           </bind>
7           <bind role="stop" component="florestaVideo"/>
8       </link>
9       <link xconnector="onVoiceRecognitionStop">
10          <bind role="onVoiceRecognition" component="florestaVideo">
11            <bindParam name="key" value="parar"/>
12            <bindParam name="user" value="U02"/>
13          </bind>
14          <bind role="stop" component="florestaVideo"/>
15      </link>
16  ...
```

Listagem 4.4: Definição de links dinâmicos gerados a partir de um link que tenha um perfil de usuário como valor do parâmetro “user”.

E finalmente, outra possibilidade de se ter a participação do usuário identificada é sem a utilização do *<userAgent>* e *<userProfile>*. Basta colocar a palavra *“all”* no parâmetro *user* dos links construídos que, de forma análoga ao que ocorre com um perfil específico, serão criados *links* dinâmicos para todos os usuários encontrados no receptor de TV digital sem fazer a validação do perfil. Nós do tipo *userSettings* com valor *“all”* resultarão na criação de outros nós do tipo *userSettings* dinamicamente, um para cada usuário encontrado no receptor de TV digital.

Note que a definição de usuários e/ou perfis e o uso do atributo *user* nos *links* é opcional. Caso não seja utilizado, qualquer usuário poderá interagir com a aplicação e habilitar a execução dos *links*.

As entidades *UserSettingsNode* e *AmbientSettingsNode* são expressas por meio de modificações no elemento *<media>* onde foram adicionados dois novos tipos, um para repre-

Atributo	Descrição	Valor	Obrigatoriedade
id	Identifica o elemento dentro da aplicação NCL.	Qualquer cadeia de caracteres que comece com uma letra ou um sublinhado ("_") e que contenha apenas letras, dígitos, "."e "_".	Obrigatório
src	Define a URI do arquivo com as propriedades do perfil de usuário.	Devem obrigatoriamente estar de acordo com a Tabela 11 em [3]	
min	Define o número mínimo de usuários que podem seguir este perfil na aplicação NCL.	número inteiro maior ou igual a zero (valor default é zero)	Opcional
max	Define o número máximo de usuários que podem seguir este perfil na aplicação NCL.	número inteiro maior ou igual ao valor do atributo min ou "unbounded" (valor default é "unbounded")	Opcional

Tabela 4.3: Descrição dos atributos do elemento <userProfile>

sentar características do usuário e outro para representar características do ambiente. Tais características podem ser estáticas lidas de arquivos ou dinâmicas lidas de sensores presentes no ambiente e/ou no usuário. Porém a linguagem NCL 3.0 não dá suporte a isso, pois conforme apresentado na norma ABNT NBR15606-2 [2], é permitido o armazenamento de variáveis em NCL 3.0 por meio de um nó *<media>* do tipo *application/x-ncl-settings*. Estas podem ser globais definidas pelo autor ou de ambiente reservadas para o sistema Ginga. Alguns valores de propriedades do nó do tipo *application/x-ncl-settings* podem ser modificadas através do uso de elos NCL. Porém, só é permitido um nó do tipo ncl-settings em cada documento NCL. Neste contexto, este trabalho propõe adicionar mais dois tipos de mídia: "*ambient-settings*"e "*user-settings*"que armazenam informações do ambiente e do usuário respectivamente. Estes com a cardinalidade maior que 1 como os outros tipos de nó *<media>*, além de suas informações poderem ser advindas de sensores que deverão atualizar estes dados periodicamente ou definidos pelo autor do documento NCL. O nó *<media>* do tipo *ambient-settings* (*application/x-ncl-ambient-settings*) poderá usar a propriedade *descriptor* para especificar um descritor que contém características de leitura das informações dos sensores como por exemplo, frequência de captura. A frequência de leitura indica a cada quanto tempo a mídia será atualizada com dados lidos do sensor. Um elemento *descriptor* associado a um nó *ambient-settings* pode referenciar o elemento *region* no qual define a região de captura de dados.

A Listagem 4.5 apresenta um exemplo da utilização de *ambient-settings*. O nó com id *propLumRoom* descreve variáveis que irão armazenar características do ambiente, neste caso luminosidade e temperatura. Este nó contém uma propriedade para cada característica.

```
1 <media type="application/x-ncl-ambient-settings" id="propLumRoom" >
2   <property name="luminosity" />
3   <property name="temperature" />
4 </media>
```

Listagem 4.5: Definição de *AmbientSettingsNode*

O nó <*media*> do tipo *user-settings* (*application/x-ncl-user-settings*) funciona de maneira semelhante ao *ambient-settings*. A principal diferença é que os sensores não estão associados a um ambiente, mas sim a um usuário especificamente. Além de poder armazenar também características estáticas vindas de arquivos armazenados localmente ou repositórios pré-definidos. Tal arquitetura será definida no Capítulo 5. Para realizar a associação de sensores a um usuário, deve-se associar a mídia *user-settings* a um *id* de usuário. A Listagem 4.6 apresenta um exemplo da utilização de *user-settings*. Pode ser observado que o *user-settings* do exemplo está associado ao *id* do usuário *userA* definido na Listagem 4.1. Desta forma, todas as características contidas no XML com as propriedades do usuário, poderão ser carregadas para o *userSettingsNode* associado a um *userAgent*.

```
1 <media type="application/x-ncl-user-settings" id="propUserA" user="
   userA">
2   <property name="heartBeat" />
3   <property name="face" />
4 </media>
```

Listagem 4.6: Definição de *UserSettingsNode*

Em NCL, utilizando conectores, relacionamentos de interação do usuário podem ser definidos como especificado por Soares et al. [64]. E a definição de um conector é feita por um conjunto de papéis que determinam a função dos participantes da relação. Cada papel descreve um evento associado a um participante da relação. Os eventos de interação possuem o atributo *key* que especifica o que foi reconhecido na interação do usuário. Na linguagem NCL, a interação multimodal é expressa com adição de eventos e novos papéis com os valores possíveis para o atributo *key*. Essa adição de eventos, através dos papeis, segue o que foi modelado na Seção 4.1 representado na Figura 4.4. Para facilitar o uso

desses novos eventos em NCL, são adicionados novos nomes reservados de papéis para o elemento <*simpleCondition*>. A Tabela 4.4 apresenta os papeis relacionados a eventos de interatividade multimodal assim como os possíveis valores para o atributo *key* propostos nesta tese.

Papel	Descrição	Atributo key
onVoiceRecognition	Quando algo dito pelo usuário for reconhecido enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	trecho de voz reconhecido
onFaceRecognition	Quando alguma expressão facial do usuário for reconhecida enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	tipo da expressão facial do usuário
onHandPoseRecognition	Quando algum gesto feito pelo usuário for reconhecido enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	tipo do gesto do usuário
onEyeGaze	Quando o usuário fixar o olhar em um região enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	-
onBodyMotionRecognition	Quando algum movimento corporal feito pelo usuário for reconhecido enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	tipo do movimento do corpo do usuário
onTouch	Quando alguma interação por toque feita pelo usuário for reconhecida enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	tipo do toque
onPointer	Quando um apontamento realizado pelo usuário for reconhecido enquanto o objeto associado a esse papel estiver sendo apresentado ou com foco.	posição do apontamento

Tabela 4.4: Papeis propostos para contemplar as interações multimodais

Conforme o trabalho de [42], NCL 4.0 contém um novo módulo chamado *Effect* dentro da área funcional *Components*, que define o elemento *effect* e seus atributos. Consequentemente os módulos *Layout* e *Descriptor* foram alterados. Mais especificamente, o *Layout* foi alterado ao acrescentar atributos ao elemento *region* e *Descriptor* foi alterado ao acres-

centar atributos ao elemento *descriptor*. As entidades NCM do tipo *SensoryEffectNode* são implementadas através do elemento <*effect*> em NCL 4.0, junto com seus atributos *id*, *type* e *descriptor*. Tais atributos vão identificar, especificar e descrever as características da ocorrência do efeito sensorial. A propriedade *type* será descrita seguindo a nomenclatura de tipos de efeitos sensoriais definidos no padrão MPEG-V [79]. Em Josué et al. [44], há uma discussão aprofundada sobre a implementação e descrição de elementos que integram efeitos sensoriais em uma aplicação multimídia imersiva.

4.2.1 Considerações Finais

Este capítulo apresentou a propostas de extensão tanto do modelo NCM quanto da linguagem NCL. Essas propostas foram consolidadas em uma nova versão da linguagem NCL, chamada NCL 4.0. Para isso, o capítulo detalhou as novas entidades que fazem parte da extensão assim como as alterações que abrangem o modelo e a linguagem. O capítulo apresentou as duas principais contribuições desta tese em seções que falam sobre multiusuário, informações de contexto e multimodalidade. NCL 4.0 reúne facilidades para especificação de aplicações multissensoriais interativas e multiusuário, possibilitando a evolução da TV digital interativa para TV imersiva multimodal e multiusuário.

O próximo capítulo apresenta a implementação de NCL 4.0 no *middleware* Ginga-NCL considerando os elementos propostos, assim como alterações necessárias para que o Ginga-NCL contemple a proposta. Desta forma, possibilitando a implementação e a execução de aplicações NCL 4.0 com as funcionalidades citadas. A implementação atual ainda não contempla o nó do tipo *ambient-settings*, que é trabalho futuro.

Capítulo 5

Implementação no Middleware Ginga-NCL

O Ginga-NCL é o subsistema do *middleware* Ginga, padrão do Sistema Brasileiro de TV Digital, que permite a execução de aplicações multimídia especificadas na linguagem NCL [2]. Atualmente, o subsistema tem como principal meio de interação de um usuário, o controle remoto ou mouse. Visando dar suporte a outras modalidades de interação, suporte multiusuário e armazenamento de informações contextualizadas por usuários (ou grupo de usuários) nas aplicações, esta tese propõe a inclusão de novos componentes no *middleware*[1].

Conforme a recomendação ITU-T H.761 [40], a arquitetura do *middleware* Ginga é composta por subsistemas que possuem um conjunto de componentes que implementam funcionalidades específicas, como mostra a Figura 5.1. O componente núcleo do subsistema *Ginga-NCL Presentation Environment* é o *NCL Player* ou *NCL Formatter*, responsável por controlar a execução da aplicação multimídia. Para que o middleware Ginga dê suporte à execução de aplicações especificadas em NCL 4.0, o *NCL Formatter* foi modificado, através da inserção de componentes novos (*Presentation Orchestrator, Preparation Orchestrator, Sensory Device Calibrator, Sensory Effect Renderer Manager, Interaction Manager*), e alteração de componentes já existentes (*Player Manager* e *NCL Context Manager*), conforme ilustrado na Figura 5.2.

Os componentes *Presentation Orchestrator, Preparation Orchestrator, Sensory Device Calibrator, Sensory Effect Renderer Manager* foram inseridos pela tese de Josué [44]. Já os componentes *Interaction Manager, User Context Manager* e *Ambient Context Manager* são propostos por esta tese.

[1] A implementação de extensão do Ginga-NCL apresentada neste trabalho pode ser acessada em bit.do/gingaMultimodalBranch.

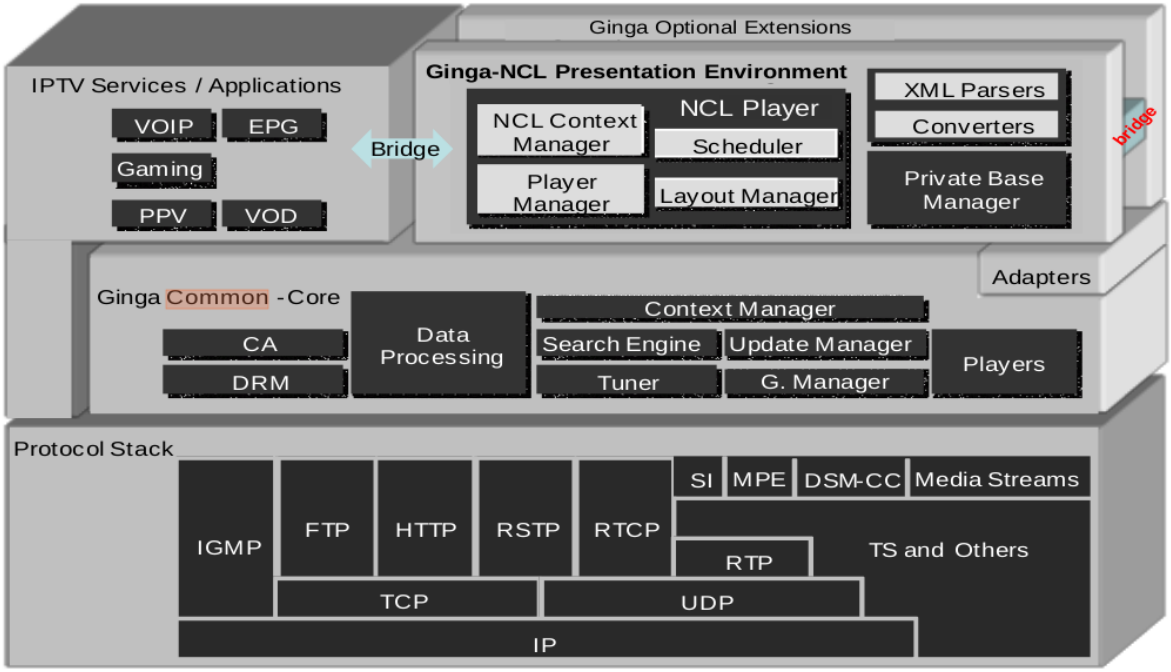


Figura 5.1: Visão de módulos da Arquitetura do Ginga [40]

Para possibilitar a preparação automática de conteúdo de mídia e efeitos sensoriais, o componente *Preparation Orchestrator* foi inserido no *NCL Formatter*. Além disso, o *Sensory Device Calibrator* foi adicionado para possibilitar a calibragem de renderizadores de efeitos sensoriais. O controle da execução da aplicação e a construção do plano de apresentação é realizado pelo componente *Presentation Orchestrator* proposto por Josué [44]. Por fim, o componente *Sensory Effect Renderer Manager* foi incorporado ao *NCL Formatter* para permitir o gerenciamento de renderizadores de efeito sensoriais.

Os componentes *User Context Manager* e *Ambient Context Manager* foram inseridos com especializações no componente *NCL Context Manager*, a fim de dar suporte ao gerenciamento de contexto do usuário e ambiente, respectivamente. Já o *Interaction Manager* foi inserido no *NCL Formatter* para dar suporte às diversas modalidades de interação do usuário, propostas por esta tese. Esses componentes serão detalhados na Figura 5.3.

A proposta desta tese foi submetida à Chamada da TV 3.0 do Fórum SBTVD[2] para especificação da camada de codificação de aplicações para o futuro sistema de TV Digital no Brasil.

Para o desenvolvimento da proposta, foi utilizada a implementação de referência da máquina de apresentação[3] do Ginga-NCL. A Figura 5.3 ilustra a arquitetura da imple-

[2]https://forumsbtvd.org.br/tv3_0/
[3]https://github.com/TeleMidia/ginga

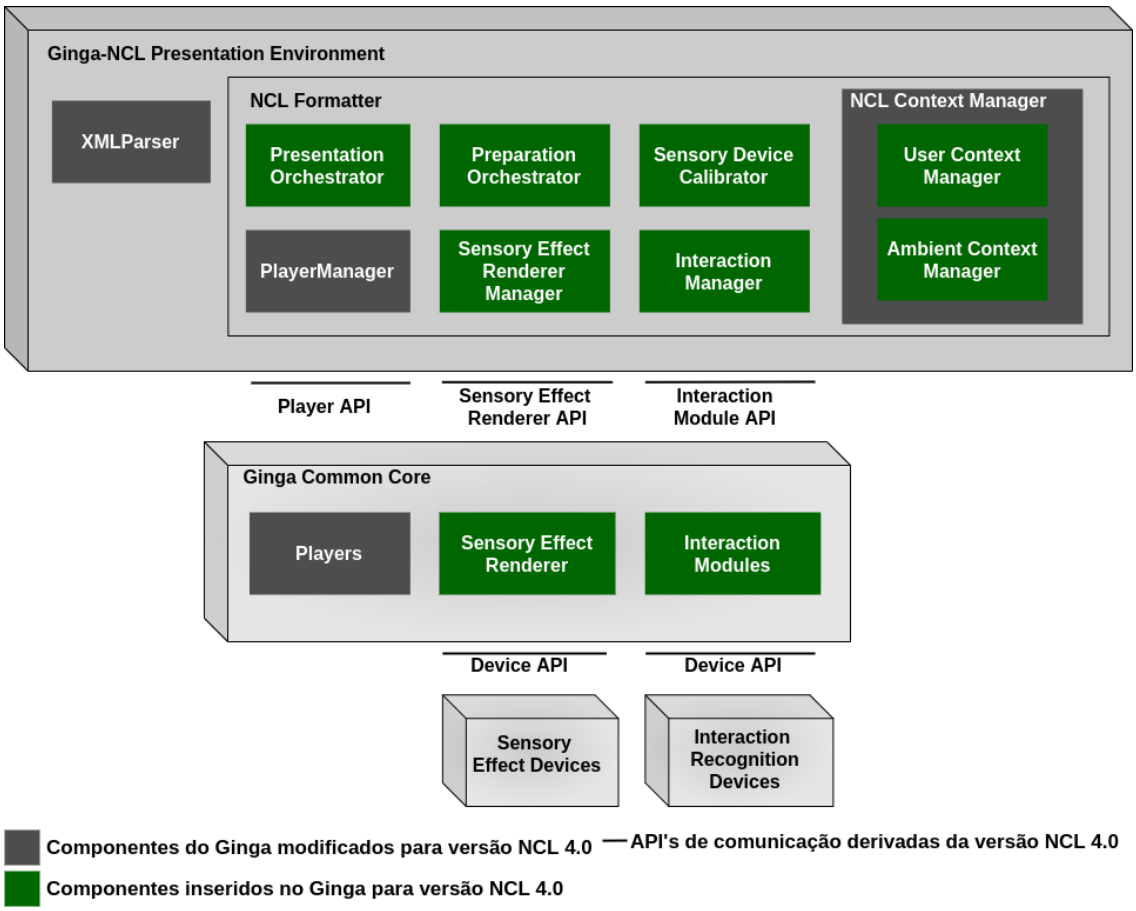


Figura 5.2: Visão de módulos da Arquitetura Estendida do Ginga para suporte à NCL 4.0

mentação estendida do *middleware* Ginga-NCL.

5.1 Interação multimodal e multiusuário

O componente *Formatter*, presente na atual versão do Ginga, é o responsável por controlar a apresentação dos objetos de mídia que compõem uma aplicação. Além disso, este elemento manipula os eventos de interação do usuário com a aplicação. Ao iniciar uma aplicação multimídia, o *Formatter* envia o documento NCL para o *Parser*, que extrai as informações sobre os elementos que compõem a aplicação e as relações de sincronização espaço-temporal definidas pelo autor da aplicação multimídia.

O controle das interações do usuário com a aplicação multimídia é realizado pelo componente *InteractionManager* proposto neste trabalho. É por meio do *InteractionManager* que os dispositivos de interação se comunicam com o *Formatter*, notificando-o quando ocorre uma interação por parte do usuário. O *InteractionManager* ativa os diferentes módulos de interação, conforme especificado pelo documento NCL, e passa, para cada um

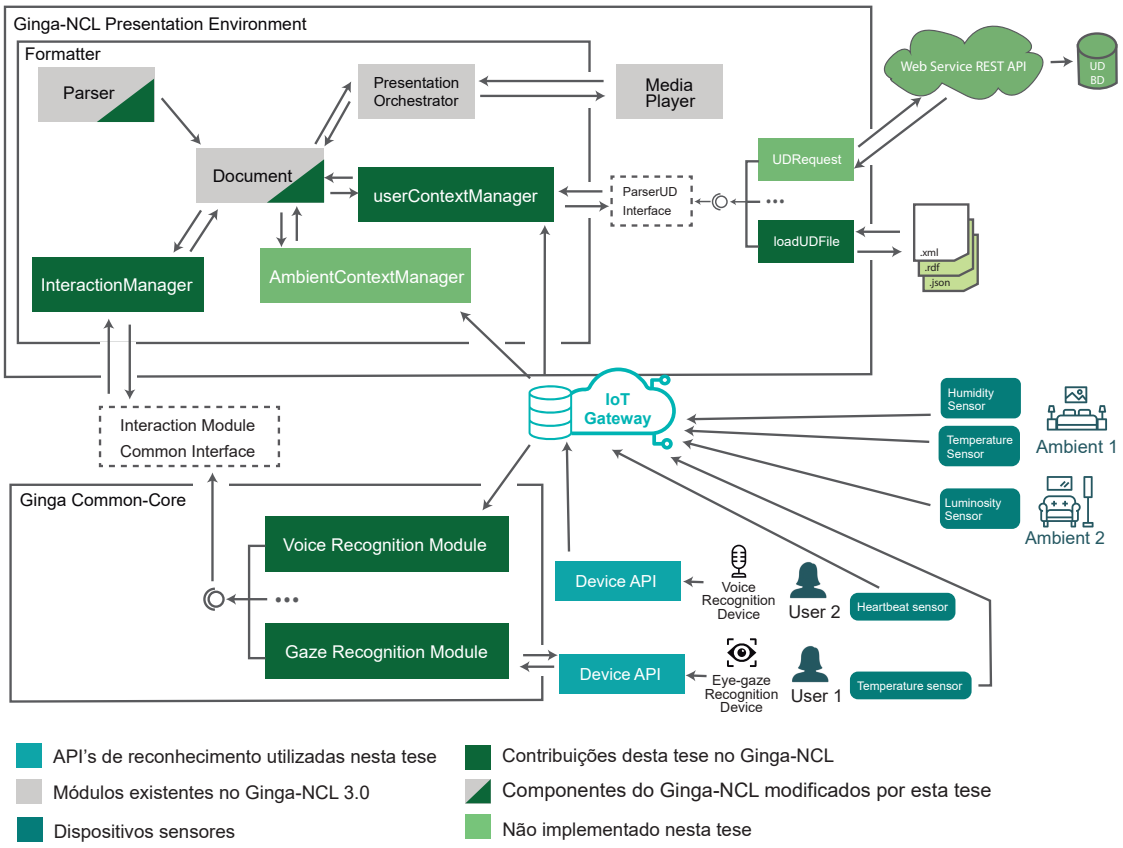


Figura 5.3: Arquitetura estendida do Ginga-NCL para suporte à interação multimodal com múltiplos usuários.

deles, as informações sobre o que deve ser monitorado durante a execução da aplicação. A fim de permitir que a implementação seja facilmente estendida para adição de novas modalidades de interação, foi criada uma interface (*Interaction Module Common Interface*), que deve ser implementada por cada módulo de interação específico. Note que, se a aplicação não utiliza um determinado tipo de interação, o módulo correspondente não será iniciado pelo *InteractionManager* desnecessariamente.

Para a implementação da interação multimodal proposta nesta tese, os componentes *Parser* e *Formatter* foram estendidos, para dar suporte ao reconhecimento de novos tipos de eventos definidos em NCL 4.0 e detalhados no Capítulo 4. Se elos da aplicação NCL associam eventos de interação a usuários ou perfis específicos, na fase de análise do documento, o *Parser* faz o mapeamento entre a interação e o usuário associado a ela, por meio da análise dos elementos *<link>* especificados. Por exemplo, ao analisar o elo definido na Listagem 5.1, o *Parser* irá informar ao *Formatter* que é preciso ativar o módulo responsável pelo reconhecimento de voz (*Voice Recognition Module*), além de enviar para esse módulo o usuário "*user1*" (atributo *user*) e palavra chave "*play*" (atributo *key*)

relacionados à interação e que ele deve notificar ao reconhecedor. Essa notificação é feita ao módulo *InteractionManager*.

