Erick Lazaro Melo

Uma arquitetura para autoria ubíqua de documentos a partir de apresentações de TV Digital

São Carlos, SP - Brasil

Fevereiro de 2010

Erick Lazaro Melo

Uma arquitetura para autoria ubíqua de documentos a partir de apresentações de TV Digital

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, da Universidade Federal de São Carlos, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação

Orientador:
Cesar Augusto Camillo Teixeira

Co-orientador:
Antônio Francisco do Prado

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

São Carlos, SP - Brasil

Fevereiro de 2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M528aa

Melo, Erick Lazaro.
Uma arquitetura para autoria ubíqua de documentos a partir de apresentações de TV Digital / Erick Lazaro Melo. -- São Carlos : UFSCar, 2010.
90 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2010.

1. Multimídia - programas de computador. 2. Televisão digital. 3. Engenharia de documentos. 4. Autoria multimídia. 5. Middleware. I. Título.

CDD: 006.7 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Arquitetura para Autoria Ubíqua de Documentos a partir de apresentações de TV Digital"

ERICK LAZARO MELO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Cesar Augusto Camillo Teixeira
(Orientador - DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(Co-orientador – DC/UFSCar)

Prof. Dr. Luiz Fernando Gomes Soares
(PUC-Rio)

Prof. Dr. Eduardo Barrére
(UFJF)

São Carlos
Fevereiro/2010

Agradecimentos

A Deus acima de tudo. Sou grato pelas bençãos derramadas em minha vida, pelo cuidado e por propiciar vida e saúde para que este trabalho pudesse ser desenvolvido.

À minha família pela paciência e compreensão nos momentos de *stress*, pela ausência nas confraternizações familiares, pela falta de tempo para me dedicar-me mais a vocês. Agradeço ainda pelos investimentos dados à minha educação. Eles foram fundamentais para que eu chegasse até aqui.

À Yara, meu amor, pela compreensão, apoio e palavras de motivação nos momentos difíceis.

Ao Caio Viel, meu braço direito. Não tenho nem palavras para expressar a minha gratidão a você...

Aos meus companheiros do *Lince* pelo companheirismo. Foi fundamental o apoio de vocês! Agradeço especialmente aos alunos de iniciação científica que contribuiram diretamente para a realização deste trabalho... Franco, Tiago e Suetônio, VALEU!!

Ao meu orientador Cesar Teixeira pela paciência e sábios conselhos. O tempo que passamos juntos foi uma grande escola...

Ao meu co-orientador Antônio Prado pelo incentivo e por acreditar na minha capacidade.

À professora Maria da Graça Pimentel pela parceria na elaboração de artigos e escrita de projetos. As discussões e sugestões contribuiram muito neste trabalho e foram fontes de aprendizado e incentivo à pesquisa.

Aos professores Luiz Fernando Soares e Eduardo Barrére pelos esforços de se deslocarem até São Carlos para participação da Banca de Defesa desta dissertação.

Às instituições parceiras do *Lince* que contribuem com os nossos trabalhos. Agradeço especialmente às agências de financiamento: CAPES, CNPq, FINEP e RNP que proveram recursos financeiros para que o trabalho pudesse ser desenvolvido.

Agradeço a todos que direta ou indiretamente contribuiram com este trabalho.

Resumo

É bastante comum pessoas realizarem comentários a respeito dos programas de TV que assistem conjuntamente. As facilidades tecnológicas disponíveis em ambientes de TV Digital podem permitir que esses comentários, e outros eventualmente realizados por interação outra além da oral, possam ser capturados e armazenados. Os comentários, devidamente associados ao programa original, podem resultar em outro documento que incorpora particularidades dessas pessoas e caracterizam um processo de autoria ou de edição. A naturalidade inerente à situação pode caracterizar como ubíquo esse processo de autoria/edição.

Este trabalho explora os conceitos de captura e acesso e propõe uma arquitetura computacional para viabilizar a autoria/edição ubíqua de documentos multimídia pelo interagente à medida que uma versão base vai sendo apresentada. O documento resultante pode substituir a versão base original, antes mesmo que sua apresentação chegue ao fim, e interferir na própria experiência original de interação, ou seja, o processo de autoria/edição desencadeia alterações no documento original que podem ser acessadas *on-the-fly*.

Os principais módulos que compõem a arquitetura são os módulos: gerenciador de edições do cliente, gerador de código, gerenciador de bases privadas, gerenciador de máquina de estados, gerenciador de entradas, gravador de *frames* e fluxos elementares, formatador de documentos multimídia e *players* de objetos de mídia.

A arquitetura proposta foi incorporada ao *middleware* Ginga para TV Digital. A linguagem *NCL* mostrou-se bastante adequada para a instanciação do modelo proposto já que propicia facilidades como edição ao vivo, múltiplos documentos e múltiplos dispositivos. O *middleware* Ginga tem a linguagem *NCL* como padrão para aplicações declarativas e tem uma arquitetura que pôde ser reusada e expandida com a incorporação dos novos componentes propostos neste trabalho. Alguns componentes da arquitetura do *middleware* apresentaram grande sinergia com o modelo proposto neste trabalho, permitindo o reúso em diversas instâncias.

Abstract

People comment about TV programs while watching. The technology found at Digital TV environment allow capturing and storing those comments. The combination between comments and the original program can result in a new multimedia document with personalized content. This is an ubiquitous authoring (editing) approach.

This work explore the capture and access concepts and propose an architecture to allow the ubiquitous authoring of multimedia documents by end-user while a base-document is presented. The result of this activity is a new multimedia document which can replace the original document on-the-fly, changing the user experience.

The main modules of the architecture are: client editing manger, code generator, private base manager, state machine manager, enhanced input manager, framebuffer recorder, elementary stream recorder, formatter (presentation engine) and media players.

The proposed architecture was integrated to the Brazilian Digital TV Middleware (*Ginga*). The NCL language features like live editing, support to multi devices and documents seems interesting to apply the proposed model. *Ginga* supports *NCL* language and has open specifications and a reference implementation that can be reused. The reference implementation was expanded by adding new components and reusing others.

Lista de Figuras

Sumário

1 Introdução

Pesquisadores da área de computação ubíqua investigam técnicas para prover serviços para usuários de forma transparente, sem que os mesmos tenham que se preocupar com procedimentos técnicos para a execução destes serviços.

Com base nos conceitos de captura e acesso [Abowd, Mynatt e Rodden 2002], propõem-se, em [Pimentel et al. 2007], que a experiência do usuário ao assistir um vídeo possa ser capturada. A experiência do usuário, através da sua interação com um vídeo, pode ser aproveitada na criação de novos documentos multimídia, que podem ser de interesse do próprio usuário ou de outras pessoas. O princípio é que interações do usuário possam promover anotações não intrusivas sobre o vídeo original, resultando em outro documento (vídeo anotado).

Os estudos nessa área foram aprofundados, culminando com o estabelecimento do paradigma *watch-and-comment* [Cattelan et al. 2008]. No paradigma *watch-and-comment*, ou WaC, são estabelecidos diversos princípios dos quais destacamos os seguintes: inexistência de restrições quanto à fonte do vídeo (o vídeo pode ser capturado ao vivo através de uma câmera, estar armazenado em um arquivo local ou acessível remotamente); inexistência de restrições quanto à linguagem do documento resultante; o documento declarativo resultante deve manter separadas da mídia original as anotações realizadas; a sessão pode ser colaborativa, síncrona e distribuída; a forma de captura da interação e a semântica não são restritas.

Abordando o que se percebe no cotidiano das pessoas pode-se observar que realizar anotações e fazer comentários sobre conteúdos são atividades bastante frequentes. Observa-se também que as facilidades tecnológicas têm viabilizado o oferecimento de serviços cada vez mais personalizados, aderentes aos perfis, gostos e preferências das pessoas, na busca de promover maior satisfação aos clientes.

A viabilização da autoria de conteúdo personalizado de forma ubíqua, aproveitando-se as anotações, comentários e interações que pessoas fazem naturalmente sobre um conteúdo base, pode contribuir com o aumento da satisfação que essas pessoas, ou terceiros, terão ao rever/interagir com o documento resultante.

Quando se fala aqui em autoria, não se trata de processos tradicionais de autoria de conteúdo que, geralmente, requerem o conhecimento de ferramentas dominadas apenas por especialistas (principalmente se o conteúdo tratar-se de vídeo; ou ainda de vídeo interativo). Reitera-se que a autoria a que se refere este trabalho baseia-se na captura de interações realizadas, em grande parte, de maneira natural por usuários, telespectadores, ou interagentes, que é o termo utilizado com maior frequência no texto para referências às pessoas que assistem e interagem com programas de TV, interativos ou não.

A flexibilização do que se refere por interação natural, ou seja, a inclusão de interações simples, porém realizadas de forma intencional para se obter uma anotação mais refinada e maior especificidade no documento que se cria, pode estender a aplicabilidade do paradigma WaC também a profissionais de mídia. Uma ferramenta simples de autoria, fundamentada em simples anotações e comentários, pode ser utilizada por diretores de cinema, vídeo ou TV para promoverem edições de alto nível em material básico, a serem refinadas posteriormente por técnicos com o uso de ferramentas especializadas.

Propõe-se neste trabalho uma arquitetura que apoie a construção de aplicações que aplicam o paradigma *WaC*. Foram levantados requisitos comuns a aplicações dessa categoria e desta forma foi enumerado um conjunto de componentes para atender a esses requisitos. Fazem parte dessa arquitetura componentes como um gerador de código, que permite a obtenção de uma representação textual de um documento multimídia em execução; um gerenciador de edições realizadas pelo cliente, que possibilita a distinção dos comandos de edição realizados pelo usuário (interagente) e emissora, permitindo que ações possam ser desfeitas; componentes para a gravação do conteúdo e sua replicação (permitindo a realização de anotações mais sofisticadas, como anotações com tinta eletrônica); mecanismos que possibilitem a obtenção da máquina de estados de um documento multimídia dentre outros.

1.1 Motivação

Tradicionalmente, o aparelho de televisão é um dispositivo passivo, cuja função essencial é exibir os sinais de vídeo recebidos da emissora ou de dispositivo local de armazenamento. Entretanto, a TV digital requer que o televisor passe a ser um dispositivo ativo, capaz de processar informações recebidas, já que além do conteúdo de vídeo e áudio, devem ser tratadas também aplicações enviadas pela emissora. Essa capacidade de processamento pode estar embutida no próprio televisor ou externa em dispositivos denominados, em língua inglesa, set-top-box. Nesse trabalho estaremos usando a denominação terminal de acesso para caracterizar o módulo

de processamento, quer esteja interno ou externo ao televisor.

Um terminal de acesso de TV Digital possui recursos semelhantes aos de um computador convencional. No cenário atual, entretanto, os recursos computacionais disponíveis em um terminal de acesso são mais restritos. Além da limitação de processamento e memória, os dispositivos de interação resumem-se à tela e alto-falante do televisor, e a um controle remoto.

Verificamos a oportunidade de explorar os conceitos do paradigma *watch-and-comment* também no ambiente de TV Digital. Embora tenha sido originalmente explorado em um ambiente de computador, o paradigma tem um grande potencial para ser aplicado também no ambiente de TV. Para isso, temos que levar em consideração a natureza da TV Digital. É um hábito comum assistir TV realizando comentários, individualmente ou coletivamente. Na TV Digital temos a oportunidade de registrar anotações e comentários feitos pelo usuário de uma forma ubíqua. Essas anotações podem ser aproveitadas tanto para vídeos gravados quanto para programas que estão sendo transmitidos pela emissora de TV.

1.2 Objetivo

A proposta deste trabalho é fornecer uma infraestrutura computacional adequada que possibilite a autoria de documentos em fase de exibição e interação por usuários leigos, não especialistas. Pretende-se fornecer ao usuário facilidades para a construção de conteúdo através da interação com a TV Digital, de forma transparente, sem a necessidade de ferramentas de autoria complexas. O processo de autoria não deve demandar procedimentos complexos. Exploramos o paradigma *watch-and-Comment* em um terminal de TV Digital convencional, respeitando suas limitações. O foco original do trabalho é o usuário final, interagente. Buscamos permitir que esse usuário deixe de ser apenas um ator passivo ao assistir TV, mas passe a exercer um papel ativo, tendo a possibilidade de produzir conteúdo. O conteúdo produzido pelo usuário traduzirá sua experiência com o conteúdo de TV em um formato que pode ser armazenado e compartilhado.

É importante observar que são suportadas modificações não planejadas previamente pelo autor do programa original. É permitido ao usuário modificar o documento em tempo de exibição. Isto significa que os comandos de edição do usuário podem afetar não somente partes do programa já apresentadas, mas também partes que ainda não foram apresentadas.

A nossa abordagem também pode ser aplicada como um ferramental complementar às ferramentas de autoria de vídeo tradicionais, em um ambiente de produção de conteúdo de TV. A ubiquidade oferecida por ferramentas que exploram o conceito *watch-and-comment* pode rev-

olucionar o trabalho em estúdios de TV, viabilizando pré-edição e anotação de conteúdo por parte de diretores, de uma forma natural. Um diretor pode transmitir suas impressões sobre um vídeo bruto ou em fase de edição adicionando marcas a esse vídeo, sem que seja necessário o domínio de ferramentas de edição de vídeo tradicionais, geralmente complexas.

Uma arquitetura é proposta neste trabalho para viabilizar o processo de autoria de forma ubíqua. Este trabalho não busca apresentar uma ferramenta de autoria, embora apresente alguns protótipos, para fins de validação da arquitetura proposta, que permitem que usuários leigos produzam documentos multimídia de forma interativa. Com essa arquitetura, é esperado que o processo de construção de ferramentas de autoria se torne menos custoso e mais poderoso.

1.3 Organização da Monografia

O Capítulo 2 apresenta alguns trabalhos (trabalhos acadêmicos e ferramentas comerciais) relacionados com a proposta apresentada, dando um panorama do estado da arte e identificando oportunidades de avanço. Nesse capítulo também são apresentados conceitos importantes utilizados no desenvolvimento do trabalho.

No Capítulo 3 é apresentado o trabalho de Engenharia de Domínio realizado para a obtenção dos requisitos comuns a aplicações de autoria ubíqua de documentos multimídia que exploram o paradigma *watch-and-comment*.

A arquitetura proposta é apresentada, de forma genérica, no Capítulo 4. No Capítulo 5 a arquitetura é apresentada de uma forma mais detalhada, através de uma instanciação no *middleware* Ginga. A validação da proposta através da construção de aplicações que utilizam-se da arquitetura proposta é descrita no Capítulo 6.

As considerações finais, destacando as contribuições do trabalho, publicações resultantes até o momento do desenvolvimento do trabalho, desdobramentos e perspectivas de trabalhos futuros são descritas no Capítulo 7.

2 *Trabalhos Relacionadas*

Nos primórdios da indústria cinematográfica a edição do conteúdo era feita de forma literalmente manual. As películas com as tomadas das câmeras eram cortadas e depois coladas na sequência desejada. Esse processo tornava a produção de vídeo complexa e custosa.

Com a possibilidade de digitalização do vídeo o processo de edição tornou-se mais poderoso e menos complexo. Ferramentas de autoria foram criadas, abstraindo parte da complexidade de edição do vídeo. Com essas ferramentas o autor pode concentrar sua atenção nas atividades criativas da produção do vídeo.

Essa motivação tem sido aplicada tanto para a construção de ferramentas de autoria de vídeo linear (não interativo) quanto para vídeo interativo. A autoria de um vídeo interativo traz uma complexidade adicional à construção de vídeos lineares: a construção de um *software*. Um vídeo interativo pode ser descrito como a combinação de diversos conteúdos audiovisuais organizados em uma aplicação (*software*).

A inserção de elementos de *software* no processo de criação de vídeos é um desafio, pois requer do profissional múltiplas competências. Além de se centrar no processo criativo de produção do vídeo é necessário que ele tenha conhecimentos computacionais (inerentes ao desenvolvimento de um *software*).

O advento da *Web 2.0* e redes sociais tem mostrado que o usuário, além de consumir conteúdo, também quer desempenhar um papel ativo na produção do conteúdo consumido. Cada vez mais o conteúdo é produzido de forma colaborativa pelos próprios usuários. A Internet se tornou uma infraestrutura para a divulgação desse conteúdo.

A nova geração de usuários de TV e computador tem demandado novos serviços que facilitem o processo de criação e publicação do conteúdo. Novos modelos de negócio e plataformas (como *Youtube*, *Wikipedia*, *ustream.tv*) têm sido construídos para dar respostas a esses anseios.

Existem diversas ferramentas de autoria para TV não-interativa, para TV interativa e para

Web [Berners-Lee et al. 1994]. Este capítulo traz uma análise de algumas dessas ferramentas, destacando o público-alvo atingido por cada uma delas. O capítulo trata ainda de alguns trabalhos relacionados a linguagens declarativas utilizadas para o sincronismo de mídias e conceitos de Engenharia de Software.

2.1 Linguagens multimídia

A especificação de uma apresentação multimídia é, de maneira geral, feita através de duas classes (paradigmas) de linguagens:

- *Linguagens imperativas* como *Java* e *C++*. Este paradigma é adotado em diversos sistemas de TV Digital. O *framework GEM* [Executable 2006] (linguagem Java) é um exemplo de plataforma que utiliza essa abordagem. Para o ambiente de *Web* também existem outras opções de linguagens como *Action Script*[1] [Moock 2007]

- *Linguagens declarativas* como as linguagens de sincronização de mídia *NCL - Nested Context Language* [Soares, Rodrigues e F. 2007] e *SMIL - Synchronized Multimedia Integration Language* [Bulterman et al. 2005]. São, em geral, projetadas para um domínio específico.

Linguagens declarativas parecem mais oportunas para a construção de aplicações multimídia. O processo de codificação torna-se menos complexo. Tarefas comuns à maioria das aplicações multimídia como o sincronismo de mídias e, particularmente, a interatividade, tornam-se tarefas relativamente simples em linguagens declarativas.

A arquitetura proposta no escopo deste trabalho levou isso em consideração. Optamos por uma abordagem direcionada a linguagens declarativas.

2.1.1 SMIL

SMIL (*Synchronized Multimedia Integration Language*) é uma linguagem padrão do W3C [2] para especificação de apresentações multimídia. Trata-se de uma linguagem declarativa (baseada em *XML*) e é adotada na construção de aplicações multimídia para a *Web* (alguns *browsers* e a plataforma *Flash*[3] a suportam).

[1] Linguagem adotada pela plataforma *Flash*
[2] *World Wide Web Consortium*
[3] Adobe Flash Player

A edição do conteúdo em tempo real não é contemplada pela linguagem e máquinas de apresentação disponíveis. Os relacionamentos na linguagem devem ser previamente definidos. O *player* de *SMIL GRiNS*[Bulterman et al. 1998], por exemplo, demanda o recebimento do documento na sua totalidade antes do início da apresentação. Alterações na estrutura da apresentação não são refletidas no cliente *on-the-fly*. Para visualizar as alterações é necessário recarregar o documento.

A linguagem suporta a especificação de regiões onde o conteúdo deve ser apresentado, mas não permite a associação de uma região a um dispositivo especificamente. Soluções alternativas são investigadas através de técnicas de distribuição automática do conteúdo (baseadas no perfil do usuário) ou através do uso de metalinguagens [Cesar et al. 2008].

As características da linguagem indicam algumas limitações. No cenário em que é desejável a alteração da estrutura do documento multimídia *on-the-fly* a linguagem parece não atender a todos os requisitos. Entretanto é válido o seu uso para a realização de anotações que serão visualizadas posteriormente. A ferramenta Ambulant Annotator [Cesar, Bulterman e Jansen 2006] é um caso de uso típico desse cenário.

2.1.2 NCL

A linguagem *NCL* (*Nested Context Language*) é uma linguagem declarativa, baseada em *XML* para a especificação de apresentações multimídia. Diferentemente de outras linguagens declarativas hipermídia, como *HTML* em que conteúdo é especificado no documento, *NCL* atua estritamente como um elemento de cola entre elementos multimídia.

A linguagem *NCL* é baseada no modelo *NCM*[Casanova et al. 1991] e foi adotada, no contexto do Sistema Brasileiro de TV Digital[SBTVD 2006], como linguagem padrão para aplicações declarativas.

A linguagem apresenta alguns recursos interessantes para o ambiente de TV Digital, dentre os quais destacamos:

- *Edição ao vivo*: o modelo *NCM*, no qual se baseia a linguagem favorece o suporte à alteração da especificação de um documento em fase de apresentação.
- *Suporte a múltiplos dispositivos*: a linguagem suporta a especificação, de forma declarativa, de uma apresentação associada a diversos dispositivos.
- *Suporte a múltiplos documentos*: a linguagem suporta a composição de uma apresentação através de diversos documentos, possibilitando o reúso de um mesmo documento em

contextos diferentes, combinado com outros documentos.

Os recursos oferecidos pela linguagem *NCL* mostram-se interessantes para uma abordagem de autoria focada no usuário, em que a ubiquidade (através de múltiplos dispositivos), possibilidade de anotação sobre um documento multimídia (múltiplos documentos) e a alteração do documento em fase de apresentação (edição ao vivo) são requeridos.

Máquina de estados do Documento

Aplicações multimídia possuem requisitos relacionados à sincronização de mídias. Em [Costa e Soares 2007] é apresentada uma estrutura que permite a interrupção de uma apresentação multimídia e recuperação de seu estado para exibição. É proposto um modelo de *Grafos Temporais Hipermídia*[Costa e Soares 2007] representando os eventos e seus relacionamentos. A Figura 2.1 apresenta a máquina de estados definida no modelo.

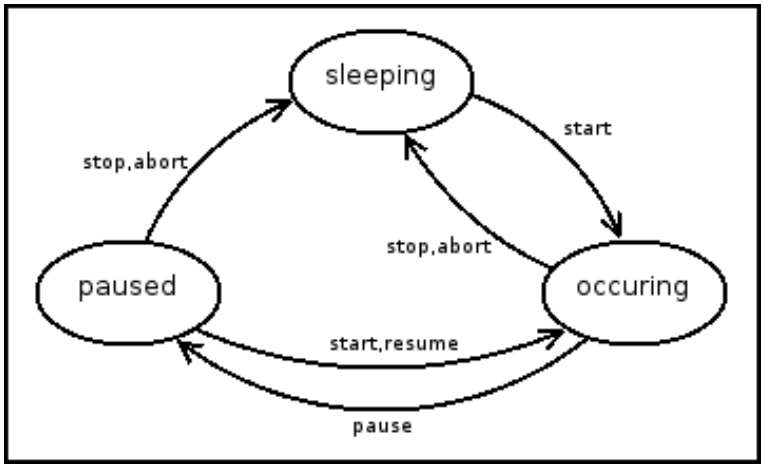

Figura 2.1: Máquina de estados - [Costa e Soares 2007]

No contexto do SBTVD[4][SBTVD 2006], onde as aplicações são especificadas declarativamente na linguagem *NCL*[ABNT 2007], a apresentação de um documento multimídia pode ser modelada conforme o modelo de *Grafos Temporais Hipermídia* .

Um documento *NCL* é composto de alguns elementos, dentre os quais destacamos:

1. *Nós de mídia* (<MEDIA>): elementos que especificam mídias (conteúdo) utilizadas na apresentação. Geralmente representam mídias com conteúdo audiovisual (vídeo, áudio, imagens) e aplicações imperativas.

[4]Sistema Brasileiro de TV Digital

2. *Âncoras* (<AREA>): definem pontos de sincronismo (temporais) associados a uma mídia. Âncoras podem ainda estar associadas a valores de atributos (variáveis) ou ainda relacionarem-se a regiões espaciais.

