UNIVERSIDADE ESTADUAL DO CEARÁ

TALES PAIVA NOGUEIRA

UML-TV:
UM PERFIL UML PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA TV DIGITAL

FORTALEZA – CEARÁ
2010

TALES PAIVA NOGUEIRA

UML-TV:
UM PERFIL UML PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA TV DIGITAL

Dissertação apresentada ao programa de Mestrado Acadêmico em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software

Orientadora: Profª. Drª. Mariela Inés Cortés

Co-Orientador: Prof. Dr. Cidcley Teixeira de Souza

FORTALEZA – CEARÁ
2010

N778u Nogueira, Tales P.

UML-TV: um perfil UML para suporte ao desenvolvimento de aplicações para TV digital/ Tales Paiva Nogueira. __ Fortaleza, 2010.

115p. ; 2il.

Orientadora: Profª. Drª. Mariela Inés Cortés.

Dissertação (Mestrado Acadêmico em Ciência da Computação) – Universidade Estadual do Ceará, Centro de Ciência e Tecnologia.

1. Engenharia de Software. 2. Engenharia de Software Orientada a Modelos. 3. Televisão Digital. 4. *UML Profiles*. 5. *Domain Specific Languages*. I. Universidade Estadual do Ceará, Centro de Ciência e Tecnologia.

CDD: 001.6

TALES PAIVA NOGUEIRA

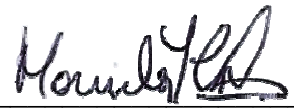
UML-TV:
UM PERFIL UML PARA SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES PARA TV DIGITAL

Dissertação submetida ao programa de Mestrado Acadêmico em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

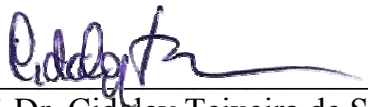
Área de concentração: Engenharia de Software

Aprovada em: 06/09/2010.

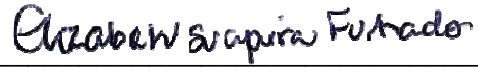
BANCA EXAMINADORA



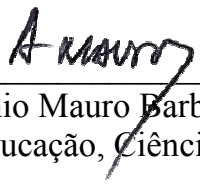
Profª. Drª. Mariela Inés Cortés
Universidade Estadual do Ceará – UECE



Prof. Dr. Cidcley Teixeira de Souza
Instituto Federal de Educação, Ciência e Tecnologia – IFCE



Profª. Drª. Maria Elizabeth Sucupira Furtado
Universidade Estadual do Ceará – UECE



Prof. Dr. Antônio Mauro Barbosa de Oliveira
Instituto Federal de Educação, Ciência e Tecnologia – IFCE

Aos meus pais, Hélio e Joice

AGRADECIMENTOS

À minha Família, em especial à minha Mãe e minhas irmãs, pelo constante incentivo à minha formação profissional, muito obrigado por tudo. Ao meu Pai, que mesmo não estando mais presente entre nós para comemorar este feito, estará sempre presente na minha memória.

À minha namorada, Josy, que aguardou pelo fim deste trabalho mais do que ninguém, agradeço pelo apoio e preocupação constantes.

Agradeço à professora Mariela, minha orientadora, pela confiança e paciência depositados nos momentos de maior dificuldade durante a elaboração deste trabalho, além das ótimas revisões do texto.

Ao professor Cidcley, meu co-orientador, agradeço pela idealização do tema, pela confiança e pela ajuda nos momentos de dúvida.

Aos integrantes do Núcleo Avançado em Engenharia de Software Distribuído e Sistemas Hipermídia (NASH) do Instituto Federal de Educação, Ciência e Tecnologia (IFCE), em especial ao Lailson, por todo o esforço desprendido na implementação das transformações de modelos que foram essenciais para a validação deste trabalho.

Ao Mestrado Acadêmico em Ciência da Computação (MACC) da UECE, desejo ainda mais sucesso e crescimento. Deixo meu agradecimento também a todos os professores e funcionários.

Ao Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) da UFC, pela flexibilização dos horários de trabalho sempre que necessário e pelo incentivo ao aprimoramento profissional e acadêmico.

A todos os colegas de mestrado, em especial ao Mário, Robson e Enyo pelos momentos de aprendizado e descontração vividos durante o curso.

O futuro não é um lugar onde estamos indo, mas um lugar que estamos criando. O caminho para ele não é encontrado, mas construído, e o ato de fazê-lo muda tanto o realizador quanto o destino.

Antoine de Saint-Exupéry

RESUMO

O desenvolvimento de aplicações hipermídia para televisão digital vem tomando grande importância no cenário mundial há alguns anos. No Brasil, o crescimento no interesse pelo desenvolvimento desse tipo de aplicação encontra-se em grande expansão na medida em que as emissoras de televisão iniciam suas transmissões em formato digital, delineando o marco de uma nova era para um dos mais importantes meios de comunicação do planeta.

No ambiente de desenvolvimento de aplicações hipermídia, pouco tem sido feito para auxiliar os desenvolvedores a construir aplicações com mais qualidade e rapidez, sem deixar de lado a atividade de documentação do software. Nesse sentido, este trabalho propõe a aplicação de conceitos de Engenharia de Software Orientada a Modelos tendo como domínio as aplicações para televisão digital.

Os principais aspectos de Engenharia de Software Orientada a Modelos utilizados neste trabalho foram o da criação de um modelo independente de plataforma através da criação de um perfil UML chamado UML-TV, que especializa diagramas de Máquinas de Estados e de Atividades com estereótipos, valores rotulados e restrições especificadas em OCL com o objetivo de modelar aplicações para televisão digital que envolvam sincronismo de mídias e interação do usuário através do controle remoto. Além disso, foram definidas regras de transformação para que o processo de desenvolvimento de aplicações hipermídia pudesse ser automatizado.

Como resultado, foi desenvolvida uma linguagem visual específica de domínio que auxilia o desenvolvimento de aplicações para televisão digital e um conjunto de regras de transformação que podem ser aplicadas para a geração automática de documentos NCL.

Palavras-chave: Engenharia de software orientada a modelos, Televisão digital, Perfil UML.

ABSTRACT

The development of hypermedia applications for digital television has been taking great importance on the world scene in the past few years. In Brazil, the growth in interest in the development of such type of applications is booming to the extent that the television stations begin broadcasting in digital format, outlining the boundary of a new era for one of the most important media in the planet.

In the field of development of hypermedia applications, little has been done to help developers building applications with better quality and faster without setting aside the software documentation activity. In this sense, this work proposes the application of concepts of Model Driven Software Engineering in the development of digital TV applications.

The main aspects of Model Driven Software Engineering used in this work were the creation of a platform independent model by creating a UML profile called UML-TV, which specializes State Machine and Activity diagrams with stereotypes, tagged values and OCL constraints aiming at modeling digital television applications that comprise media synchronization and user interaction through the remote control. Moreover, transformation rules were defined so the process of developing hypermedia applications could be automated.

As a result, a new domain-specific visual language that helps the development of applications for digital TV was developed and a set of transformation rules that can be applied to the automatic generation of NCL documents was defined.

Keywords: Model driven software engineering, Digital television, UML profile.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ACAP	Advanced Common Application Platform
ATL	ATLAS Transformation Language
ARIB	Association of Radio Industries and Bussinesses
ATSC	Advanced Television Systems Committee
CASE	Computer Aided Software Engineering
CIM	Computation Independent Model
BML	Broadcast Markup Language
DSL	Domain Specific Language
DSVL	Domain Specific Visual Language
DVB	Digital Video Broadcasting
EPG	Electronic Programming Guide
GEM	Globally Executable MHP
GPL	General Purpose Language
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
ISDB	Integrated Services Digital Broadcasting
ISDB-Tb	Integrated Services Digital Broadcast – Terrestrial, Brazilian version
M2C	Model to code
JVM	Java Virtual Machine
MDA	Model Driven Architecture
MHP	Multimedia Home Platform
MDSE	Model Driven Software Engineering
MOF	MetaObject Facility
NCL	Nested Context Language
NCM	Nested Context Model
OCL	Object Constraint Language
OMG	Object Management Group

OMMMA	Object-oriented Modeling of MultiMedia Applications
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
SBTVD	Sistema Brasileiro de Televisão Digital
SMIL	Synchronized Multimedia Integration Language
UML	Unified Modeling Language
URD	Unidade Receptora Decodificadora
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

SUMÁRIO

1. INTRODUÇÃO

A digitalização do sinal de televisão constituiu-se como um grande avanço da tecnologia de telecomunicações no mundo pelo fato da grande abrangência desse meio de comunicação. No Brasil, onde aproximadamente 94,6% das residências possuem pelo menos um aparelho de televisão (ALENCAR, 2009), o impacto dessa nova tecnologia também é muito grande, pois é de conhecimento geral que a televisão é uma das principais fontes de informação e entretenimento da população.

Dentre as novas possibilidades trazidas pela digitalização do sinal televisivo, podem ser citados serviços independentes da programação televisiva, i.e. aplicações que não estejam ligadas diretamente a um programa em exibição, as quais adicionam outras possibilidades de uso para a televisão além de informação e entretenimento. Essas aplicações só eram passíveis de utilização através de outros meios de comunicação como, por exemplo, aplicações de *internet banking* e correio eletrônico (*e-mail*), acessíveis através de computadores ou telefones.

Com o advento da televisão digital, o aparelho de TV deixa de ser um canal unidirecional por onde o telespectador apenas recebe aquilo que é transmitido pelas emissoras via *broadcast*. Ao invés disso, o antigo telespectador – agora chamado de usuário – pode interagir com a programação de diversas maneiras, dependendo dos recursos disponíveis no seu equipamento de recepção e na emissora do conteúdo digital. Essa interação pode ser classificada em três níveis: interatividade local, intermitente e permanente (OLIVEIRA; DE SOUZA, 2005). Os dois últimos níveis são condicionados à existência de um canal de retorno, cuja função é tornar o fluxo de informações, antes unidirecional, em bidirecional. Por exemplo, uma enquete que usualmente poderia ser respondida somente via rede telefônica, agora pode ser respondida instantaneamente através do controle remoto de um equipamento ligado a um canal de retorno.

Indo na direção contrária à tendência mundial de adoção de um dos três padrões vigentes de televisão digital (o americano ATSC – *Advanced Television Systems Commitee*, o europeu DVB – *Digital Video Broadcasting* e o japonês ISDB – *Integrated Services Digital Broadcasting*), o governo brasileiro iniciou um projeto que envolveu diversas instituições de pesquisa e desenvolvimento com o intuito de conceber um padrão de televisão digital que incorporasse tecnologia nacional, até então chamado de SBTVD (Sistema Brasileiro de Televisão Digital), através do Decreto nº 4901 de 26 de novembro de 2003 (República Federativa do Brasil, 2003).

Participaram desse projeto 22 consórcios formados por universidades, instituições de pesquisa e empresas privadas. Os consórcios foram divididos em temas, a saber: Camada de Interatividade; Codificação de Sinais Fonte; *Middleware*; Serviços, Aplicações e Conteúdo; Transmissão e Recepção, Codificação de Canal e Modulação; e Camada de Transporte (Ministério das Comunicações, 2003).

Com a definição do modelo brasileiro de televisão digital nos últimos anos e o início das primeiras transmissões digitais, é notável a baixa demanda pelos serviços oferecidos, que atualmente limitam-se à transmissão de parte da programação de algumas emissoras em alta definição e o surgimento das primeiras aplicações interativas. No entanto, espera-se que haja um interesse maior por parte dos produtores de conteúdo (emissoras) no desenvolvimento de aplicações interativas compatíveis com o padrão utilizado no sistema brasileiro de TV digital, hoje conhecido como ISDB-Tb (*Integrated Services Digital Broadcasting – Terrestrial, Brazillian version*) (Fórum do Sistema Brasileiro de TV Digital Terrestre, 2009). Esse interesse dos produtores de conteúdo pode se tornar ainda maior com a diminuição dos preços das URDs e a implementação de todas as funcionalidades previstas no ISDB-Tb, como a multiprogramação (possibilidade de transmitir mais um programa em um mesmo canal de radiofrequência), e recursos de interatividade.

As aplicações hipermídia podem ser dividas em dois grupos, de acordo com o paradigma de programação utilizado: aplicações imperativas e aplicações declarativas. As aplicações do primeiro tipo se caracterizam por serem escritas em

linguagens de programação tais como Java (GOSLING et al., 2005), C++ (STROUSTRUP, 1986), etc. Já as aplicações do segundo tipo são construídas através do uso de linguagens de marcação, como HTML (*HyperText Markup Language*) (BERNERS-LEE; CONNOLLY, 1995), SMIL (*Synchronized Multimedia Integration Language*) (World Wide Web Consortium, 2005), NCL (*Nested Context Language*) (ANTONACCI; SOARES, 2000), etc. Há, ainda, a possibilidade de utilizar os dois tipos de linguagem em uma mesma aplicação hipermídia; esse é o caso, por exemplo, de aplicações declarativas escritas em NCL que podem ter como objetos de mídia um ou mais NCLets (implementados em *Xlets*[1]) ou NCLuas (implementados na linguagem Lua) (SANT'ANNA; CERQUEIRA; SOARES, 2008).

O *middleware*[2] presente na unidade receptora decodificadora (URD – também conhecido como *set-top-box*) dos padrões de televisão digital de todo o mundo suportam aplicações desenvolvidas em ambos os tipos de linguagens. No caso do *middleware* do sistema brasileiro, chamado Ginga (BARBOSA; SOARES, 2008), o mesmo é composto por duas partes: Ginga-NCL (Telemídia, 2009) e Ginga-J (DE SOUZA FILHO; LEITE; BATISTA, 2007) que são especializados em processar, respectivamente, aplicações declarativas em NCL e aplicações imperativas em Java.

1.1. Problema

Um requisito muito importante para uma aplicação hipermídia diz respeito ao sincronismo espaçotemporal entre as mídias envolvidas. É imprescindível que as mídias sejam iniciadas e interrompidas no momento correto (propriedade temporal) e que sejam exibidas nas regiões adequadas da tela da televisão (propriedade espacial). Portanto, assegurar que esses aspectos sejam respeitados é um fator crítico para o sucesso da aplicação e devem ser priorizados durante seu desenvolvimento.

---

[1] Programas escritos em Java específicos para televisão digital. O nome *Xlet* é derivado da principal interface a ser implementada em aplicações desse tipo.

[2] Camada intermediária que possibilita a interoperabilidade entre o hardware e o software.

Tomando como exemplo a linguagem NCL, para criar uma aplicação que possa ser executada no Ginga-NCL, i.e. um documento NCL, o desenvolvedor pode criar o documento manualmente através de um editor de textos qualquer ou com a ajuda de um *plug-in* para o IDE (*Integrated Development Environment*) Eclipse chamado NCL Eclipse (AZEVEDO, 2008). Outra opção é utilizar a ferramenta Composer (GUIMARÃES; COSTA; SOARES, 2008), a qual oferece diversas perspectivas de visualização do documento criado (textual, estrutural, temporal e *layout*).

Além do indispensável conhecimento prévio sobre a linguagem NCL, que demanda certo tempo independentemente da metodologia a ser seguida, a primeira alternativa não se mostra atrativa pelo grande trabalho manual envolvido, mesmo quando auxiliada por uma ferramenta de apoio integrada a uma IDE. A segunda trata-se de uma ferramenta que teve seu desenvolvimento descontinuado e possui instabilidades conhecidas (Telemídia, 2009).

Nesse contexto, pode-se inferir que novas ferramentas que auxiliem a criação de aplicações voltadas para a televisão digital podem ser propostas de forma que novos paradigmas sejam utilizados na tentativa de melhorar o processo de produção das mesmas. Um paradigma que pode ser utilizado para esse fim é o da Arquitetura Orientada a Modelos - MDA (*Model Driven Architecture*) (Object Management Group, 2010), cuja principal característica é tornar os modelos, e.g. um diagrama de classes UML – *Unified Modeling Language* (RUMBAUGH; JACOBSON; BOOCH, 2004), em artefatos de primeira classe no ciclo de desenvolvimento de software. O processo envolve transformações de modelos independentes de plataforma em modelos específicos de plataformas e, posteriormente, transformações desses modelos intermediários em código fonte.