```
...
<connectorBase>
    <causalConnector id="onVoiceRecognitionStart">
        <connectorParam name="key"/>
        <connectorParam name="user"/>
        <simpleCondition role="onVoiceRecognition" key="$key" user= "$
    user"/>
        <simpleAction role="start" />
    </causalConnector>
</connectorBase>
...
<link xconnector="onVoiceRecognitionStart">
  <bind role="onVoiceRecognition" component="botanicalGardenImage">
        <bindParam name="key" value="play"/>
        <bindParam name="user" value="user1"/>
  </bind>
  <bind role="start" component="botanicalGardenVideo"/>
</link>
...
```

Listagem 5.1: Conector e elo para interação mutimodal em NCL 4.0

Os módulos de interação e o *InteractionManager* se comunicam através de métodos pré-definidos pela interface *Interaction Module Common Interface*. Por exemplo, o *InteractionManager*, ao ativar um módulo de interação, envia um objeto JSON (*JavaScript Object Notation*) com uma lista de *keys* que devem ser monitoradas pelo módulo, e os usuários relacionados àquela interação. Esta lista de *keys* enviada pelo *InteractionManager* depende do tipo de interação. Por exemplo, para o *Voice Recognition Module*, são enviadas palavras ou frases a serem reconhecidas, e para o *Hand Pose Recognition Module*, os tipos de gestos utilizados na aplicação. A Listagem 5.2 mostra um exemplo de JSON que segue o modelo para mapeamento de reconhecimento de voz, que gera o evento definido no *link* na Listagem 5.1.

```
{
    "userKeyList": [
        {
            "user": "user1",
```

```
            "key": [ "play"]
        }
    ]
}
```

Listagem 5.2: Exemplo de JSON para mapeamento de reconhecimento de voz correspondente à a Listagem 5.1.

Na arquitetura proposta ilustrada na Figura 5.3, pode-se perceber que o trabalho de reconhecimento da interação não é feito pelo formatador, evitando assim uma sobrecarga de processamento. Ele é notificado pelo *InteractionManager* apenas se ocorrer uma interação que deve ser tratada pela aplicação. Ou seja, em uma aplicação multimídia que especifica apenas interação por voz, tendo como *key* a palavra "*play*", caso o usuário diga quaisquer outras palavras diferentes, nenhuma notificação será enviada ao *InteractionManager*, e consequentemente ao formatador. Além disso, quando um módulo de interação específico notifica uma interação, ele pode também informar qual o usuário que a realizou, que no exemplo seria o "user1". Dentre os métodos que os módulos de interação devem implementar estão: *start()*, *setUserKeyList(json)* e *stop()*, que possibilitam ao *InteractionManager* iniciar, passar as informações (*keys*) que devem ser reconhecidas e parar o módulo de interação. Estes métodos são virtuais na classe *InteractionModule*, portanto para desenvolver um novo módulo de interação funcionando nesta arquitetura, é necessário estender a classe *InteractionModule*. Exemplos de subclasses de *InteractionModule* na Figura 5.3 são *Eye-gaze Recognition Module* e *Voice Recognition Module*. O modulo *userContextManeger* (UCM) inicia seu trabalho com a execução do método *start* onde recebe do *formatter* a lista de todos os *userAgents* e *userProfiles* definidos no documentos NCL. Para cada *userAgent* encontrado, o UCM irá adicioná-lo na lista de usuários e carregará todas as propriedades do arquivo em um nó do tipo *userSettingNode* correspondente. Já para o tratamento dos elementos *userProfile*, o UCM trabalha de forma diferente. Se a lista de perfil não for vazia, o UCM verifica se há arquivos (em um diretório pré definido no receptor de TV digital) com propriedades de usuários. Estas propriedades serão validadas com as propriedades do perfil e o usuário será adicionado com um usuário válido. Outra atividade do UCM é a criação de links dinâmicos, ou seja, para cada *link* onde tem um perfil em sua propriedade *user*, o UCM irá criar um *link* dinamicamente com cada usuário válido na propriedade *user* conforme explicado na Seção 4.2.

5.1.1 Reconhecimento de Voz em Ginga-NCL

Com objetivo de validar a solução proposta, esta seção apresenta a implementação do módulo *VoiceRecognitionModule* no Ginga-NCL utilizando uma API de reconhecimento de voz. A implementação de *VoiceRecognitionModule* no Ginga utiliza a interface de comunicação comum a todos os dispositivos de interação (*Interaction Module Common Interface*). Nesta implementação, o dispositivo de reconhecimento de voz se comunica com o Ginga utilizando o protocolo MQTT [37]. Para isso, foram usadas funções do *Google Cloud* para suprir a necessidade da conversão *speech-to-text* [15]. Inicialmente, é necessária a aplicação *Google Assistant* para captar a voz do usuário e uma *Action* do Google. Uma *Action* é, essencialmente, uma aplicação que roda sobre o *Google Assistant*, estendendo suas funcionalidades. Como ilustrado pela Figura 5.4, para a *Action* se comunicar com o broker MQTT, ela utiliza o serviço Google *Firebase* para que seja feita a publicação MQTT[4].

Em resumo, o *VoiceRecognitionModule* se inscreve em um tópico MQTT criado para o módulo de reconhecimento de voz. Isso é feito na execução do método *start()* e portanto a instância passa a ouvir publicações no tópico que a *Action* usa. Assim, a API do Google publica uma mensagem nesse tópico, significando que o dispositivo reconheceu o que foi dito. Quando ocorre a publicação do texto reconhecido no broker MQTT, o *VoiceRecognitionModule* compara seu conteúdo com a lista de *keys* e *users* de interesse da aplicação. Se for uma palavra ou frase associada a um usuário de interesse ao Ginga, notifica o `InteractionManager`, que por sua vez dispara o evento de reconhecimento de voz no formatador.

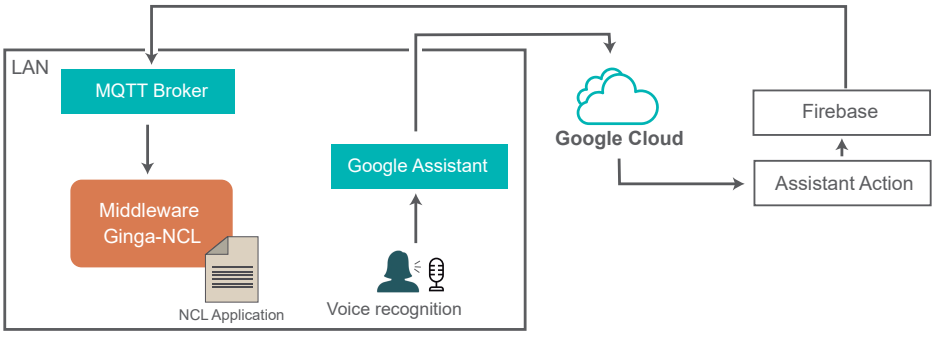


Figura 5.4: Arquitetura do reconhecimento de voz na implementação atual.

A *Action*, ao ser invocada, encaminha em forma de texto, todos comandos de voz que sejam precedidos por "TV". Desse modo, caso se queira executar na aplicação um

[4]A implementação da *Action* foi feita pelo aluno de graduação da UFF Pedro Valentim.

comando denominado "iniciar", diz-se "TV iniciar". Há, ainda, uma configuração opcional de perfis de usuário. Ao dizer "Definir perfil como...", o que vier depois é vinculado ao dispositivo corrente como sendo o perfil ou a identificação daquele usuário. Feito isso, todos os comandos enviados à aplicação são precedidos por esse perfil ou identificação, relacionado ao usuário que interagiu.

A título de exemplo, em uma aplicação que contém um professor e um aluno, de forma que cada perfil tem acesso a um conjunto de comandos diferentes, o usuário ao se identificar por professor, em seu dispositivo, invoca "Definir perfil como professor" e, analogamente, o aluno faz o mesmo. A partir disso, a aplicação será capaz de decidir, baseado nos perfis, quem invoca qual comando.

Outra implementação, seguindo a arquitetura da Figura 5.3, foi proposta por Montevecchi et al. [52], onde sensores de rastreamento ocular permitem que os usuários interajam pelo olhar com aplicativos multimídia, proporcionando assim também uma nova modalidade de interação para as aplicações de TV digital. Em [52], Montevecchi et al. propõe uma extensão Ginga-NCL para fornecer interação através de fixação do olhar usando um dispositivo rastreador de olhos e permitir o uso de um novo tipo de evento para autoria em NCL, chamado *EyeGaze*.

O trabalho de Valentim et al. [74] utiliza a arquitetura proposta na Figura 5.3 para possibilitar o reconhecimento de expressões faciais para que a TV seja agnóstica à implementação do algoritmo de reconhecimento. Como prova de conceito, a proposta do artigo foi desenvolvida sob o *middleware* Ginga-NCL. Foram realizadas duas implementações: a primeira baseada na versão atual do *middleware* Ginga e a segunda baseada em uma extensão proposta ao *middleware*, mostrando a viabilidade da proposta apresentada.

5.2 Suporte Multiusuário e Informações de Contexto

A atual versão do *middleware* Ginga-NCL captura as interações vindas de dispositivos como controle remoto, mouse ou teclado representando a interação de um usuário somente. Todas as informações da aplicação são armazenadas em um nó de conteúdo do tipo *settingsNode* onde também são armazenadas informações gerais da aplicação NCL. Só é permitido um nó desse tipo por aplicação NCL. Algumas dessas propriedades não podem ser alteradas durante a execução da aplicação. Se mais de um usuário interage com a aplicação e essa precisa mudar seu comportamento de acordo com propriedades específicas de quem participa da experiência, não seria possível construir um documento

NCL para atender a esse caso de uso na versão atual do Ginga-NCL.

O elemento XML <*userAgent*> foi implementado por meio de uma estrutura *user* que armazena propriedades de um usuário como um identificador e o caminho do arquivo onde estão suas informações individuais. Já o <*userProfile*> foi implementado com a construção da estrutura *profile* que mantém um identificador, numero mínimo e máximo de usuários que podem estar associados ao perfil e o caminho para as informações do perfil. A implementação mantém, no formatador, uma lista de estruturas do tipo *user* e outra lista do tipo *profile*. Desta forma, é possível ao formatador associar os *links* de interação com os usuários ou perfis a estas listas; e carregar informações destes usuários para nós de conteúdo do tipo *userSettingsNode* podendo também criar elos com esses nós. Finalmente, pode-se associar a essa lista sensores, que podem coletar informações dos usuários participantes da experiência multimídia, como batimento cardíaco ou temperatura.

Os elos são associados ao usuário da maneira como foi descrita na Seção 5.1, ou seja, o *InteractionManager* envia para os módulos as listas de usuários que participam dos elos por meio de seu atributo *user*. O valor do atributo *user* deve fazer parte das listas de *users* ou *profiles* mantidas pelo formatador. Assim o autor da aplicação precisa definir o elemento *userAgent* e/ou *userProfile* para poder referenciá-lo em elos com eventos de interação.

Conforme apresentado na Figura 5.3, o *UserContextManager* gerencia a importação das informações associadas tanto a *userAgent* quanto a *userProfile*. As informações podem estar dispostas em arquivos como por exemplo XML, RDF ou JSON ou ainda estarem disponíveis em um banco de dados remoto acessado por meio de API REST. O caminho do arquivo ou a string de conexão fica armazenada no atributo *src* do *userAgent* ou *userProfile*. O carregamento dessas informações é feito por um módulo que implementa a interface *ParserUDInterface*, como por exemplo, *UDRequest* e *loadUDFile*. Este módulo irá carregar todas as informações presentes no arquivo ou no banco de dados de acordo com a fonte de cada um. Para que um novo formato de arquivo seja contemplado pelo *userContextManager*, basta implementar um classe filha de *ParserUDInterface*. O módulo *userContextManager* irá passar aos "importadores" (módulos que implementam *ParserUDInterface*) todos os dados necessários para importação e armazenamento das propriedades desses usuários ou perfis.

As informações trazidas serão armazenadas em nós de conteúdo do tipo *user-settings* gerenciados pelo *UserContextManager*. Esses nós são instâncias de *UserSettingsNode* associados aos elementos <*userAgent*> ou <*userProfile*>. Como prova de conceito, foi

desenvolvido o módulo *loadUDFile* para arquivos XML seguindo o padrão MPEG-21 parte 22 [39]. Este módulo vai preencher uma instância da classe *UserSettingsNode* associado à instância de *UserAgent* respectiva. A partir do momento em que as informações estão em nós do tipo *user-settings*, poderão ser utilizadas em elos com eventos de atribuição NCL. A Listagem 5.3 mostra como poderia ser criado um nó do tipo *user-settings* para um usuário identificado como "U01"e tendo como propriedade "*heartBeat*". Além de utilizá-lo em elos com eventos de atribuição. No cenário em que possa capturar os valores lido de um sensor batimentos cardíacos e alterar a propriedades nó.