3. *Elos* (<LINK>): definem o relacionamento entre mídias. Através desta estrutura é possível a especificação de um relacionamento de mídias do tipo "*quando* **mídia 1** *terminar inicie* **mídia 2**". As relações são especificadas através de outra estrutura (conectores). A linguagem permite o uso de conectores com condições compostas, ou seja, é possível o uso de uma combinação de condições para que o evento seja iniciado. Além disso a linguagem permite que ações compostas sejam iniciadas a partir de um único elo.

Um documento multimídia é composto por diferentes mídias que podem assumir diferentes estados, de acordo com a interação realizada. A maneira clássica (como a adotada em [Pimentel et al. 2007]) de anotar conteúdo de vídeo consiste na inserção de âncoras (marcas temporais e/ou espaciais) associadas ao vídeo, permitindo que as anotações sejam posteriormente reproduzidas de forma sincronizada.

Quando é considerada a anotação sobre um documento multimídia, o cenário torna-se mais complexo. Ao invés da linha do tempo de uma única mídia (vídeo) deve ser considerado o estado do documento (composição de mídias) para a criação dos pontos de sincronização. As anotações devem ser contextualizadas com as diferentes mídias para permitir a correta sincronização.

No escopo deste trabalho, a máquina de estados do documento *NCL* foi considerada como a composição dos elementos da linguagem enumerados e seus atributos. A máquina de estados pode ser sumarizada em:

- *Estado da mídia*: o estado associado à mídia (ocorrendo, pausada ou em execução).
- *Linha do tempo da mídia*: no caso de mídias temporais o tempo da mídia é considerado para que as âncoras temporais sejam corretamente posicionadas.
- *Atributos da mídia*: outros atributos relacionados à mídia como tamanho, posicionamento e variáveis devem ser considerados.

A possibilidade de obter o estado da apresentação proporcionada pela *NCL* foi utilizada neste trabalho de forma a permitir que seja possível a adição de anotações mais ricas que utilizam-se da composição de diversas mídias para a definição dos pontos de sincronismo.

Suporte a edição ao vivo

O processo de autoria e apresentação de um documento multimídia ocorre, geralmente, em fases distintas. Usualmente o documento é codificado (autoria) pelo autor e posteriormente enviado para apresentação e interação pelo usuário (interagente).

Em alguns casos o processo usual não se aplica. Em cenários como o de transmissões de TV ao vivo a autoria acontece em tempo de apresentação. Opções de interação são planejadas antecipadamente, mas não é possível a predição de todos os eventos. Nesse caso, os eventos não previstos precisam ser incorporados pela apresentação multimídia em tempo de apresentação e interação.

No âmbito da TV Digital, especificamente no contexto do Sistema Brasileiro de TV Digital, a linguagem *NCL*, através do *middleware Ginga* apresenta soluções para a edição de um documento *on-the-fly*. Em [Costa et al. 2006] é proposta uma solução para a edição ao vivo de documento *NCL*. Esses mecanismos são projetados para os casos em que a emissora demanda a alteração do documento *on-the-fly*.

O processo de edição ao vivo realizado pelas emissoras requer recursos computacionais semelhantes aos demandados pela autoria realizada pelo interagente. É importante ressaltar que a edição feita pela emissora é planejada, em princípio, pelo autor do documento original. Dessa forma, considera-se que o autor (emissora) tem domínio sobre as alterações realizadas, garantindo a consistência desta edição. Quando é considerada a edição desencadeada por outros atores novos requisitos são acrescentados e precisam ser devidamente tratados.

No desenvolvimento do trabalho foi investigado o uso dos recursos de edição ao vivo proporcionados pela linguagem e *middleware* no contexto do usuário final (interagente), tratando das especificidades desse tipo edição.

Suporte a múltiplos documentos

Um documento hipermídia é responsável pela especificação dos objetos de mídia (vídeo, áudio, imagens, aplicações imperativas,etc) e dos relacionamentos entre esses objetos (temporais e atemporais) [Neto e Soares 2009]. Por se tratar de uma linguagem de cola, *NCL* não descreve conteúdo, mas cuida de orquestrar o relacionamento entre objetos de mídia (que contêm conteúdo). O reúso em *NCL* pode ocorrer de duas formas:

- *Reúso dos objetos de mídia*: o conteúdo de objetos de mídia como vídeo, áudio e imagens pode ser reusado. Isto implica que a autoria do conteúdo possa ser realizada uma

única vez e esse conteúdo reusado em diferentes contextos seja na mesma aplicação ou em aplicações distintas. Em *NCL*, especificamente, o reúso pode se dar ainda a nível de um objeto[5] em execução. No documento pode ser especificado que dois elementos representam a mesma instância de um objeto (o mesmo objeto em execução) ou uma nova instância do mesmo objeto.

- *Reúso de documentos*: um documento pode ser reusado por outros documentos. A composição de documentos pode promover o reúso e organização do código já que partes bem definidas e reusáveis (por exemplo um comercial interativo que é exibido em diversos programas) podem ser definidas de forma unitária e utilizadas na composição de apresentações mais complexas. Existe ainda a possibilidade de tratar partes do documento (como conectores *NCL*) como uma biblioteca de código declarativo. O uso desses recursos torna a aplicação mais clara (facilitando a depuração), testável e menos propensa a erros.

A possibilidade de compor uma apresentação a partir de outra apresentação (contemplada pela linguagem *NCL*) é importante quando é considerado o cenário de adição de anotações sobre o conteúdo. Quando o processo de autoria é realizado sobre conteúdo de terceiros (por exemplo um programa transmitido por uma emissora) é pertinente que as anotações sejam armazenadas separadamente por questões de respeito a direitos autorais. O usuário que realiza anotações é autor do seu conteúdo e tem liberdade para compartilhá-lo com outros.

O conteúdo produzido por um usuário ao anotar um documento multimídia pode se tratar tanto de mídias produzidas (anotações sobre *frames* de vídeos, anotações de áudio realizadas pelo usuário, etc) quanto documentos multimídia produzidos (especificando o relacionamento das anotações com outras mídias). Dessa forma, é desejável que o usuário detenha plenos direitos de distribuição do conteúdo (mídia e documentos) de sua autoria. A linguagem *NCL*, através dos seus recursos de múltiplos documentos, é adequada a atender esses requisitos.

Suporte a múltiplos dispositivos

O modelo tradicional de TV centra-se, basicamente, em um par de dispositivos para apresentação do conteúdo: o conjunto tela da TV/alto-falante. Assistir TV é uma prática que geralmente é feita de forma coletiva tanto em ambientes doméstico (sala de TV) quanto em outros ambientes (aeroportos, salas de espera, etc).

[5] Neste contexto é considerado um objeto (conceito de Orientação a Objetos) em memória.

Com o advento da interatividade, incorporada à TV Digital, houve uma mudança no comportamento do usuário perante a TV. No modelo tradicional, o indivíduo é um espectador (telespectador), tendo um comportamento basicamente passivo com relação à TV. Com a inserção das opções de interatividade, o indivíduo passa a ser um usuário (interagente) de um sistema computacional (*software*). Essa mudança de papel faz com que o usuário tenha a possibilidade de assumir um papel ativo, relacionando-se com o conteúdo apresentado na TV e promovendo alterações na forma como esse conteúdo é exibido (acessando conteúdo interativo, por exemplo). Esse novo comportamento traz alguns inconvenientes quando é considerada a utilização da TV de forma coletiva já que podem existir muitas opções de interatividade e, em um cenário coletivo, usuários distintos podem ter preferências distintas. Dessa forma, ocorre uma concorrência pelo uso do dispositivo de interação (controle remoto) e pela tela de apresentação (TV).

O uso de múltiplos dispositivos é uma solução para esse problema. Nesse cenário, vários usuários podem interagir de forma simultânea com o conteúdo sem concorrer com o dispositivo de interação e tela de apresentação. O conteúdo principal pode ser apresentado de forma coletiva (em um modelo similar ao tradicional) e o conteúdo referente à interatividade pode ser acessado de forma individualizada.

Em [Costa, Moreno e Soares 2009] é apresentada uma solução para o suporte a múltiplos dispositivos através do *middleware Ginga*. Com base na linguagem *NCL*, é apresentada uma solução em que são definidas classes de dispositivos que podem atuar de forma passiva (como telas para apresentação de conteúdo) e de forma ativa (oferecendo recursos de interação).

O suporte da linguagem *NCL* e do *middleware Ginga* a múltiplos dispositivos é oportuno quando busca-se um cenário ubíquo para a autoria e visualização de conteúdo obtido através de anotações feitas por interagentes, seja de forma individualizada ou colaborativa.

2.2 Autoria de vídeo interativo

Existem algumas ferramentas que dão suporte à autoria de vídeo interativo, tanto no contexto de linguagens imperativas (como Java) quanto no contexto de linguagens declarativas (como NCL e SMIL).

Linguagens declarativas tornam o processo de autoria mais próximo da linguagem natural. Através desse paradigma é especificado “o quê“ deve ser feito. Detalhes técnicos de “como“ a tarefa será realizada são abstraídos.

A autoria direcionada ao usuário final (interagente) foi explorada, no contexto deste tra-

balho, através de linguagens declarativas, que se mostram mais apropriadas para tal fim.

Algumas ferramentas de autoria direcionadas a linguagens interativas foram estudadas no desenvolvimento do trabalho. Uma breve descrição dessas ferramentas é apresentada adiante.

2.2.1 GRiNS

O GRiNS [Bulterman et al. 1998] é um ambiente de autoria para a linguagem *SMIL* (*Synchronized Multimedia Integration Language*). Possui as visões temporal, espacial e textual integradas para a construção de um documento. Na visão temporal o autor consegue manipular as relações temporais entre os elementos da apresentação multimídia. Na visão espacial é possível manipular como os elementos serão apresentados. Através de uma interface gráfica é possível definir em qual espaço da tela cada elemento visual será apresentado. Atributos como coordenadas das região, largura, altura, índice (no caso de regiões sobrepostas), dentre outros, podem ser manuseados de forma gráfica.

Na visão textual é possível a autoria do documento de forma textual. O autor pode declarar os elementos do documento multimídia, na linguagem *SMIL*, através dessa visão. Qualquer alteração nessa visão é refletida nas demais.

A combinação das diferentes visões torna o processo de desenvolvimento mais produtivo, já que desobriga o autor de conhecer profundamente a linguagem *SMIL*. O uso da perspectiva de linha de tempo facilita a visualização das relações lógicas entre as mídias. A ferramenta privilegia a sincronização. A especificação de pontos de interatividade no documento é mais fácil de ser implementada na visão textual.

2.2.2 Composer

Composer [Guimarães, Costa e Soares 2007] é uma ferramenta de autoria desenvolvida no laboratório *Telemidia*, na PUC-Rio[6]. A ferramenta permite a construção de aplicações interativas na linguagem NCL. Assim como a ferramenta *GRiNs*, oferece as visões espacial, temporal e textual. Além dessas, é possível ter-se uma visão estrutural. As quatro visões são sincronizadas, isto é, alterações em uma visão repercutem nas demais (Figura 2.2).

Para o contexto de TV Digital é uma ferramenta com uma peculiaridade bastante interessante: o suporte à edição ao vivo. A ferramenta permite que comandos de edição ao vivo (suportados pela linguagem *NCL*) sejam enviados ao documento multimídia em exibição. Através de

[6]Pontifícia Universidade Católica do Rio de Janeiro

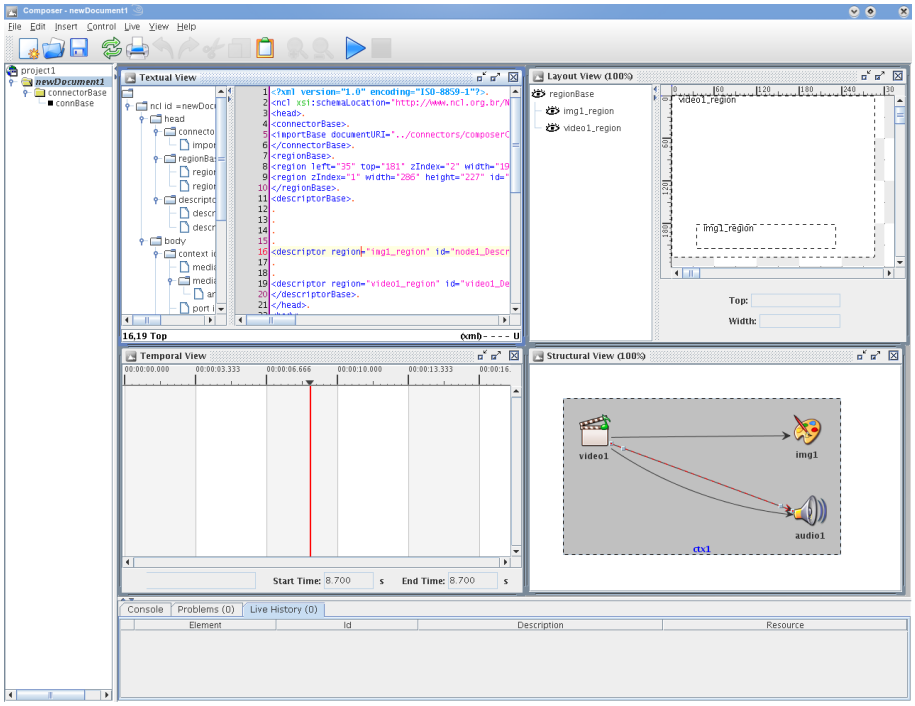

Figura 2.2: Visões da ferramenta *Composer*.

uma interface gráfica é possível que os comandos sejam enviados à máquina de apresentação. Em um cenário real de TV essa funcionalidade é primordial para os casos em que programas ao-vivo são transmitidos e que elementos não previstos precisam ser incorporados ao documento.

A ferramenta possui um formatador para a exibição de documentos NCL. Essa funcionalidade pode ser encarada como uma visão complementar às demais (estrutural, espacial, temporal e textual), permitindo a visualização da aplicação durante a autoria, dando um *feedback* mais imediato ao autor do documento. A Figura 2.3 dá um panorama da estrutura da ferramenta *Composer*.

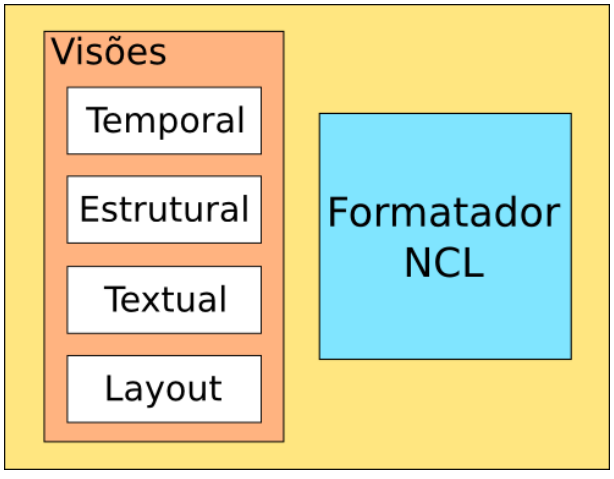

Figura 2.3: Estrutura da ferramenta *Composer*.

Partindo-se da ideia de que a apresentação do documento (através de um formatador presente na ferramenta) também é uma visão, algumas possibilidades adicionais podem ser exploradas. Tarefas menos complexas de autoria como demarcação de segmentos de mídias que devem ser eliminados ou destacados, anotações com tinta eletrônica em *frames* de vídeo, dentre outros, podem ser exploradas de forma ubíqua. As anotações feitas pelo usuário podem ser incorporadas ao documento e, através das demais visões, sofrerem os refinamentos adequados.

Na produção de conteúdo para TV é usual a realização de uma pré-edição no conteúdo. O diretor ou editor-chefe do programa faz uma análise do material capturado, dando as sugestões de como o trabalho de edição deve ser realizado. Essas sugestões, se capturadas de forma adequada, podem facilitar o trabalho dos encarregados pela edição, que contarão com um documento pré-produzido. O processo de comunicação da equipe tende a ser otimizado já que as anotações estarão sincronizadas com o conteúdo.

A adição de funcionalidades que permitam a autoria ubíqua de documentos em fase de apresentação e interação com o usuário parece oportuna na ferramenta *Composer*. A apresentação/execução do documento, através do formatador (máquina de apresentação), poderia nesse caso, ser considerada mais uma visão da ferramenta, com um comportamento ativo.

2.3 Autoria com foco no usuário final

Existem algumas ferramentas de autoria projetadas para utilização por ususários finais, isto é, usuários não especialistas em autoria de vídeo. Ferramentas com esse foco atendem a requisitos especiais, como interfaces gráficas bastante naturais, que tornam o processo de autoria mais ubíquo.

2.3.1 Ambulant Annotator

O Ambulant Annotator [Cesar, Bulterman e Jansen 2006] é uma ferramenta que permite a criação e visualização de anotações multimídia. A nova geração de computadores com telas sensíveis a toque (*tablets*,*all-in-one*) é utilizada como plataforma para que usuários realizem anotações através de comentários baseados em tinta (caneta eletrônica).

A ferramenta [Cesar, Bulterman e Jansen 2006] relaciona-se com a proposta apresentada nesta dissertação principalmente com relação ao uso de uma linguagem estruturada para a codificação das anotações (no caso *SMIL*). Entretanto, os trabalhos divergem na forma como este conteúdo é incorporado à apresentação multimídia. O trabalho ora proposto considera o cenário

de edição ao vivo do documento. Diferencia-se do *Ambulant Annotator*, pois além explorar os conceitos de anotação multimídia e permitir a geração automática de documentos multimídia, busca prover uma infraestrutura para a construção de aplicações que exploram a anotação sobre documentos multimídia. Não é objeto do trabalho descrito nesta dissertação a criação de uma aplicação específica, mas fornecer recursos para a construção de aplicações do domínio.

2.3.2 Youtube

Uma das definições clássicas, conceituada por Tim O'Reilly [OReilly] define o termo Web 2.0 como:

> "Web 2.0 é a mudança para uma internet como plataforma, e um entendimento das regras para obter sucesso nessa nova plataforma. Entre outras, a regra mais importante é desenvolver aplicativos que aproveitem os efeitos de rede para se tornarem melhores quanto mais são usados pelas pessoas, aproveitando a inteligência coletiva."

O *Youtube*, plataforma do *Google Inc.*, pode ser considerado um dos casos mais concretos do uso da *inteligência coletiva*. Trata-se de uma plataforma, utilizada por milhões de pessoas, para a divulgação de conhecimento (desde conteúdo de entretenimento a conteúdo cientifico).

Um fenômeno comum nesta plataforma é encontrar o mesmo vídeo publicado por diversas pessoas. O conteúdo principal (vídeo) adquire uma semântica distinta em cada uma das instâncias, de acordo com a percepção que os usuários têm do conteúdo de forma individual. Essa percepção, tradicionalmente, é transmitida através da descrição do conteúdo, *tags*, votação e de comentários textuais enviados pelos usuários.

No ano de 2008, a ferramenta foi incrementada e passou a permitir novos tipos de comentários no conteúdo. O usuário pode adicionar anotações ao conteúdo. Essas anotações podem ter uma sincronização temporal e espacial com o vídeo. O conteúdo original (vídeo) é preservado, mas o usuário passa a ter a opção de visualizar as anotações realizadas por outros. Dessa forma, um mesmo vídeo pode ter uma leitura completamente diferente, de acordo com as anotações associadas a ele.

As anotações realizadas pelo usuário pode adicionar conteúdo extra (informações textuais - Figura 2.4) como adicionar opções de interatividade (Figuras 2.5,2.6).

Do ponto de vista do usuário final a plataforma oferece alguns serviços interessantes:

- Possibilidade de disponibilizar conteúdo para outros. Com uma simples câmera de vídeo o usuário pode divulgar um conteúdo para milhões de pessoas.
- Possibilidade de transmitir a sua percepção sobre um mesmo conteúdo a outros, utilizando-se das anotações.
- Possibilidade de adicionar anotações a um vídeo com uma interface que não requer conhecimentos específicos de autoria de vídeos. A autoria acontece de forma ubíqua.
- Possibilidade de exercer a capacidade criativa através da produção de vídeos interativos. Esses vídeos podem ser enriquecidos com conhecimento construído por terceiros (através de *links*).

A solução se mostra interessante, mas poderia ter recursos mais poderosos como:

- Recursos para a anotação colaborativa do vídeo: vários usuários anotando um único vídeo. Um cenário em que seja possível a realização de anotações sobre vídeos já anotados parece relevante. A colaboração entre diversos usuários, exercendo a inteligência coletiva poderia produzir resultados interessantes.
- Disponibilização de recursos interativos mais complexos para usuários avançados.
- Recursos para a alteração do fluxo do vídeo (alteração da linha do tempo).
- Utilização de linguagens abertas para a sincronização de mídias, como *NCL* e *SMIL*.

Figura 2.4: Anotações realizadas pelo usuário sobre um vídeo através do *Youtube*.

Figura 2.5: Adição de opções de interação através de anotações realizadas pelo usuário sobre um vídeo através do *Youtube*.

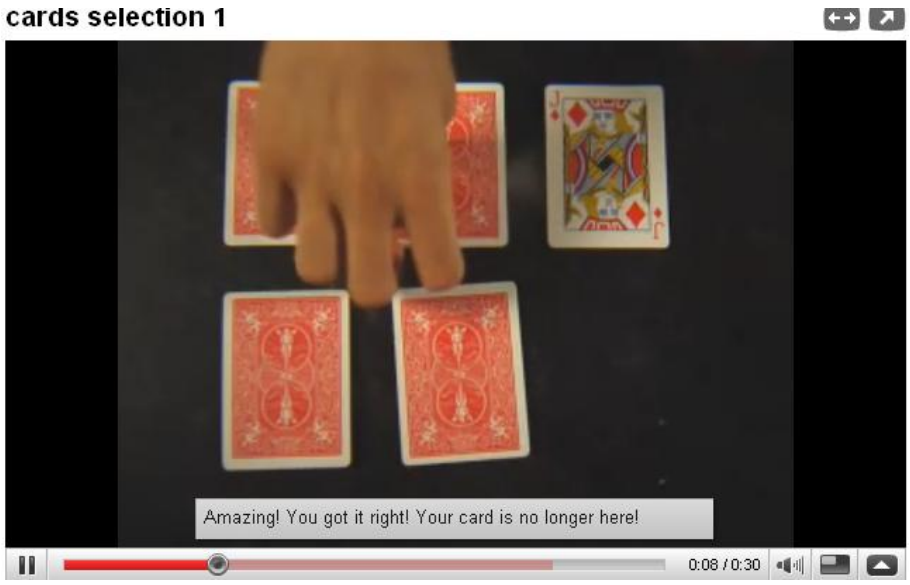

Figura 2.6: Acesso a ponto de interação de um vídeo do *Youtube*.

2.3.3 Watch and Comment Paradigm

É um hábito corriqueiro assistir e comentar vídeos. Em [Cattelan et al. 2008] foi proposto o paradigma *watch-and-comment* para a produção e edição transparente de vídeo digital interativo. O paradigma parte da premissa de que qualquer interação entre o usuário e a mídia (um comentário de voz, por exemplo) pode ser capturada e registrada em um vídeo interativo especificado por meio de uma linguagem declarativa, como *SMIL* ou *NCL*.

A captura ubíqua da interação multimodal do usuário com o vídeo é a base do paradigma, de acordo com os seguintes princípios:

1. Não há restrição com respeito à fonte de vídeo: o fluxo de mídia pode ser obtido ao vivo a partir de uma câmera, por difusão do sinal de TV, ou reproduzido de um arquivo armazenado localmente no computador, conversor digital (*set-top box*) ou reprodutor de mídia (*media player*).

2. Não há restrição com respeito ao tipo do vídeo. Podem ser considerados vídeos lineares (em quaisquer formatos) e vídeos interativos (aqueles que possuem uma aplicação associada).

3. Não há restrição com respeito à linguagem do documento resultante: por exemplo, *NCL*, *SMIL* ou qualquer outra linguagem declarativa para autoria hipermídia pode ser utilizada.