Uma das formas de se aplicar os conceitos de MDA é através da especificação de perfis UML (FUENTES-FERNÁNDEZ; VALLECILLO-MORENO, 2004), que permitem adaptar a UML a determinado domínio por meio da criação de estereótipos, valores rotulados e restrições, criando, assim, uma linguagem de domínio

específico (DSL – *Domain Specific Language*). Aliando essas técnicas, espera-se que sejam alcançados de forma satisfatória os objetivos deste trabalho.

1.2. Justificativa

Durante o processo de desenvolvimento de software de qualquer tipo, e.g. software embarcado, sistemas Web, aplicações *desktop,* etc., é comum que o código-fonte nem sempre reflita aquilo que foi especificado no início do projeto devido a mudanças nos requisitos decorrentes da evolução do software, requisições do cliente ou qualquer outro motivo. Esse problema não deixa de ocorrer em projetos de aplicações para televisão digital.

Com a aplicação das técnicas de Engenharia de Software Orientada a Modelos (MDSE[3] – *Model Driven Software Engineering*) à produção de aplicações para televisão digital, espera-se que a distância entre os modelos e o código-fonte seja diminuída. Desta forma, eventuais modificações em um dado projeto baseado nesse paradigma de desenvolvimento são realizadas primeiramente nos modelos de alto nível e depois propagadas até o código-fonte. Com isso, problemas de inconsistência entre os documentos de modelagem e o código podem ser minimizados. Outra vantagem dessa abordagem é a aproximação dos *experts* no domínio ao processo de desenvolvimento, uma vez que os modelos independentes de plataforma representam o domínio em um nível de abstração mais elevado, i.e. sem detalhes de implementação, o que torna seu entendimento mais fácil.

Além disso, outros tipos de aplicação, e.g. aplicações *desktop*, há muito tempo já contam com as vantagens advindas da MDSE através de ferramentas CASE (ORLIKOWSKI, 1993) – *Computer Aided Software Engineering*, e.g. Rational Rose[4], que possuem a funcionalidade de geração de código-fonte Java ou C++ a partir de diagramas UML. No entanto, a aplicação desse método de desenvolvimento de

[3] De acordo com Pires (2008), o termo MDSE vem sendo usado quando não se deseja restringir o tema às tecnologias, vocabulário e visão da OMG, a qual registrou as marcas MDA e MDD.

[4] www.ibm.com/software/rational

software no âmbito das aplicações televisivas ainda não foi muito explorada até o momento da escrita deste trabalho.

1.3. Objetivos

A partir da problemática apresentada, o presente trabalho visa contribuir no desenvolvimento de aplicações interativas para TV Digital que contenham sincronismo de mídias e interação via controle remoto objetivando os seguintes aspectos:

- Definição de um perfil UML que sirva à especificação de aplicações hipermídia, provendo uma linguagem visual de domínio específica que possa ser utilizada para a modelagem dessas aplicações tanto por *experts* do domínio de televisão digital quanto por projetistas de software. No contexto de MDA, o perfil aqui apresentado pode ser utilizado como base para a elaboração de modelos específicos de plataforma de forma que novas linguagens possam ser facilmente adaptadas e adicionadas ao processo de transformação.

- Apoio à geração automática de código-fonte de aplicações hipermídia utilizando o perfil criado como um modelo independente de plataforma, com o intuito de facilitar e tornar mais rápida a especificação e produção de tais conteúdos hipermídia através da formulação de regras de transformação genéricas escritas na linguagem MOFScript.

1.4. Metodologia

Inicialmente, foi obtido um entendimento teórico sobre os conceitos utilizados no decorrer do trabalho. Esses conceitos consistem em conhecimentos sobre desenvolvimento orientado a modelos, transformações de modelos, diagramas UML e

perfis UML, aplicações interativas para televisão digital e linguagens de domínio específico. As relações de sincronismo entre mídias foram levantadas para se ter uma estimativa dos elementos necessários para que essas relações fossem devidamente modeladas sem perda semântica no perfil proposto neste trabalho.

Algumas propostas de utilização de diagramas UML para a modelagem de exemplos de aplicações hipermídia foram realizadas. As aplicações modeladas foram escolhidas dentre as presentes na literatura sobre a área, e tiveram complexidade crescente na medida em que a pesquisa se desenvolveu. Além disso, aplicações de uso real foram modeladas a fim de provar a eficácia da linguagem de modelagem proposta. Dessa forma, pode-se validar a proposta inicial do trabalho.

Ao final da pesquisa sobre os diagramas UML, os diagramas de Máquinas de Estados e de Atividades foram escolhidos para compor os modelos. Depois, estereótipos e valores rotulados foram definidos e finalmente as restrições em OCL (Object Management Group, 2009) – *Object Contstraint Language* – foram especificadas formando, assim, o perfil UML-TV. A partir de modelos desenhados de acordo com as aplicações de exemplo utilizadas, foram criadas regras genéricas de transformação na linguagem MOFScript para que fosse possível gerar aplicações automaticamente tendo como entrada a representação textual dos modelos no formato XMI (Object Management Group, 2007) – *XML Metadata Interchange* – e, como saída, documentos NCL.

1.5. Estrutura da dissertação

O restante deste trabalho está organizado da seguinte maneira: o Capítulo 2 consiste no referencial teórico necessário para o entendimento dos conceitos principais que envolvem este trabalho. Este referencial envolve o estado da arte das aplicações hipermídia para televisão digital e das linguagens de programação voltadas para este domínio, do desenvolvimento orientado a modelos, das linguagens de domínio específico e da UML com ênfase nos seus mecanismos de extensão, os perfis UML.

No Capítulo 3, alguns trabalhos relacionados são discutidos. O Capítulo 4 contém a descrição do perfil UML-TV, uma das contribuições deste trabalho. No Capítulo 5, as regras genéricas de transformação implementadas para a geração de código fonte são mostradas. No Capítulo 6, um estudo de caso com uma aplicação mais próxima de um ambiente real é apresentado. Por fim, são realizadas considerações finais sobre o trabalho e são descritos os trabalhos futuros que podem dar continuidade ao que foi produzido durante a escrita desta dissertação no Capítulo 7.

2. REFERENCIAL TEÓRICO

Nas seções seguintes, os conceitos que embasam essa dissertação são apresentados. Inicialmente, o domínio das aplicações hipermídia para televisão digital é explorado, incluindo definições das principais linguagens que podem ser utilizadas para a produção de tais aplicações no padrão brasileiro. Em seguida, o paradigma da MDSE é explicado com tendo como base o padrão MDA. Depois, as características das linguagens de domínio específico são mostradas e, por fim, a linguagem de modelagem utilizada no trabalho, a UML, é descrita com ênfase nos seus mecanismos de extensão, os perfis UML.

2.1. Aplicações Hipermídia para Televisão Digital

Antes de apresentar o domínio das aplicações hipermídia para televisão digital, é importante que se faça uma distinção entre dois conceitos intimamente ligados a esse domínio: hipermídia e multimídia. Essa definição é útil porque existem diversos trabalhos na literatura que se aplicam a sistemas multimídia, mas não diretamente a sistemas hipermídia.

A primeira utilização do termo multimídia é atribuída ao artista Bobb Goldsteinn em 1966 (ALBARINO, 1966). Goldsteinn o utilizou para definir o seu trabalho *LightWorks* que envolvia apresentações de músicas sincronizadas manualmente com vídeos e imagens. Apenas nos anos 90 o termo tomou sua conotação atual: múltiplas mídias englobando texto, imagem, vídeo, animações, áudio e aplicações interativas (BADII et al., 2009).

O termo hipermídia foi usado pela primeira vez por Nelson (1965) no artigo intitulado "*Complex information processing: a file structure for the complex, the changing and the indeterminate*". No mesmo artigo, o autor cita o conceito de hipertexto definindo-o como um conjunto de materiais escritos ou imagens interconectados de forma a tornar impossível sua reprodução em papel. Similarmente, Nelson define hipermídia argumentando que sons e vídeos também podem ser agrupados como sistemas não lineares que naturalmente devem ser controlados por um sistema computadorizado.

Com a evolução dos sistemas e das mídias, temos hoje a definição de hipermídia como a união de duas tecnologias de processamento de informações: hipertexto e multimídia. Hipermídia é baseada em objetos e tempo. Os objetos podem ser ligados aos seus metadados ou a outros objetos, e o conteúdo multimídia é especificado em uma estrutura de nós e *links* orientados ao tempo que podem ser automaticamente ativados para apresentar o conteúdo (SRIVASTAVA, 2002).

Atualmente, hipertexto é apenas uma das tecnologias que podem ser utilizadas para definir quais mídias estarão presentes em uma aplicação hipermídia, estabelecendo também como essas mídias se relacionarão. Existem várias linguagens que servem a esse propósito, como as linguagens de marcação HTML (padrão predominante para páginas Web), SMIL (a recomendação do W3C – *World Wide Web Consortium* – para descrição de apresentações multimídia), NCL (uma das linguagens suportadas pelo *middleware* da televisão digital brasileira), e as linguagens de programação Java e C++, por exemplo.

Um importante subgrupo das aplicações hipermídia é o das aplicações interativas. Segundo Kiousis (2002), a interatividade pode ser definida como o poder que uma tecnologia de comunicação tem de criar um ambiente mediado, no qual os participantes podem se comunicar (um para um, um para muitos e muitos para muitos) tanto sincronamente quanto assincronamente e participar de trocas de mensagens recíprocas. No domínio da televisão digital, a interatividade tem um papel

extremamente importante por proporcionar novas formas de utilização da televisão, e.g. utilização de serviços bancários, de correio eletrônico, votações, etc.

Aplicações voltadas para televisão digital que envolvem sincronismo de mídias são normalmente compostas por um fluxo de vídeo e um fluxo de áudio principais, onde o gerenciamento da sincronização temporal e espacial das mídias é geralmente orientado a eventos. Diferentemente de uma linha do tempo, o paradigma orientado a eventos se baseia no posicionamento espaço-temporal dos eventos, independente de quando (momento absoluto no tempo) a sincronização acontece. Em programas não lineares, alguns conteúdos são conhecidos *a priori*, e.g., o fim da apresentação de um segmento de mídia com início e duração conhecidos. No entanto, outros relacionamentos espaço-temporais entre objetos de mídia podem não ser estabelecidos previamente como, por exemplo, no caso de interações do usuário. Esses relacionamentos entre objetos de mídia devem ser controlados durante a apresentação a fim de garantir que os mesmos sejam respeitados.

Eventos de apresentação e seleção (no tempo) podem ser definidos por objetos de mídia ou agentes externos, e.g., um controle remoto com botões alfanuméricos, teclas direcionais, botões de aumento e diminuição de volume, botões de mudança de canais e botões especiais como atalhos para o menu principal, para o EPG (*Electronic Programming Guide*), além dos botões de contextuais coloridos (normalmente nas cores vermelha, verde, amarela e azul). Além disso, pode-se prover ao usuário a possibilidade de interromper a aplicação e retomá-la em determinado momento no futuro. Também é possível, por exemplo, que uma aplicação interativa disponibilize ao usuário a escolha de um ângulo de visão específico que ele deseje selecionar para assistir ao conteúdo.

2.1.1. Linguagens de programação

Diversas linguagens de programação podem ser utilizadas para a construção de aplicações hipermídia para televisão digital. No entanto, algumas têm sido mais amplamente adotadas do que outras por diversos motivos. Altos índices de

popularidade como é o caso da linguagem Java, suporte de *middlewares* como é o caso de NCL e Lua no padrão ISDB-Tb podem ser citados como as linguagens mais populares para esse domínio especifico.

A Tabela 1 contém a descrição das linguagens declarativas e imperativas utilizadas nos seus respectivos ambientes nos principais sistemas de televisão digital existentes. Nota-se a hegemonia da linguagem Java no que diz respeito ao ambiente das linguagens imperativas. Já no âmbito das linguagens declarativas, há certa diversidade de linguagens de marcação (XHTML (World Wide Web Consortium, 2002), BML – *Broadcast Markup Language* – (Association of Radio Industries and Bussinesses, 2007) e NCL) e domínio da linguagem de script ECMAScript (ECMA International, 2009), base do JavaScript. Nesta subseção, serão brevemente apresentadas as linguagens utilizadas no padrão brasileiro, o ISDB-Tb: NCL, Lua e Java.

Tabela 1 – Ambientes de aplicações dos sistemas de televisão digital. Adaptado de (BARBOSA; SOARES, 2008)

Sistema	Middleware	Ambiente declarativo	Ambiente imperativo
ATSC	ACAP	ACAP-X (XHTML/ECMAScript)	ACAP-J (Java)
DVB-T	MHP	DVB-HTML (XHTML/ECMAScript)	MHP (Java)
ISDB-T	ARIB-BML	ARIB-BML (BML/ECMAScript)	GEM (Java)
ISDB-Tb	Ginga	Ginga-NCL (NCL/Lua)	Ginga-J (Java)

2.1.1.1. NCL

NCL é uma linguagem de marcação que segue o padrão XML (*Extensible Markup Language*) (World Wide Web Consortium, 2010) e que possibilita a definição de objetos de mídia e o sincronismo espaço-temporal entre os mesmos, bem como contempla aspectos de interatividade, adaptabilidade, suporte a múltiplos dispositivos e suporte à produção ao vivo de programas interativos não-lineares.

A linguagem NCL é baseada no modelo NCM (*Nested Context Model*) (SOARES, 2000), o qual define estruturas chamadas de nós e elos que são responsáveis pela constituição das aplicações interativas que seguem esse modelo. As mídias são representadas por nós de contexto e as operações de sincronismo entre elas são representadas pelos elos.

Criada no ano 2000 (ANTONACCI, 2000), a linguagem NCL foi projetada levando em consideração algumas características importantes para uma linguagem de autoria de documentos hipermídia. Dentre essas características estão: estruturação lógica do documento, representação de objetos de mídia, características de apresentação separadas do componente, reuso, definição de elos no escopo de uma composição, relacionamentos complexos entre componentes, flexibilidade na especificação temporal, e adaptação à plataforma de exibição (ANTONACCI et al., 2000).

Uma característica que diferencia NCL de linguagens mais difundidas como HTML e SMIL é a especificação espacial dos elementos. Enquanto que em HTML e SMIL as informações sobre tamanho e posição dos objetos de mídia fazem parte das declarações dos próprios objetos de mídia, em NCL essa informação fica em partes diferentes do documento. Isso favorece o reuso interno dos objetos de mídia, pois é possível definir mais de um tamanho e posição para um mesmo objeto, por exemplo. O aspecto do reuso é um ponto onde NCL é mais avançada do que outras linguagens de autoria hipermídia uma vez que possibilita a reutilização de partes do documento no próprio documento atual, ou a inclusão de um ou mais documentos NCL em outro. No caso das outras linguagens o reuso é facilitado apenas pelo fato de que o conteúdo dos objetos de mídia é separado do documento hipermídia.

No padrão ISBD-Tb, o Ginga-NCL é a parte do *middleware* responsável pela reprodução de documentos NCL (SOARES; RODRIGUES; MORENO, 2007). O padrão brasileiro suporta dois perfis da linguagem: EDTVProfile (*Enhanced Digital TV Profile*) e BDTVProfile (*Basic Digital TV Profile*) (SOARES; RODRIGUES, 2006).

2.1.1.2. Lua

Lua é uma linguagem de script criada em 1993 que hoje é utilizada em várias aplicações, e.g. robótica, processamento de imagens, bioinformática, etc. Atualmente, Lua é a linguagem mais utilizada para *scripting* em jogos (IERUSALIMSCHY, 2009) e é suportada pelo *middleware* do padrão ISBD-Tb.

No contexto do padrão ISDB-Tb, Lua se integra à linguagem NCL através do conceito de objeto de mídia NCLua, constituído por um arquivo com extensão `.lua` utilizado no parâmetro `src` de um elemento `<media>` em um documento NCL. Para a integração ao ambiente televisivo, alguns módulos foram adicionados à linguagem Lua, a saber: `event` (permite a comunicação através de eventos), `canvas` (permite a criação de gráficos e imagens), `settings` (provê uma tabela com variáveis de ambiente e variáveis personalizadas), e `persistent` (provê uma tabela com variáveis persistentes) (SANT'ANNA; CERQUEIRA; SOARES, 2008).

Diferentemente de uma execução via linha de comando, onde o programador tem maior liberdade quanto à forma de desenvolver programas Lua, a utilização dessa linguagem em aplicações hipermídia deve seguir um modelo de execução orientado a eventos. Para isso, o módulo `event` é utilizado para ligar o *script* NCLua ao formatador[5] NCL.