```
<!-- em head -->
...
<causalConnector id="onCondGteBeginStop">
    <connectorParam name="var"/>
    <compoundCondition operator="and">
        <simpleCondition role="onEndAttribution"/>
        <assessmentStatement comparator="gte">
            <attributeAssessment role="attNodeTest" eventType="
    attribution" attributeType="nodeProperty"/>
            <valueAssessment value="$var"/>
        </assessmentStatement>
    </compoundCondition>
    <simpleAction role="stop"/>
</causalConnector>

<!-- em body -->
...
<media type="application/x-ncl-user-settings" id="propU01" user="U01"
    >
    <property name="heartBeat" />
</media>

<link xconnector="onCondGteBeginStop">
    <linkParam name="var" value="200"/>
    <bind role="onEndAttribution" component="batCardiacoSensor"
    interface="heartBeat"/>
    <bind role="attNodeTest" component="batCardiacoSensor" interface=
    "heartBeat"/>
    <bind role="stop" component="video"/>
```

```
26 </link>
```

Listagem 5.3: Trecho de aplicação em NCL utilizando nó do tipo *user-settings*

As instâncias de *UserSettingsNode* podem ser usadas também para armazenar informações lidas de um sensor em um ambiente IoT. Como pode-se verificar na Figura 5.3, pode-se ter sensores (com por exemplo *Heartbeat Sensor* no User2) publicando leituras em um broker e o *userContextManager* pode ler essas publicações e atualizar os nós do tipo *UserSettingsNode* respectivos. Para isso, o gerenciador de contexto deve ter um assinante de um tópico no broker. Tomando como exemplo a arquitetura MQTT, pode-se ter um *broker* MQTT recebendo publicações dos sensores e o gerenciador de contexto pode se inscrever nos mesmos tópicos de publicações dos sensores e atualizar as propriedades respectivas nos nós de conteúdo do tipo *userSettingsNode* a partir da atualização das propriedades. Elos poderão ser ativados mudando o comportamento da aplicação de acordo com leituras feitas pelos sensores. De maneira análoga, o mesmo pode ser feito com sensores de ambiente (como por exemplo *LuminositySensor* no Ambient 2 na Figura 5.3), publicando suas leituras no servidor IoT. E o gerenciador de contexto de ambientes (*ambientContextManager*) pode pegar essas informações e atualizar os nós do tipo *ambientSettingsNode*. Como também são nós de conteúdo, a atualização de suas propriedades pode ativar elos e disparar ações na aplicação. A implementação do *ambientContextManager* é trabalho futuro.

Além de ler as informações publicadas pelos sensores de um servidor IoT, O *userContextManager* pode também publicar informações específicas para que o dispositivo IoT leia, como por exemplo, informações para calibragem do dispositivo. Assim o gerenciador pode-se comunicar com o servidor nos dois sentidos. A comunicação entre o *userContextManager* e o servidor IoT também é trabalho futuro.

Vale ressaltar que a arquitetura é extensível para vários tipos de eventos de interação, pois a implementação do *InteractionManager* recebe notificações de vários módulos de interação desde que tenham sido implementados seguindo a interface *InteractionModuleCommonInterface*. O mesmo acontece com os módulos responsáveis pela leituras das propriedades dos usuários contidas no receptor de TV digital, ou seja, o UCM segue uma interface denominada *ParserUD*. A arquitetura também é escalável para os dispositivos reconhecedores e sensores a partir do momento que a arquitetura proposta contém um *IoT Gateway* como o que foi utilizado nos testes de interação multimodal. Este trabalho apresenta testes que utilizaram um *broker* MQTT como *IoT Gateway*. Desta forma, os dispositivos reconhecedores e sensores podem publicar em tópicos específicos em que o

middleware está lendo, podendo assim, agir diante destas publicações.

5.2.1 Considerações Finais

Este capítulo apresentou a implementação da proposta desta tese no *middleware* Ginga-NCL, incluindo uma arquitetura extensível, seus módulos e funcionamento geral. O capítulo exemplificou aplicações NCL desenvolvidas e utilizadas nos testes de funcionalidades. Apresentou também o formato dos dados que são trocados entre os módulos, como por exemplo, objetos JSON. O próximo capítulo apresenta a avaliação da proposta por meio de comparação com a implementação padrão atual do Ginga-NCL e também testes de carga que atestam o desempenho do *middleware* para o suporte multimodal e multiusuário.

Capítulo 6

Avaliação

Este capítulo apresenta a avaliação da proposta desta tese considerando diferentes abordagens. Primeiramente, a Seção 6.1.1 apresenta a execução de experimentos para avaliar como a extensão proposta do middleware Ginga-NCL, que dá suporte a NCL 4.0, se comporta em comparação a uma implementação do reconhecimento de voz usando o Ginga-NCL padrão atual com scripts Lua. A Seção 6.1.2 apresenta a realização de um teste de carga da capacidade de manipulação de eventos do *middleware* Ginga-NCL, tratando vários eventos multimodais em diversos intervalos de tempo. A Seção 6.1.3 apresenta os testes para avaliar se há perda de desempenho no *middleware* no tratamento do eventos sendo disparados em vários intervalos de tempo de disparos, assim como intervalos de tempo diferentes para executar os processos de leitura do servidor MQTT. Já a Seção 6.2 apresenta a avaliação de se o tratamento de perfis de múltiplos usuários causa algum atraso significativo à execução de uma aplicação NCL 4.0 no *middleware*.

6.1 Multimodalidade

As subseções seguintes apresentam avaliação da proposta de multimodalidade, os seja, uma análise quantitativa de como o tratamento de eventos de interação multimodal está sendo realizado pelo *middleware* Ginga-NCL estendido.

6.1.1 Avaliação Comparativa NCL 3.0 x NCL 4.0

Este seção apresenta um cenário de uso para destacar como NCL 4.0 suporta o desenvolvimento de uma aplicação multimídia com interação de voz por múltiplos usuários. Será feita uma comparação da execução da aplicação utilizando a versão de NCL 4.0 proposta

nesta tese e a versão 3.0 padrão atual. Para tal, a seção descreve uma aplicação NCL que tem como objetivo representar um cenário em que o usuário "*U01*"controla um vídeo que faz um *tour* pelo Jardim Botânico. Desta maneira, apenas o usuário "*U01*" pode manipular a apresentação do vídeo. Para demonstrar a implementação do evento de voz, foi desenvolvida um aplicação que inicia um vídeo a partir do comando de "*play*" dito pelo usuário "*U01*". As Listagens 6.1 e 6.2 apresentam o conector e o elo relacionado com o início do vídeo do Jardim Botânico, ativado por comando de voz, especificadas em NCL 3.0 e NCL 4.0 respectivamente.

O exemplo da Listagem 6.1, especificado em NCL 3.0, utiliza um evento de reconhecimento de voz (*voiceRecognition*) por meio do script *scriptInt.lua* (linha 16), que o autor precisa desenvolver e iniciá-lo junto com a aplicação que neste caso foi feita por meio do *link* da linha 20. Além disso, será necessário um conector do tipo *onEndAttribution* com um *compoudCondition* para o teste que verificará se uma *string* contém a palavra que indica que o "*U01*"disse "play".

```
1  ...
2  <connectorBase>
3    <causalConnector id="onEndAttributionTestStart">
4      <connectorParam name="val"/>
5        <compoundCondition operator="and">
6          <simpleCondition role="onEndAttribution"/>
7        <assessmentStatement comparator="eq">
8          <attributeAssessment role="attNodeTest" eventType="
     attribution" attributeType="nodeProperty"/>
9          <valueAssessment value="$val"/>
10       </assessmentStatement>
11     </compoundCondition>
12       <simpleAction role="start"/>
13   </causalConnector>
14 </connectorBase>
15 ...
16 <media id="readIntModes" src="scripts/scriptInt.lua">
17   <property name="key"/>
18 </media>
19 ...
20 <link xconnector="onBeginStart">
21   <bind role="onBegin" component="botanicalGardenImage"/>
```

```
22   <bind role="start" component="readIntModes" />
23 </link>
24 <link xconnector="onEndAttributionTestStart">
25   <bind role="onEndAttribution" component="readIntModes" interface =
       "key"/>
26   <bind role="attNodeTest" component="readIntModes" interface="key">
27   <bindParam name="val" value="voice_recog/U01:play"/>
28   </bind>
29   <bind role="start" component="botanicalGardenVideo"/>
30 </link>
31 ...
```

Listagem 6.1: Conector e elo para interação mutimodal em NCL 3.0

O exemplo da Listagem 6.2, especificado em NCL 4.0, utiliza um evento de reconhecimento de voz (*voiceRecognition*) por meio do papel pré-definido *onVoiceRecognition*. Esta condição possui um atributo *key* que identifica o palavra a ser captada. Ainda, o atributo *user* indica que tal palavra deverá ser captada do usuário "*U01*".