4. O documento declarativo gerado mantém as anotações em separado da mídia original: isto significa que as anotações e edições podem ser distribuídas independentemente da mídia original — o que é uma característica importante quando se faz alusão à questão de direitos autorais.

5. Não há restrição com respeito à mídia utilizada para os comentários. A captura pode ser transparente sob a perspectiva do usuário (por exemplo, voz capturada com um microfone, tinta digital de dispositivos com interface com caneta eletrônica, gestos capturados com sensores como os acelerômetros presentes no *Wii Remote* ou no *iPhone*). A captura pode ser explícita, por exemplo palavras digitadas em um teclado e ações realizadas para produzir algum resultado (como é o caso de um clique na janela de vídeo para indicar o início ou o fim de um segmento de vídeo selecionado), ou implícita, por exemplo se comentários de voz forem ativados automaticamente quando o microfone detecta que o usuário está falando.

6. Não há restrição com respeito em como a captura da interação deva ser utilizada: o que significa que as aplicações podem inovar em termos do que fazer com a interação cap-

turada. Por um lado, uma interação de um usuário em particular pode ser associada a qualquer comando no vídeo interativo resultante (por exemplo, em um comando de edição *skip*). Nesse caso, quando o vídeo interativo resultante é posteriormente revisto, o comando correspondente é oferecido ao usuário como uma opção (isto é, o usuário decide se quer ou não saltar o trecho de vídeo indicado).

7. Não há restrição com respeito a como o documento declarativo resultante é distribuído: significando que o vídeo interativo pode ser armazenado e reproduzido apenas no dispositivo em que foi capturado (por exemplo, no caso do usuário possuir um controle remoto com facilidades de armazenamento), ou que exista um integração do ambiente do usuário com um repositório *Web*.

Como prova de conceito do potencial do paradigma WaC foi desenvolvida a ferramenta *WaCTool* [Pimentel et al. 2007]. A ferramenta explora alguns conceitos do paradigma, permitindo anotações multimodais sobre um vídeo linear. As anotações multimodais possibilitam ao usuário o acesso a comandos de edição, produzindo ao fim do processo de autoria um vídeo interativo (um documento em linguagem declarativa).

A ferramenta foi implementada na linguagem *Java* e projetada para uso em computadores com dispositivos de interação sofisticados. O protótipo foi experimentado em um *tablet PC*, com interação através de caneta. O dispositivo utilizado também dispunha de microfone para a captura de voz.

Embora o paradigma *watch-and-comment* (item 2) apresente a possibilidade de realizar anotações sobre os mais diversos vídeos não houve uma experimentação do processo de anotação sobre vídeos interativos. Este trabalho explora o paradigma nesse sentido: fornecer uma infraestrutura adequada para que anotações sejam feitas sobre vídeos interativos. Trabalhamos no uso do paradigma em um cenário de TV, em que os comentários feitos por usuários ao assistirem uma apresentação multimídia (interativa ou não) podem ser registrados e alterar a experiência do usuário *on-the-fly*. Diferentemente do que ocorre na ferramenta *WaCTool*, em que o conteúdo produzido é visualizado posteriormente propomos uma solução que permite a visualização do conteúdo em tempo de apresentação, isto é, a estrutura do documento multimídia é alterada em tempo de apresentação, dando um *feedback* imediato ao usuário.

O foco do trabalho não é a construção de uma ferramenta (como a *WaCTool*). Propomos uma infraestrutura que permite a construção de aplicações que explorem o paradigma *watch-and-comment* de forma mais sofisticada.

2.4 Linhas de Produto de Software

A construção de um software é um processo dispendioso para qualquer organização. Existe uma série de etapas que precisam ser realizadas desde o levantamento dos requisitos do software, passando pelas fases de modelagem, codificação e testes até a implantação do sistema. O processo de desenvolvimento de software pode ser comparado a uma produção industrial em alguns aspectos: diversos produtos podem ser derivados de uma linha de produtos que compartilham uma infraestrutura comum, um processo de produção ou componentes (por exemplo a indústria automobilística - vários modelos podem ser produzidos utilizando a mesma linha). Essa abordagem tem sido adotada também na área de desenvolvimento de software por diversas empresas [Northrop 2002].

Pelo levantamento realizado por [Cohen 2002], uma definição de linha de produtos de software bem aceita é dada por [Clements e Northrop 2001]:

> Uma linha de produtos de software é um conjunto de sistemas de software compartilhando um conjunto gerenciável de características que satisfazem necessidades de uma missão ou segmento de mercado específico e que são desenvolvidos a partir de um núcleo comum de artefatos (*core assets*) de uma maneira sistemática.

Esta definição introduz alguns conceitos chaves para o entendimento de linhas de produto:

- *core assets*[7]: é um conjunto de *assets* prontos para serem reusados no desenvolvimento de novos produtos. Os *core assets* podem ser componentes de software, padrões de projeto, documentos utilizados no desenvolvimento, arquitetura, cronogramas e outros artefatos. A arquitetura é compartilhada por todos os produtos, e carrega consigo as possibilidades de variabilidade da linha. O *core asset* arquitetura, em particular, é visto como ponto chave de uma linha de produtos, caso seja mal projetado pode inviabilizar todo o projeto.

- Maneira Sistemática: cada *core asset* tem um processo associado que define como o mesmo poderá ser usado no desenvolvimento de novos produtos. O conjunto desses processos combinados com processos que descrevem o acoplamento dos *core assets* definem um plano de produção para cada produto. O plano de produção define um procedimento de derivação para um novo produto [Clements e Northrop 2001].

- Missão ou segmento particular: O segmento de atuação de uma linha de produtos deve ser delimitado e bem compreendido. As estratégias de negócio são traçadas dentro desse segmento de mercado. Se o escopo do segmento for muito grande também será o seu

custo. É complexo gerenciar uma linha de produtos muito genérica, pois além da base de *core assets* ser grande, a complexidade de cada *core asset* também será, pois deverão suportar muitas variabilidades.

A principal diferença entre o desenvolvimento de sistemas convencionais individuais e a abordagem de linhas de produto é o enfoque. Nos sistemas convencionais enfoca-se um único sistema por vez, ao passo que o desenvolvimento de linhas de produto visa um conjunto de sistemas.

A aplicação dos conceitos de linha de produto mostram-se oportunas para o projeto de ferramentas de autoria. O domínio dessas ferramentas é bem conhecido e seus requisitos não possuem grandes variabilidades. Foi utilizada a abordagem de linhas de produto para a concepção de uma arquitetura que suporte a construção de aplicações de autoria de documentos multimídia, objeto deste trabalho.

3 *Engenharia de Domínio*

A Engenharia de Domínio identifica aspectos comuns de um domínio para que sejam reutilizados por várias aplicações deste domínio. O enfoque é mais abrangente, centrado num conjunto e não em uma única aplicação, atendendo necessidades e prevendo futuras expansões. Esses aspectos comuns são *core assets* tangíveis como componentes, *frameworks*, documentação, dentre outros. Trata-se de uma tarefa complexa que requer a figura de um especialista no domínio. O autor deste trabalho tem vasta experiência na área de TV Digital e esteve envolvido em diversos trabalhos (acadêmicos e técnicos) relacionados ao domínio. Esta experiência permitiu que o autor atuasse como especialista no domínio.

3.1 Descrição do domínio

O cenário apresentado nesta seção ilustra o conceito do usuário final (interagente) promover mudanças no documento em apresentação que alterem a sua experiência ao assistir TV, propostas pelo paradigma *watch-and-comment*. Nesse cenário, assumimos uma situação hipotética em que o usuário assiste um filme e pode:

- *Marcar uma cena como favorita*: O usuário pode selecionar um instante específico do vídeo para visualização posterior. Uma cópia do *frame* (*screenshot*) e o momento do documento correspondente são armazenados. A qualquer momento o usuário pode acessar a lista de favoritos e optar pela visualização do vídeo a partir de um momento;

- *Saltar um trecho do vídeo*: O usuário pode selecionar dois instantes do vídeo que delimitam um intervalo de tempo. Esse intervalo pode ser marcado como um trecho que deve ser ignorado quando o vídeo for exibido novamente. Esse salto do trecho do vídeo pode acontecer de forma automática (quando uma âncora do vídeo for atingida haverá um salto para outra âncora) ou manual, através de interação do usuário. Os *frames* correspondentes podem ser extraídos para a composição de um menu que permita a visualização apenas das cenas marcadas para “salto“;

- *Repetir um trecho do vídeo*: O usuário pode selecionar, novamente, dois instantes do vídeo que delimitam um intervalo. Esse segmento de vídeo pode ser demarcado como um segmento que deve ser repetido em *loop*. Esse tipo de opção pode ser usado para os casos em que o usuário julgue aquela cena como importante e quer a opção de repetição do segmento para uma análise mais aprofundada do conteúdo. Essa estratégia é empregada na TV comercial em diversos cenários: desde situações de comédia até programas científicos. Os *frames* correspondentes ao intervalo demarcado podem ser utilizados para a composição de um menu que facilite a navegação do usuário.

As situações apresentadas são exemplos de interações consideradas. O cenário descrito pode ser representado através de um diagrama de casos de uso, ilustrado na Figura 3.1. Os casos de uso levantados dão suporte à edição de um documento por um usuário leigo de forma transparente.

O diagrama de caos de uso apresentado na Figura 3.1 representa as interações do ator (usuário telespectador) que assiste TV. No cenário clássico, o ator dispõe de um dispositivo típico de entrada associado à TV, o controle remoto, e através dele tem acesso a todas as ações representadas no diagrama.

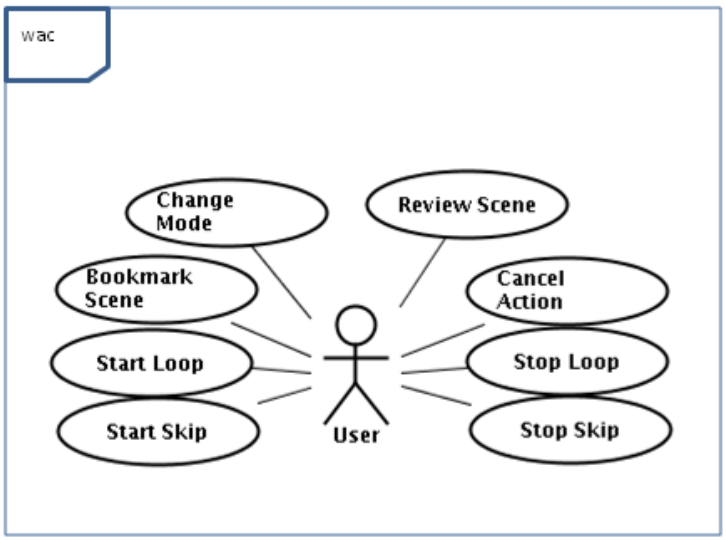

Figura 3.1: Diagrama de Casos de Uso: Através de um controle remoto o usuário pode criar e visualizar anotações

.

3.1.1 Dispositivo de Interação

Da forma como a TV Digital foi concebida, o principal dispositivo de interação entre o usuário e a TV continua sendo o controle remoto. No cenário do Sistema Brasileiro de TV Digital, um conjunto de botões é reservado para interatividade. Pelo menos quatro botões para a interação do usuário com aplicações estão disponíveis em todos os controles remotos - eles possuem cores e formas diferenciadas (círculo vermelho, triângulo amarelo, quadrado azul e losango verde) [SBTVD 2006]. A Figura 3.2 mostra um controle remoto típico para TV Digital Interativa.

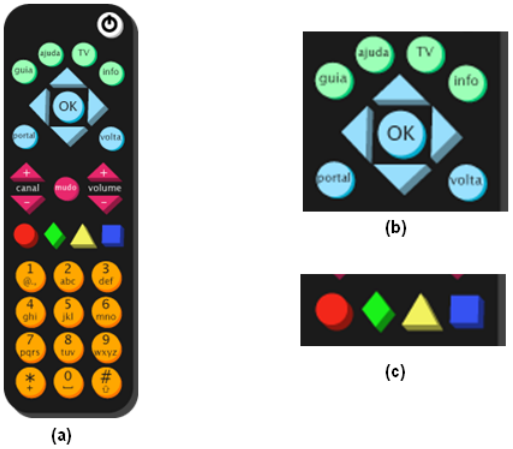

Figura 3.2: Controle Remoto para o Sistema Brasileiro de TV Digital. (a) controle remoto de TV interativa mínimo. (b) botões de navegação e botões com funções especiais. (c) botões coloridos especiais, utilizados em aplicações interativas..

Foi considerada na arquitetura proposta permitir que usuários assistam e editem um programa de TV através de um controle remoto mínimo, aproximando os conceitos de *watch-and-comment* dos usuários típicos de TV, que não dispõem de dispositivos sofisticados de interação.

Entretanto, é importante salientar que novos dispositivos de interação multimodal, suportando gestos, toque e voz, por exemplo, estão sob investigação (por exemplo [Cattelan et al. 2008], [César et al. 2007]). Uma arquitetura robusta para o suporte à autoria ubíqua de documentos deve garantir um conjunto mínimo de opções de interação, e também deve permitir a adição de novos dispositivos mais sofisticados.

3.2 Requisitos do domínio

Através de uma análise do proposto pelo paradigma *watch-and-comment* e dos experimentos realizados com usuários [Motti et al. 2009, Cattelan et al. 2008] foi identificado um conjunto de requisitos que parecem comuns a aplicações que exploram esses conceitos. O domínio foi delimitado para o ambiente de TV Digital, considerando os recursos disponíveis na plataforma.

Essa análise resultou em um conjunto de *core assets* que deram subsídio para a proposta de uma arquitetura que contemple os seguintes requisitos:

1. Aplicações que permitem a realização de anotações em um conteúdo multimídia requerem sincronização temporal. A sincronização temporal pode estar atrelada tanto a uma mídia específica como a um conjunto de mídias. Mecanismos para a recuperação do estado de uma mídia, quanto de um conjunto de mídias são necessários para a maioria das aplicações de autoria de documentos em fase de apresentação.

2. Considerando que o usuário realiza as anotações no momento em que assiste um vídeo é importante a existência de mecanismos para a captura do *frame* de vídeo correspondente para que ele possa ser apresentado posteriormente, quando o usuário decidir acessar suas marcas. Nos casos de interações mais sofisticadas, com o uso de tinta eletrônica, a possibilidade de captura do frame pode ser útil para que o usuário possa anotar sobre o *frame* (como na Figura 3.3)

3. O processo de autoria muitas vezes requer que o usuário tenha um *feedback* instantâneo da autoria realizada. Em outras palavras, que o usuário possa visualizar as alterações promovidas no documento *on-the-fly*, em tempo de interação. Mecanismos que permitam a edição do documento em fase de exibição e interação são importantes. Essa abordagem é análoga ao processo de edição ao vivo de documentos feitos por emissoras de TV: alterações no documento em apresentação são realizadas sem a necessidade de recarregá-lo.

4. Pode ser elemento do processo de autoria um documento multimídia pré-existente, que esteja sendo enviado por uma emissora de TV (*broadcaster*). Esse documento pode estar associado a um programa de TV ao vivo, isto é, que esteja sofrendo edições por parte da emissora em tempo de transmissão (eventos esportivos, por exemplo). A autoria sobre esse tipo de documento pode desencadear uma situação peculiar: o documento é editado por dois atores diferentes, simultaneamente. Isso pode gerar algumas inconsistências

e mecanismos para anular a edição do usuário, isto é, mecanismos de *rollback* para o conteúdo original veiculado pela emissora, sem edições adicionais (exceto as edições da própria emissora - controladas) são desejáveis.

5. A gravação dos fluxos elementares relacionados a um documento multimídia é um requisito transversal à maioria das aplicações de autoria, quando utilizadas em um ambiente de TV. O usuário pode realizar a anotação de um documento multimídia e a gravação dos fluxos de vídeo e áudio (fluxos elementares - *streams* associados) deve ser realizada paralelamente para permitir a visualização do conteúdo, quer seja durante a própria exibição do documento (visualização de uma cena anterior marcada) ou para visualização *a-posteriori*. Esse processo de gravação é um processo complexo, pois demanda a gravação de um conjunto de tabelas associadas que permitem a sincronização dos fluxos posteriormente.

6. Questões relativas a direitos autorais precisam ser consideradas pelas aplicações. A gravação não autorizada de fluxos pode trazer transtornos (como processos judiciais) ao usuário. O processo de gravação, portanto, deve ser controlado de modo a só permitir a gravação de conteúdos autorizados. Os direitos autoriais relacionados ao documento multimídia também precisam ser considerados. É desejável que as anotações do usuário sejam armazenadas em outro documento, de sua autoria e que possa ser livremente distribuído. O documento do usuário pode ser combinado com o documento original, atuando como uma máscara, produzindo os efeitos desejados.

7. Em um cenário típico onde o usuário interage com um controle remoto convencional ele possui um conjunto de botões disponíveis para interação limitado. Além das teclas numéricas (0 a 9) estão disponíveis alguns botões coloridos e mais algumas teclas de navegação. Essas teclas do controle remoto podem estar associadas a pontos de interatividade no documento multimídia (ex: quando o usuário pressionar o botão vermelho do controle remoto é exibida uma imagem na tela). A indisponibilidade de teclas exclusivas para a autoria realizada por cada ferramenta requer a existência de mecanismos que permitam que cada tecla possua mais de uma ação associada (o botão vermelho pode, ao mesmo tempo, ser um ponto de interação no documento multimídia e representar uma ação em uma ferramenta de autoria). Entretanto, se as ações forem desencadeadas simultaneamente resultados desastrosos podem ocorrer (opções de interatividade serem acionadas sem que este tenha sido o desejo do usuário ou ações de ferramentas de autoria ativadas involuntariamente). Com a combinação de teclas pode-se obter uma quantidade ilimitada de opções de interação. A sequência de teclas *1234Vermelho*, por exemplo,

pode ter um significado enquanto a sequência *1234Verde* pode ter outro significado. Além disso o tempo que uma tecla fica pressionada ou o intervalo de tempo entre a seleção de duas teclas podem ser relevantes na definição de uma opção de interação. Entretanto, o uso da combinação de teclas (que devem ser pressionadas em uma ordem correta ou com determinado intervalo de tempo) pode tornar o processo de autoria menos natural. A usabilidade poderia ser prejudicada nesse sentido. Isso aponta para a necessidade de mecanismos de gerenciamento dos dispositivos de entrada que permitam a diferenciação dos contextos de interação. Isso deve permitir que o usuário entre no "modo de edição" quando quiser realizar anotações.

8. É desejável que o documento multimídia resultante da autoria do usuário possa ser persistido.

9. A autoria considerada neste trabalho baseia-se na apresentação de documentos multimídia. A apresentação desses requer a existência de uma máquina de apresentação.

10. Mecanismos para a decodificação e apresentação das mídias (vídeo, áudio, imagens, etc) precisam ser providos.

11. No ambiente de TV Digital, principalmente em sistemas terrestres e por satétile, os fluxos de dados referentes a cada canal são enviados de forma multiplexada. Recursos para a demultiplexação dos fluxos são necessários para que o conteúdo possa ser apresentado ao usuário.

Figura 3.3: Anotação sobre um *frame* de vídeo

4 *Arquitetura para autoria ubíqua de documentos a partir de apresentações de TV Digital*

Levando em consideração os requisitos levantados, propomos uma arquitetura de *software* que provê uma infraestrutura para que aplicações de autoria ubíqua possam ser construídas. O foco da arquitetura proposta é fornecer um arcabouço para que desenvolvedores de *software* possam construir aplicações desse domínio com pouco esforço. As tarefas complexas, transversais a aplicações desse domínio, foram encapsuladas na forma de componentes reusáveis. Dessa forma, o desenvolvedor pode trabalhar em camadas de mais alto nível, não tendo que se preocupar com detalhes de implementação de baixo nível, dependentes de plataforma. Como consequência de se prover um nível de abstração elevado, o código construído pelo desenvolvedor tende a ser mais portável.

A Figura 4.1 fornece uma visão geral da arquitetura proposta que atende aos requisitos listados na seção 3.2, representada através de um diagrama de blocos.

A arquitetura projetada foi baseada em *middlewares* de TV Digital, aproveitando-se de componentes disponíveis nos subsistemas declarativos e propondo a adição de novos componentes com fim específico.

4.1 Componentes

A arquitetura proposta (ilustrada na Figura 4.1) tem seus componentes listados a seguir. Este capítulo traz uma descrição de alto nível dos componentes. Questões técnicas mais aprofundadas de cada componente são descritas no Capítulo 5 em que a arquitetura é instanciada em um *middleware* de TV Digital. Os componentes estão listados na ordem em que aparecem na Figura 4.1.

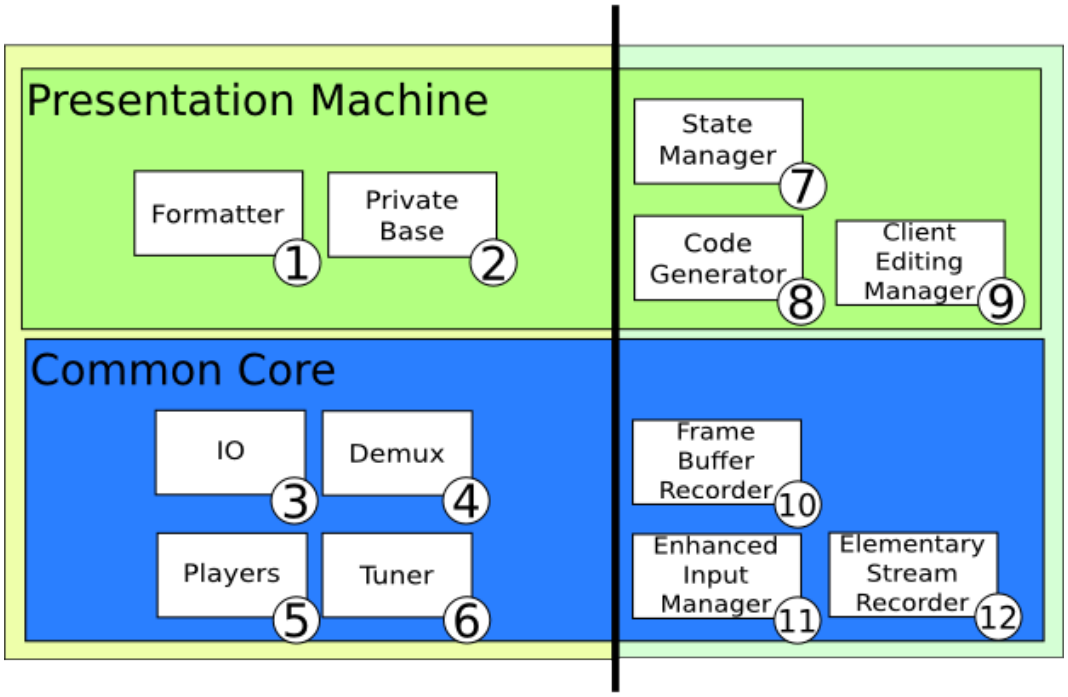

Figura 4.1: Arquitetura para a autoria ubíqua de documentos baseada em apresentações multimídia no ambiente de TV Digital

4.1.1 Formatador (*Formatter*)

O componente *formatter* implementa uma máquina de apresentação de documentos multimídia. A especificação de uma aplicação multimídia (em linguagem declarativa) é enviada ao componente que orquestra a apresentação do documento. Informações como o estado do documento devem ser providas por este componente para que a anotação possa ser sincronizada apropriadamente com o documento multimídia.

4.1.2 Gerenciador de bases privadas (*Private Base Manager*)

O conceito de bases privadas de documentos foi inspirado no modelo adotado no *middleware* Ginga [Costa et al. 2006]. Este componente provê dispositivos de encapsulamento e organização de documentos multimídia. Através dele é possível a coexistência de diversas bases privadas (que encapsulam documentos multimídia). No contexto de edição de documentos pelo usuário final (interagente) esse componente permite a separação dos documentos enviados pela emissora e de autoria do cliente, mesmo que esses possuam o mesmo identificador.