2.1.1.3. Java

Os altos índices de adoção da linguagem Java, que já era popular antes da criação dos primeiros sistemas de televisão digital, foram definitivos para adoção da linguagem pelos principais *middlewares* do mercado. A própria arquitetura da linguagem Java, que permite a portabilidade de programas, também favoreceu sua adoção. Java permite que os desenvolvedores tenham um maior controle e flexibilidade com relação a aspectos visuais de suas aplicações, além de oferecer funcionalidades de segurança e escalabilidade (SRIVASTAVA, 2002). Essa

[5] Formatador NCL é a parte do *middleware* responsável por controlar a execução de documentos NCL e garantir que as relações de sincronismo sejam respeitadas

portabilidade dos programas escritos em Java é possível graças à presença da JVM (*Java Virtual Machine*) na arquitetura dos receptores de sinal digital.

Similarmente ao conceito de *MIDlets* (Oracle, 2010a) e *Applets* (Oracle, 2010b), que representam interfaces a serem implementadas para o desenvolvimento de aplicações Java para dispositivos móveis e para navegadores de Internet, respectivamente, a API JavaTV provê a interface *Xlet* de onde provem o nome dado a aplicações Java para o ambiente da televisão digital. Essa interface é responsável por gerenciar o ciclo de vida de uma aplicação (Figura 1), que inicia no estado *Loaded*, de onde só pode ser alterado para o estado *Paused*, a partir do qual pode passar para *Active* ou *Destroyed*. Do estado *Active*, o *Xlet* pode voltar para o estado *Paused* ou ser destruído, indo para o estado *Destroyed* de onde não pode sair para nenhum outro estado.

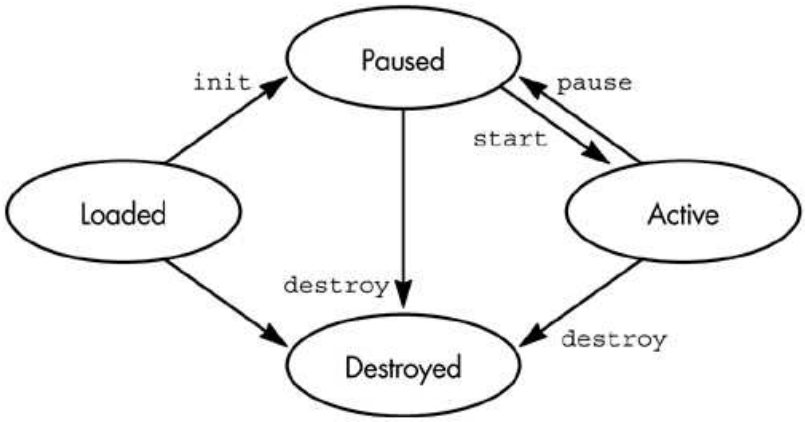


Figura 1 – Ciclo de vida de um Xlet (SCHWALB, 2003)

No contexto do *middleware* brasileiro, o Ginga-J inclui uma adaptação da API de acesso a informações de serviço do padrão ISDB (ARIB B.23), a implementação da especificação Java DTV, a qual inclui a API JavaTV, além de outras APIs de extensão (PAULINELLI; KULESZA, 2009). A especificação Java DTV é fruto de um acordo de colaboração entre o fórum brasileiro de televisão digital (SBTVD), o governo brasileiro, a *Sun Microsystems* e empresas do ramo televisivo com o intuito de especificar uma versão da API JavaTV livre de pagamentos de direitos autorais (*royalties*).

2.2. Engenharia de Software Orientada a Modelos

Nesta subseção, são abordados alguns temas que envolvem a Engenharia de Software Orientada a Modelos. Primeiramente, a Arquitetura Orientada a Modelos é descrita, seguida de uma explanação sobre Linguagens de Domínio Específico e uma revisão sobre UML e seu mecanismo de extensão, os perfis UML.

2.2.1. Arquitetura Orientada a Modelos

No desenvolvimento de software tradicional, um problema frequentemente enfrentado no início dos projetos diz respeito à pequena quantidade de código-fonte produzido. Esse aspecto, que pode ser considerado por alguns como um problema de baixa produtividade, é consequência de uma fase de exploração de fundamental importância para o sucesso do projeto, como um investimento a médio ou longo prazo. Assim, os primeiros artefatos de um projeto normalmente se caracterizam por modelos em sua maior parte representados por diagramas. No entanto, nos processos de desenvolvimento tradicionais, esses modelos geralmente perdem suas ligações com o código-fonte na medida em que o tempo passa e os modelos não são atualizados de forma a acompanhar a evolução dos sistemas.

Esse descompasso entre o que é modelado e o que é realmente implementado gera outro problema ainda mais grave durante a fase de manutenção e evolução. Isto acontece por que normalmente a equipe de manutenção não é a mesma equipe que participou do desenvolvimento do sistema. Dessa forma, há uma perda de tempo e uma diminuição de produtividade na tentativa de descobrir como o software foi desenvolvido, recriando modelos de alto nível, por exemplo.

Por outro lado, existem alguns fatores que desestimulam a adoção da prática de documentação como, por exemplo, a noção errada de alguns desenvolvedores de que só devem produzir código, uma vez que a documentação demanda muito tempo e somente será de utilidade na fase de manutenção ou evolução do sistema e não durante o desenvolvimento. Com isso, não existe uma motivação e sim a obrigação de produzi-la.

Outro fator importante a ser levado em conta durante o desenvolvimento de software é a necessidade de acompanhar o mercado e manter a compatibilidade com novos sistemas. Uma vez que os sistemas de software não vivem isolados, surge a necessidade de interoperabilidade de forma a propiciar o intercâmbio de informação mesmo entre sistemas originados a partir de diferentes tecnologias.

Todos os problemas anteriormente mencionados podem ser pelo menos parcialmente resolvidos através de uma maior atenção dada à arquitetura do sistema, que é a representação da estrutura do mesmo, em termos dos seus componentes (KLEPPE; WARMER; BAST, 2003). A arquitetura tem o papel de ajudar no entendimento do sistema, proporcionar o reuso de componentes arquiteturais, e pode ser atuante nas etapas de análise, construção e evolução do software bem como no gerenciamento do projeto.

A arquitetura de um sistema é normalmente representada por um ou mais modelos. Modelos provêem abstrações do sistema que ajudam a lidar com a complexidade das aplicações com maior facilidade, independentemente de como elas são implementadas, distribuídas e das plataformas ou tecnologias utilizadas.

Um dos padrões para desenvolvimento orientado a modelos é o desenvolvido pela OMG (*Object Management Group*), denominado MDA (*Model Driven Architecture*) (Object Management Group, 2010). Nesse padrão, os artefatos gerados ao longo do ciclo de vida do software são diferentes dos artefatos gerados durante ao longo do processo de desenvolvimento tradicional. Ao invés de documentos representados apenas como texto, modelos formais são introduzidos a partir da fase de análise tornando-os, assim, elementos de primeira classe no ciclo de vida do software.

Os modelos formais introduzidos pela MDA constituem-se no modelo independente de computação (CIM – *Computation Independent Model*), no modelo independente de plataforma (PIM – *Platform Independent Model*) e no modelo dependente de plataforma (PSM – *Platform Specific Model*). O CIM possui um maior nível de abstração, tendo como alvo o nível de negócio do sistema. O PIM descreve o sistema sem dar detalhes sobre a plataforma de execução. Já o PSM é específico de

uma tecnologia e pode ser o resultado da transformação automática de um ou mais PIMs.

Como benefício advindo da utilização de MDA, pode-se destacar uma mudança no foco do desenvolvimento, passando do código-fonte para o PIM. Como o PIM é menos complexo de se desenvolver, essa mudança implica em funcionalidades implementadas em menos tempo, melhorando a produtividade. A independência de plataforma do PIM e a possibilidade de transformá-lo em modelos correspondentes a plataformas específicas também trazem um grande avanço no aspecto da portabilidade do código para diferentes tecnologias. Existe ainda a possibilidade de criar relacionamentos chamados de pontes (*bridges*) entre PSMs derivados de um mesmo PIM. Com isso, é possível gerar pontes entre PSMs de forma que a interoperabilidade seja alcançada.

O modelo é uma representação do código, sendo o PIM o modelo de alto nível do sistema. Como as alterações são realizadas primeiramente no PIM, a possibilidade de inconsistências entre o modelo e o código pode ser reduzida, o que atenua os problemas de documentação e manutenção anteriormente mencionados.

Vale ressaltar, no entanto, que alguns "mitos" fazem parte do universo do desenvolvimento orientado a modelos. Alguns desses mitos, levantados por Lopes (2005), são os seguintes:

- *A independência total de plataforma*: Lopes cita como exemplo a diferença entre IDL-CORBA (SEETHARAMAN, 1998), que foi desenvolvido independente de linguagem de programação e sistema operacional e, portanto, pode ser considerado independente de plataforma, e UML, que pode ser considerado independente de plataforma quando utilizada para modelagem de negócios (como uma espécie de *middleware*), mas que é mais utilizada no paradigma da orientação a objetos.

- *A utilização de UML para modelar qualquer tipo de sistema*: UML não é ideal para a modelagem de certos sistemas. Outras linguagens podem ser desenvolvidas para apoiar a modelagem através de DSLs.

- *A geração automática e completa do código a partir dos modelos*: Como os modelos geralmente não definem a semântica e a lógica dos sistemas em um nível alto de detalhes, não é possível gerar todo o código de uma aplicação complexa com as ferramentas atuais.

- *Os problemas da abordagem MDA serão resolvidos com uma linguagem de transformação padronizada*: as transformações se constituem na parte mais trabalhosa do processo de MDA e ocupa uma grande parte das pesquisas na área de onde podem surgir novas propostas.

- *A utilização de modelos simplifica os sistemas complexos*: modelos ajudam a gerenciar a complexidade (tornar um sistema mais simples de ser entendido) porque divide aspectos do sistema em diferentes diagramas, permitindo a análise de aspectos específicos separadamente. No entanto, a complexidade do sistema continua sendo representada pela soma dos seus modelos.

2.2.2. Linguagens de Domínio Específico

Uma linguagem de domínio específico (DSL) ou linguagem visual de domínio específico (DSVL – *Domain Specific Visual Language*) define notações e estruturas adequadas a um domínio específico de uma aplicação. Com isso, ganhos em expressividade e facilidade de uso são obtidos, além de vantagens como custo de manutenção reduzido e aumento da produtividade (MERNIK; HEERING; SLOANE, 2005). Segundo Czarnecki (2005), algumas das vantagens que as DSLs podem proporcionar são:

- *Abstrações específicas do domínio*: uma DSL provê abstrações predefinidas para representar conceitos do domínio da aplicação;

- *Sintaxe concreta*: uma DSL oferece uma notação natural e intuitiva para um dado domínio e evita confusão sintática;

- *Verificação de erro específica do domínio*: uma DSL permite criar analisadores sintáticos que podem encontrar mais erros do que os analisadores normais e podem reportar esses erros em uma linguagem mais familiar ao usuário;

- *Otimizações específicas para o domínio*: uma DSL dá oportunidade de gerar código otimizado baseado no conhecimento sobre o domínio da aplicação, o que não é possível em um compilador comum;

- *Suporte de ferramentas específicas do domínio*: uma DSL pode melhorar o ambiente de desenvolvimento através da inclusão de editores, depuradores, etc. O conhecimento capturado por uma DSL pode ser utilizado para disponibilizar um suporte mais inteligente por parte das ferramentas aos desenvolvedores.

Em contraste com as DSLs estão as GPL – *General Purpose Language* (BANGIA, 2007) – que podem lidar com um leque maior de aplicações, mas com um custo de complexidade maior enquanto que uma DSL é capaz de lidar com apenas um ou com poucos aspectos de um sistema.

2.2.3. UML

Segundo Watson (2007), a história da modelagem gráfica de software pode ser dividida em dois períodos: antes da UML e depois da UML. Antes da criação dessa linguagem, o cenário da modelagem de software era repleto de notações incompatíveis, o que levava ao enfraquecimento do mercado de ferramentas, deixando os modelos fora do processo de desenvolvimento de software. No entanto, após a especificação da UML e sua grande adoção no processo de produção de software, as incompatibilidades de notação até então existentes foram praticamente eliminadas.

A UML, criada em 1996, hoje na sua versão 2.1.2 (Object Management Group, 2009b) é composta por 13 diagramas que definem a notação e semântica para os seguintes domínios:

- A interação do usuário ou modelo de casos de uso descreve os limites e interações entre o sistema e seus usuários, correspondendo a um modelo de requisitos, em alguns aspectos;
- O modelo de interação ou comunicação descreve como objetos do sistema interagem entre si;
- O modelo dinâmico ou de estados descreve os estados ou condições que as classes possuem ao longo do tempo. Gráficos de atividades descrevem o fluxo de trabalho que o sistema irá seguir;
- O modelo lógico ou de classes descreve as classes e objetos que fazem parte do sistema;
- O modelo físico de componentes descreve o software (e algumas vezes componentes de hardware) que fazem parte do sistema;
- O modelo físico de implantação descreve a arquitetura física e a implantação de componentes na arquitetura de hardware.

A UML foi desenvolvida seguindo uma abordagem de metamodelagem que adapta técnicas de especificação formal. Além disso, a UML segue um padrão arquitetural de quatro camadas (ver Figura 2) onde a camada M3 corresponde ao metamodelo[6] MOF (Object Management Group, 2006) – *MetaObject Facility* –, a camada M2 é a própria UML, a camada M1 são os modelos utilizados no processo de produção de software e a camada M0 são os objetos em si.

[6] Um metamodelo é um modelo que serve para definir os relacionamentos entre vários componentes de um modelo.

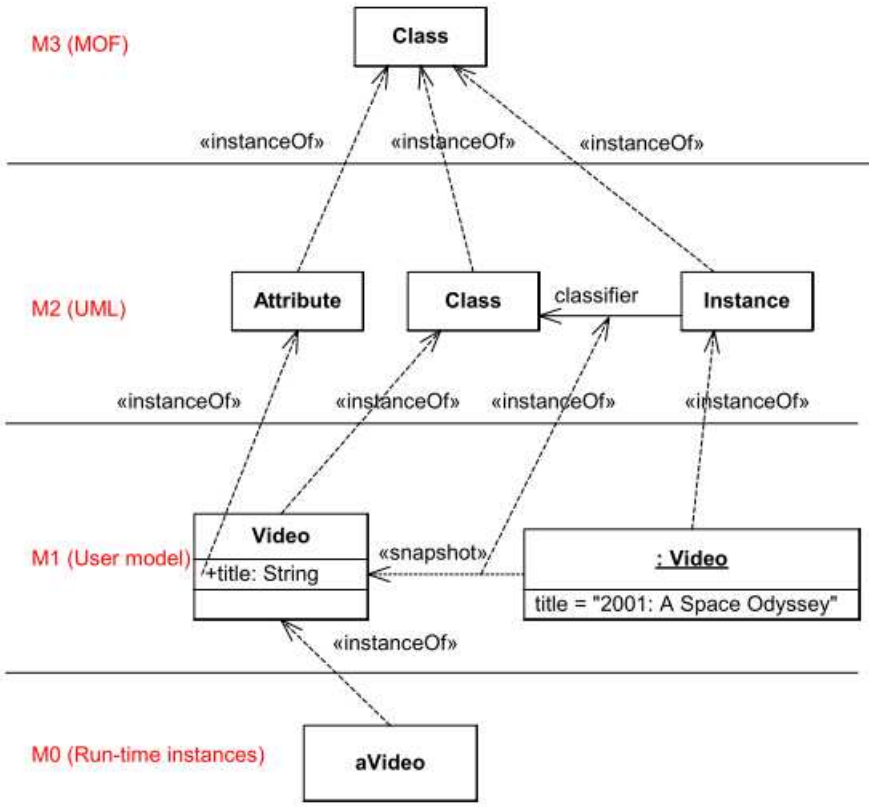


Figura 2 – Exemplo de hierarquia de quatro camadas (Object Management Group, 2009c)

O metamodelo é dividido em três pacotes principais (RUMBAUGH; JACOBSON; BOOCH, 2004), são eles: *foundation*, que define a estrutura estática, *behavioral elements*, que define a estrutura dinâmica, e *model management*, que define a estrutura organizacional dos modelos. Os dois primeiros pacotes contêm pacotes internos.