```
1  ...
2  <connectorBase>
3     <causalConnector id="onVoiceRecognitionStart">
4        <connectorParam name="key"/>
5        <connectorParam name="user"/>
6        <simpleCondition role="onVoiceRecognition" key="$key" user= "$
       user"/>
7        <simpleAction role="start" />
8     </causalConnector>
9  </connectorBase>
10 ...
11 <link xconnector="onVoiceRecognitionStart">
12   <bind role="onVoiceRecognition" component="botanicalGardenImage">
13       <bindParam name="key" value="play"/>
14       <bindParam name="user" value="U01"/>
15   </bind>
16   <bind role="start" component="botanicalGardenVideo"/>
17  </link>
18 ...
```

Listagem 6.2: Conector e elo para interação mutimodal em NCL 4.0

Para avaliar a o comportamento do evento *VoiceRecognition* na extensão proposta, foram construídas três aplicações NCL derivadas do exemplo na Listagem 6.2[1]. A primeira aplicação inicia um vídeo quando for dito "*play*"pelo usuário "*U01*". A segunda aplicação inicia automaticamente um vídeo e para este vídeo, quando for dito "*stop*" pelo usuário "*U01*". A terceira aplicação inicia automaticamente um vídeo e pausa o vídeo, quando for dito "*pause*"pelo usuário "*U01*".

As aplicações foram desenvolvidas em duas versões. Primeiro na versão em NCL 4.0 proposta nesta tese. Para executá-la, foi utilizada a versão Ginga-NCL estendida apresentada no Capítulo 5. A segunda versão das aplicações foi construída utilizando NCL 3.0 e scripts NCLua emulando o comportamento do evento *VoiceRecognition*.

Em todos os testes, a forma de disparar o evento de interação por voz foi simplificada para que o teste não fosse comprometido com o retardo de reconhecimento da API na nuvem da Google. Desta forma, foi produzido um script que inicia a aplicação e publica um texto no tópico *voiceRecognition* de um *broker* MQTT rodando na mesma máquina que o *middleware* Ginga. A publicação no tópico *voiceRecognition* sinaliza que algo foi dito.

Após a publicação no tópico, o *VoiceRecognitionModule* deve perceber uma atualização no tópico. A partir deste momento, na versão 4.0 da classe implementada seguindo a arquitetura apresentada no Capítulo 5 notifica o formatador sobre o evento de reconhecimento de voz.

Na versão das aplicações em NCL 3.0, foi implementado um script NCLua que acessa o tópico MQTT desejado e dispara um evento de atribuição na aplicação NCL, que por sua vez possui *links* para sincronizar a atribuição com uma ação no vídeo.

Cada aplicação foi construída para testar e medir o tempo gasto desde o reconhecimento de voz até o início de alguma de três ações ("start", "stop" e "pause"). Experimentos foram executados com cada uma das aplicações a fim de avaliar e comparar o tempo de resposta da proposta de NCL 4.0 e de uma aplicação NCL 3.0 com *scripts* Lua. Os resultados foram obtidos de uma média de 100 execuções. Os experimentos foram realizados em uma máquina com as seguintes configurações: Intel Core™ i5-7500T CPU @ 2.70GHz; 8GB RAM; 1TB HD e Sistema Operacional Ubuntu 18.

[1]As aplicações NCL e o *script* de testes podem ser acessados em http://bit.do/fHiw4

O tempo médio (T) calculado em cada experimento é definido por:

$$T = \frac{\sum_{i=1}^{n}(mqttPub_i + responseTime_i)}{n} \tag{6.1}$$

Onde *mqttPub* é o tempo de comunicação entre a postagem de dados no tópico do broker, o processamento do *broker* e a percepção de atualização deste tópico pelo cliente MQTT. *responseTime* é o tempo que a aplicação Ginga-NCL demora para executar a ação esperada a partir do momento em que é reconhecida a publicação no tópico MQTT.

Na Figura 6.1, são apresentados os resultados de experimentos para as três aplicações em suas duas versões com intervalo de confiança de 95%. Na cor azul, é apresentado o tempo médio das aplicações criadas com NCL 4.0. Na cor verde, pode-se observar o tempo médio das aplicações criadas com NCL 3.0 e scripts NCLua. Pode-se observar que a aplicação com a proposta deste trabalho com NCL 4.0 possui tempo de resposta abaixo de 70ms enquanto todas as aplicações feitas em NCL 3.0 superam 200ms para responder a ação.

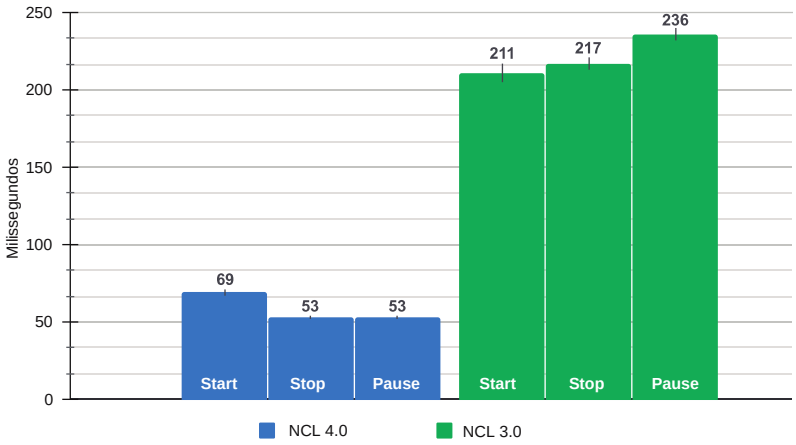


Figura 6.1: Média de tempo (ms) gasto para aplicação reagir a uma publicação MQTT de *start/stop/pause* em um vídeo.

Como pode ser visto na Figura 6.1, o tempo de resposta de uma aplicação em NCL 4.0 é aproximadamente um terço do tempo de uma aplicação feita utilizando NCL 3.0 e Lua. Além disso, em NCL 4.0, o autor da aplicação não precisa programar em uma linguagem imperativa e não precisa acoplar módulos ou scripts NCLua à sua aplicação existente. Estes resultados demonstram que a implementação de interação multimodal proposta no Ginga oferece bom desempenho.

Com o uso de NCL 4.0, também há uma diminuição do esforço de autoria. O autor da aplicação, usando NCL 4.0, deve declarar apenas 1 conector e 1 *link* para prover a interação por voz. A aplicação tem um total de 49 linhas de código para descrever seu

comportamento. Por outro lado, em NCL 3.0, o autor necessita de scripts NCLua para realizar o acesso aos tópicos MQTT e criação de um conector de atribuição para emular um evento de reconhecimento de voz. A aplicação NCL 3.0 tem um total de 158 linhas de código.

Adicionalmente, a forma de descrever interação multimodal por meio de eventos da linguagem NCL favorece a legibilidade do código NCL. Pois quando o autor define um conector com o papel *onVoiceRecognition* (evento de interação por voz), é mais fácil de entender que se trata de um reconhecimento de voz do que ter que criar um conector com papel *onEndAtribution* (evento de atribuição) para tratar o evento de reconhecimento de voz gerado pelo *script* Lua.

6.1.2 Avaliação de Carga

Para avaliar a capacidade do *middleware* de lidar com inúmeros eventos multimodais, esta seção apresenta vários testes realizados que usaram várias aplicações NCL simples. Essas aplicações possuem elos que são disparados com eventos de interação por meio de voz e gesto. As interações foram representadas no teste por meio de publicações em tópicos MQTT nos quais os dispositivos responsáveis por essas interações multimodais publicam. Isso foi possível utilizando um *script*, que faz as publicações no *broker* MQTT. O *middleware* se registrou nesses tópicos por meio dos módulos de interação (IMs). Sempre que ocorreu uma publicação, os IMs sinalizaram para o *middleware*, que por sua vez sinalizou para todos objetos interessados nos eventos na aplicação NCL. Assim, o teste utilizou uma aplicação NCL que alterava uma variável cada vez que um evento de interação era percebido pelo middleware, ou seja, para confirmar que a ação aconteceu e que o evento foi tratado. Para avaliar esse comportamento, a aplicação foi executada com os módulos do *middleware* ouvindo eventos a cada 10 milissegundos (ms). Este é um parâmetro do Ginga-NCL que pode ser configurado. O teste foi executado em seis cenários distintos, considerando intervalos de $10ms$; $15ms$; $20ms$; $30ms$; $40ms$ e $50ms$ para geração de eventos de interação, ou seja, publicações no *broker* MQTT pelo *script* de teste. Para cada cenário, o *script* emulou 100 eventos com o intervalo de tempo correspondente entre eles e contabilizou quantos eventos foram tratados pelo Ginga-NCL. Para cada cenário, o teste foi repetido 100 vezes e, em seguida, foi calculada a média de eventos tratados em cada intervalo. A Figura 6.2 mostra os resultados obtidos com intervalo de confiança de 95%. Pode-se ver que, para cenários com geração de eventos espaçados de $20ms$ ou mais, o Ginga obteve uma excelente taxa de tratamento de eventos, atingindo em média 99,9%

de eventos tratados. Para o cenário de eventos gerados de $10ms$ em $10ms$, o Ginga não conseguiu tratar 21,26% dos eventos gerados pelo *script* de teste, já que o *middleware* foi configurado para verificar eventos a cada $10ms$.

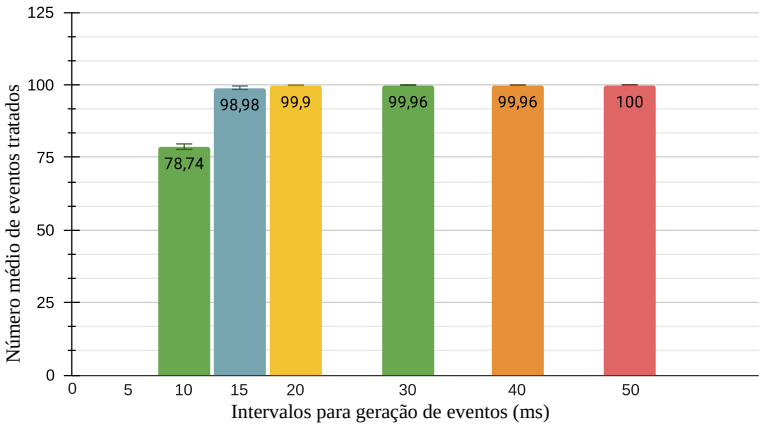


Figura 6.2: Média do número de eventos tratados pelo Ginga-NCL considerando diferentes intervalos para geração de eventos de interação

6.1.3 Avaliação de Desempenho

Para a avaliação de desempenho da proposta, foi avaliado o quanto o tratamento de novos eventos pelo *middleware* Ginga-NCL causa sobrecarga na execução de aplicações NCL. Para isso, primeiramente o teste mediu o atraso causado pela escuta dos módulos de interação (IMs) sem acionar os eventos e depois tratando eventos de interação. O teste utilizou *scripts* que realizaram as publicações MQTT em intervalos de $20ms$, que é o menor intervalo com 99,9% dos eventos tratados, conforme apresentado na Figura 6.2. Para isso, uma aplicação NCL foi criada com uma imagem de fundo com âncoras temporais que iniciaram quatro outras imagens. Para cada segundo, uma nova imagem foi iniciada. A visão temporal deste aplicativo é mostrada na Figura 6.3. A aplicação foi executada em três cenários diferentes:

(1) Rodar a aplicação sem os IMs (Ginga-NCL estendido sem iniciar os IMs);

(2) Rodar a aplicação com os IMs sem receber eventos de interação (IMs iniciados com Ginga-NCL estendido);

(3) Executar a aplicação com os IMs e receber 300 eventos de interação (150 eventos de reconhecimento de voz e 150 de reconhecimento de gestos) (Ginga-NCL estendido iniciando e sobrecarregando IMs).

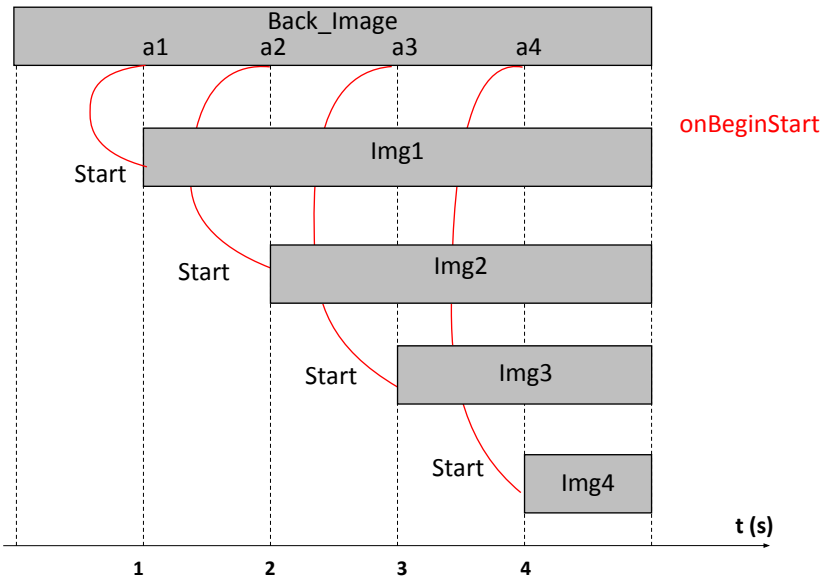


Figura 6.3: Visão temporal da aplicação usada para avaliar o atraso de exibição de imagens em três diferentes cenários

A aplicação foi executada 100 vezes em cada cenário e foi calculada a média de atraso de início de cada imagem (*img1, img2, img3 e img4*) associada às âncoras temporais com intervalo de confiança de 95%.

A Figura 6.4 mostra o atraso médio da exibição de cada uma das quatro imagens nesses três cenários diferentes. Observe que nos cenários (2) e (3), o *middleware* desperta o assinante MQTT (módulo que ouve os IMs) a cada $10ms$. No cenário (2), pode-se observar que, sem disparar os eventos, há um atraso da ordem de $9ms$ na exibição da imagem *img1*. No cenário (3), ocorre o disparo de 300 eventos de interação no intervalo de $20ms$, com os módulos do Ginga-NCL acordando e ouvindo a cada $10ms$. Pode-se ver na Figura 6.4 que a extensão proposta não impacta o *middleware* Ginga quando nenhum tratador de evento multimodal está ativo no middleware no experimento. Além disso, para o cenário (2), o atraso médio resultante foi de $\approx 6.75ms$ e, para o cenário (3), o atraso médio foi de $\approx 11.25ms$. A implementação dos módulos tratadores de eventos multimodais adiciona pelo menos um atraso de $\approx 5ms$ na implementação estendida do Ginga-NCL. No entanto, esse atraso não aumenta à medida que o número de eventos aumenta. Além disso, o pior caso de atraso no cenário (3) foi de $13ms$ para *img1*. De acordo com o trabalho de Card et al. [19], são necessários 230 ms para um humano ter ciência de algo captado visualmente. Considerando esse tempo, como o cenário de

teste mais sobrecarregado adiciona à percepção de *img1* um atraso de $5,65\%$ do tempo identificado em [19], conclui-se que a implementação estendida do *middleware* Ginga-NCL não causa sobrecarga significativa na apresentação de *img1*.

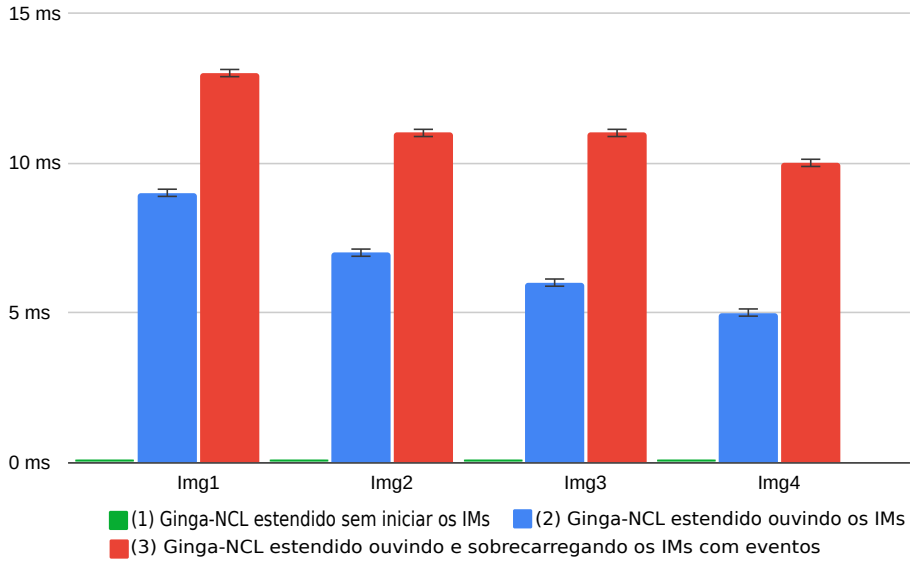


Figura 6.4: Média de atraso (ms) de exibição de cada mídia imagem na aplicação para cada cenário

6.2 Multiusuário

Esta seção apresenta a avaliação da proposta de suporte multiusuário, ou seja, uma análise quantitativa de como o *middleware* Ginga-NCL se comporta com o tratamento das funcionalidades para uma aplicação multiusuário especificadas neste trabalho. Para isso, esta seção apresenta a avaliação em duas partes. A primeira mede o tempo gasto no reconhecimento dos usuários cujas propriedades estão armazenadas em arquivos XML. A maior sobrecarga de processamento acontece quando a aplicação tem um perfil de usuário a ser verificado. Conforme apresentado no Capítulo 4, quando a aplicação tem algum *userProfile*, todas as suas propriedades são comparadas com as propriedades de todos os usuários que estão armazenados no receptor de TV digital. Assim, somente os usuários que tenham tais propriedades irão ter suas interações consideradas pelo *middleware*. A segunda parte mede o tempo gasto para criação de todos o *links* dinamicamente a partir dos usuários encontrados e que atendam ao perfil.

Os experimentos foram realizados em uma máquina com as seguintes configurações: Intel Core™ i5-7500T CPU @ 2.70GHz; 8GB RAM; 1TB HD e Sistema Operacional Ubuntu 18.

Na primeira parte da avaliação, cada aplicação foi construída para testar e medir o tempo gasto para reconhecer todos os usuários com o perfil especificado na aplicação. Para isso, o experimento avaliou 5 cenários onde foram variados o número de usuários e o número de propriedades comparadas com as propriedades do perfil usado. Cada cenário foi executado 100 vezes e os resultados são apresentados com intervalo de confiança de 95%. A Tabela 6.1 apresenta os cenários com a média dos tempos gastos para carregar os usuários em 100 execuções.

Cenário	Descrição	Média (ms)	Confiança
1	1 usuário com 5 propriedades	0	1,09E-09
2	5 usuários com 5 propriedades	1	2,04E-09
3	10 usuários com 5 propriedades	1	7,76E-10
4	5 usuários com 10 propriedades	1	1,06E-09
5	10 usuários com 10 propriedades	1	1,47E-09
6	15 usuários com 10 propriedades	2	2,61E-08
7	20 usuários com 10 propriedades	2	1,09E-09

Tabela 6.1: Cenários contemplados no experimento variando números de usuários e propriedades

Na segunda parte da avaliação, cada aplicação foi construída para testar e medir o tempo gasto para criar os *links* dinamicamente para cada usuário. Para isso, o experimento avaliou nove cenários onde foram variados o número de usuários e o número de *links* com o perfil usado. Cada cenário foi executado 100 vezes e os resultados são apresentados com intervalo de confiança de 95%. A Tabela 6.2 apresenta os cenários com a média dos tempos gastos para carregar os usuários em 100 execuções.

O gráfico da Figura 6.5 exibe o tempo médio gasto para criar 50 links dinâmicos em diferentes quantidades de usuário. Podemos notar que não houve diferença significativa entre a criação de 750 e 1000 links. Para grupos de 15 e 20 usuários respectivamente.

Como mostram as Tabelas 6.1 e 6.2, o tempo gasto no pior caso do experimento ainda se mostra bem pequeno, pois fazendo o somatório do tempo para carregar 20 usuários contendo 10 propriedades comparadas mais o tempo gasto em uma aplicação com 50 *links*, o total é de 17 ms durante a carregamento da aplicação, ou seja, antes da apresentação das mídias. As aplicações utilizadas no teste estão disponíveis no Apêndice A. Vale salientar que o número de 10 usuários já reflete um número grande de pessoas que compartilhariam um único receptor de TV digital.

Cenário	Descrição	Média (ms)	Confiança
1	1 link com 1 usuário	0	0
2	10 links com 1 usuário	0	5,89E-10
3	50 links com 1 usuário	0	9,40E-10
4	1 link com 5 usuários	0	3,94E-10
5	10 links com 5 usuários	0	6,61E-10
6	50 links com 5 usuários	1	1,59E-09
7	1 link com 10 usuários	0	3,94E-10
8	10 links com 10 usuários	0	1,12E-09
9	50 links com 10 usuários	7	8,65E-09
10	50 links com 15 usuários	15	1,03E-08
11	50 links com 20 usuários	15	1,28E-08

Tabela 6.2: Cenários contemplados no experimento variando números de links e usuários

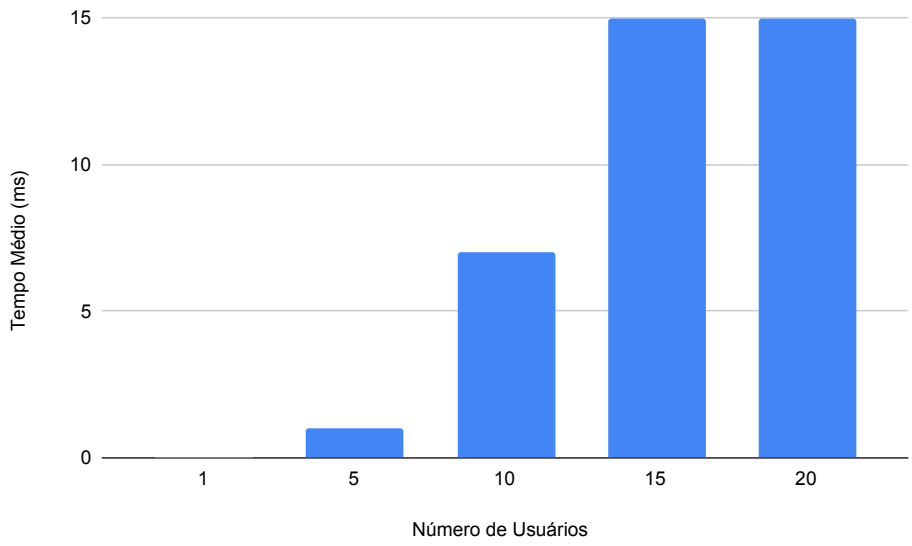


Figura 6.5: Média de tempo para carregar 50 *links* dinâmicos em relação ao número de usuários

6.2.1 Considerações Finais

Este capítulo apresentou a avaliação da proposta desta tese quanto ao uso de várias modalidades de interação. Esta avaliação foi feita em três abordagens: comparativa entre NCL 3.0 e NCL 4.0, avaliação de carga de eventos e avaliação de desempenho.

O capítulo também apresentou a avaliação de aplicações multiusuário em duas partes, considerando reconhecimentos dos usuários pelo *middleware* e sobrecarga da criação de *links* dinâmicos.

Capítulo 7

Cenários de Uso

Este capítulo apresenta alguns cenários de uso cujas funcionalidades propostas nesta tese são pertinentes.

7.1 Multimodalidade

Esta seção apresenta cenários de uso que exploram diferentes modalidades de interação.

7.1.1 Combinação de Modalidades Diferentes de Interação

A aplicação desenvolvida para ilustrar a combinação de modalidades diferentes de interação combina interação via voz e teclas do controle remoto. Ela possui três vídeos que poderão ser apresentados de acordo com o desejo do usuário. No primeiro momento, o usuário escolhe dentre três opções para exibir o primeiro vídeo. Selecionando um dos botões, o vídeo respectivo será apresentado. A interface inicial da aplicação é apresentada na Figura 7.1.

Após o início de qualquer um dos vídeos, o usuário poderá dizer "trocar" e então os botões com as opções aparecerão novamente possibilitando a troca do vídeo que está sendo apresentado. Ao selecionar outro vídeo, o atual será parado e o escolhido será exibido. O código NCL da aplicação é apresentado na Listagem 7.1.

Figura 7.1: Interface da aplicação com interação multimodal na versão NCL 4.0.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ncl id="apNCL40MultModal" xmlns="http://www.ncl.org.br/NCL4.0
       /EDTVProfile">
3   <head>
4    <regionBase>
5      <region id="backReg" width="100%" height="100%" zIndex="0" />
6      <region id="florestaVideoReg" width="100%" height="100%" zIndex="
       1" />
7      <region id="btnFlorestReg" bottom="5%" left="5%"  width="20%"
       height="20%" zIndex="2"/>
8      <region id="btnAutumnReg" bottom="5%" left="30%" width="20%"
       height="20%" zIndex="2"/>
9      <region id="btnSnowReg"    bottom="5%" left="55%" width="20%"
       height="20%" zIndex="2"/>
10   </regionBase>
11
12   <descriptorBase>
13       <descriptor id="backDesc"  region="backReg"/>
14       <descriptor id="florestaVideoRegDesc"  region="florestaVideoReg
       "/>
15       <descriptor id="btnFlorestDesc" region="btnFlorestReg"
       focusIndex="0" moveUp="2" moveDown="1" moveLeft="2" moveRight="1"
       focusBorderColor="yellow" focusBorderWidth="2"/>
16       <descriptor id="btnAutumnDesc" region="btnAutumnReg"
```

```
    focusIndex="1" moveUp="0" moveDown="2" moveLeft="0" moveRight="2"
    focusBorderColor="yellow" focusBorderWidth="2"/>
17      <descriptor id="btnSnowDesc" region="btnSnowReg"     focusIndex=
    "2" moveUp="1" moveDown="0" moveLeft="1" moveRight="0"
    focusBorderColor="yellow" focusBorderWidth="2" />
18  </descriptorBase>
19  <connectorBase>
20      <causalConnector id="onVoiceRecognitionStart">
21          <connectorParam name="key"/>
22          <connectorParam name="user"/>
23          <simpleCondition role="onVoiceRecognition" key="$key" user=
    "$user"/>
24          <simpleAction role="start" max="unbounded"/>
25      </causalConnector>
26      <causalConnector id="onBeginStart">
27          <simpleCondition role="onBegin"/>
28          <simpleAction role="start" max="unbounded"/>
29      </causalConnector>
30      <causalConnector id="onSelectionStartStop">
31          <simpleCondition role="onSelection"/>
32          <compoundAction operator="seq">
33              <simpleAction role="start" max="unbounded"/>
34              <simpleAction role="stop" max="unbounded"/>
35          </compoundAction>
36      </causalConnector>
37      <causalConnector id="onBeginStop">
38          <simpleCondition role="onBegin"/>
39          <simpleAction role="stop" max="unbounded"/>
40      </causalConnector>
41  </connectorBase>
42</head>
43<body>
44  <port id="pBack" component="back" />
45    <media id="back" src="images/blue.jpg" descriptor="backDesc"/>
46    <media id="florestVideo" src="videos/forest_720.mp4" descriptor="
    florestaVideoRegDesc"/>
47    <media id="autumnVideo" src="videos/autumn_720.mp4" descriptor="
    florestaVideoRegDesc"/>
```

```
48      <media id="snowVideo" src="videos/snow_720.mp4" descriptor="
    florestaVideoRegDesc"/>
49      <media id="btnFlorestVideo" src="images/florest.png" descriptor="
    btnFlorestDesc"/>
50      <media id="btnAutumnVideo" src="images/autumn.png" descriptor="
    btnAutumnDesc"/>
51      <media id="btnSnowVideo" src="images/snow.png" descriptor="
    btnSnowDesc"/>
52      <link xconnector="onBeginStart">
53          <bind role="onBegin" component="back"/>
54          <bind role="start" component="btnFlorestVideo"/>
55          <bind role="start" component="btnAutumnVideo"/>
56          <bind role="start" component="btnSnowVideo"/>
57      </link>
58      <link xconnector="onBeginStop">
59          <bind role="onBegin" component="florestVideo"/>
60          <bind role="stop" component="btnFlorestVideo"/>
61          <bind role="stop" component="btnAutumnVideo"/>
62          <bind role="stop" component="btnSnowVideo"/>
63      </link>
64      <link xconnector="onBeginStop">
65          <bind role="onBegin" component="autumnVideo"/>
66          <bind role="stop" component="btnFlorestVideo"/>
67          <bind role="stop" component="btnAutumnVideo"/>
68          <bind role="stop" component="btnSnowVideo"/>
69      </link>
70      <link xconnector="onBeginStop">
71          <bind role="onBegin" component="snowVideo"/>
72          <bind role="stop" component="btnFlorestVideo"/>
73          <bind role="stop" component="btnAutumnVideo"/>
74          <bind role="stop" component="btnSnowVideo"/>
75      </link>
76      <link xconnector="onSelectionStartStop">
77        <bind role="onSelection" component="btnSnowVideo"/>
78        <bind role="start" component="snowVideo"/>
79        <bind role="stop" component="autumnVideo"/>
80        <bind role="stop" component="florestVideo"/>
81      </link>
```

```
82      <link xconnector="onSelectionStartStop">
83        <bind role="onSelection" component="btnFlorestVideo"/>
84        <bind role="start" component="florestVideo"/>
85        <bind role="stop" component="autumnVideo"/>
86        <bind role="stop" component="snowVideo"/>
87      </link>
88      <link xconnector="onSelectionStartStop">
89        <bind role="onSelection" component="btnAutumnVideo"/>
90        <bind role="start" component="autumnVideo"/>
91        <bind role="stop" component="snowVideo"/>
92        <bind role="stop" component="florestVideo"/>
93      </link>
94      <link xconnector="onVoiceRecognitionStart">
95            <bind role="onVoiceRecognition" component="back">
96              <bindParam name="key" value="trocar"/>
97            </bind>
98            <bind role="start" component="btnFlorestVideo"/>
99            <bind role="start" component="btnAutumnVideo"/>
100           <bind role="start" component="btnSnowVideo"/>
101     </link>
102  </body>
103 </ncl>
```