4.1.3 Gerenciador de dispositivos de interação (*Input/Output*)

Um terminal básico de TV Digital dispoõe de três principais elementos para interação: tela e alto-falantes como dispositivos de saída e controle remoto como dispositivo de entrada. Este componente trata da gerência desses recursos, permitindo que a interação possa acontecer apro-

priadamente com o controle remoto e que conteúdo possa ser apresentado ao usuário através da tela e alto-falantes.

4.1.4 Demultiplexador (*Demux*)

No ambiente de TV Digital, principalmente em sistemas terrestres e por satétile, os fluxos de dados referentes a cada canal são enviados de forma multiplexada. Este componente provê recursos para a demultiplexação dos fluxos são necessários para que o conteúdo possa ser apresentado ao usuário. Plataformas de TV Digital que encapsulam as mídias em um único fluxo demandam esse componente. Em plataformas onde os fluxos são obtidos de forma direta (por exemplo através do protocolo RTP - *Real Time Transport Protocol*), este componente pode não ser requerido, mas é necessário que existam mecanismos para a obtenção do relógio de referência para a sincronização do conteúdo.

4.1.5 Exibidores de mídia (*Players*)

Uma apresentação multimídia referencia um conjunto de mídias (vídeo, áudio, etc). Exibidores (tocadores) apropriados para cada tipo de mídia são requeridos para a decodificação do conteúdo a ser apresentado. Qualquer plataforma para apresentação de conteúdo multimídia demanda a existência desse tipo de dispositivo.

4.1.6 Sintonizador (*Tuner*)

A sintonização do canal para o recebimento da programação é necessária para que o usuário possa acessar conteúdo veiculado por terceiros. Este componente é um elemento genérico em sistemas de TV Digital provendo os recursos necessários para que a conexão com o provedor de serviço seja estabelecida e, dessa forma, o conteúdo seja recebido pelo terminal de acesso do usuário. O processo de sintonização depende da camada física de rede, podendo estar relacionado a uma faixa de frequência do espectro (sistemas terrestres e por satélite) ou a endereços lógicos na rede (IPTV), por exemplo.

4.1.7 Gerenciador da máquina de estados (*State Machine Manager*)

É desejável que as anotações realizadas pelo usuário possam ser vistas no contexto original em que foram feitas.

Este componente permite a obtenção do estado da apresentação de um documento. O estado de uma apresentação pode ser descrito como uma máquina de estados. A possibilidade de obtenção dessa máquina de estados possibilita a adição de anotações sincronizadas com o documento como um todo. É importante ressaltar que é possível a existência de anotações sobre um conteúdo composto, isto é, uma composição de diferentes mídias, que podem estar em

diferentes estados (ocorrendo, pausada, parada) e em diferentes momentos (tempo). As âncoras de sincronização devem retratar este cenário (condições compostas).

4.1.8 Gerador de código (*Code Generator*)

A edição ao vivo de um documento multimídia ocorre, geralmente, em memória através de comandos enviados ao formatador do documento. Quando a edição ao vivo é considerada um mecanismo para autoria, é desejável que o resultado da edição possa ser persistido ao fim da interação, para que o usuário possa visualizar o conteúdo produzido em outros momentos ou que possa compartilhar esse documento (que é de sua autoria) com terceiros. Este componente permite a representação textual de um modelo abstrato de um documento instanciado em memória possibilitando, deste modo, a persistência das edições realizadas pelo interagente.

4.1.9 Gerenciador de Edições do Cliente (*Cliente Editing Manager*)

A gerência das edições realizadas pelo interagente no documento multimídia é essencial. É importante diferenciar as edições feitas pelo usuário das realizadas pela emissora. Em um cenário real é possível que um documento seja editado, simultaneamente, por atores distintos: usuário final (interagente) e emissora(*broadcaster*).

A diferenciação das edições realizadas por cada um desses atores permite, por exemplo, fazer um *rollback* das alterações realizadas pelo usuário, exibindo apenas o conteúdo enviado pela emissora. Essa gerência introduz mecanismos que garantem a recuperação da apresentação em caso de falhas. Aplicações que utilizem-se dessa infraestrutura têm, assim, o seu comportamento controlado. Esse controle pode permitir que determinados elos de sincronismo possam ser ignorados. A existência desse componente de controle previne que resultados catastróficos sejam produzidos de forma irreversível. O componente não evita que ações desastradas sejam realizadas, mas fornece mecanismos de recuperação.

4.1.10 Gravador de *frames* (*Framebuffer Recorder*)

A abordagem ubíqua requer a captura de forma transparente da interação do usuário com o vídeo interativo. Considerando que o usuário, ao assistir um conteúdo, realiza uma anotação, é desejável que o *frame* do vídeo em exibição no instante da interação seja registrado. Esse *frame* é armazenado e pode ser apresentado novamente quando o usuário desejar rever as marcas. Esses *frames* de vídeo podem ser usados de formas diferentes, de acordo com a semântica dada pelas aplicações - o importante é que o *frame* representa a informação visualizada pelo usuário no momento em que a anotação foi feita, auxiliando, portanto, o usuário na identificação, de forma visual, das cenas com anotações dentre diversas opções (por exemplo escolher uma cena dentre um conjunto de cenas favoritas selecionadas).

Quando dispositivos de interação mais sofisticados (como telas sensíveis a toque) estão disponíveis aplicações podem se aproveitar dos *frames* capturados para a realização de anotações mais complexas como anotações em tinta eletrônica.

É importante ressaltar que o contexto do documento no momento em que o *frame* de vídeo é capturado também precisa ser considerado já que a visualização do conteúdo demanda não somente o *frame* de vídeo a ser apresentado. O *frame* precisa estar associado ao contexto original (que também é obtido, através do componente *Gerenciador da máquina de estados*) de quando foi capturado.

O componente ainda traz recursos para a transmissão de um *frame* ou de um conjunto de *frames* (*streaming*) através de uma rede doméstica. Essa funcionalidade pode permitir que o conteúdo da TV seja transmitido a outros dispositivos da rede que não possuem um sintonizador de TV. Uma conexão de rede é suficiente para que o conteúdo da TV seja replicado para outras telas de dispositivos da rede doméstica do usuário. Um cenário possível de uso desta funcionalidade é a projeção do conteúdo da TV na tela da geladeira quando o usuário se desloca da sala para a cozinha. Em um ambiente ubíquo é possível que sensores reconheçam a movimentação do usuário e projetem a imagem da TV em diversas telas, sem que seja necessário a existência de um sintonizador de TV acoplado a cada tela.

4.1.11 Gerenciador de entradas (*Enhanced Input Manager*)

Em uma plataforma de apresentação de documentos de hipermídia é necessário que existam mecanismos para gerenciar as diversas interações do usuário. As interações podem acontecer através de dispositivos tradicionais, como um controle remoto ou através de dispositivos não convencionais. Em um outro cenário onde está disponível apenas um controle remoto tradicional é desejável que o Gerenciador de entradas seja capaz de multiplexar funções diferentes para uma mesma tecla, dependendo do contexto de utilização que o usuário deseja no momento.

4.1.12 Gravador de fluxos elementares (*Elementary Stream Recorder*)

A abordagem do paradigma *watch-and-comment* requer que o usuário tenha uma infraestrutura que permita a gravação de documentos interativos. Tradicionalmente plataformas de TV Digital possuem mecanismos para a reprodução dos fluxos elementares (vídeo, áudio e dados) associados a um programa. Plataformas de TV mais sofisticadas possuem recursos de gravação (PVR - *Personal Video Recorder*) , como o TiVO[1], em que é possível o agendamento de gravações e pequenas edições no conteúdo (geralmente cortes). A gravação, em geral, consiste em um único fluxo de mídia.

[1] http://www.tivo.com

Este componente fornece uma estrutura de *PVR*, permitindo que todo o conteúdo seja gravado e visualizado posteriormente, oferecendo as opções de interatividade originais. O componente oferece serviços mais robustos que os gravadores de vídeo digital tradicionais. Nesses gravadores o usuário tem a opção de armazenar fluxos de vídeo. Este componente expande esta funcionalidade, permitindo a gravação de aplicações e marcas de sincronismo.

5 *GingaWaC*

A arquitetura proposta neste trabalho foi avaliada no contexto do *middleware* Ginga. Foi implementada na linguagem C++ e integrada à implementação de referência do *middleware* Ginga. Essa instanciação da arquitetura no *middleware* Ginga foi denominada GingaWaC.

O *middleware* Ginga possui características bastante apropriadas para a aplicação dos conceitos de autoria de documentos multimídia pelo usuário final. Isso possibilitou o *reúso* de diversos componentes. Alguns componentes foram utilizados na sua forma original, isto é, sem a necessidade de modificações. Entretanto alguns componentes não atendiam a todos os requisitos necessários e foram necessárias algumas alterações. Foram feitos três tipos de alterações:

- Herança: alguns componentes atendiam aos requisitos de forma parcial. Estes tiveram o comportamento expandido através de novas classes que complementaram os requisitos do componente;
- Modificadores de visibilidade: alguns componentes atendiam aos requisitos funcionais, mas alguns métodos e atributos estavam com modificadores de visibilidade inadequados. Métodos com os modificadores *private* e *protected* tiveram que ser modificados para se tornarem métodos públicos;
- Inserções de trechos de código: alguns componentes sofreram alterações mais substanciais. Tiveram o comportamento modificado ou complementado através da inserção e, em alguns casos, substituição de trechos de código.

5.1 Arquitetura do Middleware Ginga

A Figura 5.1 apresenta a arquitetura da implementação de referência do *Ginga*, em sua versão *0.10.1* encontrada no *Portal do Software Público*[1]

[1] http://www.softwarepublico.gov.br

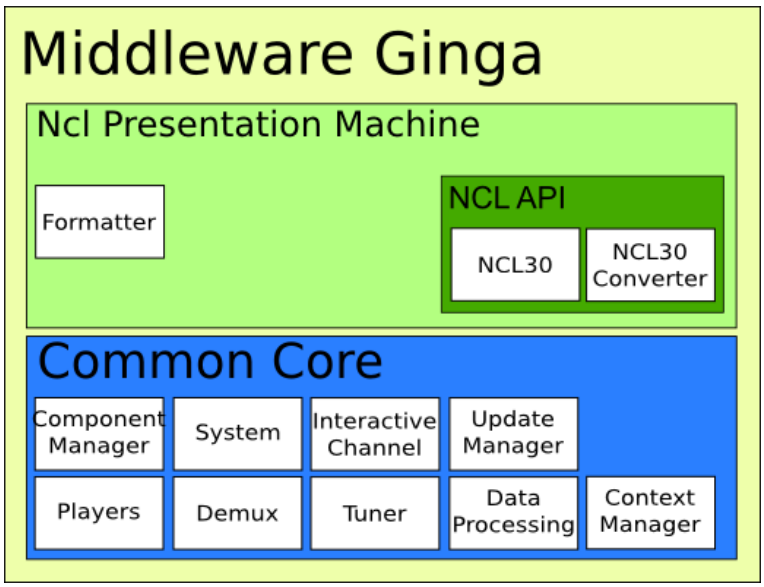

Figura 5.1: Arquitetura da implementação de referência *middleware Ginga*

Segue-se uma descrição de cada um dos módulos organizados nas camadas *NCL Presentation Machine* e *Common Core*

5.1.1 Common-Core

Na camada *Common-Core* tem-se os seguintes componentes:

- gingacc-cm(Component Manager): Componente que permite o carregamento e descarregamento dos demais componentes de forma dinâmica;
- gingacc-system (System): Componente que controla os dispositivos de entrada e saída do terminal de acesso. Funciona como interface do middleware com o *DirectFB*;
- gingacc-ic (Interactive Channel): Componente responsável por controlar e acessar o canal de retorno. Através deste componente é possível uma comunicação bi-direcional, em que aplicações em execução no terminal de acesso possam enviar dados a provedores de serviço (emissora, por exemplo);
- gingacc-um (Update Manager): Componente que oferece serviços para atualização de versões do *middleware*. Atualização ocorrem, principalmente, por meio da substituição de componentes existentes por outros;
- gingacc-player (Players): Componente responsável pela reprodução de diversos tipos de mídias. Essas mídias vão desde conteúdo audiovisual (vídeo, áudio, imagens) a aplicações em linguagem imperativa (Java e Lua).

- gingacc-tuner (Tuner): Responsável por realizar a sintonização de um canal;
- gingacc-tsparser (Demux): Responsável por realizar a desmultiplexação do *Transport Stream* enviado pela emissora, obtendo os fluxos elementares e tabelas de informação;
- gingacc-dataprocessing: Responsável por manter as informações sobre os elementos do *Transport Stream*, bem como montar e gerenciar o carrossel de dados e obter os documentos NCL que devem ser apresentados;
- gingacc-contextmanager: Armazena algumas informações de contexto como nome de usuário, faixa etária, etc. Essas informações podem ser usadas para fins de personalização e adaptação de conteúdo.

5.1.2 NCL Presentation Machine

Na camada *NCL Presentation Machine* tem-se os seguintes componentes:

- ncl30: Provê uma representação a nível de objetos das entidades de um documento *NCL*.
- ncl30-converter: Responsável por realizar o *parse* nos arquivos NCL e transformá-los em uma representação a nível de objetos do componente ncl30.
- formatter: Máquina de apresentação de documentos *NCL*.

5.2 Extensão do Middleware Ginga

A arquitetura original do *middleware* Ginga foi estendida para atender aos requisitos levantados para a construção de ferramentas de autoria com foco no usuário final (Seção 3.2). A Figura 5.2 mostra um diagrama com os módulos da arquitetura resultante da implementação de referência do *middleware* Ginga após a incorporação de mecanismos para autoria focada no usuário. Os componentes numerados na Figura 5.2 são os componentes comuns ao *middleware* Ginga e à arquitetura proposta neste trabalho. Os componentes que foram modificaçãos (ou construidos) são apresentados em blocos com fundo branco; componentes que foram preservados em suas versões originais são apresentados em fundo escuro. Os componentes exclusivos da arquitetura proposta neste trabalho estão destacados no lado direito da Figura 5.2.

A Figura 5.3 apresenta a arquitetura resultante através da notação *UML*, dando uma visão a nível de projeto da implementação obtida.

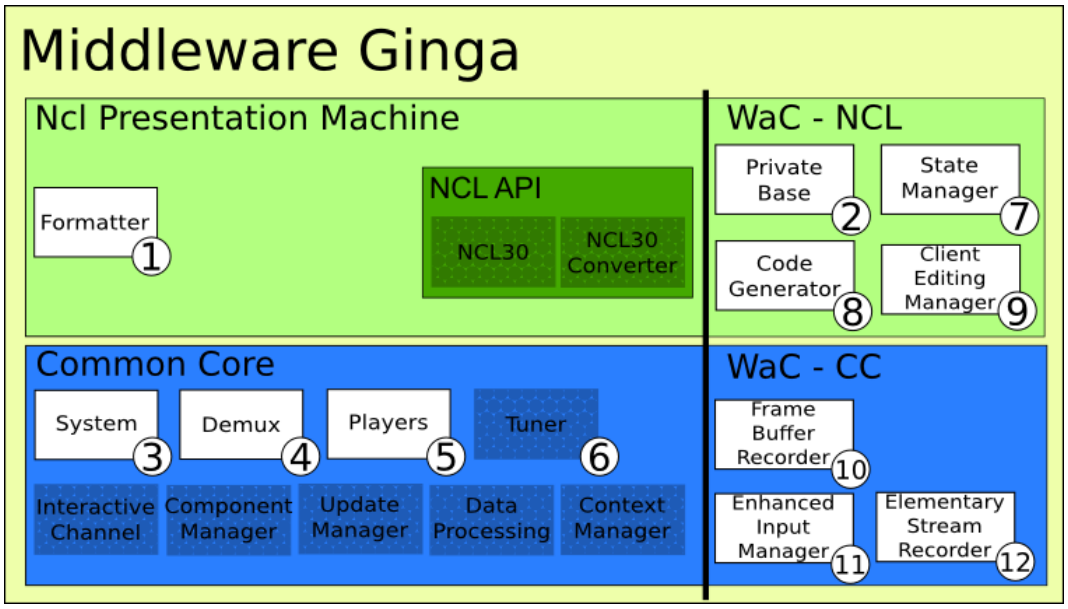

Figura 5.2: Arquitetura para a autoria ubíqua de documentos baseada em apresentações multimídia integrada ao *middleware* Ginga.

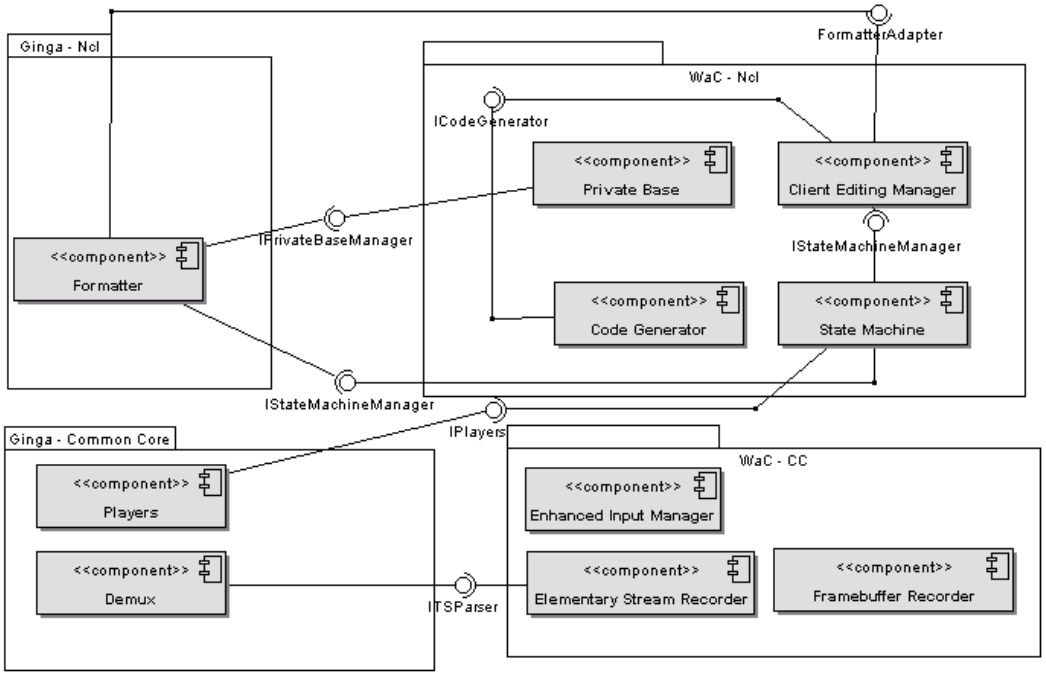

Figura 5.3: Diagrama de componentes da arquitetura integrada à implementação de referência do *Ginga*.

Na Seção 5.3 serão apresentados as alterações promovidas em alguns componentes do *middleware* e as razões para tais modificações. Em alguns componentes são descritas abordagens alternativas que poderiam ser adotadas.

5.3 Componentes

5.3.1 Novos Componentes

Esta seção descreve os novos componentes desenvolvidos para o *Ginga WaC*.

gingacc-capturer (Gravador de *frames* - *Framebuffer Recorder*)

Este módulo é a implementação do componente 4.1.10 da arquitetura. Permite a captura do conteúdo apresentado na TV como um vídeo (contendo o áudio) da apresentação; vídeo e áudio de forma independente; e imagens (*screenshots*). O conteúdo capturado pode ser salvo em arquivos (o que é útil para aplicações de anotação em documentos *NCL*) ou transferidos para outros dispositivos da rede doméstica (através do protocolo *UPnP - Universal Plug and Play*, por exemplo). O módulo foi subdividido em dois componentes, um deles responsável pela captura do conteúdo apresentado pelo *middleware* Ginga (áudio, vídeo, imagem) e outro responsável pela replicação do conteúdo (gravação do conteúdo em forma de arquivo) e transmissão do conteúdo para outros dispositivos.

Para a codificação e decodificação do conteúdo capturado, gravado e/ou retransmitido, foram utilizadas bibliotecas do *FFMpeg*[2]. A biblioteca do *FFMpeg* pode decodificar, converter e codificar arquivos de mídia de diversas fontes e formatos para outros diversos destinos (arquivos, servidores, etc) e formatos, fornecendo uma série de *encoders* e de *decoders*, facilitando o trabalho através da disponibilização de uma API.

O processo de captura do conteúdo foi implementado através da classe *GingaFFMpeg*. Esta classe foi desenvolvida com base no código fonte da biblioteca *FFMpeg* de forma a compatibiliza-se com o ambiente do Ginga.

Para a captura de conteúdo foi criado um componente capaz de capturar recursos multimídia que estejam sendo apresentados pelo *middleware* nas seguintes formas:

- *imagem da tela*: captura da imagem em exibição em um determinado instante. A imagem é uma réplica idêntica da imagem visualizada pelo usuário no momento em que o compo-

[2]http://ffmpeg.org/

nente foi acionado. A imagem poderá ser salva em diversos formatos, tais como JPEG[3], GIF[4] e PNG [5].

- *vídeo (apenas imagem)*: o vídeo em exibição no aparelho televisor poderá ser salvo da maneira como é apresentado no televisor, de forma idêntica. Dessa forma, toda imagem, com ou sem interação do usuário, será capturada de forma idêntica à imagem exibida no aparelho;

- *som*: apenas o áudio poderá ser capturado, sem imagem. Todo som reproduzido pela transmissão da televisão poderá ser obtido pelo componente.

- *vídeo (imagem e som)*: o conteúdo será copiado da transmissão do televisor. Toda imagem e todo som reproduzidos serão obtidos.

Após capturado, o conteúdo de TV Digital pode ter dois destinos: ser salvo em forma de arquivo no terminal de acesso ou ser transmitido para outro dispositivo da rede doméstica, viabilizando a visualização do conteúdo em outros equipamentos que não disponham de um sintonizador de TV. A transmissão de dados em rede é feita por meio do protocolo *RTP* (*Real Time Transport Protocol*). Desta forma, as opções de retransmissão são:

- *Retransmitir uma imagem da televisão para outro equipamento*: a imagem (*screenshot*) será capturada e, em seguida, enviada através do protocolo *RTP* para o endereço de IP na porta especificada;

- *Retransmitir um vídeo em exibição na televisão em outro equipamento*: o vídeo em veiculação na televisão será capturado e transmitido em tempo real (*on-the-fly*) por meio do protocolo RTP para o endereço e portas especificados da rede local. Através de um reprodutor de mídia compatível com a transmissão por RTP é possível assistir ao conteúdo da televisão em outros aparelhos, simultaneamente à exibição do vídeo no aparelho de televisão.

- *Retransmitir o áudio sendo tocado na televisão em outro equipamento*: Ocorre de forma análoga ao vídeo, mas utiliza o áudio ao invés de imagens.

- *Retransmitir o áudio e o vídeo da televisão em outro equipamento*: este caso difere dos demais, pois o protocolo RTP exige que apenas uma *stream* (fluxo de dados) seja transmitida por vez. Dessa forma, o áudio e o vídeo, após capturados, são codificados e

[3]ISO/IEC 10918-1
[4]Graphics Interchange Format
[5]Portable Network Graphics - ISO/IEC 15948:2004

transmitidos em duas *streams* de saída diferentes; são necessários, portanto, dois *sockets* diferentes para a transmissão (ao menos as portas deverão ser distintas). Para a visualização do conteúdo, o reprodutor de mídia deve suportar conexão para as diversas *streams* produzidas (no mínimo duas, para o caso de áudio e vídeo). A captura dos fluxos já integrados em um único *stream*, encapsulados em um *Transport Stream* é desejável, mas os resultados obtidos até o momento não contemplam esta opção.