O pacote *foundation* tem os seguintes subpacotes (que também contêm outros subpacotes): *core*, *data types* e *extension mechanisms*. O pacote *core* descreve estruturas estáticas da UML como atributos, classificadores e operações, relacionamentos (generalização, associação e dependência) e metaclasses abstratas, e.g. *Element*. O pacote *data types* possui definições de classes de tipos de dados utilizados no metamodelo. O pacote *extension mechanisms* define os mecanismos de restrição, estereótipos e valores rotulados.

O pacote *behavior elements* contém os subpacotes *common behavior*, *collaborations*, *use cases* e *state machines*. O primeiro pacote tem definições de sinal, operação e ação. O segundo especifica colaboração, interação, mensagem, associação

e papel classificador. O terceiro descreve ator e caso de uso. E o quarto pacote descreve máquina de estado, incluindo estado, pseudoestados, eventos, sinais, transições e condições de guarda, além de estruturas de modelos de atividades.

O pacote *model management* não tem subpacotes e contém definições para pacote, modelo e subsistema, além de descrever visibilidade de espaços de nomes (*namespaces*) e pacotes. A Figura 3 mostra os pacotes que formam o metamodelo, seus respectivos subpacotes e seus relacionamentos.

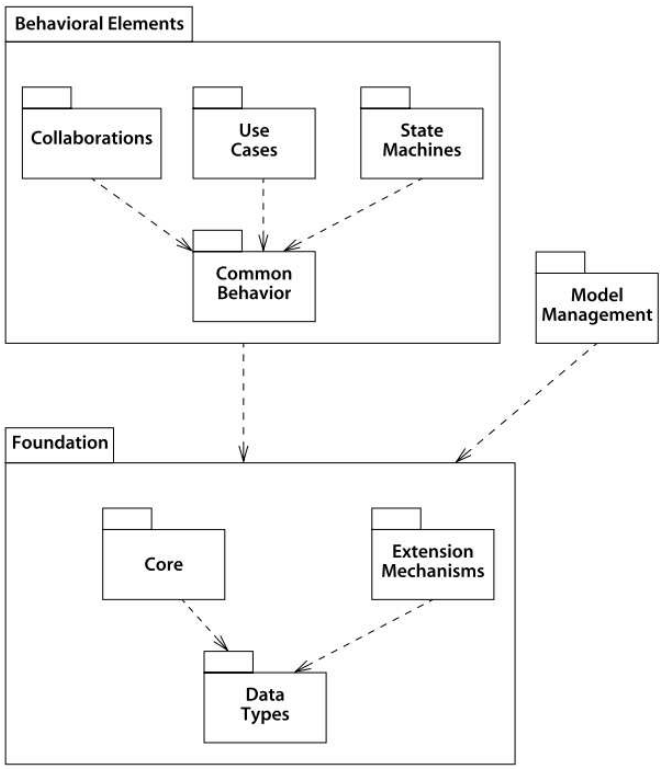


Figura 3 – Visão dos pacotes do metamodelo UML (RUMBAUGH; JACOBSON; BOOCH, 2004)

A UML conta com 13 diagramas em sua versão mais recente. Como pode ser visto na Figura 4, os diagramas podem ser classificados em diagramas estruturais e diagramas comportamentais, que têm uma subclassificação chamada de diagramas de interação (GUEDES, 2008).

Os diagramas estruturais constituem-se nos diagramas de Classes, de Objetos, de Componentes, de Implantação, de Pacotes, e de Estrutura Composta e podem ser utilizados para refletir os aspectos estáticos das aplicações.

Os diagramas comportamentais englobam os diagramas de Atividades, de Máquina de Estados, de Casos de Uso e o subgrupo dos diagramas de interação composto pelos diagramas de Comunicação, de Sequência, de Tempo, e de Visão Geral. Esses diagramas podem ser utilizados para modelar os aspectos dinâmicos das aplicações.

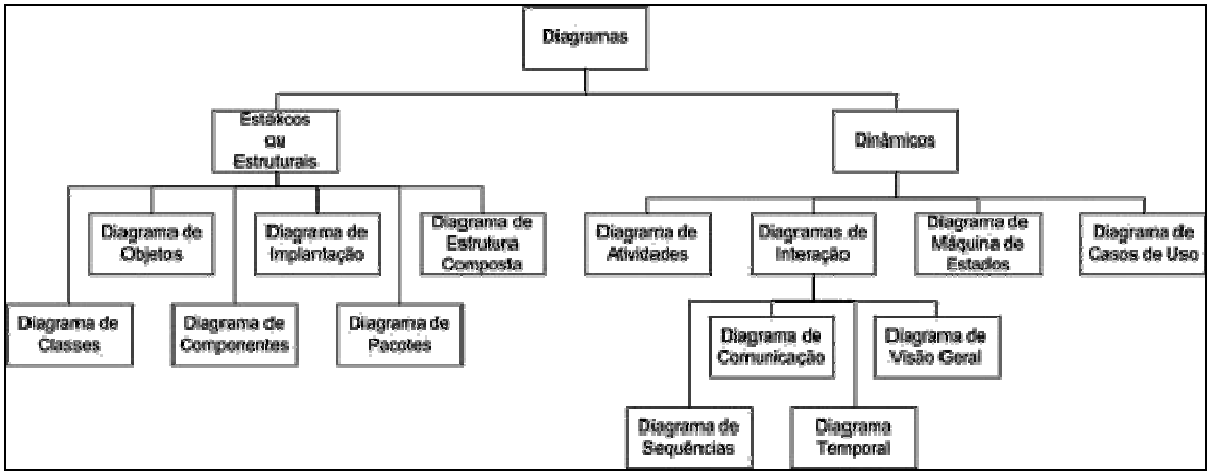


Figura 4 – Classificação dos diagramas UML

No escopo deste trabalho, somente os diagramas de Máquina de Estados e de Atividades são utilizados. O diagrama de Máquina de Estados pode ser utilizado com o objetivo de representar as mudanças de um objeto ao longo da execução do sistema. Segundo Guedes (2008), esse diagrama também pode ser usado para representar os estados de um caso de uso ou mesmo de estados gerais de um subsistema ou de um sistema por completo.

Diagramas de Atividades são utilizados para descrever fluxos de execução de atividades. Segundo Ambler (2007), uma atividade pode ser tanto física, e.g. “verificar formulários”, quanto eletrônica, e.g. “mostrar tela de cadastro de aluno”. Esse tipo de diagrama pode ter diversos níveis de complexidade como, por exemplo, modelar a lógica interna de um componente complexo ou ilustrar processos de negócio em alto nível (GRÄSSLE; BAUMANN; BAUMANN, 2005).

2.2.3.1. Perfis UML

Pelo fato de ter vários diagramas que podem ser aplicados em conjunto ou individualmente para a modelagem de sistemas computacionais em geral, a UML é

considerada uma GPL e, por esse motivo, para permitir uma personalização e extensão da sua sintaxe e semântica, a UML provê mecanismos de adaptação a domínios específicos chamados de perfis UML.

São três os mecanismos de extensão que integram os perfis UML, a saber: os estereótipos, os valores rotulados (*tagged values*) e as restrições. Além disso, ícones e símbolos também podem ser utilizados em um perfil UML com o objetivo de tornar mais intuitiva a modelagem de sistemas para um domínio específico.

Um estereótipo é definido por um nome e o conjunto de elementos do metamodelo da UML nos quais ele pode ser aplicado. Este recurso permite atribuir uma nova semântica a elementos já existentes do modelo.

Um valor rotulado é um meta-atributo adicionado a uma metaclasse do metamodelo. Ele possui um nome, um tipo e é associado a um estereótipo específico. Além de poder definir características de um objeto com estereótipo aplicado, um atributo também pode ser utilizado para especificar restrições (POOLEY; WILCOX, 2004).

Restrições podem ser aplicadas a estereótipos, por exemplo, para definir as propriedades de boa formação de um modelo com o objetivo de assegurar que todas as relações entre os elementos são coerentes com o domínio da aplicação. Elas podem ser expressas em qualquer linguagem (FUENTES-FERNÁNDEZ; VALLECILLO-MORENO, 2004), incluindo linguagem natural e OCL.

3. *TRABALHOS RELACIONADOS*

Vários trabalhos que propõem DSLs através de perfis UML com o objetivo de adaptar diagramas UML a domínios específicos podem ser citados (AAGEDAL; ECKLUND, 2002) (FONTOURA; PREE; RUMPE, 2000) (UETANABARA JÚNIOR; CAMARGO; CHAVEZ, 2009) (ZIADI; HELOUET; JEZEQUEL, 2004). No trabalho de Ziadi, Helouet e Jezequel (2004), por exemplo, é apresentado um perfil UML para linhas de produto de software que especifica a variabilidade de produtos por meio de diagramas de classe e de sequência. Esses trabalhos e vários outros demonstram a versatilidade de utilização dos perfis UML. No entanto, nenhum perfil UML específico para o domínio da televisão digital foi encontrado até a data da escrita deste trabalho.

Trabalhos que aplicam diretamente os conceitos da arquitetura orientada a modelos ao desenvolvimento de aplicações hipermídia para televisão digital ainda são escassos na literatura. De acordo com as pesquisas realizadas, os trabalhos que se destacam tratam de ferramentas de autoria com notação própria ou modelos de desenvolvimento que utilizam *templates* ou componentes. Ainda assim, a maioria desses trabalhos é voltada apenas à linguagem NCL, limitando seu campo de atuação. Nas subseções que compõem este capítulo, serão apresentados trabalhos que tratam de modelagem de aplicações multimídia e/ou hipermídia e desenvolvimento baseado em componentes para televisão digital.

3.1. Towards a UML Extension for Hypermedia Design

Baumeister, Koch e Mandel (1999) propuseram uma extensão da UML para projetos hipermídia focando na modelagem dos aspectos de navegação e de interface de usuário. Os autores consideram que o projeto de uma aplicação hipermídia consiste em três modelos: conceitual, de navegação, e de apresentação. O modelo conceitual consiste em um diagrama de Classes que identifica os objetos do domínio da aplicação e seus relacionamentos. Similarmente, o modelo de navegação é representado em um diagrama de Classes onde são definidos os nós navegacionais, em conjunção com um diagrama de Objetos que mostra como esses nós navegacionais são visitados. Finalmente, o modelo de apresentação descreve a interface de usuário através de objetos compostos e seu comportamento dinâmico através de diagramas Máquina de Estados.

O modelo conceitual, resultante da fase de análise do projeto, define as classes relevantes ao domínio do problema e as associações entre as mesmas. O objetivo desse modelo é capturar a semântica do domínio sem se preocupar com aspectos de navegação e apresentação. Um exemplo desse diagrama pode ser visto na Figura 5.

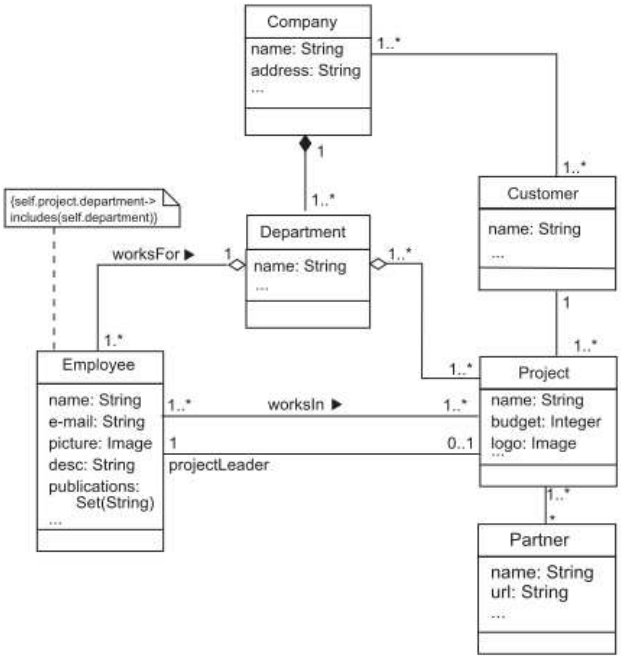


Figura 5 – Exemplo de modelo conceitual de uma aplicação hipermídia (BAUMEISTER; KOCH; MANDEL, 1999)

Um exemplo de diagrama navegacional pode ser visto na Figura 6. Nota-se que foram utilizados ícones para facilitar a identificação da função de cada estereótipo, similarmente à abordagem adotada no presente trabalho. O modelo de classes navegacionais é um subgrafo do diagrama conceitual, pois as classes que não são necessárias à navegação são retiradas ou reduzidas a atributos no novo diagrama. O modelo de estrutura navegacional, baseado no modelo de classes navegacionais, define como os objetos são visitados. Para isso, os autores introduziram novos elementos de modelagem: menus, índices, nós externos e contextos de navegação.

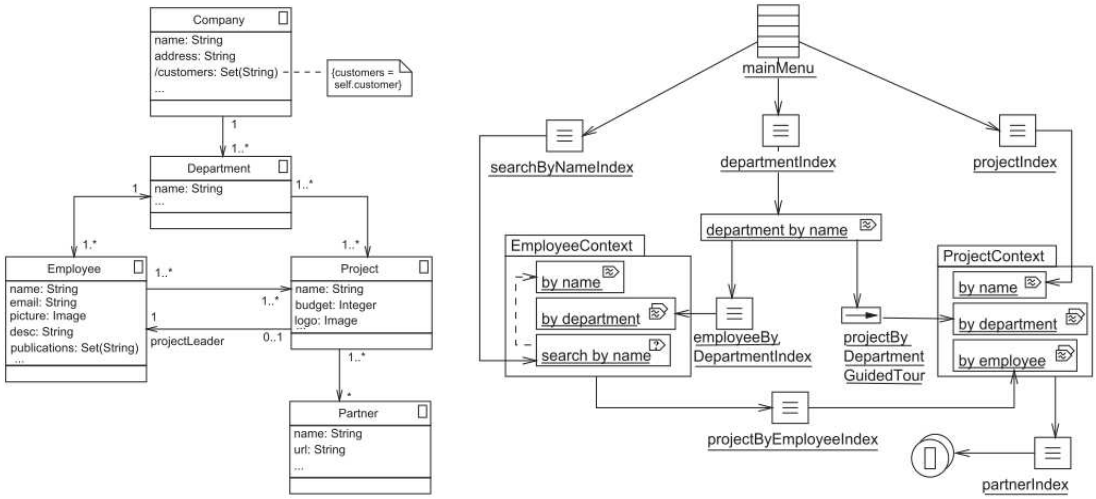


Figura 6 – Exemplo de modelos de classes navegacionais (esquerda) e de estrutura navegacional (direita) (BAUMEISTER; KOCH; MANDEL, 1999)

A última parte do processo de *design* é a modelagem da apresentação da aplicação ao usuário. Para isso, dois diagramas são utilizados no intuito de especificar quais elementos de interface de usuário são usados (modelo estático de apresentação) e para especificar o comportamento desses elementos (modelo dinâmico de apresentação). Ambos os modelos são exemplificados na Figura 7. Vale ressaltar que a representação gráfica dos elementos é muito ligada a HTML, o que pode ser justificado pelo fato de que a Web era a mais importante plataforma para aplicações hipermídia no período em que o trabalho foi escrito. Esse aspecto do trabalho o deixa mais longe do domínio das aplicações para televisão digital, foco desta dissertação.