Listagem 7.1: Código da aplicação NCL com interação multimodal.

7.1.2 Composição de Modalidades Diferentes de Interação

A aplicação desenvolvida para ilustrar a composição de modalidades diferentes de interação combina interação via voz e pose das mãos. Ela possui um vídeo que poderá ser pausado somente se o usuário "U01" disser pause e mostrar a pose de mão aberta diante da câmera. O código NCL da aplicação é apresentado na Listagem 7.1.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <ncl id="apl_2ModalidadesInt" xmlns="http://www.ncl.org.br/NCL4.0
    /EDTVProfile">
3  <head>
4   <regionBase>
5     <region id="florestaVideoReg" width="100%" height="100%" zIndex =
      "0" />
```

```
</regionBase>
<descriptorBase>
    <descriptor id="florestaVideoRegDesc"  region="florestaVideoReg
" />
</descriptorBase>
<connectorBase>
  <causalConnector id="onVoiceRecognitionStop">
      <connectorParam name="val"/>
      <connectorParam name="key"/>
      <connectorParam name="user"/>
      <simpleCondition role="onVoiceRecognition" key="$key" user =
"$user"/>
      <simpleAction role="stop" max = "unbounded"/>
  </causalConnector>
  <causalConnector id="onHandPoseRecognitionSet">
    <connectorParam name="key"/>
    <connectorParam name="user"/>
    <connectorParam name="var"/>
    <simpleCondition role="onHandPoseRecognition" key="$key" user =
"$user"/>
    <simpleAction role="set" value="$var"/>
  </causalConnector>
  <causalConnector id="onVoiceRecognitionSet">
    <connectorParam name="key"/>
    <connectorParam name="user"/>
    <connectorParam name="var"/>
    <simpleCondition role="onVoiceRecognition" key="$key" user = "$
user"/>
    <simpleAction role="set" value="$var"/>
  </causalConnector>
  <causalConnector id="onVoiceRecognitionTestPause">
      <connectorParam name="key"/>
      <connectorParam name="user"/>
      <compoundCondition operator="and">
          <simpleCondition role="onVoiceRecognition" key="$key"
user = "$user"/>
          <assessmentStatement comparator="eq">
              <attributeAssessment role="test" eventType="
```

```
    attribution" attributeType="nodeProperty"/>
39                  <valueAssessment value="true"/>
40              </assessmentStatement>
41           </compoundCondition>
42           <simpleAction role="pause" max = "unbounded"/>
43       </causalConnector>
44       <causalConnector id="onHandPoseRecognitionTestPause">
45           <connectorParam name="key"/>
46           <connectorParam name="user"/>
47           <compoundCondition operator="and">
48               <simpleCondition role="onHandPoseRecognition" key="$key"
    user = "$user"/>
49               <assessmentStatement comparator="eq">
50                   <attributeAssessment role="test" eventType="
    attribution" attributeType="nodeProperty"/>
51                   <valueAssessment value="true"/>
52               </assessmentStatement>
53           </compoundCondition>
54           <simpleAction role="pause" max = "unbounded"/>
55       </causalConnector>
56   </connectorBase>
57 </head>
58 <body>
59       <port id="pVideo" component="florestaVideo" />
60       <media id="florestaVideo" src="videos/forest_720.mp4" descriptor=
    "florestaVideoRegDesc">
61           <property name="voice" value="false"/>
62           <property name="hand" value="false"/>
63       </media>
64       <link xconnector="onHandPoseRecognitionSet">
65           <bind role="onHandPoseRecognition" component="florestaVideo">
66               <bindParam name="key" value="open"/>
67               <bindParam name="user" value="U01"/>
68           </bind>
69           <bind role="set" component="florestaVideo" interface="hand">
70               <bindParam name="var" value="true"/>
71           </bind>
72       </link>
```

```
73      <link xconnector="onVoiceRecognitionSet">
74        <bind role="onVoiceRecognition" component="florestaVideo">
75          <bindParam name="key" value="pause"/>
76          <bindParam name="user" value="U01"/>
77        </bind>
78        <bind role="set" component="florestaVideo" interface="voice">
79          <bindParam name="var" value="true"/>
80        </bind>
81      </link>
82      <link xconnector="onVoiceRecognitionTestPause">
83        <bind role="onVoiceRecognition" component="florestaVideo">
84          <bindParam name="key" value="pause"/>
85          <bindParam name="user" value="U01"/>
86        </bind>
87        <bind role="test" component="florestaVideo" interface="hand"/>
88        <bind role="pause" component="florestaVideo"/>
89      </link>
90      <link xconnector="onHandPoseRecognitionTestPause">
91        <bind role="onHandPoseRecognition" component="florestaVideo">
92          <bindParam name="key" value="open"/>
93          <bindParam name="user" value="U01"/>
94        </bind>
95        <bind role="test" component="florestaVideo" interface="voice"/>
96        <bind role="pause" component="florestaVideo"/>
97      </link>
98      <link xconnector="onVoiceRecognitionStop">
99        <bind role="onVoiceRecognition" component="florestaVideo">
100         <bindParam name="key" value="stop"/>
101         <bindParam name="user" value="U01"/>
102       </bind>
103       <bind role="stop" component="florestaVideo"/>
104     </link>
105   </body>
106 </ncl>
```

Listagem 7.2: Código da aplicação NCL com composição interação multimodal.

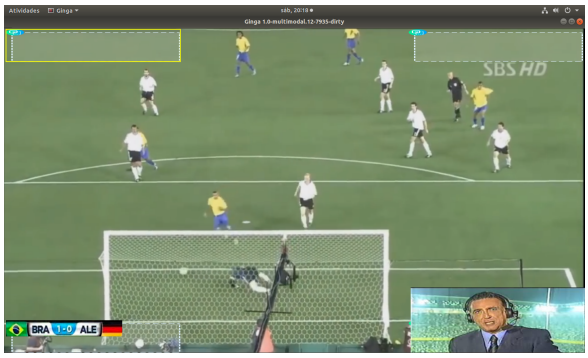
7.1.3 Cenário de Uso *"Put That There"*

Turk et al. [72] discutem sobre como a famosa aplicação "*Put That There*" de Bolt et al. [17] abriu caminho para vários sistemas que pretendem integrar diferentes modos de interação em uma variedade de áreas de aplicação. Em [17], os comandos de voz "*Put that*" e "*there*" processados em conjunto com o gesto de apontar destacam a grande utilidade da interação multimodal em um sistema de gerenciamento de dados espaciais. Assim, como prova de conceito e para ilustrar um cenário de uso pertinente para os usuários de TV digital, esta seção apresenta uma aplicação que usa esse exemplo de interação multimodal no contexto de uma experiência de TV. Com a diferença de que, neste cenário de uso, o usuário seleciona com o controle remoto ao invés de apontar com as mãos como na proposta de Bolt et al. [17].

A aplicação representa um cenário onde o usuário assiste a uma partida de futebol apresentada na TV e dois objetos de mídia sobrepostos ao vídeo principal: o placar e o vídeo do narrador, como podemos ver na Figura 7.2a. O usuário pode selecionar o placar ou o vídeo do narrador e mudar sua posição na tela. Assim que o usuário selecionar um objeto de mídia usando as teclas do controle remoto e clicando no *enter*, e disser "*put that*", ele será identificado. Em seguida, as novas regiões espaciais para possíveis reposicionamentos do objeto aparecem na TV. A Figura 7.2b mostra como a aplicação apresenta duas novas regiões superiores possíveis no momento exato em que o usuário seleciona o placar e diz "*put that*". O usuário então seleciona a região para a qual deseja que a mídia seja movida, usando as teclas do controle remoto e o *enter*, e diz "*there*". A Figura 7.2c mostra a região da esquerda escolhida logo após o usuário tê-la selecionado. Quando ele diz "there", o aplicativo altera o objeto (o placar na figura) colocando-o no local selecionado, conforme mostrado na Figura 7.2d. Depois disso, as regiões de posicionamento desaparecem, como podemos ver na Figura 7.2e.

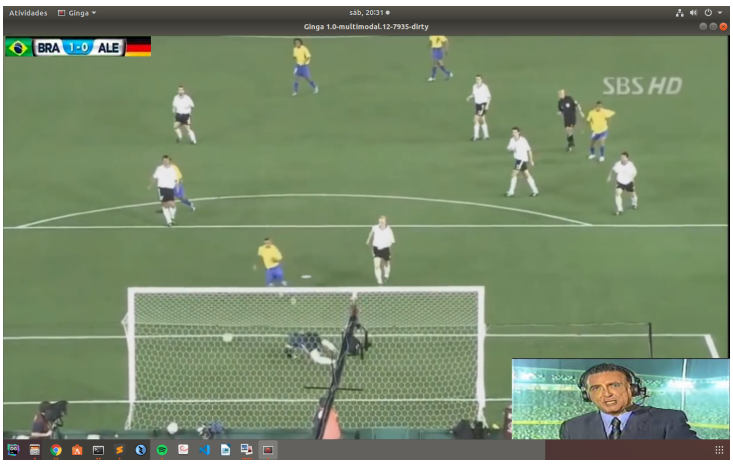
(a) Estado inicial

(b) Depois que o usuário disser "put that"



(c) Depois que o usuário seleciona o ponto esquerdo

(d) Depois que o usuários disser "there"



(e) Depois que a interação terminar

Figura 7.2: Instantes do aplicativo em execução após cada interação do usuário

7.2 Multiusuário

Esta seção apresenta cenários de uso onde a participação e identificação do usuário representa uma interessante funcionalidade tendo em vista a importância do uso da TV na vida das pessoas.

7.2.1 Personalização da Interação

Este cenário de uso exemplifica como criar *links* personalizados para que usuários com determinado perfil possam interagir com uma aplicação NCL. A aplicação inicia com a apresentação de um vídeo em que somente usuários de um determinado perfil podem pausar, tocar ou parar a apresentação da mídia. A codificação do documento NCL é apresentada na Listagem 7.3. Neste exemplo, considere que somente os usuários *U01* e *U02* atendem ao perfil *profile.xml* indicado na aplicação (linha 31 da Listagem 7.3). Suas propriedades, incluindo o *id*, estão especificadas nos arquivos *usr1.xml* e *usr2.xml* respectivamente, armazenados no receptor de TV digital. Além de a aplicação habilitar a interação somente de usuários que atendam ao perfil, essas interações podem ser identificadas pelo parâmetro *user* dos *links* como, por exemplo, na linha 40 onde o *link* só será disparado se o usuário com o perfil de *profile1* fizer a interação de voz.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ncl id="aplNCL40MultUser" xmlns="http://www.ncl.org.br/NCL4.0
       /EDTVProfile">
3   <head>
4    <regionBase>
5      <region id="florestaVideoReg" width="100%" height="100%" zIndex="
       0" />
6    </regionBase>
7    <descriptorBase>
8        <descriptor id="florestaVideoRegDesc"  region="florestaVideoReg
       " />
9    </descriptorBase>
10   <connectorBase>
11        <causalConnector id="onVoiceRecognitionPause">
12            <connectorParam name="key"/>
13            <connectorParam name="user"/>
14            <simpleCondition role="onVoiceRecognition" key="$key" user=
       "$user"/>
```

```
15            <simpleAction role="pause" />
16        </causalConnector>
17        <causalConnector id="onVoiceRecognitionResume">
18            <connectorParam name="key"/>
19            <connectorParam name="user"/>
20            <simpleCondition role="onVoiceRecognition" key="$key" user=
    "$user"/>
21            <simpleAction role="resume" />
22        </causalConnector>
23        <causalConnector id="onVoiceRecognitionStop">
24            <connectorParam name="key"/>
25            <connectorParam name="user"/>
26            <simpleCondition role="onVoiceRecognition" key="$key" user=
    "$user"/>
27            <simpleAction role="stop" />
28        </causalConnector>
29    </connectorBase>
30     <userBase>
31       <userProfile id="profile1" src="profiles/profile.xml"/>
32     </userBase>
33   </head>
34 <body>
35   <port id="pVideo" component="florestaVideo" />
36   <media id="florestaVideo" src="videos/forest_720.mp4" descriptor="
     florestaVideoRegDesc"/>
37     <link xconnector="onVoiceRecognitionPause">
38            <bind role="onVoiceRecognition" component="florestaVideo">
39              <bindParam name="key" value="pausar"/>
40              <bindParam name="user" value="profile1"/>
41            </bind>
42            <bind role="pause" component="florestaVideo"/>
43     </link>
44     <link xconnector="onVoiceRecognitionResume">
45            <bind role="onVoiceRecognition" component="florestaVideo">
46              <bindParam name="key" value="tocar"/>
47              <bindParam name="user" value="profile1"/>
48            </bind>
49            <bind role="resume" component="florestaVideo"/>
```

```
50      </link>
51      <link xconnector="onVoiceRecognitionStop">
52          <bind role="onVoiceRecognition" component="florestaVideo">
53            <bindParam name="key" value="parar"/>
54            <bindParam name="user" value="profile1"/>
55          </bind>
56          <bind role="stop" component="florestaVideo"/>
57      </link>
58  </body>
59 </ncl>
```

Listagem 7.3: Código da aplicação NCL com identificação multiusuário.