O componente foi implementado segundo o paradigma de Orientação a Objetos. As seguintes clases compõem o componente:

- *Classe ImageCapture*: administra a interface de acesso ao componente básico de captura de *screenshots*.
- *Classe AudioCapture*: administra a interface de acesso ao componente básico de captura de áudio.
- *Classe VideoCapture*: administra a interface de acesso ao componente básico de captura de vídeo; esta classe também pode administrar a captura de áudio, para o caso de captura de áudio com vídeo.
- *Classe GingaFFMpeg*: classe adaptada do *FFMpeg* para compatibilidade interna com o Ginga. Responsável pela captura de áudio, pela captura de vídeo, pela codificação de dados para uso do telespectador e também pela transmissão de conteúdo através da rede e/ou por persistir o conteúdo capturado.

Ao contrário do código do *FFMpeg* original, o processamento de dados do código adaptado ocorre sempre por meio de *threads*. Isto evita que o conteúdo em exibição na televisão tenha que ser parado quando ocorre a captura, evitando-se atrasos e interrupções da programação exibida pela televisão. Os componentes criados para o Ginga utilizam essa versão adaptada do *FFMpeg* para a captura do conteúdo e sua manipulação.

Na Figura 5.4 é apresentado um diagrama de classes em alto nível. Classes internas do componente que não possuem interface com outros componentes foram ocultadas.

gingacc-streamrecorder (Gravador de fluxos elementares - *Elementary Stream Recorder*)

Este componente implementa o Item 4.1.12 da arquitetura e responde por persistir os fluxos enviados pela emissora individualmente. Esses *streams* podem ser, entre outras coisas, fluxos

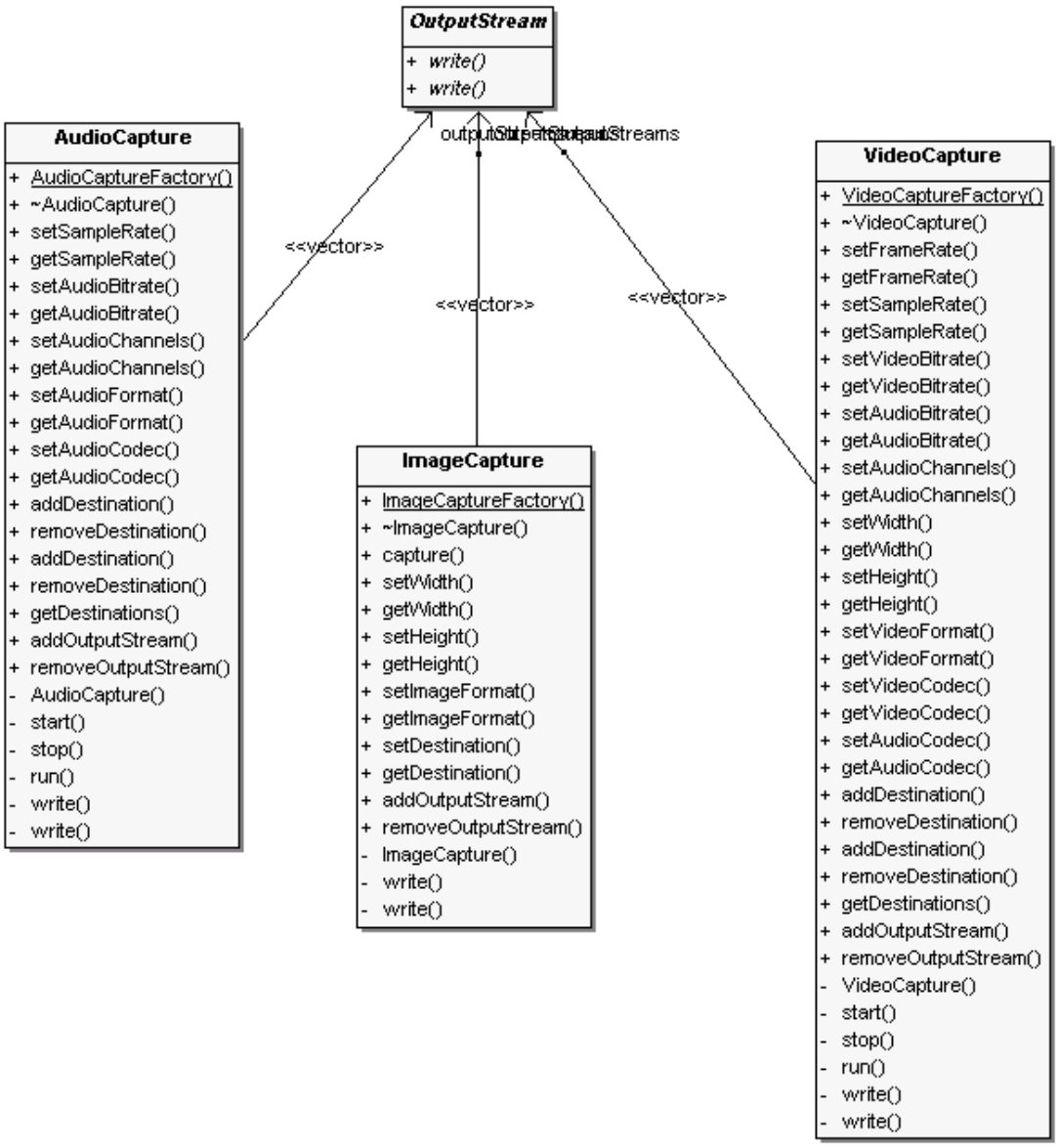

Figura 5.4: Diagrama de classes do componente *gingacc-capturer*

de vídeos ou áudio enviados pela emissora para apresentação em um dado programa interativo. O armazenamento desses fluxos permite a reapresentação de um programa mantendo as opções de interatividade originais. As opções de interatividade são mantidas a nível de documento e mídias. Determinados programas ao vivo podem oferecer opções de interação através do canal de retorno. Nesses casos o acesso aos recursos de interação ficam indisponíveis posteriormente.

Através desse componente é possível o agendamento e gerência da gravação de programas e gerência dos fluxos armazenados no terminal de acesso. Este componente simplifica o processo de criação de aplicações da classe de um PVR (*Personal Video Recorder*).

O componente acessa diversos serviços oferecidos por outros componentes de *middleware*, como o *gingacc-tsparser* e *gingacc-tuner*. Algumas modificações nas versões originais dos componentes foram promovidas. Essas alterações são descritas na Seção 5.3.2.

O componente foi implementado segundo o paradigma de Orientação a Objetos. As seguintes classes e estruturas compõem o componente:

- *Header*: armazena informações extraídas das tabelas de informação relativas aos fluxos que compõe um programa transmitido.
- *Record*: trata-se da principal classe do componente, utilizando serviços oferecidos por outros componentes efetua a captura do *payload* dos fluxos.

A Figura 5.5 mostra o diagrama de classes deste componente. Em branco temos as classes pertencentes a esse módulo. Em cinza claro temos as classes da implementação de referência do Ginga que foram modificadas para a construção desse módulo. Em cinza escuro temos as classes da implementação de referência que não foram modificadas mas que provêm serviços ao componente.

O componente não pôde ser validado devido à indisponibilidade de um gerador de *Transport Streams*. O desenvolvimento foi feito com base nas especificações [ABNT 2007].

gingacc-enhancedim (Gerenciador de entradas - *Enhanced Input Manager*)

O componente *Enhanced Input Manager* (4.1.11) é composto por uma classe que estende a classe *InputManager*, a *EnhancedInputManager*, provendo todas as funcionalidades originais mais o controle de modos (modos de entrada). Está prevista a adição da funcionalidade de interação multimodal, que permite que dispositivos que capturem eventos multimodais possam enviar eventos à *EnhancedInputManager* e também que módulos ou programas que queiram receber tais eventos multimodais possam se registrar para tal.

O componente é composto pelas seguintes classes:

- *IEnhancedInputManager*: estende a interface da *IInputManager*. É através desta interface que os outros módulo do Ginga terão acesso aos serviços oferecidos por este módulo.
- *EnhancedInputManager*: estende as funcionalidades da *InputManager*, adicionando as funcionalidades para interações multimodais e modos de entradas distintos para multiplexar diversas funcionalidades para os botões do controle.

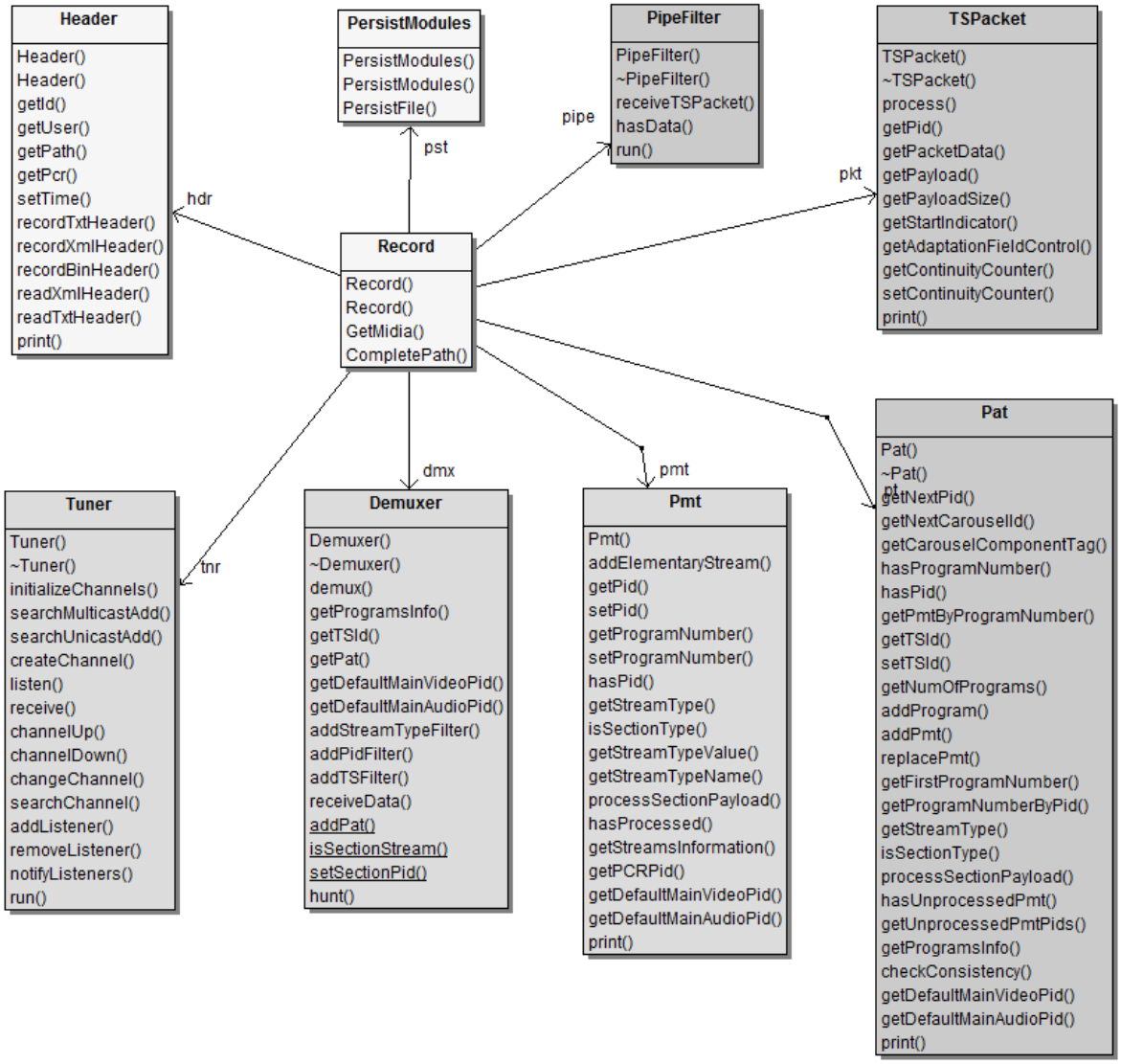

Figura 5.5: Diagrama de classes do componente *gingacc-streamrecorder*

- *IInputMode*: interface que todas as classes que proporcionem modos de entrada devem implementar.

- *InputModeModifier*: classe abstrata que contém algumas funcionalidades para modos de entrada que não redirecionam a entrada, apenas a modificam de algum modo (por exemplo convertendo um botão em outro). Um componente que deseja essa funcionalidade deve estender essa classe.

- *MultimodalEvent*: classe que implementa a interface *IInputEvent*, pois representa um evento dentro do *middleware* Ginga. Possui um único atributo que é o identificador do tipo de evento multimodal que ele representa. Cada dispositivo multimodal deve criar uma extensão própria dessa classe, onde adicionará os métodos e atributos necessários

para representar um evento multimodal daquele tipo.

- *InputModeRedirecter*: classe abstrata que contém algumas funcionalidades para modos de entrada que redirecionam as entradas enviadas pelo usuário. Um modo que deseje esta funcionalidade deve estender essa classe. Essa classe também implementa a interface *IInputEventListener*, pois a classe que irá implementar este tipo de modo deve estar preparada para receber os eventos.

- *IMultimodalEventListener*: essa interface estende a interface *IInputEventListener*. Ela deve ser implementada por qualquer classe que deseje se registrar na *EnhancedInputManager* para receber eventos multimodais de algum tipo. Esse tipo é passado como parâmetro quando a classe se registra.

- *IMultimodalAdapter*: essa interface deve ser implementada por todas as classes que captam ou controlam eventos multimodais. Classes que implementem essa interface funcionam como uma camada de adaptação que permitem a comunicação entre os dispositivos e o *EnhancedInputManager*.

A Figura 5.6 mostra o diagrama de classes do componente gingacc-enhancedim. Em cinza claro estão as classes pertencentes ao componente gingacc-system que foram modificados, em cinza escuro as classes que não foram modificados. Em branco estão as novas classes que integram este componente.

ncl30-generator (Gerador de código - *Code Generator*)

Este componente (item 4.1.8 da arquitetura) converte os objetos do módulo *ncl30* para código *NCL* equivalente, isto é, converte uma representação abstrata a nível de objetos em uma representação textual do documento.

O componente consiste, basicamente, em uma classe para cada classe que representa uma entidade *NCL* concreta do módulo *ncl30*. As classes desse componente são construídas estendendo-se a classe original. Elas possuem o sufixo **generator** em seus nomes e acrescentam um método que responde pela geração do código referente à entidade.

Foi desenvolvida a classe *NclGenerator*, composta por uma série de métodos estáticos que convertem os objetos comuns do módulo *ncl30* em objetos equivalentes do componente *ncl30-generator*, capazes de gerar código XML (*NCL*).

Esse componente possui, no estágio atual, algumas limitações relacionadas à incapacidade

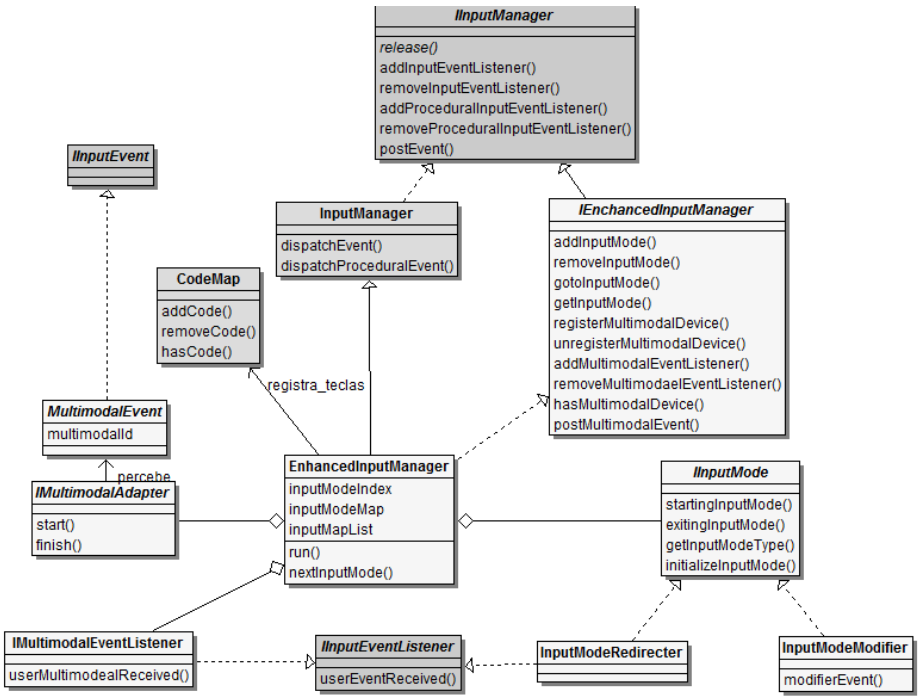

Figura 5.6: Diagrama de classes do componente *Gerenciador de entradas*

de gerar código para transições, meta-dados e a parte de *switchs* da linguagem *NCL*. Em futuras versões espera-se sanar tais limitações.

A Figura 5.7 mostra parte do diagrama de classe do componente *ncl30-generator*. Em branco temos as classes pertencentes a este componente e em cinza escuro temos as classes pertencentes ao módulo *ncl30*. Não foi necessário modificar nenhuma classe da implementação de referência do Ginga para criamos este componente. Para facilitar o entendimento estão representadas no diagrama apenas algumas entidades *NCL*.

ncl30-privatebase (Gerenciador de bases privadas - *Private Base Manager*)

Este componente traz uma implementação simples do *private base manager* (item 4.1.2). A implementação de referência do Ginga utilizada, por padrão, cria uma única base privada e adiciona todos os documentos a ela. Com este componente é criado o suporte a diversas bases privadas(cada uma com um objeto *DocumentManager* associado). Esse componente gerencia a criação e recuperação de bases privadas por meio da classe *PrivateBaseManager*.

A Figura 5.8 mostra o diagrama de classes do componente *ncl30-privatebase*. Em cinza claro são apresentadas as classes pertencentes que demandaram modificações. Em cinza escuro as classes dos componentes *ncl30-conveter* e *ncl30* em suas versões originais, das quais existem dependências. Em branco temos as classes deste componente.

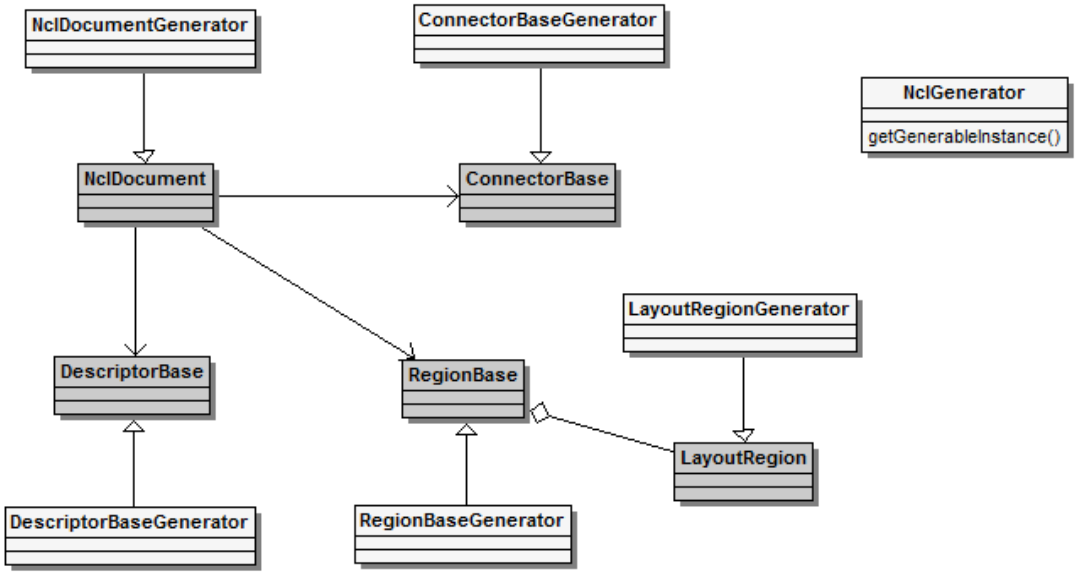

Figura 5.7: Diagrama de classes do componente *Gerador de código*

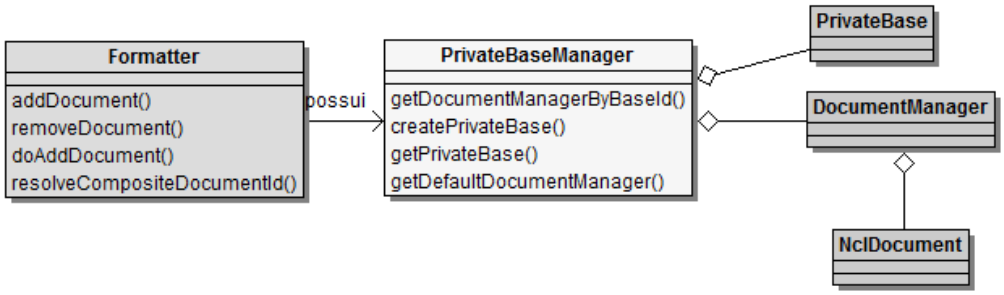

Figura 5.8: Diagrama de classes do componente *Gerenciador de bases privadas*

wac-editing (Gerenciador de edições do cliente - *Client Editing Manager*)

Componente (item 4.1.9 da arquitetura) responsável por efetuar a edição ao vivo do usuário e alternar entre o modo de exibição do usuário e o modo de exibição da emissora.

O componente é composto pelas classes enumeradas abaixo:

- *ClientEditingManager*: possuiu os métodos de edição ao vivo pelo cliente. Registra as edições realizadas para que seja possível desfazê-las posteriormente, caso necessário. Para isto existe uma interface com a classe *Formatter* do módulo ginga-ncl através da interface *IFormatterAdapter*.
- *IFormatterAdapter*: interface para enviar mensagens ao *Formatter* os comandos de edição ao vivo.

- *ISchedulerAdapter*: interface para o envio de mensagens como a finalização de um *player* ao componente *Scheduler*.

- *ModeManager*: classe responsável pela comunicação com o componente *FormatterScheduler* através da interface *IScheduler Adapter* para a eliminação dos objetos desnecessários. Os *links* da linguagem *NCL* passam por uma análise que permite a identificação se o elo (*link*) deve ser disparado, de acordo com o modo de exibição corrente. Através desta classe é, ainda, permitida a troca dos modos de exibição, alternando entre os modos de exibição *Cliente* (contendo anotações do usuário) e *Emissora* (contendo somente o conteúdo da emissora).

- *ILinkAction*: interface que os objetos do tipo *LinkAction* do módulo *ginga-ncl* implementam para que seja possível verificar o tipo dos *links*(elos).

- *EditingCommand*: classe que representa um comando de edição do usuário. Armazena o objeto que foi adicionado ao documento, informações sobre onde ele foi adicionado e o tipo de comando executado. É útil para desfazer alterações ou recuperar o modo de exibição da emissora, removendo todos os comandos que geram nós de mídias, por exemplo. Os comandos são gerados a partir das chamadas de edição à classe *ClientEditingManager*, onde são armazenados em estruturas de dados apropriadas.

- *Imode*: interface para os serviços fornecidos pela classe *ModeManager*.

- *IClientEditing*: interface para os serviços fornecidos pela classe *ClientEditingManager*.

A Figura 5.9 mostra o diagrama de classes do componente *wac-editing*. Em branco as classes deste componente. Em cinza claro as classes do componente *ginga-ncl* modificadas. Em cinza escuro as classes do componente ginga-ncl que não foram modificadas mas que oferecem serviços às classes do componente *wac-editing*.

wac-state (Gerenciador da máquina de estados)

Este componente provê serviços para a coleta do estado de execução do documento *NCL*. Trata-se da implementação do componente 4.1.7 da arquitetura. Através do estado de cada *player* em execução no módulo *ginga-ncl* é obtida a máquina de estados do documento *NCL* em apresentação.