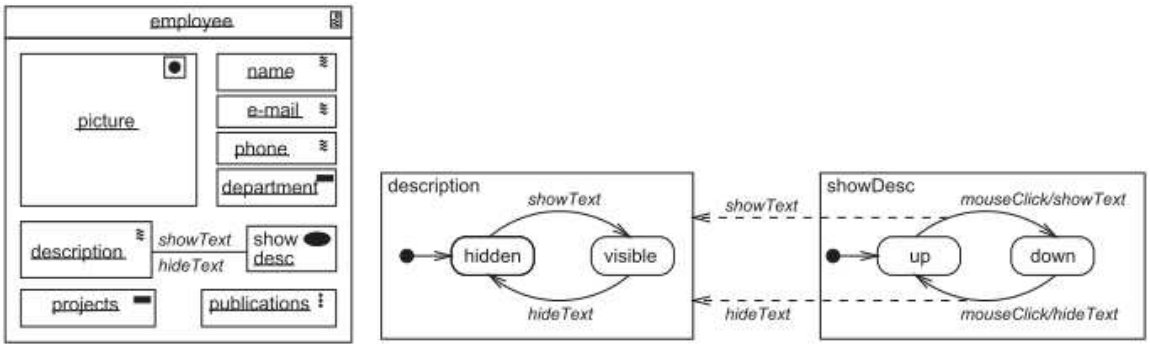


Figura 7 – Exemplo de objeto de apresentação (direita) e de diagrama de Máquina de Estados especificando o comportamento de um botão (esquerda) (BAUMEISTER; KOCH; MANDEL, 1999)

O modelo estático é representado por diagramas de objetos compostos. Foram definidos os seguintes estereótipos de interface de usuário: `<<anchor>>`, `<<text>>`, `<<button>>`, `<<video>>`, `<<audio>>`, `<<image>>`, `<<form>>`, `<<collection>>`, `<<anchored collection>>`, `<<presentational object>>` e `<<external application>>`. O modelo dinâmico de apresentação utiliza diagramas de Máquina de Estados para definir a reação dos elementos estáticos a eventos externos, e.g. movimentos do mouse, cliques e pressionamentos de teclas como também a eventos internos, e.g. *timers*. Mais uma vez, é possível notar a forte ligação desse perfil com aplicações voltadas para ambientes diferentes da televisão digital, visto que a utilização de dispositivos de entrada como mouse e teclado não fazem parte da maioria dos cenários de interação com aplicações televisivas.

Além de explorar uma classe de aplicações hipermídia diferente, i.e. aplicações para televisão digital, o trabalho aqui apresentado não se limita apenas à modelagem desse tipo de aplicação, mas também provê meios de gerar automaticamente o seu código-fonte.

3.2. UML-based Behavior Specification of Interactive Multimedia Applications

Sauer e Engels (2001) propuseram uma extensão da UML para a especificação integrada de sistemas multimídia baseada em um método de desenvolvimento orientado a objetos. Para isso, focaram na especificação do comportamento interativo e temporal do sistema através dos diagramas de máquina de

estados e de seqüência, respectivamente. O perfil UML proposto se adapta à abordagem OMMMA (ENGELS; SAUER, 2002) – *Object-oriented Modeling of MultiMedia Applications* – descrita em um trabalho anterior dos mesmos autores (SAUER; ENGELS, 1999).

Os objetos multimídia são marcados com o estereótipo `<<media>>` enquanto que outros estereótipos são usados para os demais tipos de objetos presentes na aplicação. O estereótipo `<<application>>` é usado para diferenciar objetos de mídias de classes gerais da aplicação. Para partes mais complexas que envolvem relacionamentos temporais e espaciais entre objetos `<<application>>`, foi criado o estereótipo `<<scenario>>`.

Diferentemente da abordagem adotada no presente trabalho, o trabalho de Sauer e Engels utiliza diagramas de sequência para a modelagem do comportamento temporal da apresentação. No entanto, este diagrama pode não ser o mais adequado para a modelagem de aplicações hipermídia interativas uma vez que não é possível descrever graficamente em uma linha de tempo o momento em que o usuário interage com a aplicação. Além disso, para aplicações complexas, esse tipo de diagrama pode ficar pouco inteligível devido às suas limitações no posicionamento dos elementos que o compõem. Para contornar esse problema, Sauer e Engels convencionaram que as transições dos diagramas de Máquina de Estados passassem a ser referências a diagramas de Sequência como mostra a Figura 8.

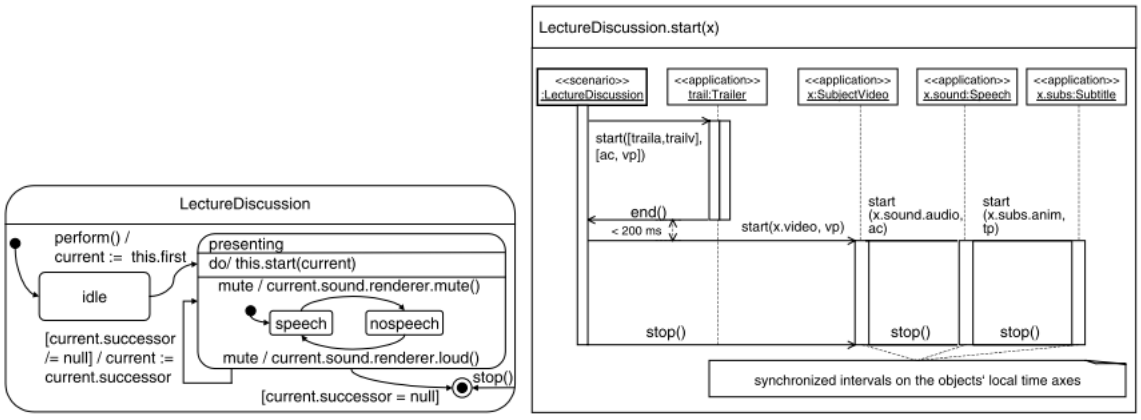


Figura 8 – Exemplos de diagrama de Máquina de Estados e de Sequência (SAUER; ENGELS, 2001)

Nota-se que a linguagem utilizada nos diagramas possui características muito ligadas ao código-fonte, como instruções semelhantes a chamadas de métodos, o que pode dificultar o entendimento dos modelos por envolvidos no projeto que não sejam programadores. Nem mesmo a utilização de representações gráficas mais intuitivas como acontece em (BAUMEISTER; KOCH; MANDEL, 1999) e no perfil UML-TV aqui apresentado é feita, o que facilitaria o entendimento dos diagramas. Além disso, os autores se limitam apenas à especificação do perfil, sem vislumbrar transformações desse modelo em outros modelos intermediários ou em código-fonte como é o caso do trabalho aqui apresentado.

3.3. UML Extensions for Hypermedia Navigation and Presentation Modeling

Um perfil UML é apresentado em (BARNAGHI; KAREEM, 2005) com o objetivo de modelar aspectos de navegação e apresentação de cenários hipermídia. Os autores empregaram os diagramas de Máquina de Estados e de Atividades para alcançar seus objetivos.

A proposta é baseada no conceito de segmentos, i.e., um subconjunto da apresentação que pode ser acessado através de *links* ou navegação sequencial. Uma apresentação hipermídia é então formada por diversos segmentos, e.g. em uma apresentação de *slides*, cada *slide* é considerado como um segmento. Assim, são definidos dois estereótipos: `<<segment>>` e `<<presentation>>`. O estereótipo `<<segment>>` é subdivido em três partes: `Elements` (descreve os elementos do segmento e seus atributos espaciais), `Constraints` (define restrições e regras de sincronização) e `Messages` (descreve mensagens e interações). Ao longo do trabalho as notações gráficas utilizadas são mapeadas em código-fonte SMIL, mas nenhuma menção a apoio automatizado para este processo de transformação é feita. O estereótipo `<<presentation>>` pode possuir diversas subdivisões denominadas *areas* que são identificadas como regiões.

A representação do *layout* é feita dentro da representação da classe onde é aplicado o estereótipo `<<presentation>>`, o que pode ser um problema em projetos maiores e mais complexos. Os valores rotulados não são claramente especificados e os exemplos apresentados não foram suficientes para avaliar se aqueles que foram mostrados são adequados para uma grande variedade de aplicações.

O trabalho apresenta versões modificadas dos diagramas de Atividades e de Máquina de Estados. No diagrama de Atividades (Figura 9), um novo elemento (visualmente parecido com um nó de decisão) chamado de `Anchor` é adicionado, mas não é satisfatoriamente explicado pelos autores, que informam apenas que esse elemento descreve os caminhos de navegação entre links**.** Na versão do diagrama de Máquina de Estados (Figura 10) apresentado, cada segmento é um estado e as transições são realizadas de acordo com eventos que não são bem explicados no artigo.

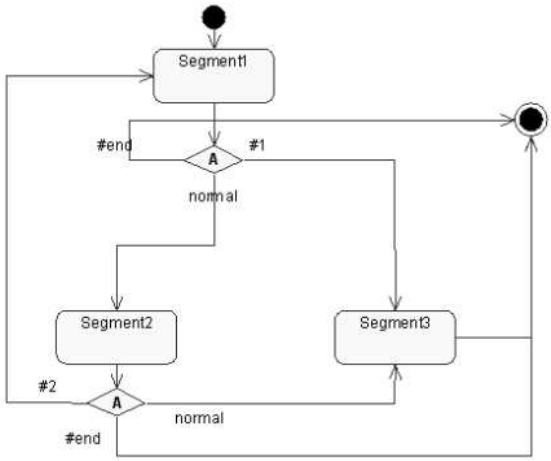


Figura 9 – Diagrama de Atividades para modelagem de aplicações hipermídia (BARNAGHI; KAREEM, 2005)

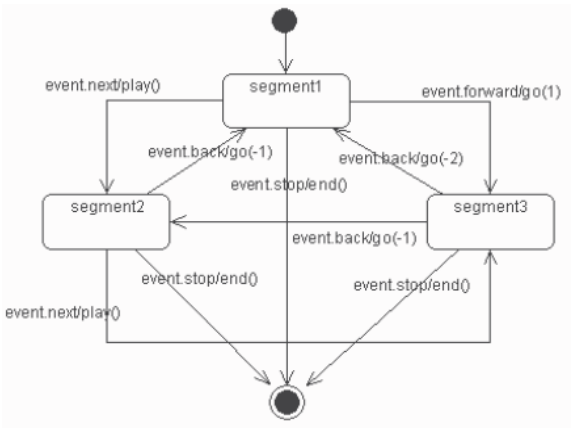


Figura 10 – Diagrama de Máquina de Estados para modelagem de aplicações hipermídia (BARNAGHI; KAREEM, 2005)

Os eventos são especificados nas transições assim como no perfil UML-TV, no entanto as ações a serem realizadas ao entrar em um estado também são definidas nas transições, deixando o diagrama visualmente poluído. Além disso, não são especificadas as ações que podem ocorrer (no diagrama da Figura 10 podemos enumerar somente as ações `play`, `go` e `end`). No perfil proposto nesta dissertação, as ações são especificadas por diagramas de Atividades secundários, o que torna mais clara e flexível a modelagem.

3.4. StoryToCode

Em (MARQUES NETO; SANTOS, 2009), os autores apresentam um modelo de desenvolvimento de aplicações chamado StoryToCode voltado para o padrão brasileiro de televisão digital. Nesse modelo, que tem como foco o desenvolvimento baseado em componentes, é proposta a transformação de *storyboards*[7] em diagramas de classe para então transformá-los em código-fonte NCL e/ou Java.

O StoryToCode constitui-se em três partes assim denominadas: *storyboard*, arquitetura de elementos e geração de código. A primeira parte, o *storyboard*, é a descrição de alto nível da aplicação, o que no contexto da MDA corresponde ao CIM e que pode ser considerada fracamente estruturada e informal, mas que se constitui como uma linguagem próxima aos *experts* no domínio da televisão. A segunda parte, a arquitetura de elementos, é definida com base em NCL e representada em UML (ver Figura 11) para que os elementos abstratos necessários para a transformação dos *storyboards* possam ser especificados de forma não ambígua. A terceira parte, a geração de código, é composta por um componente de transformação chamado de *transformation machine* que é encarregado de gerar o código-fonte a partir do conjunto de elementos da segunda parte de acordo com regras de transformação específicas de cada plataforma alvo (TV, Web, etc.).

[7] Esquemas gráficos dispostos em sequência utilizados para planejar a cenas de filme, animações ou conteúdo interativo, e.g. um site.

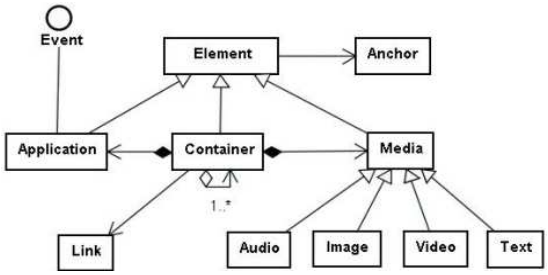


Figura 11 – Hierarquia de elementos do StoryToCode (MARQUES NETO; SANTOS, 2009)

Como os próprios autores comentam, o StoryToCode é apenas inspirado no padrão MDA, e não configura uma aplicação desse paradigma de desenvolvimento no escopo das aplicações hipermídia, diferentemente da proposta do presente trabalho. Com isso, alguns problemas podem ser apontados, como a impossibilidade de transformação do modelo de mais alto nível em modelos intermediários e a falta de formalização das transformações do modelo intermediário para o código-fonte.

Pode-se perceber um risco de distanciamento do que é modelado e o que é implementado, pois caso sejam necessárias alterações no *storyboard* de uma aplicação hipermídia, as mesmas não poderão ser propagadas automaticamente para o modelo intermediário e, consequentemente, para o código-fonte. Para isso, é necessário que alguma pessoa envolvida no processo de desenvolvimento verifique o que foi alterado no *storyboard* para então realizar essas mudanças no conjunto de elementos, o que torna o processo mais suscetível a erros.

Com relação à transformação M2C (*Model to Code*), é citado no trabalho a implementação de duas instâncias da *transformation machine* as quais recebem uma representação do conjunto de elementos (um diagrama de classes) no formato XMI. No entanto, não são fornecidos maiores detalhes sobre essas transformações, o que não permite uma avaliação mais aprofundada da aplicação das regras de transformação. Além disso, não foram utilizadas linguagens de transformação de modelos, tais como QVT (Object Management Group, 2008) – *Query View Transformation* –, ATL (JOUAULT et al., 2006) – *Atlas Transformation Language* –, ou MOFScript (OLDEVIK et al., 2005), por exemplo[8].

[8] MARQUES NETO, M. C. (Apresentação durante o Simpósio Brasileiro de Sistemas Multimídia e Web) Comunicação pessoal, 2009.

4. PERFIL UML-TV

Nesta seção, o perfil UML-TV é apresentado. Inicialmente, seu metamodelo e os elementos básicos do perfil são introduzidos. Posteriormente, com o intuito de tornar a apresentação do conteúdo mais didática, serão utilizados alguns exemplos extraídos do tutorial (SOARES NETO et al., 2007), o qual descreve aplicações hipermídia escritas em NCL. O nível de complexidade dos exemplos é gradual, servindo bem ao propósito de introdução da nova notação aqui proposta na forma do perfil UML-TV.

4.1. Metamodelo UML-TV

Todos os elementos da UML e seus relacionamentos fazem parte do modelo conceitual da linguagem, também chamado de metamodelo. A partir da extensão da UML com elementos do domínio especifico das aplicações hipermídia para televisão digital, foi desenvolvido o metamodelo UML-TV, que define as relações fundamentais entre os elementos do perfil. Esse metamodelo é mostrado na Figura 12, onde as classes que contém o estereótipo `<<metaclass>>` são provenientes da UML e as demais do perfil UML-TV.

A partir do metamodelo, algumas restrições podem ser inferidas como, por exemplo, que uma ação (*Action*) só pode se relacionar com uma mídia (*Media*) ou com uma área de visualização (*DisplayArea*) e que uma mídia visualizável (*VisibleMedia*) se relaciona com uma área de visualização através de um elemento *Display*. O conjunto completo das restrições do perfil UML-TV será apresentado no decorrer do presente capítulo.