7.2.2 Identificação do usuário

Este cenário de uso ilustra a identificação de um usuário que interage com uma aplicação de TV digital. A aplicação desenvolvida possui uma mídia que será executada com o início do documento NCL. O conteúdo do vídeo apresenta uma floresta com neve e em um determinado momento será apresentada uma imagem vendendo um passeio de esqui. No momento em que algum usuário diz "comprar", a aplicação vai exibir na tela o identificador do usuário que comprou o passeio personalizando a interação via voz. Porém, aplicação só irá responder se a interação for realizada por um dos usuários que atendam ao perfil definido no *profile.xml*. Neste cenário é necessário trabalhar com id do usuário, para isso a mídia que está envolvida na interação precisa ter uma propriedade *usr*, pois o *middleware* irá armazenar o id do usuário nesta propriedade toda vez que houver a interação. Assim, na aplicação apresentada na Listagem 7.4, a mídia *ImgPropaganda* (imagem com a propaganda) tem a propriedade *usr* (linha 37). Quando o usuário disser "comprar", a propriedade *usr* será alterada para o id do usuário que falou. Imediatamente o valor da propriedade será pego a passado para um *script* NCLua que escreve o valor da propriedade na tela da TV. Isso é possível por causa do *link* da linha 48.

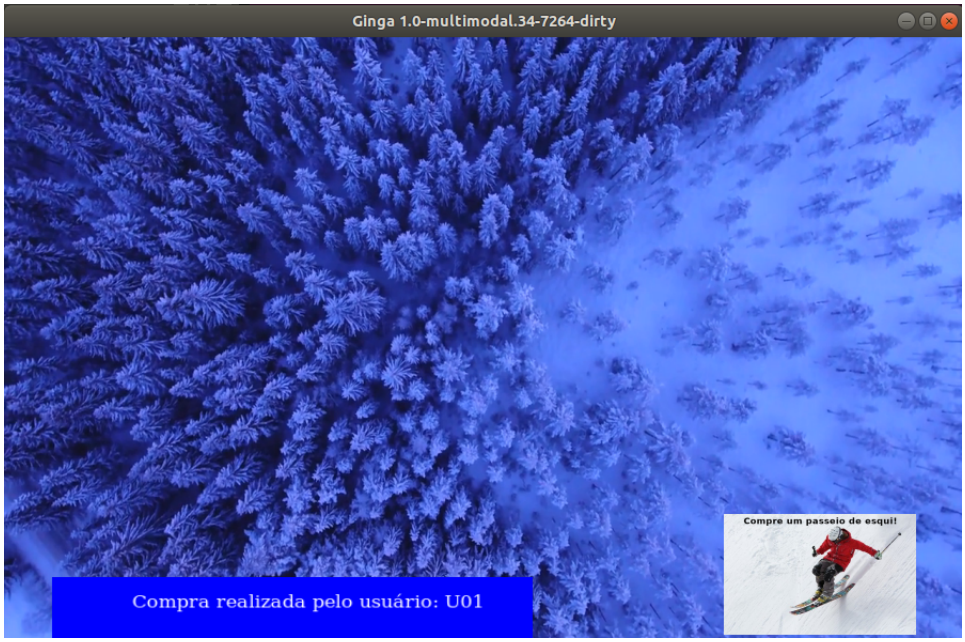


Figura 7.3: Interface da aplicação com interação e identificação multiusuário.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="aplMultiUser" xmlns="http://www.ncl.org.br/NCL3.0
    /EDTVProfile">
<head>
  <regionBase>
    <region id="VideoReg" width="100%" height="100%" zIndex="0" />
    <region id="ImgPropagandaReg" right="5%" bottom="5%" width="20%"
    height="20%" zIndex="2"/>
    <region id="rg1" left="5%" bottom="5%" height="10%"  width="50%"
    />
  </regionBase>
  <descriptorBase>
      <descriptor id="VideoDesc"  region="VideoReg"/>
      <descriptor id="ImgPropagandaDesc"  region="ImgPropagandaReg"
    />
      <descriptor id="desc1"  region="rg1"/>
  </descriptorBase>
  <userBase>
      <userProfile id="profile1" src="profiles/profile.xml"/>
  </userBase>
  <connectorBase>
      <causalConnector id="onBeginStart">
```

```
19          <simpleCondition role="onBegin"/>
20          <simpleAction role="start" max="unbounded"/>
21        </causalConnector>
22        <causalConnector id="onVoiceRecognitionSet">
23          <connectorParam name="key"/>
24          <connectorParam name="user"/>
25          <connectorParam name="var"/>
26          <simpleCondition role="onVoiceRecognition" key="$key" user="$
    user"/>
27          <simpleAction role="set" value="$var"/>
28        </causalConnector>
29    </connectorBase>
30 </head>
31 <body>
32      <port id="pInicio" component="lua" />
33      <port id="pVideo" component="snowVideo"/>
34      <media id="lua" src="scripts/script.lua" type= "application/x-
    ginga-NCLua" descriptor="desc1">
35        <property name="usr" value="false"/>
36      </media>
37      <media id="ImgPropaganda" src="images/passeioEsqui.jpeg"
    descriptor="ImgPropagandaDesc">
38        <property name="usr" value="false"/>
39      </media>
40      <media id="snowVideo" src="videos/snow_720.mp4" descriptor="
    VideoDesc">
41        <property name="usr" value="false"/>
42        <area id="aPropaganda" begin="2s" />
43      </media>
44      <link xconnector="onBeginStart">
45          <bind role="onBegin" component="snowVideo" interface="
    aPropaganda"/>
46          <bind role="start" component="ImgPropaganda"/>
47      </link>
48      <link xconnector="onVoiceRecognitionSet">
49        <bind role="onVoiceRecognition" component="ImgPropaganda">
50            <bindParam name="key" value="comprar"/>
51            <bindParam name="user" value="profile1"/>
```