O componente é composto pelas seguintes classes e interfaces:

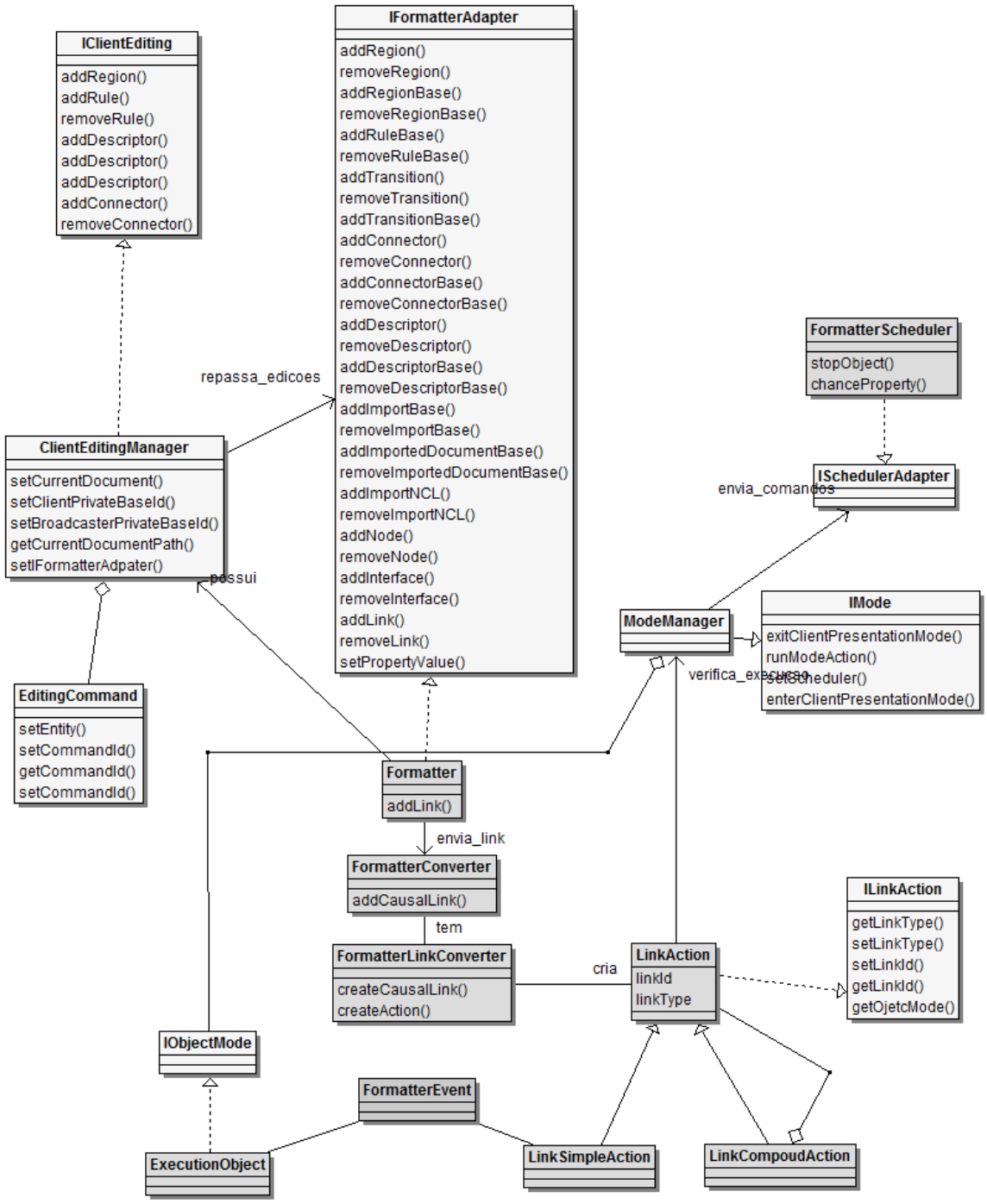

Figura 5.9: Diagrama de classes do componente *wac-editing*

1. *IPlayerWac*: interface utilizada para obtenção dos estados dos *players* das classes do *ginga-ncl*. A classe *PlayerAdapter* implementa esta interface, tendo um método adicionado que permite a captura dos estados dos *players*.

2. *StateManager*: é a classe responsável por administrar os *players* em execução e fornecer o estado da apresentação através de uma instância da classe *PresentationState*.

3. *PresentationState*: representa o estado da apresentação *NCL* em um dado momento. É composta por instâncias da classe *PlayerStateWac*.

4. *PlayerStateWac*: é uma especialização da classe *PlayerState* do módulo *Players*.

A Figura 5.10 mostra o diagrama de classes do componente *wac-state*. As classes introduzidas na arquitetura do *middleware* estão apresentadas no diagrama em branco. Em cinza claro estão representadas as classes do módulo *ginga-ncl* e em cinza escuro as classes do módulo *gingacc-player*.

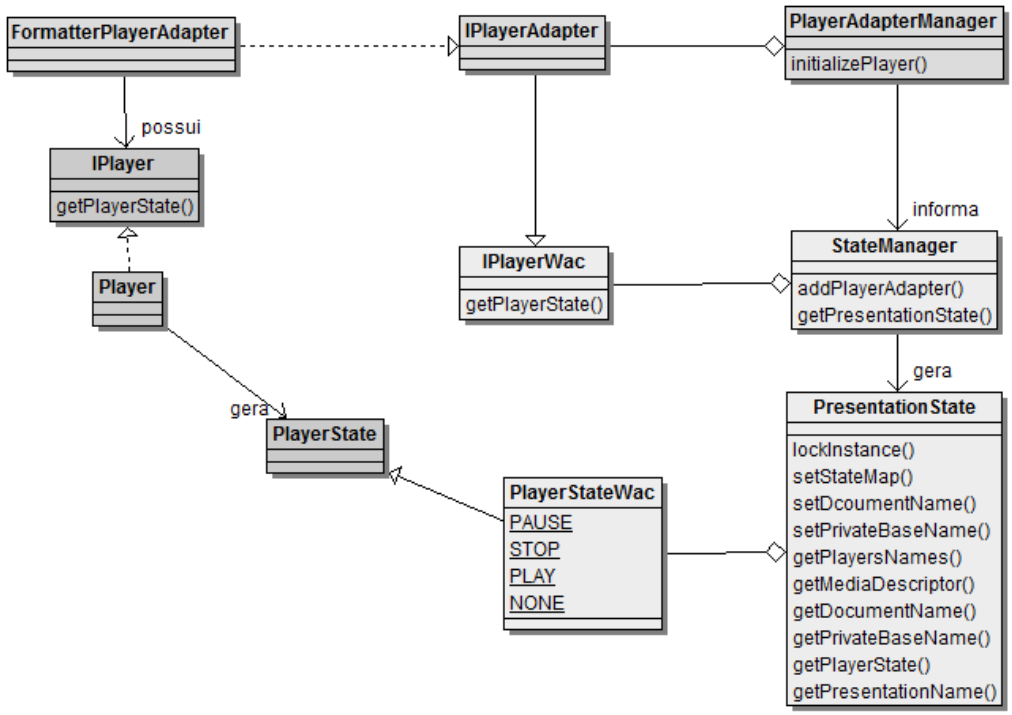

Figura 5.10: Diagrama de classes do componente *wac-machinestate*

5.3.2 Componentes Reusados

Esta seção apresenta componentes comuns entre a arquitetura do *middleware* Ginga e a arquitetura proposta neste trabalho.

gingacc-tsparser (Demultiplexador - *Demux*)

Este componente é o responsável por extrair as informações e fluxos individuais de vídeo e áudio de um fluxo *MPEG-2 Transport Stream* enviados pelo radiodifusor (emissora). Este componente foi alterado para que os fluxos pudessem ser obtidos individualmente e gravados pelo componente *gingacc-streamrecorder*. As modificações realizadas neste componente foram:

- Na classe *Demuxer* os modificadores de acesso de alguns métodos foram alterados de *private* para *public*, pois o componente *gingacc-streamrecorder* requer acesso a esses métodos para realizar a captura dos fluxos.
- Na classe *Pmt* foi trocado o modificador de acesso de um atributo responsável por identificar cada um dos fluxos *MPEG2-TS*. Isto é necessário para que o componente *gingacc-streamrecorder* possa identificar quantos e quais são os fluxos elementares existentes em cada programa.

gingacc-tuner (Sintonizador - *Tuner*)

Este componente é o responsável pela sintonização de um canal. É através dele que é possível a escolha do canal a ser exibido. Através do *Tuner* é feita a comunicação com o *hardware* para que os processos de sintonização e demultiplexação do fluxo o fluxo *MPEG-2 Transport Stream* sejam processados.

Nesse módulo foram alterados modificadores de acesso de alguns dos atributos que armazenam informações do canal da classe *Tuner*. Esses atributos tiveram os modificadores alterados de *private* para *public* para suportar algumas operações requeridas pelo componente *gingacc-streamrecorder*.

gingacc-player (Exibidores de Mídia - *Players*)

Este componente é o responsável pela exibição das diversas mídias suportadas pelo *middleware* (formatos de vídeo, áudio e imagens, dentre outros).

Foi necessário modificar esse componente para permitir que informações como estado da mídia, tempo, tamanho e localização dos *players* fossem obtidas. Essas informações são necessárias para a identificação do estado da apresentação do documento multimídia.

As modificações realizadas neste componente foram:

- Foi adicionada a classe *PlayerState*. Essa classe é responsável por encapsular o estado em um determinado instante de um *player*. Informações como estado de execução (pausado, executando, parado), tempo de execução, tamanho e posição na tela são armazenados.

- Na classe *Player* e interface *IPlayer* foi adicionado um método que retorna o estado do *player*. Esse método constrói e retorna uma instancia de *PlayerState* que contém o estado atual do *player* no instante em que o método foi chamado. O método foi implementado somente na classe *Player*, porém foi necessário adicionar sua assinatura na interface *IPlayer* já que é através desta interface que outros componentes manipulam os *players*.

gingacc-system (Gerenciador de dispositivos de interação - *Input/Output*)

Esse componente responde por diversas tarefas no *middleware*, dentre elas a manipulação dos dispositivos de entrada e saída de dados. Uma aplicação que realize anotações em documentos *NCL* provavelmente necessitaria de uma interface de entrada bastante rica, com a utilização de diversos comandos e botões do controle remoto. Isto é complicado de ser feito em um terminal de acesso tradicional, pois o principal dispositivo de entrada, o controle remoto, possui um número restrito de teclas, além de existir uma grande probabilidade de alguns botões estarem em uso por aplicações interativas (enviadas pela emissora, por exemplo). Por essas razões, foi necessário criar uma maneira diferente de explorar o controle remoto para realizar diferentes interações, em diferentes contextos.

Foi adotada uma abordagem de incluir no *middleware* a opção de suporte a diferentes modos de interação. Dessa forma é reservado um único botão para a troca dos modos de interação, possibilitando a adição de múltiplas semânticas a cada botão, de acordo com os modos de interação. Isso permitiu um aumento significativo da capacidade do controle remoto como dispositivo de entrada.

Inicialmente, realizamos modificações no componente da implementação de referência para adicionar as novas funcionalidades. No desenvolvimento do trabalho, percebemos que as modificações feitas eram muito substanciais, justificando a criação de um componente para esse fim específico. Dessa forma foi criado um novo componente para tratar essas questões, mas algumas modificações na versão original do componente foram mantidas:

- Adição de métodos e atributos na classe *CodeMap* para que novos botões do controle pudessem ser mapeados. A idéia é utilizar esses novos botões em conjunto com os modos de interação, conseguido aplicar uma maior semântica aos novos botões.

- Modificações nos modificadores de acesso de métodos e atributos da classe *InputManager* de *public* para *protected*.

O novo módulo criado estende a classe *InputManager* e incrementa sua funcionalidade ao adicionar recursos para a manipulação dos modos de entrada.

ginga-ncl (Formatador - *Formatter*)

Este é um dos módulos que compões o núcleo do subsistema *Ginga-NCL* da implementação de referência do *middleware*. O componente responde pela apresentação dos documentos *NCL*. O componente foi substancialmente alterado pois centraliza algumas das atividades necessárias para a edição do documento multimídia. Através desse componente é feito o controle das modificações na apresentação de um documento *NCL*, através dos comandos de edição ao vivo. Além disso esse componente permite que informações relacionadas aos *players* e nós de mídia de um documento *NCL* sejam obtidas. As modificações nesse componente foram:

- Implementação das interfaces *IFormatterAdapter* nos módulos *WaC*. As classes desenvolvidas em nossos componentes precisam enviar mensagens para o *Formatter* (para executar as edições ao vivo). O sentido inverso (envio de mensagens da classe *Formatter* para os módulos *WaC*) também foi suportado. Para resolver esse problema de referência cruzada, fizemos o *Formatter* implementar a interface *IFormatterAdapter* pela qual as classes dos módulos *WaC* se comunicam com o formatador.

- Adição de dipositivos para gerência das bases privadas. Na versão do *middleware* utilizada como base para o desenvolvimento era considerada apenas uma base privada. Entretanto com a adição da possibilidade de uso de mais de uma base privada foi adicionada uma referência ao *PrivateBaseManager* no formatador.

- Adição de métodos que permitem a manipulação de documentos em bases privadas.

Como não tínhamos bases privadas diferenciadas, todos os documentos eram adicionados em uma mesma base genérica. Agora os documentos podem ser adicionados a bases privadas específicas. Assim, precisamos de métodos com estas assinaturas. Os métodos antigos que adicionavam documentos à base privada genérica ainda existem para que seja mantida a compatibilidade com a interface anteriormente utilizada. Quando os métodos antigos são utilizados, eles operam sobre uma base privada genérica.

Analisando o componente, ao fim do desenvolvimento, observamos uma solução alternativa que seria a transferência de parte das responsabilidades desses métodos para o módulo de edição

WaC, preservando o componente *Formatter* mais próximo de sua versão original. Constatamos que o *Formatter* precisa diferenciar apenas as edições ao vivo que adicionem entidades do tipo *Link*. Uma possível abordagem seria diferenciar apenas a adição de *links* ao documento NCL, mantendo os métodos de adição das demais entidades preservado. Através dos *links* é possível neutralizar as ações desempenhadas pelo usuário.

Para mantermos a compatibilidade com a interface padrão do Ginga, criamos o conceito de identificador composto dos documentos, que é formado pelo nome do documento mais o nome da base privada onde este está localizado. Assim, em todos os pontos do código fonte desta classe onde nos referenciávamos ao identificador do documento, foi necessário colocar códigos especiais (basicamente o método *resolveCompositeDocumentId*) para recuperamos os nomes da base privada e do documento e tratá-los apropriadamente. Acredito que poderíamos remover esta modificação da classe *Formatter*, passando os identificadores compostos para o módulo que gerencia bases privadas e permitir que ele faça este tratamento.

A classe *LinkAction* foi alterada de forma a implementar a interface *ILinkAction* do módulo WaC, permitindo a comunicação com ela. Além disso foram adicionados novos métodos e atributos para a diferenciação dos *links* do cliente e da emissora.

Nas classes *LinkSimpleAction* e *LinkCompoundAction*, fizemos uma pequena modificação no método *run* para chamar o verificador de modos antes de disparar os eventos. É através desta chamada que o programa é capaz de, em tempo de execução, cancelar a execução de um *link* caso este seja um *link* do cliente e estivermos no modo de exibição da emissora, o que gera a impressão de termo dois documentos distintos em execução simultânea.

6 *Avaliação da Proposta*

Os casos de uso considerados para a delimitação do domínio do trabalho (Figura 3.1) foram utilizados como base para avaliação da proposta. Foi construído um conjunto de aplicações que apresentam ao usuário opções de edição (autoria) do documento utilizando os componentes da arquitetura instanciados no *middleware* Ginga através da implementação *GingaWaC*.

As aplicações *provas de conceito* foram desenvolvidas na linguagem *C++*. O desenvolvimento poderia ter sido conduzido com o uso de outras linguagens como *Lua* e *Java* que são suportadas pelo *middleware*. Para que os componentes possam ser acessados nos ambientes *Lua* e *Java* é necessário que novas *APIs*[1] sejam incorporadas a esses ambientes. Essa incorporação pode realizada através de adaptadores que expõem os serviços de *middleware* às aplicações. O desenvolvimento desses adaptadores não foi contemplado no desenvolvimento do *GingaWaC*.

6.1 Aplicações

As seguintes aplicações foram desenvolvidas:

1. Change Mode
2. Bookmark
3. Skip
4. Loop
5. Review

[1] *Application Programming Interfaces*

6.1.1 *Change Mode*

As aplicações desenvolvidas demandam o uso de alguns botões do controle remoto para a disponibilização das opções de autoria. Como alguns botões podem estar associados a aplicações interativas, é inviável o mapeamento das teclas de forma direta. Este caso de uso apresenta uma funcionalidade, disponível ao interagente, que minimiza esse problema. São disponibilizados ao usuário dois modos: *modo de interação* (*watch mode*) e *modo de edição* (*watch and comment mode*).

Assumimos o uso de uma tecla especial no controle remoto para ativar o modo de edição. Na implementação realizada o usuário entra no modo de edição *watch and comment mode* ao pressionar o botão de mudança de modo quando encontra-se no *modo de interação*(assistindo TV e interagindo com o conteúdo).

A Figura 6.1 mostra o *feedback* recebido pelo usuário quando o botão de alterância de modo é pressionado. Para indicar ao interagente que o modo de edição está ativo um ícone com esta indicação é apresentado na tela. Estando nesse modo as operações disponíveis são indicadas na parte inferior da tela.

Nesta implementação, especificamente, é disponibilizado o acesso às aplicações de **BOOKMARK**, **SKIP** e **LOOP**. Cada aplicação reserva um conjunto de botões para suas operações aplicando a semântica adequada. Por exemplo: em um cenário normal, no *modo de interação*, o botão verde poderia estar programado pelo autor do documento multimídia (emissora) para apresentar um produto disponível para compra, associado a uma determinada cena; no *modo de edição* o mesmo botão pode ser programado para adicionar a cena à lista de favoritos do usuário.

Para a implementação deste caso de uso foi utilizado o componente de *Gerenciamento de entradas*.

6.1.2 *Bookmark*

Foi implementada uma aplicação que permite a marcação de instantes de um vídeo (interativo). Quando o usuário entra no *modo de edição*, a operação *Bookmark* é ativada através do botão verde do controle remoto. Na Figura 6.2(a) a operação *BOOKMARK* é apresentada no rodapé da tela.

Quando o usuário pressiona o botão verde o *frame* é extraído e disponibilizado para acesso através de uma opção da aplicação *Review*. Considerando que o conteúdo é gravado na íntegra

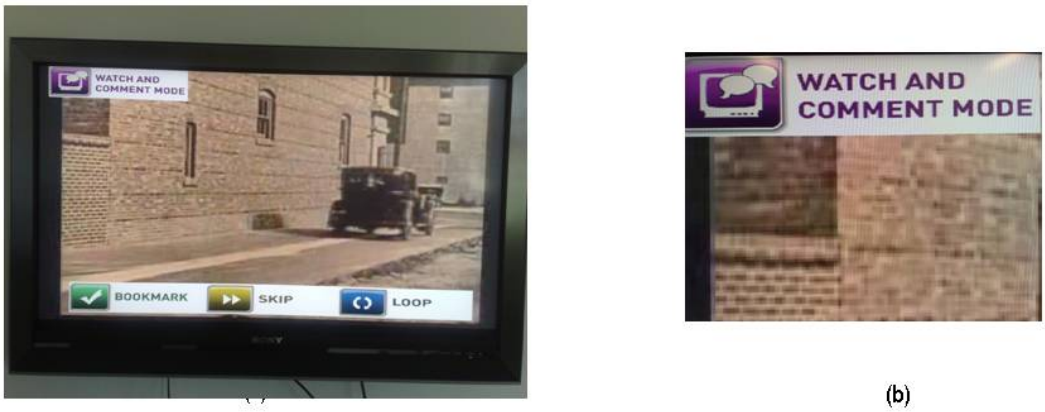

Figura 6.1: Alternância de modos de interação

(através do componente de gravação de fluxos elementares) é possível que o usuário possa fazer um salto para um momento anterior do vídeo. Esse salto dentro do vídeo pode ser feito de maneira dirigida, através de um índice (cenas preferidas) construído pelo próprio usuário. Esse índice é acessível pela aplicação *Review* onde os *frames* que representam instantes selecionados são apresentados na forma de miniaturas (Figura 6.4).

6.1.3 *Skip (Start and Stop)*

Implementamos uma aplicação que permite ao usuário marcar um segmento de um vídeo (delimitando o início e o fim) para ser suprimido do vídeo (*skipped*) quando o vídeo é exibido novamente, em outra oportunidade.

A implementação funciona da seguinte forma: quando o usuário entra no *modo de edição*, a operação *Start Skip* é ativada através do botão amarelo do controle remoto. Na Figura 6.2(a) a operação *SKIP* é apresentada no rodapé da tela. Caso o usuário selecione uma cena (através do botão amarelo), a aplicação exibe a mensagem de *WAITING STOP SKIP* como uma operação disponível no rodapé da tela (Figura 6.2(b)). A partir deste ponto o usuário tem a opção de selecionar o fim do segmento do vídeo (*Stop Skip*) ou cancelar a seleção do início do segmento.

Para selecionar o *frame* associado com o fim do segmento o usuário pode pressionar o mesmo botão (amarelo) utilizado para a marca de (*Start Skip*). O *frame* de vídeo correspondente às marcas de início e fim são armazenados. Na implementação desta aplicação o *frame* utilizado para demarcação do segmento de início é utilizado como um ícone apresentado ao usuário quando este acessa a lista de segmentos do tipo *SKIP* .

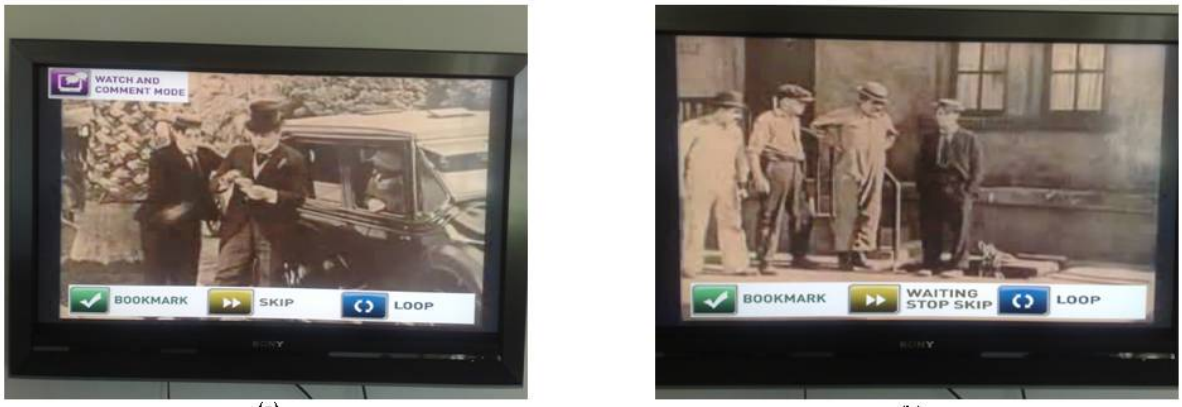

Figura 6.2: Aplicação *Skip*

6.1.4 *Loop (Start and Stop)*

Dentre as aplicações projetadas para permitir a autoria do interagente foi implementado um protótipo que permite a seleção de um segmento do vídeo para exibição em *Loop*. O usuário pode marcar um instante de início do segmento para ser repetido automaticamente até a marca de fim do segmento.

Quando o usuário está no *modo de edição* o comando de *Start Loop* pode ser ativado através do botão azul do controle remoto. Na Figura fig:skip(a) a opção LOOP é a última opção à esquerda na lista apresentada no rodapé da tela.