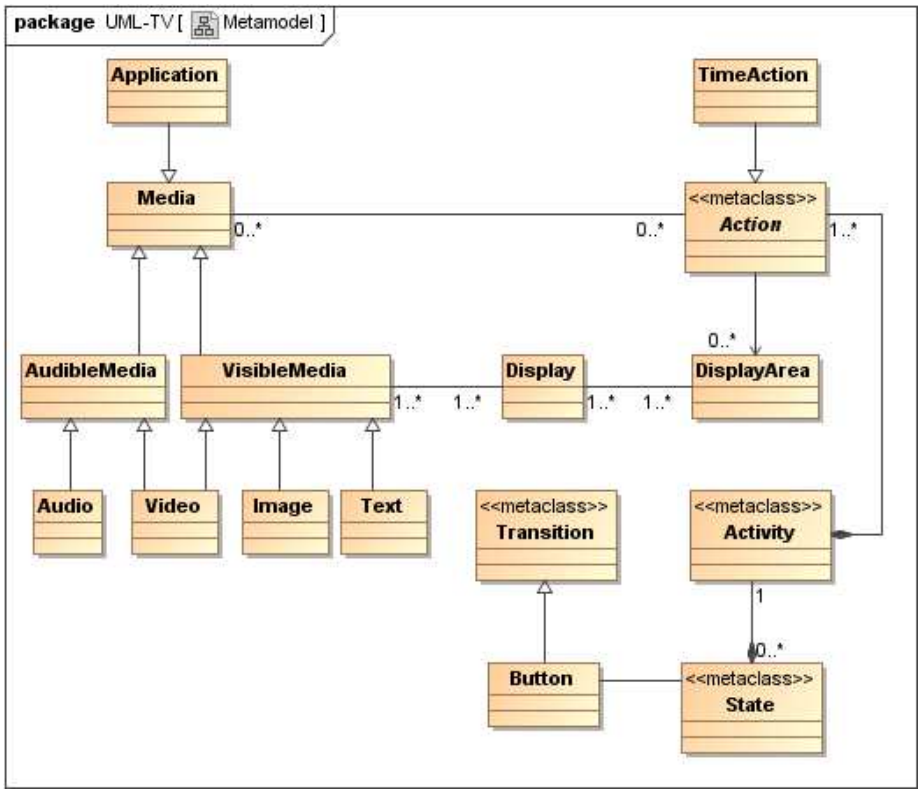


Figura 12 – Metamodelo do perfil UML-TV

4.2. Elementos do perfil UML-TV

Utilizando o mecanismo de extensão que os perfis UML propiciam (estereótipos, valores rotulados e restrições), inicialmente, foram definidos alguns estereótipos para os elementos que podem compor os diagramas de Atividades e de Máquina de Estados. A Tabela 2 contém os estereótipos aplicados à metaclasse *ObjectNode* da UML. A maioria desses estereótipos representa as mídias normalmente presentes em uma aplicação hipermídia.

Tabela 2 – Estereótipos aplicados à metaclasse ObjectNode

Ícone	Nome	Descrição
	Video	Mídias de vídeo
	Image	Mídias de imagem
	Audio	Mídias de áudio
	Text	Mídias de texto
	Application	Aplicações externas (NCLua, Xlet, etc.)
	DisplayArea	Objeto que representa uma região onde uma mídia pode ser apresentada

A Tabela 3 contém o único estereótipo aplicado à metaclasse *Action* da UML. Já que o próprio elemento *Action* representa uma unidade fundamental de funcionalidade executável (Object Management Group, 2009b), o mesmo também é utilizado diretamente nos diagramas do perfil UML-TV, como será mostrado posteriormente nos exemplos de utilização do perfil.

Tabela 3 – Estereótipos aplicados à metaclasse Action

Ícone	Nome	Descrição
	TimeAction	Ação disparada em determinado momento no tempo, relativa a uma mídia

Na Tabela 4, estão descritos os estereótipos aplicados à metaclasse *Transition*. Esses elementos representam os eventos causais (botões do controle remoto) ou temporais (*TimeTransition*) que podem causar uma mudança no estado da aplicação hipermídia, fazendo com que o sistema execute outras ações. A lista dos botões do controle remoto incluídos foi baseada na norma NBR 15604 (Associação Brasileira de Normas Técnicas, 2007) que trata dos receptores compatíveis com o padrão ISDB-Tb, mas que também são de uso universal em outros padrões de televisão

digital. Na Tabela 4, em alguns casos, apenas um ícone é mostrado por questões de melhor aproveitamento do espaço, e.g. botões numéricos do controle remoto.

Tabela 4 – Estereótipos aplicados à metaclasse Transition

Ícone	Nome	Descrição
	RED	Botão vermelho do controle remoto
	GREEN	Botão verde do controle remoto
	YELLOW	Botão amarelo do controle remoto
	BLUE	Botão azul do controle remoto
	TimeTransition	Transição disparada em determinado momento no tempo relativo à execução da aplicação
	KEY_0, KEY_1, KEY_2, KEY_3, KEY_4, KEY_5, KEY_6, KEY_7, KEY_8 e KEY_9	Botões numéricos
	CH_UP e CH_DOWN	Botões de mudança de canal
	VOL_UP e VOL_DOWN	Botões de controle de volume
	ARROW_UP, ARROW_DOWN, ARROW_RIGHT e ARROW_LEFT	Setas direcionais para cima, para baixo, para a direita e para a esquerda
	KEY_EPG	Botão que ativa o guia eletrônico de programação
	KEY_OK	Botão de confirmação
	KEY_BACK	Botão para voltar para a operação anterior
	KEY_MENU	Botão para acessar o menu
	KEY_INFO	Botão de informação

A Tabela 5 mostra os eventos que podem ser usados nos diagramas de Atividades para indicar o que ocorrerá com as mídias. A utilização desses eventos será ilustrada na subseção 4.3 a seguir. Vale ressaltar que os eventos `pause` e `resume` se

aplicam apenas em mídias contínuas (áudio e vídeo), enquanto que para as demais mídias (texto, imagem e aplicação) apenas os eventos `start` e `stop` são aplicáveis.

Tabela 5 – Eventos válidos do perfil UML-TV

Nome do evento	Descrição
start	Inicia a execução do objeto de mídia alvo
stop	Finaliza a execução o objeto de mídia alvo
pause	Para a execução do objeto de mídia alvo
resume	Retoma a execução o objeto de mídia alvo

A Figura 13 mostra o diagrama geral do perfil proposto para TV digital, com os estereótipos definidos e seus respectivos valores rotulados. Nela, pode-se verificar a relação de realização dos estereótipos abstratos `VisibleMedia` e `AudibleMedia`, que concentram as propriedades inerentes às mídias visualizáveis e às mídias audíveis. Além disso, pode-se verificar a generalização dos estereótipos `Video`, `Text`, `Image` e `Audio` para o estereótipo abstrato `Media`, o qual agrupa atributos comuns a todas as mídias, e.g., o caminho do arquivo (`filePath`). No caso dos botões do controle remoto, alguns deles foram omitidos no diagrama para melhor aproveitamento do espaço de forma a não prejudicar a visualização de todos os elementos do diagrama. No entanto, todos eles são generalizações do estereótipo abstrato `Button`. Na notação utilizada pelo software de modelagem utilizado, o MagicDraw[9], as metaclasses da UML em que os estereótipos podem ser aplicados são indicadas entre colchetes.

[9] www.magicdraw.com

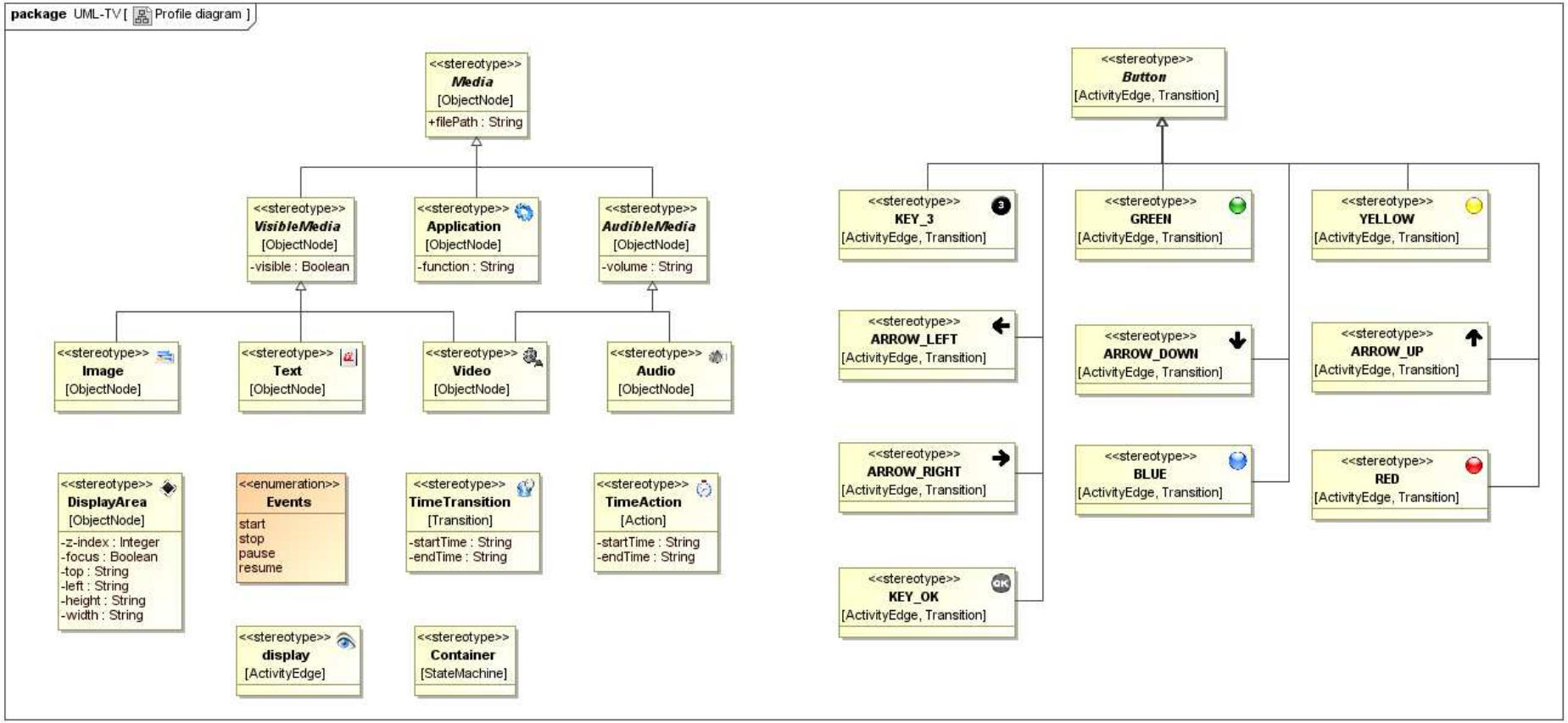


Figura 13 – Diagrama do perfil UML-TV

Alguns dos estereótipos possuem valores rotulados para que informações adicionais (parâmetros) possam ser adicionadas aos modelos. A Tabela 6 contém a lista completa de valores rotulados organizados por estereótipo.

Tabela 6 – Valores rotulados do perfil UML-TV

Estereótipo	Valor rotulado	Descrição
TimeTransition	startTime	Especifica em que momento (em segundos) determinada ação irá ocorrer relativamente à aplicação
	endTime	Especifica em que momento (em segundos) determinada ação irá terminar relativamente à aplicação
TimeAction	startTime	Especifica em que momento (em segundos) determinada ação irá ocorrer relativamente uma mídia
	endTime	Especifica em que momento (em segundos) determinada ação irá terminar relativamente a uma mídia
Media	filePath	Especifica o caminho de determinada mídia no sistema de arquivos (e.g. "..\movie\sample.avi") ou referencia um fluxo de transporte (*transport stream*)
VisibleMedia	visible	Especifica se determinada mídia está visível ou não
AudibleMedia	volume	Especifica o nível do volume de uma mídia com som
Application	function	Especifica qual função chamar quando a mídia for executada
DisplayArea	z-index	Especifica em qual nível na coordenada z está a área de exibição com o objetivo de definir relações de sobreposição de regiões
	focus	Especifica se determinada área de visualização possui o foco ou não
	top	Determina a distância do ponto superior esquerdo da área de visualização até a origem da coordenada vertical (topo da tela)
	left	Determina a distância do ponto superior esquerdo da área de visualização até a origem da coordenada horizontal (lado esquerdo da tela)
	height	Determina a altura da área de visualização
	width	Determina a largura da área de visualização

Para complementar o perfil, um conjunto de restrições foi especificado para garantir que os modelos gerados com base no perfil UML-TV estejam em conformidade com sua definição. Tais restrições, em sua maioria escritas em OCL, são descritas a seguir:

- Restrição 1: Dada uma ação do tipo `TimeAction` ou uma transição do tipo `TimeTransition`, o valor do atributo `startTime` de deve ser menor que o valor da valor rotulado `endTime`.

```
context TimeAction
inv: self.startTime < self.endTime

context TimeTransition
inv: self.startTime < self.endTime
```

- Restrição 2: Uma ação pode se relacionar apenas com áreas de visualização, mídias ou com estados iniciais (apenas como alvo da associação nesse último caso). Esse relacionamento deve ser realizado através do elemento `ActivityEdge` (generalização tanto de `ControlFlow` quanto de `ObjectFlow`).

```
context ActivityEdge
inv: self.source.oclIsKindOf(Pseudostate) and self.source-
>forAll(p | p.kind = #initial) implies
self.target.oclIsKindOf(Action) or
self.target.oclIsKindOf(ObjectNode) implies
self.source.oclIsKindOf(Action) or self.target.oclIsKindOf(Action)
implies self.source.oclIsKindOf(ObjectNode)
```

- Restrição 3: Mídias e áreas de visualização devem se relacionar através de setas de fluxo com o estereótipo `<<display>>` onde a origem é uma mídia e o destino é uma área de visualização.

```
context Display
inv: self.source.oclIsKindOf(Media) and
self.target.oclIsKindOf(DisplayArea)
```

- Restrição 4: Todo estado deve ter uma atividade associada (especificada em sua propriedade `doActivity`).

```
context State
inv: self.doActivity->notEmpty()
```

- Restrição 5: Uma, e somente uma, máquina de estados de um modelo pode conter o estereótipo `<<Container>>`.

```
context Model
inv: self->select(oclIsKindOf(Container)).size() = 1
```

- Restrição 6: Quando um atributo de uma mídia ou de uma área de exibição for alterado, o nome do atributo deve ser separado pelo sinal "=" (igualdade) do valor atribuído. Se mais de uma propriedade for alterada concomitantemente, deve-se utilizar vírgulas para separar as novas atribuições. A especificação dessa restrição não pode ser feita através de OCL, mas o formato da *string* de atribuição pode ser validado com a seguinte expressão regular: **`\w+\s*=\s*([0-9]+[%]?|\w+)[,]?`**.

- Restrição 7: Eventos de mídia devem fazer parte da enumeração `Events` nas formas "`event`" ou "`event@id`" ou alterar um ou mais valores rotulados. Onde "id" se refere a uma marcação especificada na mídia alvo, e.g. uma tag `<div>` em uma mídia do tipo HTML. A especificação dessa restrição não pode ser realizada com OCL.

- Restrição 8: Deve existir pelo menos um diagrama de máquina de estados no modelo e, em decorrência da Restrição 4, também devem existir pelo menos um diagrama de atividades no modelo.

```
context Model
inv: self->exists(s : StateMachine | s.oclIsKindOf(StateMachine))
and self.exists(a : Activity | a.oclIsKindOf(Activity))
```

- Restrição 9: Deve existir um, e somente um nó inicial em cada diagrama.

```
context StateMachine
inv: self->select(p | p.oclIsKindOf(PseudoState) and p.kind =
#initial)->size() = 1

context Activity
inv: self->select(p | p.oclIsKindOf(PseudoState) and p.kind =
#initial)->size() = 1
```

- Restrição 10: Nomes de mídias e de áreas de visualização (todos os elementos do tipo `ObjectNode`) devem ser únicos.

```
context ObjectNode
inv: self->isUnique(self.name)
```

- Restrição 11: Apenas uma região pode existir em cada máquina de estados.

```
context StateMachine
inv: self->select(oclIsKindOf(Region))->size() = 1
```

Dessa forma, com a especificação de todos os estereótipos, valores rotulados e restrições necessárias, o perfil UML-TV pode ser utilizado para a modelagem de aplicações hipermídia para televisão digital.

4.3. Exemplos de utilização

As subseções seguintes ilustram exemplos extraídos do trabalho de Soares Neto et al. (2007), no intuito de explicar a utilização do perfil UML-TV com mais detalhes. Cada exemplo apresenta um nível de complexidade superior em relação ao exemplo anterior, partindo da simples reprodução de um objeto de mídia, passando por apresentações de vídeo sequenciais, sincronismo de legendas com vídeo, interação do usuário através do controle remoto, seleção de vídeos pelo usuário, até a reprodução de um menu de DVD. Buscando evitar repetições desnecessárias, apenas as características que mudarem de um exemplo para o outro serão apresentadas. Também foram omitidos alguns valores rotulados, e.g., os caminhos dos arquivos, para que a visualização das figuras ficasse mais clara.