```
52          </bind>
53          <bind role="getValue" component="ImgPropaganda" interface="usr"
        />
54              <bind role="set" component="lua" interface="usr">
55                  <bindParam name="var" value="$getValue"/>
56              </bind>
57      </link>
58  </body>
59  </ncl>
```

Listagem 7.4: Código da aplicação NCL com interação multiusuário.

No próximo capítulo serão abordadas as considerações finais desta tese contendo conclusões e trabalhos futuros.

Capítulo 8

Conclusão

Esta tese de doutorado propôs uma extensão ao *middleware* Ginga-NCL denominada NCL 4.0 para representar a interação multimodal em aplicações de TVDI com suporte a múltiplos usuários. A solução proposta para a Questão 1 apresenta novos tipos de eventos NCL para representar diferentes modos de interação do usuário. A personalização da interação, a validação do perfil e identificação do usuário, configura uma solução para a Questão 2. Com a abordagem desta tese, um autor de aplicativo de TV digital pode usar apenas o paradigma declarativo para especificar sua aplicação, sem a necessidade de programação em scripts Lua.

Foram especificados novos módulos para a Linguagem NCL 4.0, além de alterar alguns módulos existes. Todas as especificações foram feitas em XML Schema, os novos módulos estão no Apêndice B e o todo esquema esta disponível[1].

A implementação da proposta no *middleware* Ginga-NCL foi apresentada e uma avaliação de desempenho foi feita usando os eventos *VoiceRecognition* e *GestureRecognition* publicado por Barreto et al. [10]. Na implementação, foi utilizado o protocolo MQTT e a API do Google para reconhecer a fala. Vale a pena destacar que outros eventos de interação foram implementados usando a arquitetura proposta nesta tese. Os eventos de fixação dos olhos (*EyeGaze*) de Montevecchi et al. [52] e reconhecimento de expressões faciais (*FaceRecognition*) de Valentim et al. [74].

Foram desenvolvidas aplicações NCL com eventos multimodais e conduzidos três experimentos para avaliar se a extensão proposta fornece um tempo de resposta de interação com o usuário adequado. O primeiro experimento foi conduzido para comparar o tempo de resposta de uma interação *VoiceRecognition* entre o Ginga-NCL padrão e a extensão

[1] https://github.com/FabioBarr/NCL4.0

proposta nesta tese. O experimento mostrou que a extensão proposta tem um tempo de resposta consideravelmente menor. O segundo experimento foi conduzido para avaliar a taxa de tratamento de eventos na extensão Ginga-NCL proposta. Os resultados de desempenho com Ginga-NCL estendido mostram que ele se comporta de forma adequada. O terceiro experimento foi um teste de carga para avaliar como a extensão proposta adiciona um atraso à apresentação da mídia no Ginga-NCL. Este experimento foi conduzido com várias publicações sobre tópicos em um *broker* MQTT reservado para receber publicações de dispositivos de reconhecimento de fala e gestos. Tais resultados indicam um bom desempenho, pois apresentam um atraso de aproximadamente $13ms$.

Para avaliação das funcionalidades multiusuário de NCL 4.0, mais dois experimentos foram realizados. O primeiro mediu o tempo gasto para analisar todos os usuários cadastrados em um receptor de TV digital, verificando as propriedades de um determinado perfil indicado pela aplicação NCL 4.0 mostrando em seu pior caso um atraso de $1ms$. O segundo experimento avaliou o tempo gasto na criação de *links* dinâmicos para contemplar todos os usuários cadastrados no receptor, tendo em seu pior caso, no cenário que considerou 50 *links* e 20 usuários, um atraso de $15ms$.

A proposta de extensão também adicionou elementos para especificação de múltiplos usuários possibilitando relacioná-los a eventos de interação e a variáveis de contexto. Tais variáveis poderão conter informações de usuários e/ou perfis de maneira individualizada por meio dos elementos *<userProfile>* e *<userAgent>*. As informações são armazenadas em elementos também propostos nesta tese. A nova entidade estende *SettingsNode* e se chama *UserSettingsNode* e associar-se ao *UserAgent*. As informações para as propriedades de *UserSettingsNode*, podem ser trazidas de um arquivo XML por exemplo, mas a arquitetura proposta permite outras especificações. Foram desenvolvidos casos de uso para mostrar a viabilidade de vários usuários participar da experiência e ter suas informações sendo armazenadas e tradadas de forma individualizada. A proposta desta tese foi submetida à Chamada da TV 3.0 do Fórum SBTVD[2] para especificação da camada de codificação de aplicações para o futuro sistema de TV Digital no Brasil.

8.1 Limitações

Esta proposta tem duas limitações principais. A primeira se refere ao tipo da multimodalidade de interação, ou seja, esta proposta considera a multimodalidade de interação

[2]https://forumsbtvd.org.br/tv3_0/

exclusiva e independente.

Outra limitação está na implementação do uso do parâmetro *user* no elo. É possível indicar somente um perfil ou somente um *user* nos elos que tratam eventos de interação. Para considerar diferentes usuários, é necessário criar diferentes elos.

8.2 Trabalhos Futuros

Além das interações realizadas diretamente pelos usuários, pode-se, por meio da integração de sensores às aplicações multimídia, captar as reações do usuário que está consumindo o conteúdo assim como informações do ambiente de execução. A partir desta coleta de dados, a aplicação pode se adaptar, reagindo conforme o estado do usuário ou do ambiente. Então como trabalho futuro é planejada a implementação da parte da arquitetura de sensoriamento do usuário e do ambiente. A modelagem da relação dos sensores de ambiente com *AmbientSettingsNode*, principalmente em ambientes que possuem mais de um sensor do mesmo tipo, assim como modelar os critérios utilizados para leitura dos dados de um ambiente é um trabalho futuro interessante.

Outro trabalho futuro importante é avaliar as propostas de NCL 4.0 com autores de aplicações multimídia, através de testes de usabilidade da linguagem.

Um outro trabalho futuro é o desenvolvimento de ferramentas de autoria gráfica para as novas funcionalidades propostas, de forma a definir graficamente uma aplicação com interação multimodal com múltiplos usuários.

Outra sugestão é estender a pesquisa para sistemas de interação que possibilitem a fusão das interações e a captação paralela, considerado como trabalho futuro.

Além disso, estender a linguagem HTML com os novos tipos de eventos de interação também é um trabalho futuro importante. Pois outros sistema de TV digital possuem o interpretador HTML além do middleware Ginga já possuir o interpretador HTML5.

Referências

[1] Speech recognition grammar specification. http://www.w3.org/TR/speech-grammar/, 2004.

[2] ABNT. Digital terrestrial television - data coding and transmission specification for digital broadcasting - part 2: Ginga-ncl for fixed and mobile receivers - xml application language for application coding, 2011. ABNT NBR 15606-2:2011 standard.

[3] ABNT. Nbr 15606-1: Data coding and transmission specification for digital broadcasting. part 1 - data coding.

[4] Akscyn, R.; McCracken, D.; Yoder, E. Kms: a distributed hypermedia system for managing knowledge in organizations. In *Proceedings of the ACM conference on Hypertext* (1987), ACM, pp. 1–20.

[5] Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (1983), 832–843.

[6] Allen, J. F. Maintaining knowledge about temporal intervals. In *Readings in qualitative reasoning about physical systems*. Elsevier, 1990, pp. 361–372.

[7] Ayers, Y.; Cohen, A.; Bulterman, D., et al. Synchronized multimedia integration language (smil) 2.0. *W3C Recommendations* (2001).

[8] Barnes, D. H.; Allen, B. L.; Greene, D. A. Graphical user interface (gui) language translator, Oct. 26 1999. US Patent 5,974,372.

[9] Barreto, F.; Abreu, R. S.; dos Santos, J. A.; Muchaluat-Saade, D. C. Authoring sensory effects in ncl. In *Anais Estendidos do XXV Simpósio Brasileiro de Sistemas Multimídia e Web* (2019), SBC, pp. 191–192.

[10] Barreto, F.; de Abreu, R. S.; Montevecchi, E. B. B.; Josué, M. I. P.; Valentim, P. A.; Muchaluat-Saade, D. C. M. Extending ginga-ncl to specify multimodal interactions with multiple users. In *Anais do XXVI Simpósio Brasileiro de Multimídia e Web* (2020), SBC, pp. 212–219.

[11] Barreto, F.; Montevecchi, E. B. B.; Abreu, R.; dos Santos, J. A.; Muchaluat-Saade, D. C. Ncl: Storing user settings in media node. In *Anais Estendidos do XXV Simpósio Brasileiro de Sistemas Multimídia e Web* (2019), SBC, pp. 201–202.

[12] Barreto, F.; Montevecchi, E. B. B.; Abreu, R.; dos Santos, J. A.; Muchaluat-Saade, D. C. Providing multi-user in ncl with useragent and user-profile. In *Anais Estendidos do XXV Simpósio Brasileiro de Sistemas Multimídia e Web* (2019), SBC, pp. 205–206.

[13] Barreto, F.; Montevecchi, E. B. B.; Abreu, R.; dos Santos, J. A.; Muchaluat-Saade, D. C. Providing multimodal user interaction in ncl. In *Anais Estendidos do XXV Simpósio Brasileiro de Sistemas Multimídia e Web* (2019), SBC, pp. 203–204.

[14] Batista, C. E. C.; Soares, L. F. G.; de Souza Filho, G. L. Estendendo o uso das classes de dispositivos ginga-ncl. In *Anais Principais do XVI Simpósio Brasileiro de Sistemas Multimídia e Web* (2010), SBC, pp. 27–34.

[15] Bijl, D.; Hyde-Thomson, H. Speech to text conversion, Jan. 9 2001. US Patent 6,173,259.

[16] Blakowski, G.; Steinmetz, R. A media synchronization survey: Reference model, specification and case studies. *Journal on Selected Areas in Communications 14*, 1 (January 1996), 5–35.

[17] Bolt, R. A. "put-that-there" voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques* (1980), pp. 262–270.

[18] Campbell, B.; Goodman, J. M. Ham: A general-purpose hypertext abstract machine. In *Proceedings of the ACM conference on Hypertext* (1987), ACM, pp. 21–32.

[19] Card, S.; MORAN, T.; Newell, A. The model human processor- an engineering model of human performance. *Handbook of perception and human performance. 2*, 45–1 (1986).

[20] Carvalho, L.; Macedo, H. Estendendo a ncl para promover interatividade vocal em aplicações ginga. In *Anais Principais do XVI Simpósio Brasileiro de Sistemas Multimídia e Web* (2010), SBC, pp. 227–234.

[21] Consortium, W. W. W., et al. Namespaces in xml 1.1.

[22] de Farias, B. C.; de Lima Filho, E. B.; Maia, O. B.; Souto, E. Extensions to middleware ginga for integration with iot environments. In *2020 IEEE International Conference on Consumer Electronics (ICCE)* (2020), IEEE, pp. 1–5.

[23] de Lima, E. S.; Feijó, B.; Barbosa, S.; da Silva, F. G.; Furtado, A. L.; Ciarlini, A. E.; Pozzer, C. T. Multimodal, multi-user and adaptive interaction for interactive storytelling applications. In *2011 Brazilian Symposium on Games and Digital Entertainment* (2011), IEEE, pp. 206–214.

[24] Farias, F.; Montevecchi, E.; Bokehi, J.; Santana, R.; Muchaluat-Saade, D. Memo-vr: Exercício cognitivo para idosos utilizando realidade virtual e interface com as mãos. In *Anais do XX Simpósio Brasileiro de Computação Aplicada à Saúde* (2020), SBC, pp. 434–439.

[25] Fu, Z.; Zhou, Y. Research on remote learning in multimodal interaction. In *International Association of Societies of Design Research conferences* (2019).

[26] Furht, B. *Encyclopedia of multimedia*. Springer Science & Business Media, 2008.

[27] Gilroy, G.; Ellinoy, B. J.; Nelson, G. E.; Cantrill, S. V. Integration of pharmacy into the computerized problem-oriented medical information system (promis)—a demonstration project. *American Journal of Health-System Pharmacy 34*, 2 (1977), 155–162.

[28] Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems 29*, 7 (2013), 1645–1660.

[29] Guedes, A. L.; de Albuquerque Azevedo, R. G.; Colcher, S.; Barbosa, S. D. 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), Webmedia '16, ACM, pp. 39–46.

[30] Guedes, Á. L.; de Albuquerque Azevedo, R. G.; Colcher, S.; Barbosa, S. D. Extending ncl to support multiuser and multimodal interactions. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web* (2016), pp. 39–46.

[31] Guedes, A. L. V.; Azevedo, R. G. d. A.; Moreno, M. F.; Soares, L. F. G. Specification of multimodal interactions in ncl. In *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web* (2015), ACM, pp. 181–187.

[32] Guedes, Á. L. V.; de Albuquerque Azevedo, R. G.; Barbosa, S. D. J. Extending multimedia languages to support multimodal user interactions. *Multimedia Tools and Applications* (2016), 1–30.

[33] Halasz, F.; Schwartz, M. The dexter hypertext reference model. *Commun. ACM 37*, 2 (Feb. 1994), 30–39.

[34] Halsall, F. *Multimedia communications: applications, networks, protocols, and standards*. Pearson education, 2001.

[35] Hardman, H. L. *Modeling and Authoring Hypermedia Documents*. Tese de Doutorado, Universität Amsterdam, 1998.

[36] Hardman, L.; Bulterman, D. C. A.; van Rossum, G. The amsterdam hypermedia model: Adding time and context to the dexter model. *Commun. ACM 37*, 2 (Feb. 1994), 50–62.

[37] Hunkeler, U.; Truong, H. L.; Stanford-Clark, A. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)* (2008), IEEE, pp. 791–798.

[38] Ierusalimschy, R.; De Figueiredo, L. H.; Filho, W. C. Lua—an extensible extension language. *Software: Practice and Experience 26*, 6 (1996), 635–652.

[39] ISO/IEC. *Information technology — Multimedia framework (MPEG-21) — Part 22: User Description*. International Organization for Standardization, 2019. ISO/IEC 21000-22:2019.

[40] ITU. Nested context language (ncl) and ginga-ncl for iptv services. http://www.itu.int/rec/T-REC-H.761-200904-S, 2009. ITU-T Recommendation H.761.

[41] Jalal, L.; Popescu, V.; Murroni, M. Iot architecture for multisensorial media. In *2017 IEEE URUCON* (2017), IEEE, pp. 1–4.

[42] Josué, M.; Abreu, R.; Barreto, F.; Mattos, D.; Amorim, G.; dos Santos, J.; Muchaluat-Saade, D. Modeling sensory effects as first-class entities in multimedia applications. In *Proceedings of the 9th ACM Multimedia Systems Conference* (New York, NY, USA, 2018), MMSys '18, ACM, pp. 225–236.

[43] Josué, M.; Muchaluat-Saade, D.; Moreno, M. Preparation of media object presentation and sensory effect rendering in mulsemedia applications. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web* (2018), ACM, pp. 45–52.

[44] Josué, M. I. P. *Preparação de Objetos de Mídia e Efeitos Sensoriais para Formatação de Documentos Mulsemídia.* Tese de Doutorado, Instituto de Computação, UFF, Niterói, Brasil, 2021.

[45] Kim, S.; Han, J. Text of white paper on mpeg-v. In *San Jose, USA. MPEG Group Meeting, ISO/IEC JTC* (2014), vol. 1.

[46] Klyne, G.; Carroll, J. J. Resource description framework (rdf): Concepts and abstract syntax.

[47] Kopp, S.; Krenn, B.; Marsella, S.; Marshall, A. N.; Pelachaud, C.; Pirker, H.; Thórisson, K. R.; Vilhjálmsson, H. Towards a common framework for multimodal generation: The behavior markup language. In *International workshop on intelligent virtual agents* (2006), Springer, pp. 205–217.

[48] Luque, F. P.; Galloso, I.; Feijoo, C.; Martín, C. A.; Cisneros, G. Integration of multisensorial stimuli and multimodal interaction in a hybrid 3dtv system. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 11*, 1s (2014), 1–22.

[49] Mattos, D. P. D.; Muchaluat-Saade, D. C.; Ghinea, G. Beyond multimedia authoring: On the need for mulsemedia authoring tools. *ACM Comput. Surv. 54*, 7 (July 2021).

[50] McCracken, D. L.; Akscyn, R. M. Experience with the zog human-computer interface system. *International Journal of Man-Machine Studies 21*, 4 (1984), 293–310.

[51] McGill, M.; Williamson, J. H.; Brewster, S. A. A review of collocated multi-user tv. *Personal and Ubiquitous Computing 19*, 5 (2015), 743–759.

[52] Montevecchi, E. B. B.; Josué, M. I.; Barreto, F.; de Abreu, R. S.; Muchaluat-Saade, D. C. Providing eye gaze interaction for ginga-ncl applications. In *Proceedings of the Brazilian Symposium on Multimedia and the Web* (2020), pp. 297–303.

[53] Muchaluat-Saade, D. *Relations in Hypermedia Authoring Languages: Improving Reuse and Expressiveness.* Tese de Doutorado, PhD Thesis, Informatics Department, PUC-Rio, Rio de Janeiro, Brazil, 2003.

[54] Muchaluat-Saade, D. C. *Relations in Hypermedia Authoring Languages: Improving Reuse and Expressiveness.* Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, March 2003.

[55] Muchaluat-Saade, D. C.; Pita, R.; Garcés, L.; Oliveira, B.; Arenas, C.; Ivanov, M.; Montevecchi, E.; Abreu, R.; Barreto, F.; Santos, J., et al. Minicursos do sbcas 2020.

[56] Muchaluat-Saade, D. C.; Soares, L. F. G. Xconnector & xtemplate: Improving the expressiveness and reuse in web authoring languages. *The New Review of Hypermedia and Multimedia Journal 8*, 1 (2002), 139–169.

[57] Nery e Silva, L. D.; Tavares, T. A.; de Souza Filho, G. L. Desenvolvimento de programas de tvdi explorando as funções inovadoras do ginga-j. In *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web* (2008), pp. 75–82.

[58] Pedrosa, D.; Martins Jr, J. A. C.; Maria da Graça, C. P.; Melo, E. Componente de interação multimodal no ginga. In *Anais Estendidos do XVI Simpósio Brasileiro de Sistemas Multimídia e Web* (2010), SBC, pp. 197–202.

[59] Pereira, D. M. G.; da Silva, F. J.; Neto, C. d. S. S.; Guedes, Á. L. V., et al. A middleware perspective for integrating ginga-ncl applications with the internet of things. In *Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web* (2017), SBC, pp. 70–75.

[60] Pereira, D. M. G.; e Silva, F. J. d. S.; Carlos de Salles, S. N.; dos Santos, D. V.; Coutinho, L. R.; Guedes, Á. L. An ontology-based approach to integrate tv and iot middlewares. *Multimedia Tools and Applications 80*, 2 (2021), 1813–1837.

[61] Rodrigues, R. F. *Formatação e Controle de Apresentações Hipermídia com mecanismos de Adaptação Temporal.* Tese de Doutorado, Pontifical Catholic University of Rio de Janeiro, Brazil, 2003.

[62] Sant'Anna, F.; Cerqueira, R.; Soares, L. F. G. Nclua: objetos imperativos lua na linguagem declarativa ncl. In *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web* (2008), ACM, pp. 83–90.

[63] Shneiderman, B.; Plaisant, C. *Designing the user interface: strategies for effective human-computer interaction.* Pearson Education India, 2010.

[64] Soares, L. F. G.; Barbosa, S. D. Programando em ncl 3.0: Desenvolvimento de aplicações para o middleware ginga. *Campus, Rio de Janeiro, RJ* (2009).

[65] Soares, L. F. G.; Barbosa, S. D. J. *Programando em NCL 3.0: Desenvolvimento de aplicações para o middleware Ginga 2a. Edição Versão 2.1.* Elsevier Campos, 2011.

[66] Soares, L. F. G.; Rodrigues, R. F. Nested context model 3.0 part 1 - ncm core. Tech. rep., Informatics Department, PUC-Rio, Rio de Janeiro, May 2005.

[67] Soares, L. F. G.; Rodrigues, R. F.; Moreno, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society 12*, 4 (2007), 37–46.

[68] Soares, L. F. G.; Rodrigues, R. F.; Moreno, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society 13*, 1 (2007), 37–46.

[69] Soares, L. F. G.; Rodrigues, R. F.; Saade, D. C. M. Modeling, authoring and formatting hypermedia documents in the hyperprop system. *Multimedia Systems 8*, 2 (2000), 118–134.

[70] Talavera, L. E.; Endler, M.; Vasconcelos, I.; Vasconcelos, R.; Cunha, M.; e Silva, F. J. d. S. The mobile hub concept: Enabling applications for the internet of mobile things. In *2015 IEEE International conference on pervasive computing and communication workshops (percom workshops)* (2015), IEEE, pp. 123–128.

[71] Thompson, H. S.; Beech, D.; Maloney, M.; Mendelsohn, N. Xml schema part 1: structures second edition. *W3C recommendation 39* (2004).

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

[73] Turunen, M.; Kallinen, A.; Sànchez, I.; Riekki, J.; Hella, J.; Olsson, T.; Melto, A.; Rajaniemi, J.-P.; Hakulinen, J.; Mäkinen, E., et al. Multimodal interaction with speech and physical touch interface in a media center application. In *Proceedings of the International Conference on Advances in Computer Enterntainment Technology* (2009), pp. 19–26.

[74] Valentim, P. A.; Barreto, F.; Muchaluat-Saade, D. C. Possibilitando o reconhecimento de expressões faciais em aplicações ginga-ncl. In *Anais Estendidos do XXVI Simpósio Brasileiro de Sistemas Multimídia e Web* (2020), SBC, pp. 53–56.

[75] W3C. Extensible markup language (xml) 1.0 (fifth edition), 2008. World-Wide Web Consortium Recommendation.

[76] W3C. Html5: A vocabulary and associated apis for html and xhtml. https://www.w3.org/TR/html5/, 2014. World-Wide Web Consortium Recommendation.

[77] Waltl, M.; Timmerer, C.; Hellwagner, H. Increasing the user experience of multimedia presentations with sensory effects. In *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10* (2010), IEEE, pp. 1–4.

[78] Want, R. An introduction to rfid technology. *IEEE pervasive computing*, 1 (2006), 25–33.

[79] Yoon, K.; Kim, S.-K.; Han, J. J.; Han, S.; Preda, M. *MPEG-V: bridging the virtual and real world*. Academic Press, 2015.

APÊNDICE A – Códigos Utilizados na Avaliação

A.1 Avaliação comparativa

A.1.1 Aplicações desenvolvidas para os testes em NCL 3.0

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="aplTesteStart" xmlns="ht tp://www.ncl.org.br/NCL3.0
    /EDTVProfile">
 <head>
  <regionBase>
    <region id="jardimBotanicoImgReg" width="100%" height="100%"
   zIndex="0"/>
      <region id="jardimBotanicoVideoReg" bottom ="10%" width="50%"
   height="45%" zIndex="2"/>
  </regionBase>
  <descriptorBase>
      <descriptor id="jardimBotanicoImgDesc"  region="
   jardimBotanicoImgReg"/>
      <descriptor id="jardimBotanicoVideoDesc"  region="
   jardimBotanicoVideoReg"/>
  </descriptorBase>
    <connectorBase>
  <causalConnector id="onBeginStart">
    <simpleCondition role="onBegin"/>
    <simpleAction role="start" max="unbounded"/>
  </causalConnector>
  <causalConnector id="onEndAttributionTestPause">
      <connectorParam name="val"/>
      <compoundCondition operator="and">
        <simpleCondition role="onEndAttribution"/>
```

```
21        <assessmentStatement comparator="eq">
22          <attributeAssessment role="attNodeTest" eventType="
    attribution" attributeType="nodeProperty"/>
23          <valueAssessment value="$val"/>
24        </assessmentStatement>
25      </compoundCondition>
26        <simpleAction role="pause"/>
27    </causalConnector>
28      <causalConnector id="onPauseStop">
29        <simpleCondition role="onPause"/>
30        <simpleAction role="stop"  max="unbounded"/>
31      </causalConnector>
32    </connectorBase>
33  </head>
34  <body>
35    <port id="pVideo" component="botanicalGardenImage" />
36    <media id="botanicalGardenImage" src="images/jardimBotanico.jpg"
    descriptor="jardimBotanicoImgDesc"/>
37  <media id="jardimBotanicoVideo" src="videos/JardimBotanico.mp4"
    descriptor="jardimBotanicoVideoDesc"/>
38  <media id="readIntModes" src="scripts/scriptInt.lua">
39    <property name="key"/>
40  </media>
41  <link xconnector="onBeginStart">
42    <bind role="onBegin" component="botanicalGardenImage"/>
43    <bind role="start" component="readIntModes" />
44    <bind role="start" component="jardimBotanicoVideo" />
45  </link>
46  <link xconnector="onEndAttributionTestPause">
47    <bind role="onEndAttribution" component="readIntModes" interface
    = "key"/>
48    <bind role="attNodeTest" component="readIntModes" interface="key"
    >
49      <bindParam name="val" value="voice_recog/professor:pause"/>
50    </bind>
51    <bind role="pause" component="jardimBotanicoVideo"/>
52  </link>
53  <link xconnector="onPauseStop">
```

```
54      <bind role="onPause" component="jardimBotanicoVideo" />
55        <bind role="stop" component="botanicalGardenImage"/>
56        <bind role="stop" component="readIntModes"/>
57    </link>
58 </body>
59 </ncl>
```

Listagem A.1: Aplicação em NCL 3.0 utilizanda na avaliação comparativa para a ação pause

```
1  local server = 'localhost'
2  local port=1883
3  local mqtt = require("mqtt")
4  local int cont = 0
5  local topicos= {'voice_recog', 'onFaceRecognition', '
       onGestureRecognition',
6                  'onEyeMotion', 'onMotion', 'onTouch', 'onPointer'}
7  function tprint (tbl, indent)
8   if not indent then indent = 0 end
9   for k, v in pairs(tbl) do
10   formatting = string.rep("  ", indent) .. k .. ": "
11  end
12 end
13 local callback = function(topic, payload)
14   cont = cont +1
15   local evt1 = {
16             class  = 'ncl',
17             type   = 'attribution',
18             name   = 'key',
19             value = topic.."/"..payload,
20       }
21       evt1.action = 'start'
22       event.post(evt1)
23       evt1.action = 'stop'
24       event.post(evt1)
25 end
26 local client = mqtt.client.create(server,port, callback)
27 client:connect('pegaInteracao')
28 function update ()
```

```
29      client:handler()
30      event.timer(500, update)
31  end
32  function tratador (evt)
33    if evt.action == 'start' then
34      for i, v in ipairs(topicos) do
35        client:subscribe({v})
36      end
37      update()
38    end
39    if evt.action == 'stop' then
40      client:unsubscribe({topico})
41      client:disconnect()
42      client:destroy()
43    end
44  end
45  event.register(tratador)
```

Listagem A.2: Script Lua utilizado pela aplicação em NCL 3.0 utilizanda na avaliação comparativa para a ação pause

A.1.2 Aplicações desenvolvidas para os testes em NCL 4.0

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ncl id="aplTesteStart" xmlns="ht tp://www.ncl.org.br/NCL3.0
     /EDTVProfile">
3   <head>
4    <regionBase>
5    <region id="jardimBotanicoImgReg" width="100%" height="100%"
      zIndex="0" />
6    <region id="jardimBotanicoVideoReg" bottom ="10%" width="50%"
      height="45%" zIndex="1"/>
7    </regionBase>
8    <descriptorBase>
9      <descriptor id="jardimBotanicoImgDesc"  region="
     jardimBotanicoImgReg"/>
10     <descriptor id="jardimBotanicoVideoDesc"  region="
     jardimBotanicoVideoReg"/>
```

```
11    </descriptorBase>
12    <connectorBase>
13      <causalConnector id="onVoiceRecognitionPause">
14        <connectorParam name="key"/>
15        <connectorParam name="user"/>
16        <simpleCondition role="onVoiceRecognition" key="$key" user="$
      user"/>
17        <simpleAction role="pause" />
18      </causalConnector>
19      <causalConnector id="onPauseStop">
20        <simpleCondition role="onPause"/>
21        <simpleAction role="stop" max="unbounded"/>
22      </causalConnector>
23      <causalConnector id="onBeginStart">
24        <simpleCondition role="onBegin"/>
25        <simpleAction role="start" />
26      </causalConnector>
27    </connectorBase>
28  </head>
29 <body>
30    <port id="pVideo" component="botanicalGardenImage" />
31    <media id="botanicalGardenImage" src="images/jardimBotanico.jpg"
      descriptor="jardimBotanicoImgDesc"/>
32    <media id="jardimBotanicoVideo" src="videos/JardimBotanico.mp4"
      descriptor="jardimBotanicoVideoDesc"/>
33    <link xconnector="onVoiceRecognitionPause">
34      <bind role="onVoiceRecognition" component="botanicalGardenImage">
35        <bindParam name="key" value="pause"/>
36        <bindParam name="user" value="professor"/>
37      </bind>
38      <bind role="pause" component="jardimBotanicoVideo"/>
39    </link>
40    <link xconnector="onPauseStop">
41      <bind role="onPause" component="jardimBotanicoVideo" />
42      <bind role="stop" component="botanicalGardenImage"/>
43      <bind role="stop" component="jardimBotanicoVideo"/>
44    </link>
45    <link xconnector="onBeginStart">
```