O comportamento é similar à aplicação *Skip*. Quando a tecla azul é pressionada a aplicação captura o frame correspondente e exibe o ícone *WAITING STOP LOOP*. Caso o botão azul seja pressionado novamente o segmento é delimitado. O usuário possui, ainda, a opção de cancelar a operação, anulando um *frame* já marcado.

6.1.5 *Review*

A aplicação de *review* foi projetada para permitir que o usuário acesse, através de um menu, opções de visualização de trechos da mídia anotados. Neste caso foi considerado o uso de um botão que ativa o menu (uma situação análoga ao *modo de edição*).

A implementação realizada funciona da seguinte forma: estando no *modo de interação* (acessível pela tecla de mudança de modos) o menu de *review* é apresentado quando a tecla *info* é pressionada (Figura 6.3(a)).

Quando uma das opções do menu é selecionada pelo usuário (Figura 6.3(b)) uma lista de

frames correspondentes à opção é apresentada. Essa lista traz miniaturas correspondentes ao início dos segmentos de vídeo demarcados. A lista de *frames* é navegável, permitindo ao usuário navegar pelos segmentos do vídeo (Figura fig:reviewscene).

Esta aplicação ilustra claramente a necessidade de permitir que o usuário (interagente) possa navegar pelas anotações em tempo de interação. Aplicações que permitam a adição de marcas mais semânticas como em uma aplicação do domínio de esportes, em que usuários desejam selecionar "melhores momentos" (como um gol no futebol) e visualizá-las a qualquer tempo aproveitam-se dos recursos oferecidos pela arquitetura proposta neste trabalho.

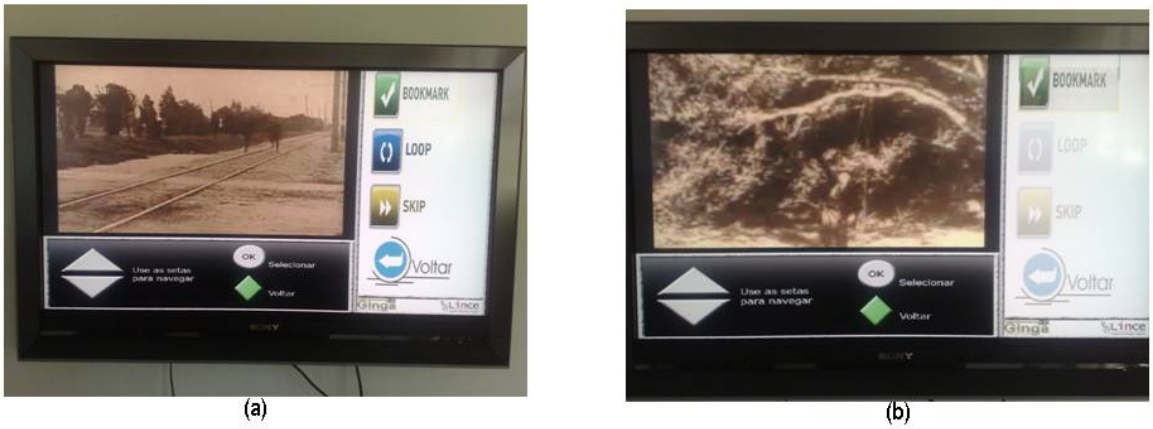

Figura 6.3: Aplicação *Review*

Figura 6.4: Visualização de uma cena

6.2 Documento Estruturado

Esta seção apresenta o resultado da autoria de um documento baseada na apresentação de um outro documento *NCL*. Para facilitar a descrição consideramos uma aplicação simples, composta por um vídeo principal e algumas imagens, adicionadas ao documento caso o usuário pressione os botões coloridos do controle remoto.

O conteúdo resultante da autoria está apresentado no mesmo documento *NCL*. Isso poderia ser feito em outro documento utilizando o suporte de múltiplos documentos da linguagem. Entretanto, para facilitar o entendimento, isso será feito no mesmo documento. Mais adiante é apresentado um exemplo em que o reúso de documentos é considerado.

O documento NCL original, enviado pela emissora, é apresentado no Apêndice A.1.

Considerando que o cliente deseje realizar uma anotação do tipo *Skip* no vídeo (utilizando a aplicação *Skip*) o botão de troca de modo deverá ser pressionando para que o usuário possa acessar a aplicação de anotação, através do *modo de edição*.

Quando o botão é pressionado, o componente *Enhanced Input Manager* recebe um evento do controle contendo o código da tecla de mudança de modo. Ao detectar esta tecla o *Enhanced Input Manager* percorre sua lista circular de modos de entrada diferentes e aciona o próximo modo, que neste caso, é o *modo de edição*.

O componente *Client Editing Manager* é então notificado de que o *modo de edição* foi acionado pelo usuário. O componente então exibe na tela uma imagem informando que o *modo de edição (Watch and Comment Mode)* é o modo de entrada atual.

O usuário então pressiona o botão do controle remoto correspondente à operação de anotação da aplicação *Skip*. Assim que o evento com o botão pressionado chega ao componente *Enhanced Input Manager*, ele o encaminha até a classe responsável por tratar os eventos do modo de anotação (que poderia ser uma aplicação Lua, por exemplo).

A aplicação responsável por tratar os eventos do modo de edição recebe então o evento e inicia a anotação de *skip*. Neste momento são adicionados conectores, áreas, descritores e nós de mídia que são usados na operação de *skip* ao documento. Para adicionar os elementos são utilizados os componente *Code Generator* e a API de edição ao vivo disponível na classe *Formatter* da implementação de referência do Ginga.

As estruturas incorporadas ao documento *NCL* são apresentadas na Figura 6.5.

Regiões e descritores que irão ligar os nós de mídia que são utilizados para mostrar a ex-

```
<causalConnector id="_WAC__connSetState1">
<simpleCondition role="onSelection" eventType="selection" transition="stops" />
<compoundAction operator="seq" >
<simpleAction role="abort" eventType="presentation" max="unbounded" qualifier="seq" />
<simpleAction role="start" eventType="presentation" max="unbounded" qualifier="seq" />
</compoundAction>
</causalConnector>

<causalConnector id="_WAC__connOnStateStart">
<compoundCondition operator="and" >
<simpleCondition role="onBegin" eventType="presentation" transition="starts" max="unbounded" />
</compoundCondition>
<compoundAction operator="seq" >
<simpleAction role="start" eventType="presentation" max="unbounded" qualifier="par" />
</compoundAction>
</causalConnector>

<causalConnector id="_WAC__connOnStateStop">
<compoundCondition operator="and" >
<simpleCondition role="onBegin" eventType="presentation" transition="starts" max="unbounded" />
</compoundCondition>
<compoundAction operator="seq" >
<simpleAction role="stop" eventType="presentation" max="unbounded" qualifier="par" />
</compoundAction>
</causalConnector>
```

Figura 6.5: Adição de conectores ao documento *NCL*

istência de um trecho de skip no documento anotado também são incorporados ao documento (Figura 6.6).

```
<region id="_WAC__RskipIcon" left ="80%" top ="5%" height="15%" width="15%" />
<region id="_WAC__RskipImg" left ="70%" top ="15%" height="20%" width="20%" />
<descriptor id="_WAC__desc1" region="_WAC__RskipIcon" focusIndex="1" />
<descriptor id="_WAC__descImg" region="_WAC__RskipImg" focusIndex="1" />
```

Figura 6.6: Regiões e Descritores

Uma mídia com o ícone que identifica uma anotação *skip* é utilizada para informar ao usuário (interagente) que existe uma anotação de *skip* associada ao segmento em apresentação (Figura 6.7. Caso seja de interesse do usuário fazer um salto no segmento, ele poderá ser feito através de um botão do controle remoto.

```
<media id="_WAC__SkipImg" type="image/jpeg" src="/misc/wacStafs/skip.jpg" descriptor="_WAC__desc1"  >
<area id="_WAC__SkipImg" begin="0s" />
</media>
```

Figura 6.7: Mídias

É necessário adicionar uma âncora temporal em todas as medias em execução para marcar o estado do documento onde a anotação de *skip* começou. Para isso, o componente *State Machine Manager* é acionado para a obtenção do estado da apresentação naquele momento. Com as informações do estado as âncoras temporais necessárias são geradas e adicionadas ao documento. Neste caso, apenas uma âncora é adicionada ao vídeo principal (Figura 6.8).

Estando demarcado o início do trecho de *skip* a aplicação aguarda um comando do usuário informando o fim do segmento de vídeo (Figura 6.2). Quando o fim do trecho for sinalizado (através de um botão do controle remoto) um evento será enviado à classe que trata os eventos no

```
<area id="_WAC__area1" begin="10.91s" />
```

Figura 6.8: Âncoras

modo de edição, finalizando a adição das entidades necessárias para a consolidação da anotação de *skip*.

Assim, a aplicação *Skip* aciona o componente *gingacc-capturer* para registrar o *frame* associado ao momento da anotação (*screenshot*). Essa imagem será um novo elemento de mídia do documento multimídia (Figura 6.9.

```
<media id="_WAC__imageAdd2" src="/tmp/image1.png" descriptor="_WAC__descImg"  >
<area id="_WAC__imageAdd2" begin="0s" />
</media>
```

Figura 6.9: Mídias

Após isso a aplicação obtém o estado da apresentação através do componente *State Machine Manager* adicionando as âncoras necessárias para a demarcação do estado final. Novamente, neste caso, somente uma âncora é adicionada ao vídeo principal.

Por fim, são adicionado três elos (*links* - Figura 6.10) responsáveis por:

1. apresentar a existência de uma anotação de *skip* (através de um ícone) quando o documento chegar no estado que demarca o inicio da anotação de *skip*;
2. retirar a marcação (ícone de *skip*) ao fim do segmento;
3. realizar o salto da cena, posicionando a apresentação no instante final do segmento de *skip*, caso o usuário pressione o botão de acionamento da operação.

É importante ressaltar que nos elos adicionados, bem como nos demais elementos adicionados, temos o identificador iniciado pelo padrão _WAC_, sinalizando que este elemento refere-se a uma anotação realizada pelo usuário (interagente). Para elos (*links*), esta marcação especial no identificador também é utilizada para verificar a origem do elo, diferenciando os elos da emissora dos elos do usuário. Dependendo do modo de exibição, *links* do usuário podem não ser disparados, caso desejado.

Continuando o cenário de uso da aplicação, temos uma nova ação desencadeada pelo usuário: é pressionado o botão de troca de modo e o componente *Enhanced Input Manager* configura o sistema para o modo padrão de exibição da aplicação *NCL*. O usuário agora interage com a aplicação acionando os botões coloridos que fazem as outras imagens de mídia do documento aparecerem na tela.

```
<link id="_WAC__linkS1" xconnector="_WAC__connOnStateStart" >
<bind component="video" role="onBegin" interface="_WAC__area1" />
<bind component="_WAC__SkipImg" role="start" interface="_WAC__SkipImg" />
<bind component="_WAC__imageAdd2" role="start" interface="_WAC__imageAdd2" />
</link>

<link id="_WAC__linkS2" xconnector="_WAC__connOnStateStop" >
<bind component="video" role="onBegin" interface="_WAC__area2" />
<bind component="_WAC__SkipImg" role="stop" interface="_WAC__SkipImg" />
<bind component="_WAC__imageAdd2" role="stop" interface="_WAC__imageAdd2" />
</link>

<link id="_WAC__linkS3" xconnector="_WAC__connSetState1" >
<bind component="_WAC__SkipImg" role="onSelection" interface="_WAC__SkipImg" />
<bind component="video" role="abort" interface="video" />
<bind component="imagem1" role="abort" interface="imagem1" />
<bind component="imagem2" role="abort" interface="imagem2" />
<bind component="imagem3" role="abort" interface="imagem3" />
<bind component="imagem4" role="abort" interface="imagem4" />
<bind component="_WAC__SkipImg" role="abort" interface="_WAC__SkipImg" />
<bind component="_WAC__imageAdd2" role="abort" interface="_WAC__imageAdd2" />
<bind component="video" role="start" interface="_WAC__area2" />
</link>
```

Figura 6.10: Elos

A seguir o interagente decide realizar outra anotação de *skip*. Volta para o modo de anotação e realiza os mesmos procedimentos anteriormente descritos. Novamente são adicionadas âncoras temporais para representar o estado do documento no começo e no fim do segmento do vídeo, além de um nó de mídia para representar a cena são adicionados os três elos necessários.

Ao final da anotação, o usuário volta para o modo de exibição e termina de assistir o programa.

Graças ao componente *StreamRecorder*, o vídeo principal poderia ser salvo no terminal de acesso, permitindo ao cliente assistir novamente a apresentação *NCL* contendo as suas anotações.

Ao visualizar novamente o documento (agora anotado) o usuário é direcionado ao *modo de exibição de conteúdo anotado*. Desta forma é possível a interação do usuário com o conteúdo anotado, ao invés de interagir com a aplicação original. Caso um mesmo botão esteja associado a duas ações distintas como exibir uma propaganda e acionar a marcação de *skip* problemas poderiam ocorrer. Entretanto por estar no modo de visualização de anotações a interação será conduzida com as anotações feitas no documento. Ações indesejadas (associadas com a interação original do programa) não seriam disparadas a menos que o usuário alterne para o *modo de interação*. Esta situação não acontece no exemplo tratado nesta seção.

Durante a reapresentação do programa, o componente *Formatter* identifica que o estado que denota o começo da primeira aplicação de *skip* foi atingido. Antes de disparar o elo (*link*) que inicia a exibição do ícone de *skip* e do *screenshot* da cena para onde o programa deve ser reposicionado é verificado o padrão do identificador do elo. Caso este tenha o prefixo **_WAC_** o link só deve ser disparado caso o *modo de exibição de anotações* esteja ativo. Estando neste

modo o elo é disparado e o ícone e a imagem da cena de *skip* são mostrados ao usuário.

O usuário tem então a opção de seleção do botão de *skip*. Caso isto aconteça, outro elo é acionado, reposicionando a apresentação no estado que representa o fim do *skip*. O Apêndice A.2 apresenta o código fonte final do documento, após a edição do usuário.

6.3 Reúso de documentos

O funcionamento da arquitetura é apresentado de forma detalhada na seção anterior em que é ilustrado o cenário de anotação sobre o conteúdo hipoteticamente enviado por uma emissora.

Em um cenário real em que deseja-se preservar os direitos autoriais sobre as anotações, isto é, separar o conteúdo gerado como resultado das anotações (de autoria do usuário - interagente) do conteúdo enviado pela emissora é desejável que o documento resultante seja armazenado de forma independente, permitindo a sua livre distribuição (a critério do usuário) [Pimentel et al. 2008].

Os recursos de reúso da linguagem *NCL* [Neto e Soares 2009] apresentam-se como uma solução para esse problema. A arquitetura proposta contempla essa possibilidade. Limitações motivadas por restrições relacionadas a direitos autorais não foram avaliadas na estruturação desta proposta. Fornecemos, entretanto, recursos para que o processo de edição utilize os recursos da linguagem, mantendo o conteúdo da emissora e do usuário armazenados de forma independente. A opção de manutenção do sincronismo, através da combinação de múltiplos documento é permitida.

Esta seção apresenta a estrutura de um documento *NCL* que carrega anotações feitas por um usuário mas que possui relações (inclusive temporais) com um documento produzido por terceiros.

A Figura 6.11 apresenta um documento *NCL* referente a um jogo de futebol teoricamente transmitido por uma emissora. Trata-se de um programa interativo que permite a opção de exibição das estatísticas do jogo (gols, posse de bola, cartões amarelos, etc). Essa opção de interatividade é disponibilizada ao usuário através do botão vermelho do controle remoto (linhas 21 a 32 da Figura 6.11). Quando o usuário pressiona o botão vermelho uma imagem com as estatísticas do jogo é exibida na tela. Quando o usuário pressiona o botão novamente essa imagem é removida da tela.

Suponha que exista uma aplicação da categoria *watch-and-comment* em um terminal de acesso de TV Digital, que incorpore a arquitetura que propomos neste trabalho, e que per-

```
01<ncl id="program">
02 <head>
03 <regionBase>
04 <region id="rgScreen" width="100%" height="100%">
05 <region id="rgGame" width="100%" height="100%"/>
06 <region id="rgStats" left="30%" width="40%" height="30%"/>
07 </region>
08 </regionBase>
09 <descriptorBase>
10 <descriptor id="dGame" region="rgGame"/>
11 <descriptor id="dStats" region="rgStats"/>
12 </descriptorBase>
13 <connectorBase>
14 <importBase alias="connBase" documentURI="soccer.conn"/>
15 </connectorBase>
16 </head>
17 <body id="body">
18 <port id ="pInicio" component="game"/>
19 <media id="game" src="game.mov" descriptor="dGame"/>
20 <media id="stats" src="stats.png" descriptor="dStats"/>
21 <link id="linkStatsOn" xconnector="connBase#OnKeySelectionStart">
22 <bind component="game" role="onKeySelection">
23 <bindParam name="keyCode" value="RED"/>
24 </bind>
25 <bind component="stats" role="start"/>
26       </link>
27 <link id="linkStatsOff" xconnector="connBase#OnKeySelectionStop">
28 <bind component="stats" role="onKeySelection">
29 <bindParam name="keyCode" value="RED"/>
30 </bind>
31 <bind component="stats" role="stop"/>
32       </link>
33 </body>
34</ncl>
```

Figura 6.11: Aplicação original (enviada pela emissora)

mita operações de anotação como as proporcionadas pela aplicação *Bookmark*. Através dela o usuário poderia selecionar duas cenas do jogo de futebol como cenas favoritas (por exemplo cenas de gol ou lances polêmicos). Essas anotações poderiam gerar um documento similar ao apresentado na Figura 6.12.

Observe que um novo documento foi produzido integralmente. Nesse documento constam, exclusivamente, anotações do usuário. As referências ao documento original são feitas através dos seguintes elementos:

- declaração no cabeçalho da importação do documento (linhas 18 a 20 da Figura 6.12);
- declaração do documento importado como um contexto do novo documento (linha 25 da Figura 6.12). Desta forma o documento original é reusado no contexto das anotações como um contexto *NCL*;
- declaração de uma porta que referencia o contexto (linha 23 da Figura 6.12) importado.

Para simplificar a descrição estamos referenciando a mesma base de conectores nos dois documentos. Bases de conectores distintas poderiam ser usadas. Operações mais complexas que demandariam um sincronismo mais rico também poderiam ser exploradas. O objetivo desta seção é descrever a possibilidade de manter anotações e documento original armazenados de maneira distinta, por isso não foram exploradas opções de sincronização mais sofisticadas entre

```
01<ncl id="annotation">
02 <head>
03 <regionBase>
04 <region id="rgScene" left="10%" top="10%" width="80%" height="80%" zIndex="2"/>
05 <region id="rgScene1" left="10%" top="10%" width="200" height="50" zIndex="3"/>
06 <region id="rgScene2" left="10%" top="20%" width="200" height="50" zIndex="3"/>
07 <region id="rgLogo" right="10%" bottom="10%" width="30" height="30" zIndex="3"/>
08 </regionBase>
09 <descriptorBase>
10 <descriptor id="dScene" region="rgScene"/>
11 <descriptor id="dScene1" region="rgScene1" focusIndex="1" moveDown="2" />
12 <descriptor id="dScene2" region="rgScene2" focusIndex="2" moveUp="1" />
13 <descriptor id="dLogo" region="rgLogo"/>
14 </descriptorBase>
15 <connectorBase>
16 <importBase alias="connBase" documentURI="soccer.conn"/>
17    </connectorBase>
18 <importedDocumentBase>
19 <importNCL alias="import" documentURI="program.ncl"/>
20 </importedDocumentBase>
21 </head>
22 <body id="annot">
23 <port id ="pInicio" component="importado"/>
24 <port id ="pLogo" component="logo"/>
25 <context id="importado" refer="import#body"/>
26 <media id="logo" src="logo.png" descriptor="dLogo"/>
27 <media id="recorded" src="recorded.mov" descriptor="dScene">
28 <area id="a1" begin="10s" end="13s"/>
29 <area id="a2" begin="40s" end="45s"/>
30 </media>
31 <media id="scene1" src="scene1.png" descriptor="dScene1"/>
32 <media id="scene2" src="scene2.png" descriptor="dScene2"/>
33 <link id="showScene1" xconnector="connBase#OnSelectionStopStart">
34 <bind component="scene1" role="onSelection"/>
35 <bind component="recorded" interface="a1" role="start"/>
36 <bind component="scene1" role="stop"/>
37 <bind component="scene2" role="stop"/>
38 </link>
39 <link id="showScene2" xconnector="connBase#OnSelectionStopStart">
40 <bind component="scene2" role="onSelection"/>
41 <bind component="recorded" interface="a2" role="start"/>
42 <bind component="scene1" role="stop"/>
43 <bind component="scene2" role="stop"/>
44 </link>
45 <link id="showMenu" xconnector="connBase#OnKeySelectionStart">
46 <bind component="logo" role="onKeySelection">
47 <bindParam name="keyCode" value="BLUE"/>
48 </bind>
49 <bind component="scene1" role="start"/>
50 <bind component="scene2" role="start"/>
51     </link>
52 <link id="stopMenu" xconnector="connBase#OnKeySelectionStopPause">
53 <bind component="scene1" role="onKeySelection">
54 <bindParam name="keyCode" value="BLUE"/>
55 </bind>
56 <bind component="scene1" role="stop"/>
57 <bind component="scene2" role="stop"/>
58     </link>
59 </body>
60</ncl>
```

Figura 6.12: Anotações realizadas pelo usuário (interagente)

elementos dos documentos. Detalhes de implementação utilizados considerando a arquitetura proposta não são objeto desta seção, por terem sido descritos detalhadamente anteriormente.

7 *Considerações Finais*

Neste trabalho cumprimos um ciclo que incluiu desde a investigação do domínio, passando pela modelagem e implementação de uma arquitetura que suporte os conceitos de *watch-and-comment* também discutidos neste trabalho. O paradigma foi substancialmente investigado no decorrer deste trabalho.

Aplicações desenvolvidas como objeto de validação da proposta mostraram-se importantes para a avaliação do uso dos diversos componentes da arquitetura. Essas aplicações ajudaram no refinamento da proposta em um processo iterativo em que novos elementos, que não foram identificados na fase de análise, puderam ser incorporados à proposta no decorrer do desenvolvimento.

As principais contribuições deste trabalho referem-se à investigação de conceitos de autoria direcionada ao usuário final (interagente); investigação do uso de linha de produto de *software* no domínio de ferramentas de autoria; definição de uma arquitetura computacional para suporte a estas atividades e avaliação da linguagem *NCL* e *middleware* Ginga para este fim.

7.1 Desdobramentos

O desenvolvimento deste trabalho teve como resultado, além desta dissertação de mestrado e publicações outros desdobramentos:

1. Projeto de pesquisa *i2TVD*: Projeto financiado pela *CAPES* visa a formação de recursos humanos na área de TV Digital. Fruto de uma parceria estabelecida entre *UFSCar*, *USP* e *FUCAPI* o projeto, tem entre seus objetivos, a investigação de técnicas que permitam a autoria ubíqua de documentos multimídia e a captura, automática, de informações de contexto associadas ao conteúdo para apoio a aplicações avançadas que envolvam trabalho colaborativo.

2. Projeto Ginga FrEvo & GingaRAP: Projeto financiado pela RNP que objetiva a evolução

do *middleware Ginga*. Este trabalho integra uma parte do projeto, denominada *GingaWaC*. A arquitetura proposta neste trabalho e os componentes desenvolvidos, como prova de conceito, são entregáveis do projeto.

3. Criação de um arcabouço (a nível de investigação de conceitos e protótipos) para o desenvolvimento de outros trabalhos como os apresentados em [Freitas e Teixeira 2009, Pimentel et al. 2009].

7.2 Trabalhos Futuros

Ao fim do trabalho identificamos oportunidades para a adição de funcionalidades que permitam um processo de autoria ubíqua de forma colaborativa. O uso de redes *peer-to-peer* pode ser um caminho para isso.