Rumbaugh, Jacobson e Booch (2004) dividem as visões proporcionadas pela UML em três: classificação estrutural, comportamento dinâmico e gerenciamento de modelo. As visões de comportamento dinâmico descrevem o comportamento de um sistema ao longo do tempo. Nas aplicações para televisão digital, um aspecto muito importante diz respeito aos relacionamentos entre as mídias, que é o que o perfil UML-TV se propõe a modelar. Assim, o comportamento das mídias ao longo do tempo é o alvo da modelagem nesse caso. Os diagramas UML voltados à modelagem desse aspecto são os diagramas de Máquina de Estados, de Atividades, de Sequência e de Colaboração.

O diagrama de Sequência, como já exposto, tem limitações com relação ao posicionamento dos elementos, o que pode levar a uma dificuldade maior de entendimento dos diagramas em aplicações complexas, além da impossibilidade de situar eventos causais em uma linha de tempo, e.g. o pressionamento de botões do controle remoto. Os diagramas de Colaboração constituem-se em uma alternativa com poucos recursos, que se diferencia dos diagramas de sequência apenas pela maior liberdade no posicionamento dos elementos.

Os dois diagramas utilizados pelo perfil UML-TV são os diagramas de Atividade e de Máquina de Estados, pois possuem conceitos que podem ser mais bem utilizados na modelagem de aplicações hipermídia, e.g. *forks* e junções. Além disso, a integração entre esses diagramas possibilita a separação da modelagem em quantos diagramas sejam necessários, o que facilita o entendimento de cada parte da aplicação.

4.3.1. Reprodução de um objeto de mídia

Este primeiro exemplo trata da execução de um vídeo em determinada área da tela da televisão. Por mais simples que seja a aplicação hipermídia que esteja sendo modelada, é obrigatória a criação de pelo menos um diagrama de Máquina de Estados e um diagrama de Atividades, de acordo com a Restrição 8 do perfil UML-TV.

O diagrama de Máquina de Estados contém os estados e as transições entre os possíveis estados da aplicação. Entende-se por estado qualquer situação em que a aplicação hipermídia encontre-se, algo como uma fotografia do estado das mídias que compõem a apresentação. As transições podem ser disparadas por eventos causais ou temporais. Os eventos causais são aqueles iniciados por alguma interação do usuário com a aplicação, e.g. o pressionamento de uma tecla do controle remoto. Os eventos temporais são disparados por indicações de determinados momentos no tempo em que devem ocorrer as transições.

Cada um dos estados é associado a um diagrama de Atividades onde são descritas as ações de sincronismo de mídias que serão executadas durante aquele estado (Restrição 4). Essa associação do diagrama de Máquina de Estados com o

diagrama de Atividades é feita com a especificação do campo `doActivity` de cada estado, onde é possível determinar que uma atividade específica seja executada quando a aplicação hipermídia atingir aquele estado.

Uma aplicação computacional, seja ela hipermídia ou não, necessita de um ponto de entrada, ou seja, é necessário que o autor do sistema indique a partir de qual estrutura seu programa será executado. Em *Java Standard Edition*, por exemplo, o programador é obrigado a escrever um método público, estático, sem tipo de retorno, chamado `main` e contendo como único argumento ou um *array* de objetos da classe `String` ou um *varargs*[10]. Já em NCL, é necessário especificar pelo menos uma porta no documento hipermídia que aponte para um nó de mídia presente no contexto `body` ou que mapeie outro contexto, que por sua vez deve ter uma porta apontando para outro contexto ou para um nó de mídia, e assim sucessivamente. No perfil UML-TV, o ponto de entrada da aplicação hipermídia é representado por um nó inicial presente no diagrama de estados distinguido com o estereótipo `<<Container>>`. Como especificado na Restrição 9, só é permitida a existência de um, e apenas um, nó inicial por diagrama no perfil UML-TV, seja ele um diagrama de Atividades ou de Máquina de Estados. Esse nó inicial também um elemento obrigatório nos dois diagramas.

A Figura 14 mostra o diagrama de Máquina de Estados para o exemplo desta seção. Este é o diagrama de Máquina de Estados mais simples possível e aceito como válido pelo perfil UML-TV. Pode-se notar a existência do nó inicial (o nome do nó – `init` – ou qualquer outro nome não é obrigatório) e um estado chamado `state1` que tem associado a ele uma ação chamada `play`. Como todo diagrama de Máquina de Estados na UML deve conter um estado final ou um retorno a um estado anterior, isso também é respeitado no perfil UML-TV. Já na Figura 15 é possível verificar que o estereótipo `<<Container>>` foi aplicado a essa máquina de estados.

[10] Vararg é uma característica da linguagem Java que deixa transparente para o programador a passagem de arrays de tamanho variável.

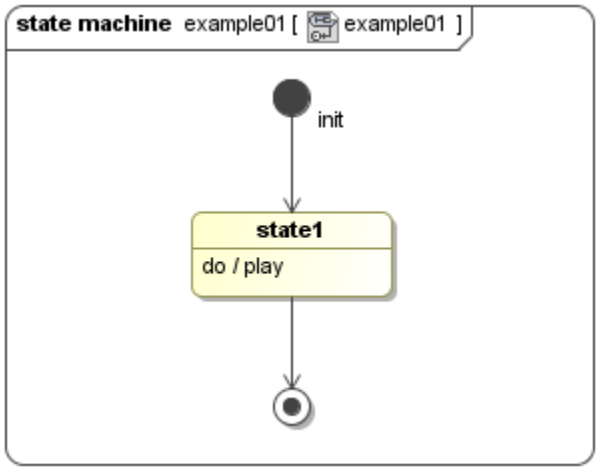


Figura 14 – Diagrama de Máquina de Estados de uma aplicação simples

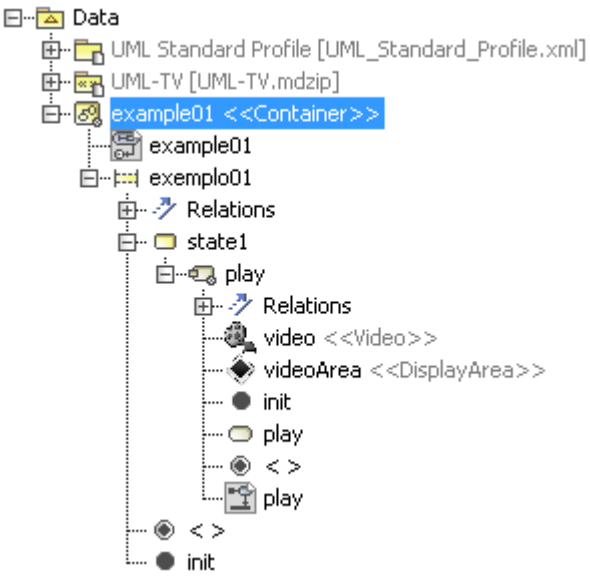


Figura 15 – Árvore contendo todos os diagramas e seus respectivos elementos do exemplo

O diagrama de Atividades associado ao estado `state1` pode ser visualizado na Figura 16. A correlação entre os diagramas é feita através dos nomes dos mesmos, por isso é importante que não haja coincidência de nomes entre diagramas pertencentes ao mesmo projeto. Isso é facilmente evitado pela maioria das ferramentas de modelagem, mas caso o desenvolvimento da aplicação seja realizado por diferentes pessoas ou em ferramentas diferentes, esse cuidado deve ser tomado a fim de que sejam evitadas ambiguidades.

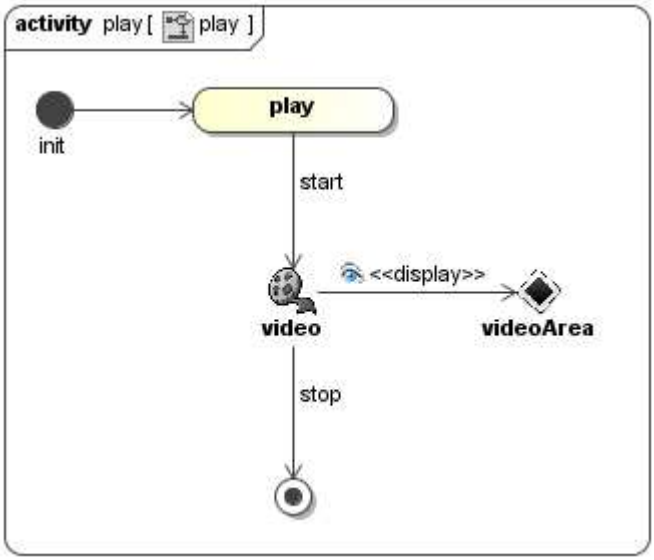


Figura 16 – Diagrama de Atividades da reprodução de um vídeo

Assim como no diagrama de Máquina de Estados, o diagrama de Atividades é iniciado por um nó inicial único. A partir do nó inicial, uma ação deve ser disparada, que por sua vez poderá alterar as propriedades de quantas mídias sejam necessárias. Caso mais uma mídia seja utilizada, um *fork* é adicionado ao diagrama para indicar o paralelismo das ações.

Por questões de organização visual do digrama e também por restrições da própria notação UML, não é possível ligar um nó inicial diretamente a um nó de objeto. Assim, faz-se necessária a adição de uma ação logo após o nó inicial. O nome da ação, chamada de `play` neste exemplo, não tem nenhuma relação com o nome do diagrama ou com qualquer outra restrição imposta pelo perfil UML-TV.

A partir da ação `play` do diagrama de Atividades, há uma seta de fluxo de controle (*Control Flow* na UML) apontando para o vídeo a ser executado durante a aplicação e indicando o evento que acontecerá com o vídeo através da propriedade `name` do fluxo de controle. Neste caso, o fluxo indica que o vídeo será iniciado. Isso é definido pela palavra `start` como valor da propriedade `name` da seta de fluxo de controle.

Outros eventos além de `start` podem ser utilizados nos modelos do UML-TV. Esses eventos são especificados na enumeração `Events` presente na especificação

do perfil (Figura 13). Dessa forma, uma das restrições do UML-TV é que os valores usados nas setas de fluxo façam parte da lista `Events` ou alterem algum valor rotulado do objeto alvo (Restrição 7). Modelos que não cumprem essa restrição tornam o projeto fora de conformidade com o perfil.

Uma vez que é definido o que acontece com o vídeo, é preciso indicar onde o vídeo será exibido durante sua execução. Isso é modelado através do estereótipo `<<display>>` aplicado sobre um fluxo de objeto (*Object Flow* na UML), ligando o vídeo que será exibido a uma área de visualização – no caso deste exemplo, a área chamada `videoArea`.

O estereótipo `<<display>>` é necessário porque não são todos os casos em que uma associação entre um objeto de mídia e uma área de exibição denota uma relação de exibição do primeiro objeto no segundo. É possível, por exemplo, que um evento temporal em uma mídia altere alguma propriedade da área de visualização como o seu tamanho. Um caso desse tipo será exemplificado com mais detalhes na subseção “Redimensionando uma região durante a exibição de uma imagem”, mais adiante neste trabalho.

Caso seja necessária a especificação de uma mídia para servir como referência para o final da aplicação, isso pode ser modelado através da adição de um nó final ligado à mídia através de uma seta contendo o evento que levará ao fim da aplicação hipermídia. No exemplo em questão, o fim do vídeo indica o final da execução da aplicação. A inclusão de um estado final em diagramas de Atividades não é necessária caso não existam mídias contínuas no diagrama.

4.3.2. Iniciando e terminando dois objetos de mídia simultaneamente

O segundo exemplo descreve uma aplicação semelhante à do exemplo anterior, envolvendo a exibição de um vídeo, mas adicionando outra mídia, um texto, em outra área da tela da televisão. Como o diagrama de Máquina de Estados dessa aplicação é praticamente idêntico ao anterior (a única diferença é o nome da atividade presente na propriedade `doActivity`), o mesmo não será reproduzido novamente.

A Figura 17 mostra o diagrama de Atividades desse exemplo. A única alteração com relação ao exemplo anterior é que a atividade `playVideo` inicia duas mídias ao mesmo tempo e em áreas de visualização distintas. Nesse caso, pode ser verificado o uso de um *fork* com o objetivo de deixar mais claro que o início das mídias se dá no mesmo instante de tempo.

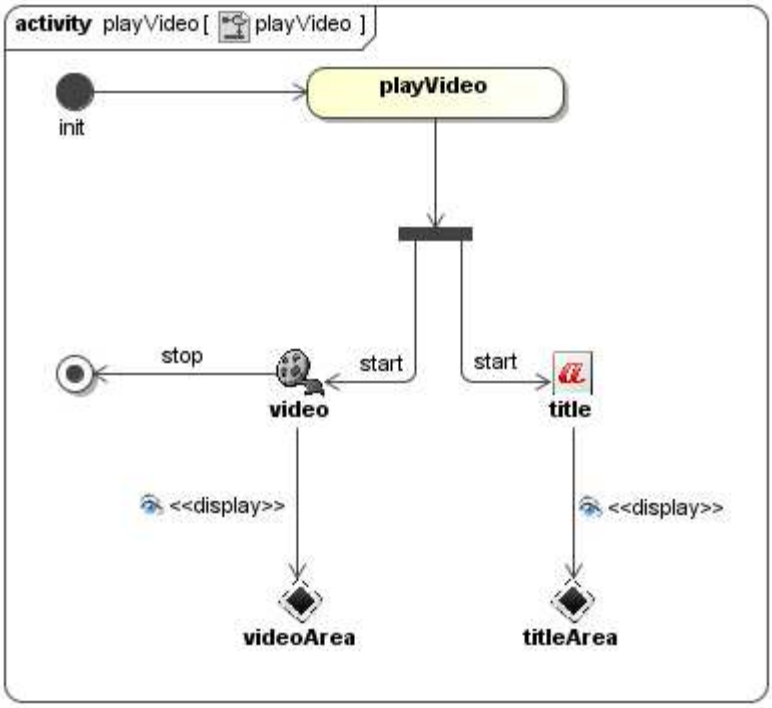


Figura 17 – Diagrama de Atividades da exibição de duas mídias

4.3.3. Iniciando um objeto de mídia quando outro termina

O exemplo a seguir constitui-se na modelagem de uma aplicação contendo apenas a exibição de um vídeo que é iniciado sincronamente com um texto. Assim como nas subseções anteriores, o diagrama de Máquina de Estados do caso apresentado nesta é muito semelhante ao dos exemplos anteriores, tendo diferença apenas no nome da atividade associada ao primeiro e único estado da aplicação, o que não é algo importante para a sua modelagem.

Já o diagrama de Atividades, mostrado na Figura 18, introduz alguns elementos novos na notação do perfil UML-TV. Como pode ser notado, a primeira atividade do diagrama (`playVideo1`) inicia um vídeo chamado `video1` na área de

visualização denominada `area1`. Ao final do primeiro vídeo, outra ação (`playVideo2`) é disparada, iniciando o vídeo chamado `video2` na mesma área de visualização onde antes estava sendo exibido o vídeo anterior, `area1`.

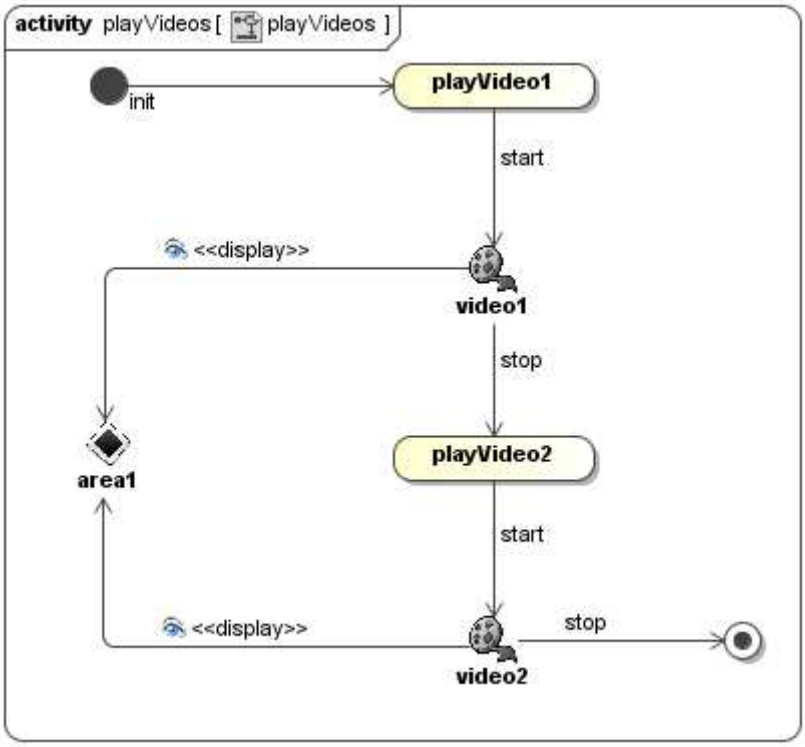


Figura 18 – Diagrama de Atividades da execução de um vídeo após o término de outro vídeo

Neste exemplo, a característica mais importante a ser observada é como se modelam casos em que mídias podem disparar ações de sincronismo. No caso, o momento em que a ação `playVideo2` será ativada é indicado pelo nome dado ao fluxo partindo de `video1` para a atividade: `stop`.