```
46      <bind role="onBegin" component="botanicalGardenImage" />
47      <bind role="start" component="jardimBotanicoVideo"/>
48    </link>
49  </body>
50 </ncl>
```

Listagem A.3: Aplicação em NCL 4.0 utilizanda na avaliação comparativa para a ação pause

A.2 Avaliação de carga

A.2.1 Aplicações desenvolvidas para os testes da capacidade de tratar eventos em NCL 4.0

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <ncl id="aplTesteEventos" xmlns="http://www.ncl.org.br/NCL3.0
    /EDTVProfile">
3  <head>
4   <regionBase>
5     <region id="backgroundImgReg" width="50%" height="50%" zIndex="0"
    />
6     </regionBase>
7   <descriptorBase>
8     <descriptor id="bkgImgDesc"  region="backgroundImgReg"/>
9   </descriptorBase>
10  <userBase>
11       <userAgent id="joao"/>
12   </userBase>
13   <connectorBase>
14     <causalConnector id="onBeginStart">
15      <simpleCondition role="onBegin"/>
16      <simpleAction role="start" max="unbounded"/>
17     </causalConnector>
18    <causalConnector id="onVoiceRecognitionSet">
19      <connectorParam name="val"/>
20      <connectorParam name="key"/>
21      <connectorParam name="user"/>
```

```
22        <simpleCondition role="onVoiceRecognition" key="$key" user="$
    user"/>
23        <simpleAction role="set" value ="$val" />
24      </causalConnector>
25      <causalConnector id="onFaceRecognitionSet">
26        <connectorParam name="val"/>
27        <connectorParam name="key"/>
28        <connectorParam name="user"/>
29        <simpleCondition role="onFaceRecognition" key="$key" user="$user
    "/>
30        <simpleAction role="set" value="$val" />
31      </causalConnector>
32      <causalConnector id="onVoiceRecognitionStop">
33        <connectorParam name="val"/>
34        <connectorParam name="key"/>
35        <connectorParam name="user"/>
36        <simpleCondition role="onVoiceRecognition" key="$key" user="$
    user"/>
37        <simpleAction role="stop" max = "unbounded"/>
38      </causalConnector>
39    </connectorBase>
40  </head>
41  <body>
42      <port id="pBkg" component="fundoImage" />
43      <media id="fundoImage" src="images/Backg.png" descriptor="
    bkgImgDesc">
44          <property name="evento" value="false"/>
45      </media>
46      <link xconnector="onVoiceRecognitionSet">
47        <bind role="onVoiceRecognition" component="fundoImage">
48          <bindParam name="key" value="play"/>
49          <bindParam name="user" value="joao"/>
50        </bind>
51        <bind role="set" component="fundoImage" interface="evento">
52                <bindParam name="val" value="1"/>
53        </bind>
54      </link>
55      <link xconnector="onFaceRecognitionSet">
```

```
56        <bind role="onFaceRecognition" component="fundoImage">
57          <bindParam name="key" value="happy"/>
58          <bindParam name="user" value="joao"/>
59        </bind>
60        <bind role="set" component="fundoImage" interface="evento">
61                <bindParam name="val" value="1"/>
62        </bind>
63      </link>
64      <link xconnector="onVoiceRecognitionStop">
65        <bind role="onVoiceRecognition" component="fundoImage">
66          <bindParam name="key" value="end"/>
67          <bindParam name="user" value="joao"/>
68        </bind>
69        <bind role="stop" component="fundoImage"/>
70      </link>
71  </body>
72  </ncl>
```

Listagem A.4: Aplicação em NCL 4.0 utilizanda na avaliação da capacidade de captura de eventos do ginga

A.3 Avaliação de Desempenho

A.3.1 Aplicações desenvolvidas para os testes de desempenho ao tratar os eventos em NCL 4.0

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <ncl id="aplMultiModal" xmlns="http://www.ncl.org.br/NCL3.0
    /EDTVProfile">
3  <head>
4   <regionBase>
5     <region id="backgroundImgReg" width="50%" height="50%" zIndex="0"
    />
6       <region id="Img1Reg" width="50%" height="50%" top="20%" zIndex=
    "0"/>
7       <region id="Img2Reg" width="50%" height="50%" top="40%" zIndex=
    "0"/>
```

```
8       <region id="Img3Reg" width="50%" height="50%" top="60%" zIndex=
    "0"/>
9       <region id="Img4Reg" width="50%" height="50%" top="80%" zIndex=
    "0"/>
10     </regionBase>
11     <descriptorBase>
12       <descriptor id="bkgImgDesc"  region="backgroundImgReg"/>
13       <descriptor id="Img1Desc"  region="Img1Reg" />
14       <descriptor id="Img2Desc"  region="Img2Reg" />
15       <descriptor id="Img3Desc"  region="Img3Reg" />
16       <descriptor id="Img4Desc"  region="Img4Reg" />
17     </descriptorBase>
18     <userBase>
19       <userAgent id="joao"/>
20     </userBase>
21
22   <connectorBase>
23       <causalConnector id="onBeginStart">
24        <simpleCondition role="onBegin"/>
25        <simpleAction role="start" max="unbounded"/>
26       </causalConnector>
27       <causalConnector id="onVoiceRecognitionSet">
28        <connectorParam name="val"/>
29        <connectorParam name="key"/>
30        <connectorParam name="user"/>
31        <simpleCondition role="onVoiceRecognition" key=$key user="$
    user"/>
32        <simpleAction role="set" value="$val" />
33       </causalConnector>
34       <causalConnector id="onFaceRecognitionSet">
35         <connectorParam name="val"/>
36         <connectorParam name="key"/>
37         <connectorParam name="user"/>
38         <simpleCondition role="onFaceRecognition" key="$key" user="$
    user"/>
39         <simpleAction role="set" value ="$val" />
40       </causalConnector>
41       <causalConnector id="onVoiceRecognitionStop">
```

```
42          <connectorParam name="val"/>
43          <connectorParam name="key"/>
44          <connectorParam name="user"/>
45          <simpleCondition role="onVoiceRecognition" key="$key" user =
    "$user"/>
46          <simpleAction role="stop" max = "unbounded"/>
47        </causalConnector>
48    </connectorBase>
49  </head>
50  <body>
51    <port id="pBkg" component="fundoImage" />
52    <media id="fundoImage" src="images/Backg.png" descriptor="
     bkgImgDesc">
53      <area id="a1" begin="1s"/>
54      <area id="a2" begin="2s"/>
55      <area id="a3" begin="3s"/>
56      <area id="a4" begin="4s"/>
57      <property name="evento" value="false"/>
58    </media>
59    <media id="Img1" src="images/abelha.jpg" descriptor="Img1Desc"/>
60    <media id="Img2" src="images/golfinho.jpg" descriptor="Img2Desc"/>
61    <media id="Img3" src="images/askRed.png" descriptor="Img3Desc"/>
62    <media id="Img4" src="images/play.jpg" descriptor="Img4Desc"/>
63    <link xconnector ="onBeginStart">
64      <bind role = "onBegin" component="fundoImage" interface="a1"/>
65      <bind role = "start" component="Img1"/>
66    </link>
67    <link xconnector ="onBeginStart">
68      <bind role = "onBegin" component="fundoImage" interface="a2"/>
69      <bind role = "start" component="Img2"/>
70    </link>
71    <link xconnector ="onBeginStart">
72      <bind role = "onBegin" component="fundoImage" interface="a3"/>
73      <bind role = "start" component="Img3"/>
74    </link>
75    <link xconnector ="onBeginStart">
76      <bind role = "onBegin" component="fundoImage" interface="a4"/>
77      <bind role = "start" component="Img4"/>
```

```
78    </link>
79    <link xconnector="onVoiceRecognitionSet">
80      <bind role="onVoiceRecognition" component="fundoImage">
81        <bindParam name="key" value="play"/>
82        <bindParam name="user" value="joao"/>
83      </bind>
84      <bind role="set" component="fundoImage" interface= "evento">
85        <bindParam name="val" value="1"/>
86      </bind>
87    </link>
88    <link xconnector="onFaceRecognitionSet">
89      <bind role="onFaceRecognition" component="fundoImage">
90        <bindParam name="key" value="happy"/>
91        <bindParam name="user" value="joao"/>
92      </bind>
93      <bind role="set" component="fundoImage" interface= "evento">
94          <bindParam name="val" value="1"/>
95      </bind>
96    </link>
97    <link xconnector="onVoiceRecognitionStop">
98      <bind role="onVoiceRecognition" component="fundoImage">
99        <bindParam name="key" value="end"/>
100       <bindParam name="user" value="joao"/>
101     </bind>
102     <bind role="stop" component="fundoImage"/>
103     <bind role="stop" component="Img1"/>
104     <bind role="stop" component="Img2"/>
105     <bind role="stop" component="Img3"/>
106     <bind role="stop" component="Img4"/>
107   </link>
108 </body>
109 </ncl>
```

Listagem A.5: Aplicação em NCL 4.0 utilizanda na avaliação de desempenho na execução um aplicação com o tratamentos de eventos no ginga

A.4 Avaliação Multiusuário

A.4.1 Aplicações desenvolvidas para os testes de multiusuário em NCL 4.0

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="apTesteUser" xmlns="http://www.ncl.org.br/NCL4.0/EDTVProfile
  ">
 <head>
  <regionBase>
      <region id="florestaVideoReg" width="100%" height="100%"
   zIndex="0" />
  </regionBase>
  <descriptorBase>
      <descriptor id="florestaVideoRegDesc"  region="florestaVideoReg
   "/>
  </descriptorBase>
  <connectorBase>
      <causalConnector id="onVoiceRecognitionPause">
        <connectorParam name="key"/>
        <connectorParam name="user"/>
        <simpleCondition role="onVoiceRecognition" key="$key" user="$
   user"/>
        <simpleAction role="pause" />
      </causalConnector>
      <causalConnector id="onBeginStop">
        <simpleCondition role="onBegin"/>
        <simpleAction role="stop" />
      </causalConnector>
      <causalConnector id="onPauseStop">
        <simpleCondition role="onPause"/>
        <simpleAction role="stop" />
      </causalConnector>
  </connectorBase>
  <userBase>
      <userProfile id="profile1" src="profiles/profile.xml"/>
  </userBase>
 </head>
```

```
<body>
  <port id="pVideo" component="florestaVideo" />
  <media id="florestaVideo" src="images/florest.png" descriptor="
   florestaVideoRegDesc"/>
    <link xconnector="onVoiceRecognitionPause">
      <bind role="onVoiceRecognition" component="florestaVideo">
        <bindParam name="key" value="pausar1"/>
        <bindParam name="user" value="profile1"/>
      </bind>
      <bind role="pause" component="florestaVideo"/>
    </link>
    <link xconnector="onVoiceRecognitionPause">
          <bind role="onVoiceRecognition" component="florestaVideo">
            <bindParam name="key" value="pausar2"/>
            <bindParam name="user" value="profile1"/>
          </bind>
          <bind role="pause" component="florestaVideo"/>
    </link>
    .
    .
    .
    <link xconnector="onVoiceRecognitionPause">
      <bind role="onVoiceRecognition" component="florestaVideo">
        <bindParam name="key" value="pausar50"/>
        <bindParam name="user" value="profile1"/>
      </bind>
      <bind role="pause" component="florestaVideo"/>
    </link>
    <link xconnector="onBeginStop">
      <bind role="onBegin" component="florestaVideo"/>
      <bind role="stop" component="florestaVideo"/>
    </link>
 </body>
</ncl>
```

Listagem A.6: Aplicação em NCL 4.0 utilizanda na avaliação de MultiUsuário na execução um aplicação com o tratamentos de eventos no ginga

APÊNDICE B – Esquema NCL 4.0

B.1 NCL40User.xsd

```
<!--
XML Schema for the NCL modules
This is NCL
Public URI: http://www.ncl.org.br/NCL4.0/modules/NCL40User.xsd
Author:
Revision: 14/11/2019
Schema for the NCL Effect module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:user="http://www.ncl.org.br/NCL4.0/User"
  targetNamespace="http://www.ncl.org.br/NCL4.0/User"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="userBasePrototype">
     <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <complexType name="userParamPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="required"/>
  </complexType>

  <complexType name="userProfilePrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="user:userParam"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
```

```
28      <attribute name="min" type="int" use="optional"/>
29      <attribute name="max" type="int" use="optional"/>
30      <attribute name="src" type="string" use="required"/>
31    </complexType>
32
33    <complexType name="userAgentPrototype">
34      <sequence minOccurs="0" maxOccurs="unbounded">
35        <element ref="user:userParam"/>
36      </sequence>
37      <attribute name="id" type="ID" use="required"/>
38      <attribute name="src" type="string" use="optional"/>
39    </complexType>
40
41
42  <!-- declare global elements in this module -->
43    <element name="userBase" type="user:userBasePrototype"/>
44    <element name="userParam" type="user:userParamPrototype"/>
45    <element name="userAgent" type="user:userAgentPrototype"/>
46    <element name="userProfile" type="user:userProfilePrototype"/>
47
48  </schema>
```

Listagem B.1: Esquema do modulo user

B.2 Alteração no NCL30ConnectorCausalExpression.xsd

```
1  <!--
2  XML Schema for the NCL modules
3  This is NCL
4  Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights
       Reserved.
5  See http://www.telemidia.puc-rio.br
6  Public URI: http://www.ncl.org.br/NCL3.0/modules/
       NCL30ConnectorCausalExpression.xsd
7  Author: TeleMidia Laboratory
8  Revision: 19/09/2006
9  Schema for the NCL Connector Causal Expression module namespace.
10 -->
```

```
<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 the definitions in the modules namespaces -->
<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="onBeginPreparation" />
    <enumeration value="onEndPreparation" />
    <enumeration value="onAbortPreparation" />
    <enumeration value="onVoiceRecognition" />
    <enumeration value="onFaceRecognition" />
    <enumeration value="onGestureRecognition" />
  </restriction>
</simpleType>

<simpleType name="maxUnion">
  <union memberTypes="positiveInteger
   connectorCausalExpression:unboundedString"/>
```

```
43 </simpleType>
44
45 <simpleType name="unboundedString">
46   <restriction base="string">
47     <pattern value="unbounded"/>
48   </restriction>
49 </simpleType>
50
51 <complexType name="simpleConditionPrototype">
52   <attribute name="role" type="
       connectorCausalExpression:conditionRoleUnion" use="required"/>
53   <attribute name="eventType" type="
       connectorCommonPart:eventPrototype" use="optional"/>
54   <attribute name="key" type="string" use="optional"/>
55   <attribute name="transition" type="
       connectorCommonPart:transitionPrototype" use="optional"/>
56   <attribute name="delay" type="string" use="optional"/>
57   <attribute name="min" type="positiveInteger" use="optional"/>
58   <attribute name="max" type="connectorCausalExpression:maxUnion" use
       ="optional"/>
59   <attribute name="qualifier" type="
       connectorCommonPart:logicalOperatorPrototype" use="optional"/>
60 </complexType>
61
62 <complexType name="compoundConditionPrototype">
63   <attribute name="operator" type="
       connectorCommonPart:logicalOperatorPrototype" use="required"/>
64   <attribute name="delay" type="string" use="optional"/>
65 </complexType>
66
67 <simpleType name="actionRoleUnion">
68   <union memberTypes="string
       connectorCausalExpression:actionNamePrototype"/>
69 </simpleType>
70
71 <simpleType name="actionNamePrototype">
72   <restriction base="string">
73     <enumeration value="start" />
```

```
74      <enumeration value="stop" />
75      <enumeration value="pause" />
76      <enumeration value="resume" />
77      <enumeration value="abort" />
78      <enumeration value="set" />
79      <enumeration value="startPreparation" />
80      <enumeration value="stopPreparation" />
81      <enumeration value="abortPreparation" />
82    </restriction>
83  </simpleType>
84
85  <simpleType name="actionOperatorPrototype">
86    <restriction base="string">
87      <enumeration value="par" />
88      <enumeration value="seq" />
89    </restriction>
90  </simpleType>
91
92  <complexType name="simpleActionPrototype">
93    <attribute name="role" type="
        connectorCausalExpression:actionRoleUnion" use="required"/>
94    <attribute name="eventType" type="
        connectorCommonPart:eventPrototype" use="optional"/>
95    <attribute name="actionType" type="
        connectorCausalExpression:actionNamePrototype" use="optional"/>
96    <attribute name="delay" type="string" use="optional"/>
97    <attribute name="value" type="string" use="optional"/>
98    <attribute name="repeat" type="positiveInteger" use="optional"/>
99    <attribute name="repeatDelay" type="string" use="optional"/>
100   <attribute name="min" type="positiveInteger" use="optional"/>
101   <attribute name="max" type="connectorCausalExpression:maxUnion" use
        ="optional"/>
102   <attribute name="qualifier" type="
        connectorCausalExpression:actionOperatorPrototype" use="optional"/
        >
103 </complexType>
104
105 <complexType name="compoundActionPrototype">
```

```
106   <choice minOccurs="2" maxOccurs="unbounded">
107     <element ref="connectorCausalExpression:simpleAction" />
108     <element ref="connectorCausalExpression:compoundAction" />
109   </choice>
110   <attribute name="operator" type="
        connectorCausalExpression:actionOperatorPrototype" use="required"/
        >
111   <attribute name="delay" type="string" use="optional"/>
112 </complexType>
113
114   <!-- declare global elements in this module -->
115 <element name="simpleCondition" type="
      connectorCausalExpression:simpleConditionPrototype" />
116 <element name="compoundCondition" type="
      connectorCausalExpression:compoundConditionPrototype" />
117 <element name="simpleAction" type="
      connectorCausalExpression:simpleActionPrototype" />
118 <element name="compoundAction" type="
      connectorCausalExpression:compoundActionPrototype" />
119
120 </schema>
```

Listagem B.2: Esquema do conector causal contendo os novos papeis propostos nesta tese