Experimentações em ambientes públicos, em que uma tela grande com a mesma informação é apresentada a todos e cada usuário interage de forma individual através de dispositivos móveis, colaborando para a construção do conteúdo também mostram-se pertinentes de serem realizadas. A utilização do conhecimento coletivo para a construção do conhecimento pode ser aplicado em diversos cenários, produzindo uma grande quantidade de resultados. Essas experimentações poderiam ser enriquecidas com o uso de dispositivos mais sofisticados como telas sensíveis a toque e sensores de movimento.

A incorporação na arquitetura de dispositivos que permitam anotação sobre conteúdo não-declarativo é um trabalho que poderia ser conduzido no futuro. Neste trabalho optamos por uma abordagem declarativa, mas isso não invalida a possibilidade de anotação sobre conteúdo não declarativo (aplicações em Java, por exemplo). Uma alternativa poderia ser a disponibilização de *APIs* nestas linguagens, para que o autor da aplicação a ser anotada forneça tais funcionalidades.

Investigações de como utilizar o conteúdo produzido através das anotações do usuário para a extração de conhecimento podem produzir resultados importantes. Quando consideramos milhares de usuários interagindo com o conteúdo e adicionando uma marca de *skip* em um segmento ou uma marca de *bookmark*, por exemplo, podemos identificar alguns comportamentos. Uma ação dessa natureza poderia sinalizar, por exemplo, que determinado trecho de um programa é desinteressante para uma parcela da população. Isso pode orientar o trabalho das produtoras em produções futuras. Esse tipo de informação é obtido atualmente através de métricas simples como audiência em determinado horário e canal. Entretanto a informação obtida através das anotações possui um nível de refinamento maior que

pode ser de grande valor. Realizamos investigações preliminares do uso dessas informações [Baladrón et al. 2008, Teixeira et al. 2009] e identificamos um potencial de pesquisa nesse quesito.

Integração com ferramentas de autoria mais completas, permitindo que a autoria realizada de forma ubíqua possa ser complementada e refinada com a utilização de outros recursos das ferramentas (como as visões temporal, espacial, textual e de *layout*). Estudamos uma integração com a ferramenta *Composer*, ampliando as possibilidades de desenvolvimento de *plugins* para a ferramenta. Essa integração, embora não consolidada, está em fase de análise.

Estudos mais aprofundados na área de Engenharia de Software referentes à obtenção de métricas sobre o grau de reúso dos componentes da arquitetura são oportunos. Pela escassez de ferramentas do domínio optamos, no escopo deste trabalho, identificar alguns *core assets* de forma empírica. Uma avaliação mais formal, através de métricas para cálculo do reúso, poderia induzir à supressão de alguns componentes ou adição de novos componentes à arquitetura.

Referências Bibliográficas

[ABNT 2007]ABNT. *NBR 15603*: Televisão digital terrestre — multiplexação e serviços de informação (si). [S.l.], nov. 2007.

[ABNT 2007]ABNT. *NBR 15606-2*: Ginga-ncl para receptores fixos e móveis – linguagem de aplicação xml para codificação de aplicações. [S.l.], nov. 2007.

[Abowd, Mynatt e Rodden 2002]ABOWD, G. D.; MYNATT, E. D.; RODDEN, T. The human experience. *IEEE pervasive computing*, IEEE Computer Society, p. 48–57, 2002.

[Baladrón et al. 2008]BALADRÓN, C. et al. Integrating User-Generated Content and Pervasive Communications. *IEEE Pervasive Computing*, v. 7, n. 4, p. 58–61, 2008.

[Berners-Lee et al. 1994]BERNERS-LEE, T. et al. The world-wide web. *Commun. ACM*, ACM, New York, NY, USA, v. 37, n. 8, p. 76–82, 1994. ISSN 0001-0782.

[Bulterman et al. 2005]BULTERMAN, D. et al. Synchronized Multimedia Integration Language (SMIL 2.1). *http://www.w3.org/TR/2005/REC-SMIL2-20051213/*, 2005.

[Bulterman et al. 1998]BULTERMAN, D. et al. GRiNS: A GRaphical INterface for creating and playing SMIL documents. *Computer Networks and ISDN systems*, Elsevier, v. 30, n. 1-7, p. 519–529, 1998.

[Casanova et al. 1991]CASANOVA, M. et al. The nested context model for hyperdocuments. In: ACM NEW YORK, NY, USA. *Proceedings of the third annual ACM conference on Hypertext*. [S.l.], 1991. p. 193–201.

[Cattelan et al. 2008]CATTELAN, R. G. et al. Watch-and-comment as a paradigm toward ubiquitous interactive video editing. *ACM Trans. Multimedia Comput. Commun. Appl.*, ACM, v. 4, n. 4, p. 1–24, 2008. ISSN 1551-6857.

[Cesar et al. 2008]CESAR, P. et al. Enhancing social sharing of videos: fragment, annotate, enrich, and share. ACM New York, NY, USA, 2008.

[Cesar, Bulterman e Jansen 2006]CESAR, P.; BULTERMAN, D.; JANSEN, A. The ambulant annotator: empowering viewer-side enrichment of multimedia content. In: ACM. *Proceedings of the 2006 ACM symposium on Document engineering*. [S.l.], 2006. p. 187.

[César et al. 2007]CÉSAR, P. et al. An architecture for non-intrusive user interfaces for interactive digital television. In: *EuroITV'07 (LNCS 4471)*. [S.l.: s.n.], 2007. p. 11–20.

[Clements e Northrop 2001]CLEMENTS, P.; NORTHROP, L. *Software product lines*. [S.l.]: Addison-Wesley Reading MA, 2001. ISBN 0-201-70332-7.

[Cohen 2002]COHEN, S. *Product line state of the practice report*. [S.l.]: CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2002.

[Costa, Moreno e Soares 2009]COSTA, R.; MORENO, M.; SOARES, L. Ginga-ncl: Suporte a múltiplos dispositivos. In: *Webmedia'09 - XV Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2009.

[Costa e Soares 2007]COSTA, R.; SOARES, L. Modelo Temporal Hipermídia para Suporte a Apresentação em Ambientes Interativos. *XIII Simpósio Brasileiro de Sistemas Multimídia e Web–WebMedia 2007*, 2007.

[Costa et al. 2006]COSTA, R. M. d. R. et al. Live editing of hypermedia documents. In: *ACM DocEng'06 – Proceedings of the 2006 ACM Symposium on Document Engineering*. [S.l.]: ACM Press, 2006. p. 165–172. ISBN 1-59593-515-0.

[Executable 2006]EXECUTABLE, M. G.-G. *A Guide to Platform Harmonisation*. 2006.

[Freitas e Teixeira 2009]FREITAS, G. B. de; TEIXEIRA, C. A. C. Ubiquitous services in home networks offered through digital tv. In: SHIN, S. Y.; OSSOWSKI, S. (Ed.). *SAC*. [S.l.]: ACM, 2009. p. 1834–1838. ISBN 978-1-60558-166-8.

[Guimarães, Costa e Soares 2007]GUIMARÃES, R.; COSTA, R.; SOARES, L. Composer: Ambiente de Autoria de Aplicações Declarativas para TV Digital Interativa. In: *Webmedia'07 - XIII Brazilian Symposium On Multimedia And The Web*. [S.l.: s.n.], 2007.

[Moock 2007]MOOCK, C. *Essential actionscript 3.0*. [S.l.]: O'Reilly, 2007.

[Motti et al. 2009]MOTTI, V. et al. Collaborative synchronous video annotation via the watch-and-comment paradigm. In: ACM NEW YORK, NY, USA. *Proceedings of the seventh european conference on European interactive television conference*. [S.l.], 2009. p. 67–76.

[Neto e Soares 2009]NETO, C.; SOARES, L. Reúso e importação em nested context language. In: *Webmedia'09 - XV Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2009.

[Northrop 2002]NORTHROP, L. SEI's software product line tenets. *IEEE software*, IEEE Computer Society, p. 32–40, 2002.

[OReilly]OREILLY, T. What is Web 2.0: Design patterns and business models for the next generation of software.

[Pimentel et al. 2010]PIMENTEL, M. et al. End-user live editing of iTV programmes. *International Journal of Advanced Media and Communication*, Inderscience, v. 4, n. 1, p. 78–103, 2010.

[Pimentel et al. 2007]PIMENTEL, M. et al. Enhancing multimodal annotations with pen-based information. In: IEEE COMPUTER SOCIETY. *Proceedings of the Ninth IEEE International Symposium on Multimedia Workshops*. [S.l.], 2007. p. 207–213.

[Pimentel et al. 2008]PIMENTEL, M. G. et al. End-user editing of interactive multimedia documents. In: *ACM DocEng'08 – Proceedings of the 2008 ACM Symposium on Document Engineering*. [S.l.: s.n.], 2008. p. 298–301.

[Pimentel et al. 2008]PIMENTEL, M. G. et al. Ubiquitous end-user live editing of interactive multimedia programs. In: *Webmedia'08 - XIV Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2008. p. 123–129.

[Pimentel et al. 2009]PIMENTEL, M. G. C. et al. Watch-and-Comment as an Approach to Collaboratively Annotate Points of Interest in Video and Interactive-TV Programs. *Mobile TV: Customizing Content and Experience*, Springer, p. 349, 2009.

[Pimentel et al. 2007]PIMENTEL, M. G. P. et al. Enhancing multimodal annotations with pen-based information. In: . [S.l.: s.n.], 2007. v. 2, p. 207–212.

[SBTVD 2006]SBTVD. Brazilian Digital TV System Reference Model. *http://sbtvd.cpqd.com.br*, 2006.

[Soares, Rodrigues e F. 2007]SOARES, L. F. G.; RODRIGUES, R. F.; F., M. M. Ginga-ncl: The declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society*, v. 12, n. 4, p. 37–46, 2007.

[Teixeira et al. 2009]TEIXEIRA, C. et al. User-media interaction with interactive TV. In: ACM. *Proceedings of the 2009 ACM symposium on Applied Computing*. [S.l.], 2009. p. 1829–1833.

APÊNDICE A – Código NCL *de conteúdo anotado*

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<ncl id="state_save" xmlns="http://www.ncl.org.br/NCL3.0/profileName">
<head>

<regionBase device="systemScreen(0)" >
    <region id="rgTV" left ="0" top ="0" height="100%" width="100%" >
        <region id="rgImagem1" left ="0%" top ="80%" height="20%" width="20%" />
        <region id="rgImagem2" left ="20%" top ="80%" height="20%" width="20%" />
        <region id="rgImagem3" left ="40%" top ="80%" height="20%" width="20%" />
        <region id="rgImagem4" left ="60%" top ="80%" height="20%" width="20%" />
        <region id="rgImagem5" left ="60%" top ="0%" height="20%" width="20%" />
        <region id="rgVideoPrincipal" left ="20" top ="20" height="80%" width="80%" />
    </region>
</regionBase>

<descriptorBase>
    <descriptor id="dVideo" region="rgVideoPrincipal" >
        <descriptorParam name="soundLevel" value="1" />
    </descriptor>
    <descriptor id="dImagem1" region="rgImagem1" />
    <descriptor id="dImagem2" region="rgImagem2" />
    <descriptor id="dImagem3" region="rgImagem3" />
    <descriptor id="dImagem4" region="rgImagem4" />
    <descriptor id="dRetornar" region="rgImagem5" focusIndex="1" focusBorderWidth="−2" />
</descriptorBase>

<connectorBase>
    <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin" eventType="presentation" transition="starts" />
        <simpleAction role="start" eventType="presentation" qualifier="seq" />
    </causalConnector>

    <causalConnector id="onKeySelectionStartN">
```

```
        <connectorParam name="keyCode" />
        <simpleCondition role="onSelection" eventType="selection" transition="stops" key="\$keyCode" />
        <simpleAction role="start" eventType="presentation" max="unbounded" qualifier="seq" />
    </causalConnector>
</connectorBase>

</head>

<body>
    <port id="pInicio" component="video" interface="video" />

    <media id="video" type="video/x-msvideo" src="/misc/ncl30/live/media/video1.avi" descriptor="dVideo" />

    <media id="imagem1" type="image/png" src="/misc/ncl30/live/media/amostra1.png" descriptor="
        dImagem1" />

    <media id="imagem2" type="image/png" src="/misc/ncl30/live/media/amostra2.png" descriptor="
        dImagem2" />

    <media id="imagem3" type="image/png" src="/misc/ncl30/live/media/amostra3.png" descriptor="
        dImagem3" />

    <media id="imagem4" type="image/png" src="/misc/ncl30/live/media/amostra3.png" descriptor="
        dImagem4" />

    <link id="lSelectGreenVideo" xconnector="onKeySelectionStartN" >
        <bind component="video" role="onSelection" interface="video" >
        <bindParam name="keyCode" value="GREEN" /></bind>
        <bind component="imagem2" role="start" interface="imagem2" />
    </link>

    <link id="lSelectRedVideo" xconnector="onKeySelectionStartN" >
        <bind component="video" role="onSelection" interface="video" >
        <bindParam name="keyCode" value="RED" /></bind>
        <bind component="imagem1" role="start" interface="imagem1" />
    </link>

    <link id="lSelectBlueVideo" xconnector="onKeySelectionStartN" >
        <bind component="video" role="onSelection" interface="video" >
        <bindParam name="keyCode" value="BLUE" /></bind>
        <bind component="imagem4" role="start" interface="imagem4" />
    </link>
```

```
    <link id="lSelectYellowVideo" xconnector="onKeySelectionStartN" >
        <bind component="video" role="onSelection" interface="video" >
        <bindParam name="keyCode" value="YELLOW" /></bind>
        <bind component="imagem3" role="start" interface="imagem3" />
    </link>

</body>

</ncl>
```

Listagem A.1: Documento NCL enviado pela emissora

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<ncl id="state_save" xmlns="http://www.ncl.org.br/NCL3.0/profileName">
<head>

<regionBase device="systemScreen(0)" >
    <region id="_WAC__RskipIcon" left ="80%" top ="5%" height="15%" width="15%" />
    <region id="_WAC__RskipImg" left ="70%" top ="15%" height="20%" width="20%" />
    <region id="rgTV" left ="0" top ="0" height="100%" width="100%" >
    <region id="rgImagem1" left ="0%" top ="80%" height="20%" width="20%" />
    <region id="rgImagem2" left ="20%" top ="80%" height="20%" width="20%" />
    <region id="rgImagem3" left ="40%" top ="80%" height="20%" width="20%" />
    <region id="rgImagem4" left ="60%" top ="80%" height="20%" width="20%" />
    <region id="rgImagem5" left ="60%" top ="0%" height="20%" width="20%" />
        <region id="rgVideoPrincipal" left ="20" top ="20" height="80%" width="80%" />
    </region>
</regionBase>

<descriptorBase>
    <descriptor id="dVideo" region="rgVideoPrincipal" >
        <descriptorParam name="soundLevel" value="1" />
    </descriptor>
    <descriptor id="dImagem1" region="rgImagem1" />
    <descriptor id="dImagem2" region="rgImagem2" />
    <descriptor id="dImagem3" region="rgImagem3" />
    <descriptor id="dImagem4" region="rgImagem4" />
    <descriptor id="dRetornar" region="rgImagem5" focusIndex="1" focusBorderWidth="−2" />
    <descriptor id="_WAC__desc1" region="_WAC__RskipIcon" focusIndex="1" />
    <descriptor id="_WAC__descImg" region="_WAC__RskipImg" focusIndex="1" />
</descriptorBase>

<connectorBase>
```

```
    <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin" eventType="presentation" transition="starts" />
        <simpleAction role="start" eventType="presentation" qualifier="seq" />
    </causalConnector>

    <causalConnector id="onKeySelectionStartN">
        <connectorParam name="keyCode" />
        <simpleCondition role="onSelection" eventType="selection" transition="stops" key="\$keyCode" />
        <simpleAction role="start" eventType="presentation" max="unbounded" qualifier="seq" />
    </causalConnector>

    <causalConnector id="_WAC__connSetState1">
        <simpleCondition role="onSelection" eventType="selection" transition="stops" />
        <compoundAction operator="seq" >
            <simpleAction role="abort" eventType="presentation" max="unbounded" qualifier="seq" />
            <simpleAction role="start" eventType="presentation" max="unbounded" qualifier="seq" />
        </compoundAction>
    </causalConnector>

    <causalConnector id="_WAC__connOnStateStart">
        <compoundCondition operator="and" >
            <simpleCondition role="onBegin" eventType="presentation" transition="starts" max="unbounded"
                />
        </compoundCondition>
        <compoundAction operator="seq" >
            <simpleAction role="start" eventType="presentation" max="unbounded" qualifier="par" />
        </compoundAction>
    </causalConnector>

    <causalConnector id="_WAC__connOnStateStop">
        <compoundCondition operator="and" >
            <simpleCondition role="onBegin" eventType="presentation" transition="starts" max="unbounded"
                />
        </compoundCondition>
        <compoundAction operator="seq" >
            <simpleAction role="stop" eventType="presentation" max="unbounded" qualifier="par" />
        </compoundAction>
    </causalConnector>
</connectorBase>

</head>

<body>
```

```
<port id="pInicio" component="video" interface="video" />

<media id="video" type="video/x−msvideo" src="/misc/ncl30/live/media/video1.avi" descriptor="dVideo"
    >
    <area id="video" begin="0s" />
    <area id="_WAC__area1" begin="10.91s" />
    <area id="_WAC__area2" begin="21.554s" />
    <area id="_WAC__area3" begin="39.806s" />
    <area id="_WAC__area4" begin="49.149s" />
</media>

<media id="imagem1" type="image/png" src="/misc/ncl30/live/media/amostra1.png" descriptor="
    dImagem1" >
    <area id="imagem1" begin="0s" />
    <area id="_WAC__area3" begin="15.314s" />
    <area id="_WAC__area4" begin="24.641s" />
</media>

<media id="imagem2" type="image/png" src="/misc/ncl30/live/media/amostra2.png" descriptor="
    dImagem2" >
    <area id="imagem2" begin="0s" />
    <area id="_WAC__area3" begin="14.736s" />
    <area id="_WAC__area4" begin="24.063s" />
</media>

<media id="imagem3" type="image/png" src="/misc/ncl30/live/media/amostra3.png" descriptor="
    dImagem3" >
    <area id="imagem3" begin="0s" />
    <area id="_WAC__area3" begin="14.254s" />
    <area id="_WAC__area4" begin="23.581s" />
</media>

<media id="imagem4" type="image/png" src="/misc/ncl30/live/media/amostra3.png" descriptor="
    dImagem4" >
    <area id="imagem4" begin="0s" />
    <area id="_WAC__area3" begin="13.658s" />
    <area id="_WAC__area4" begin="22.986s" />
</media>

<media id="_WAC__SkipImg" type="image/jpeg" src="/misc/wacStafs/skip.jpg" descriptor="
    _WAC__desc1" >
    <area id="_WAC__SkipImg" begin="0s" />
</media>
```

```
<media id="_WAC__imageAdd2" src="/tmp/image1.png" descriptor="_WAC__descImg" >
    <area id="_WAC__imageAdd2" begin="0s" />
</media>

<media id="_WAC__imageAdd3" src="/tmp/image2.png" descriptor="_WAC__descImg" >
    <area id="_WAC__imageAdd3" begin="0s" />
</media>

<link id="lSelectGreenVideo" xconnector="onKeySelectionStartN" >
    <bind component="video" role="onSelection" interface="video" >
    <bindParam name="keyCode" value="GREEN" /></bind>
    <bind component="imagem2" role="start" interface="imagem2" />
</link>

<link id="lSelectRedVideo" xconnector="onKeySelectionStartN" >
    <bind component="video" role="onSelection" interface="video" >
    <bindParam name="keyCode" value="RED" /></bind>
    <bind component="imagem1" role="start" interface="imagem1" />
</link>

<link id="lSelectBlueVideo" xconnector="onKeySelectionStartN" >
    <bind component="video" role="onSelection" interface="video" >
    <bindParam name="keyCode" value="BLUE" /></bind>
    <bind component="imagem4" role="start" interface="imagem4" />
</link>

<link id="lSelectYellowVideo" xconnector="onKeySelectionStartN" >
    <bind component="video" role="onSelection" interface="video" >
    <bindParam name="keyCode" value="YELLOW" /></bind>
    <bind component="imagem3" role="start" interface="imagem3" />
</link>

<link id="_WAC__linkS1" xconnector="_WAC__connOnStateStart" >
    <bind component="video" role="onBegin" interface="_WAC__area1" />
    <bind component="_WAC__SkipImg" role="start" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd2" role="start" interface="_WAC__imageAdd2" />
</link>

<link id="_WAC__linkS2" xconnector="_WAC__connOnStateStop" >
    <bind component="video" role="onBegin" interface="_WAC__area2" />
    <bind component="_WAC__SkipImg" role="stop" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd2" role="stop" interface="_WAC__imageAdd2" />
```

```
</link>

<link id="_WAC__linkS3" xconnector="_WAC__connSetState1" >
    <bind component="_WAC__SkipImg" role="onSelection" interface="_WAC__SkipImg" />
    <bind component="video" role="abort" interface="video" />
    <bind component="imagem1" role="abort" interface="imagem1" />
    <bind component="imagem2" role="abort" interface="imagem2" />
    <bind component="imagem3" role="abort" interface="imagem3" />
    <bind component="imagem4" role="abort" interface="imagem4" />
    <bind component="_WAC__SkipImg" role="abort" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd2" role="abort" interface="_WAC__imageAdd2" />
    <bind component="video" role="start" interface="_WAC__area2" />
</link>

<link id="_WAC__linkS4" xconnector="_WAC__connOnStateStart" >
    <bind component="video" role="onBegin" interface="_WAC__area3" />
    <bind component="imagem1" role="onBegin" interface="_WAC__area3" />
    <bind component="imagem2" role="onBegin" interface="_WAC__area3" />
    <bind component="imagem3" role="onBegin" interface="_WAC__area3" />
    <bind component="imagem4" role="onBegin" interface="_WAC__area3" />
    <bind component="_WAC__SkipImg" role="start" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd3" role="start" interface="_WAC__imageAdd3" />
</link>

<link id="_WAC__linkS5" xconnector="_WAC__connOnStateStop" >
    <bind component="video" role="onBegin" interface="_WAC__area4" />
    <bind component="imagem1" role="onBegin" interface="_WAC__area4" />
    <bind component="imagem2" role="onBegin" interface="_WAC__area4" />
    <bind component="imagem3" role="onBegin" interface="_WAC__area4" />
    <bind component="imagem4" role="onBegin" interface="_WAC__area4" />
    <bind component="_WAC__SkipImg" role="stop" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd3" role="stop" interface="_WAC__imageAdd3" />
</link>

<link id="_WAC__linkS6" xconnector="_WAC__connSetState1" >
    <bind component="_WAC__SkipImg" role="onSelection" interface="_WAC__SkipImg" />
    <bind component="video" role="abort" interface="video" />
    <bind component="imagem1" role="abort" interface="imagem1" />
    <bind component="imagem2" role="abort" interface="imagem2" />
    <bind component="imagem3" role="abort" interface="imagem3" />
    <bind component="imagem4" role="abort" interface="imagem4" />
    <bind component="_WAC__SkipImg" role="abort" interface="_WAC__SkipImg" />
    <bind component="_WAC__imageAdd2" role="abort" interface="_WAC__imageAdd2" />
```

```
        <bind component="_WAC__imageAdd3" role="abort" interface="_WAC__imageAdd3" />
        <bind component="video" role="start" interface="_WAC__area4" />
        <bind component="imagem1" role="start" interface="_WAC__area4" />
        <bind component="imagem2" role="start" interface="_WAC__area4" />
        <bind component="imagem3" role="start" interface="_WAC__area4" />
        <bind component="imagem4" role="start" interface="_WAC__area4" />
    </link>

</body>
</ncl>
```

Listagem A.2: Documento Final