4.3.4. Sincronizando um vídeo com diferentes arquivos de legenda

A aplicação hipermídia resultante da modelagem apresentada nesta subseção também exibe um vídeo, mas também são mostradas mídias de texto no formato de legendas em determinados momentos do mesmo. Nenhum elemento novo foi introduzido no diagrama de Máquina de Estados desse exemplo com relação aos exemplos anteriores.

No diagrama de Atividades apresentado na Figura 19, são detalhadas três ações que ocorrem durante a exibição do vídeo. Aos cinco segundos do vídeo, a ação `showSubtitle1` é ativada, iniciando a exibição do texto `subtitle1` na região `subtitleArea`. Aos nove segundos do vídeo, essa ação deixa de ser executada, voltando à exibição apenas do vídeo, sem nenhuma legenda. O mesmo comportamento acontece dos dez aos 14 segundos, mas dessa vez mostrando o texto `subtitle2` na mesma área destinada às legendas e também dos 15 aos 19 segundos do vídeo, mostrando a mídia `subtitle3`. Dessa forma, fica claro que a referência de tempo de uma ação desse tipo é a mídia que a dispara.

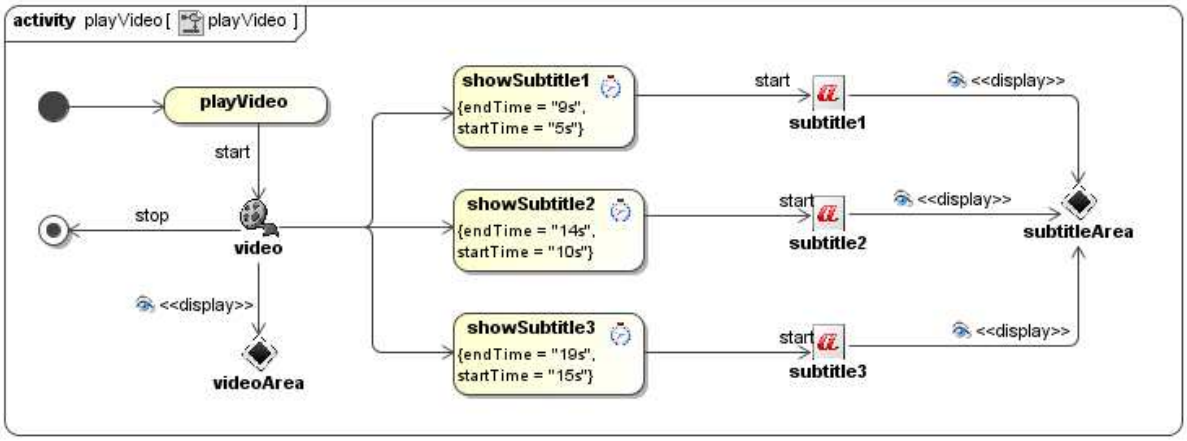


Figura 19 – Diagrama de Atividades da sincronização de um vídeo com arquivos de texto

O momento em que essas ações irão ser disparadas está definido nos valores rotulados de cada uma delas, pois as mesmas tiveram a aplicação do estereótipo `<<TimeAction>>`, que contém as propriedades `startTime` e `endTime`, onde o projetista da aplicação deve inserir os valores, em segundos, de início e de fim da ação, respectivamente, relativos à(s) mídia(s) que as disparam.

Ao final da ação, convencionou-se que o seu alvo volta ao estado em que se encontrava antes da ação acontecer. Tomando como exemplo a modelagem apresentada na Figura 19, a mídia de texto `subtitle3` é iniciada aos 15 segundos do vídeo (indicado pelo atributo `name` da seta que liga a ação à mídia), mas nenhuma indicação explícita indica que aos 19 segundos do vídeo, o texto terá sua exibição interrompida. Dessa forma, fica implícito que, ao alcançar o tempo especificado no atributo `endTime`, um evento `stop` será enviado à mídia `subtitle3`. Por outro lado,

caso a ação indicasse um evento `stop`, ao final do tempo especificado seria enviado um evento `start`. O mesmo vale para os eventos `pause` e `resume`.

4.3.5. Sincronizando um vídeo com um arquivo de legenda segmentado

O resultado da execução da aplicação modelada nesta subseção é idêntico ao resultado do exemplo anterior, i.e. um vídeo é exibido do começo ao fim tendo sincronizado a ele três legendas que aparecem em determinados instantes no tempo. A diferença na modelagem é a presença de apenas um arquivo de texto onde todas as legendas se encontram separadas por elementos `div` do HTML. Por depender de marcações do HTML, o formato do arquivo de texto utilizado deve ser desse tipo e deve conter os elementos correspondentes identificados através do atributo `id` do elemento `div`.

A Figura 20 mostra o diagrama de Atividades referente a esta aplicação hipermídia onde se pode perceber a existência de apenas duas mídias (`video` e `subtitles`), mas três relacionamentos de sincronismo entre elas. Os tempos indicados nas atividades marcadas com o estereótipo `<<TimeAction>>` se referem à mídia de vídeo e o segmento da mídia de texto que será mostrado é indicado após o caractere "@" (arroba) junto à ação que será executada (`start`, no caso). Por exemplo, quando forem decorridos nove segundos do vídeo, será iniciada a exibição do `div` com identificador `sub1` da mídia `subtitles`.

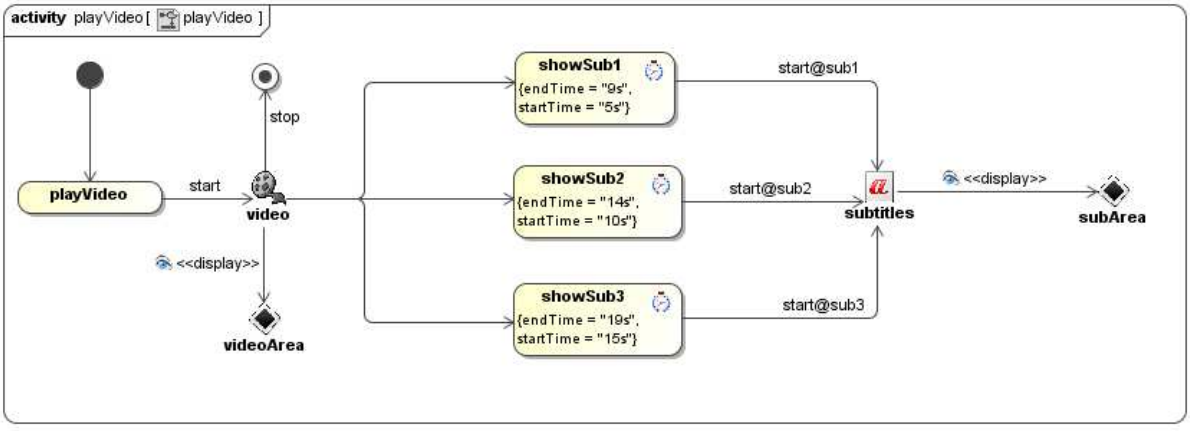


Figura 20 – Diagrama de Atividades da sincronização de um vídeo com partes de um arquivo de texto

Modelar aplicações dessa forma pode diminuir consideravelmente a complexidade dos diagramas do projeto, visto que um objeto de mídia pode concentrar a funcionalidade de dois ou mais objetos. A indicação de início de uma mídia a partir de determinado ponto também pode ser utilizada com mídias contínuas (vídeo e áudio) através da indicação do instante de tempo a partir do qual a mídia será exibida, em segundos.

4.3.6. Exibindo um vídeo em loop até a intervenção do usuário

Esta subseção descreve a primeira aplicação que utiliza elementos de interatividade do perfil UML-TV. Aqui, um vídeo é exibido em *loop* até que o usuário pressione o botão verde do controle remoto. Quando a interação com o usuário ocorrer, outro vídeo é iniciado na mesma área em que o anterior era exibido. Além disso, uma mídia de imagem com o desenho de um botão verde é exibida para indicar ao usuário a oportunidade de interatividade.

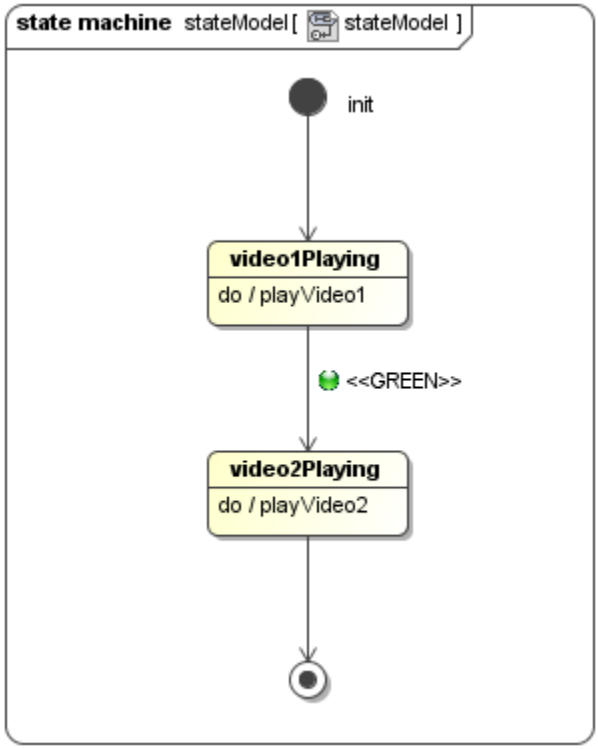


Figura 21 – Diagrama de Máquina de Estados de uma aplicação com interatividade

A Figura 21 mostra o diagrama de Máquina de Estados que modela a situação descrita no parágrafo anterior. Como diferença com relação aos exemplos até

aqui apresentados, destaca-se a adição de um elemento de interação: a aplicação do estereótipo `<<GREEN>>` à transição que parte do estado `video1Playing` para o estado `video2Playing`.

Visualizando a atividade inicial da aplicação, `playVideo1`, na Figura 22, é possível notar que a condição de *loop* imposta pelos requisitos da aplicação é satisfeita através da chamada da mesma ação que dispara o início do vídeo `video1` no momento em que o mesmo vídeo chega ao seu fim. Adicionalmente, a imagem de um botão verde é exibida em uma região da tela à parte.

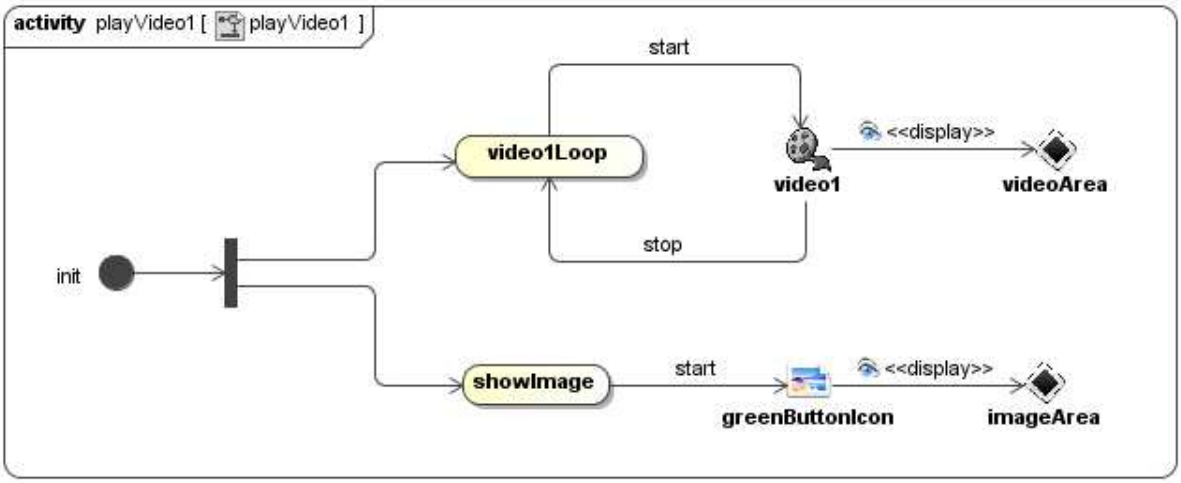


Figura 22 – Diagrama de Atividades da exibição de um vídeo em *loop*

Nesse caso, foi necessário que duas ações fossem utilizadas para especificar o que iria acontecer no começo da atividade, pois uma das ações (`video1Loop`) pode ser reativada em determinado momento. Apesar de serem duas ações diferentes, consideramos neste trabalho que elas acontecem de forma simultânea no início da atividade de acordo com o *fork* posicionado no início do diagrama.

Quando o usuário pressionar o botão verde do controle remoto, a aplicação hipermídia passará por uma mudança de estado indo para o estado `video2Playing` que tem associado a ele a atividade `playVideo2` mostrada na Figura 23.

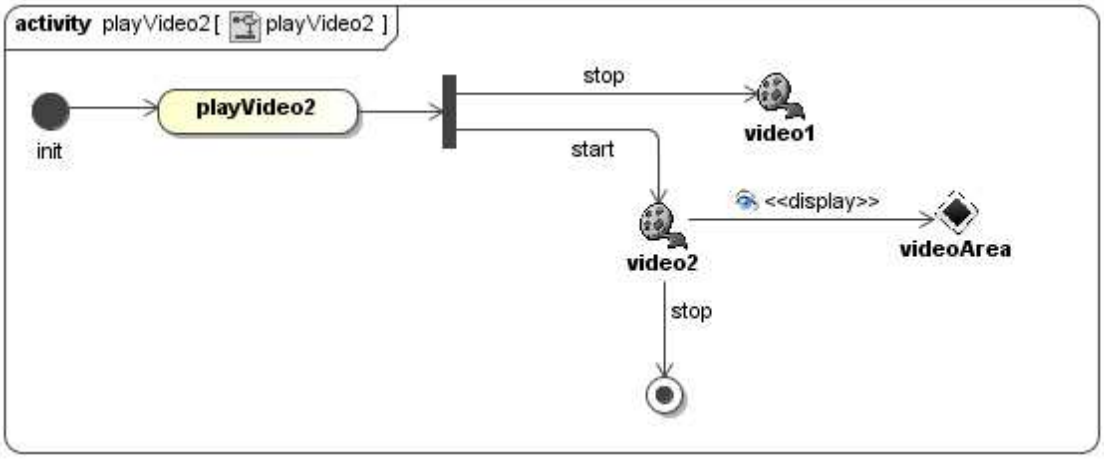


Figura 23 – Diagrama de Atividades da exibição de um vídeo

Nessa atividade, ocorre apenas a simples execução de um vídeo chamado `video2`. Para especificar que a área de visualização onde é exibido o vídeo chamado `video2` é a mesma área onde o vídeo chamado `video1` também é exibido, o mesmo nome para a área visualização (`videoArea`) foi utilizado.

4.3.7. Redimensionando uma região durante a exibição de uma imagem

Assim como os objetos de mídia, as áreas de visualização também têm propriedades como tamanho, posição, etc. Portanto, também deve ser possível alterar esses atributos durante a execução da aplicação hipermídia.

Neste exemplo, a área de visualização onde um vídeo é exibido é redimensionada em determinado momento do vídeo. O instante de tempo onde isso ocorre é definido por uma ação temporal (um elemento Action com o estereótipo `<<TimeAction>>` aplicado). No mesmo instante em que esse redimensionamento ocorre, uma imagem é exibida em outra área já dimensionada de forma que tanto o vídeo quanto a imagem fiquem visíveis. A modelagem desse cenário pode ser vista na Figura 24.

No perfil UML-TV, quando há uma alteração de atributos decorrente da execução de uma ação, os novos valores são indicados na propriedade `name` do fluxo de objetos que liga a ação ao objeto que sofre a alteração. No caso das ações temporais, como ocorre nesse caso, considera-se que após o término da ação, indicado

pelo valor rotulado `end`, os valores dos atributos voltam ao estado em que se encontravam antes do início da ação.

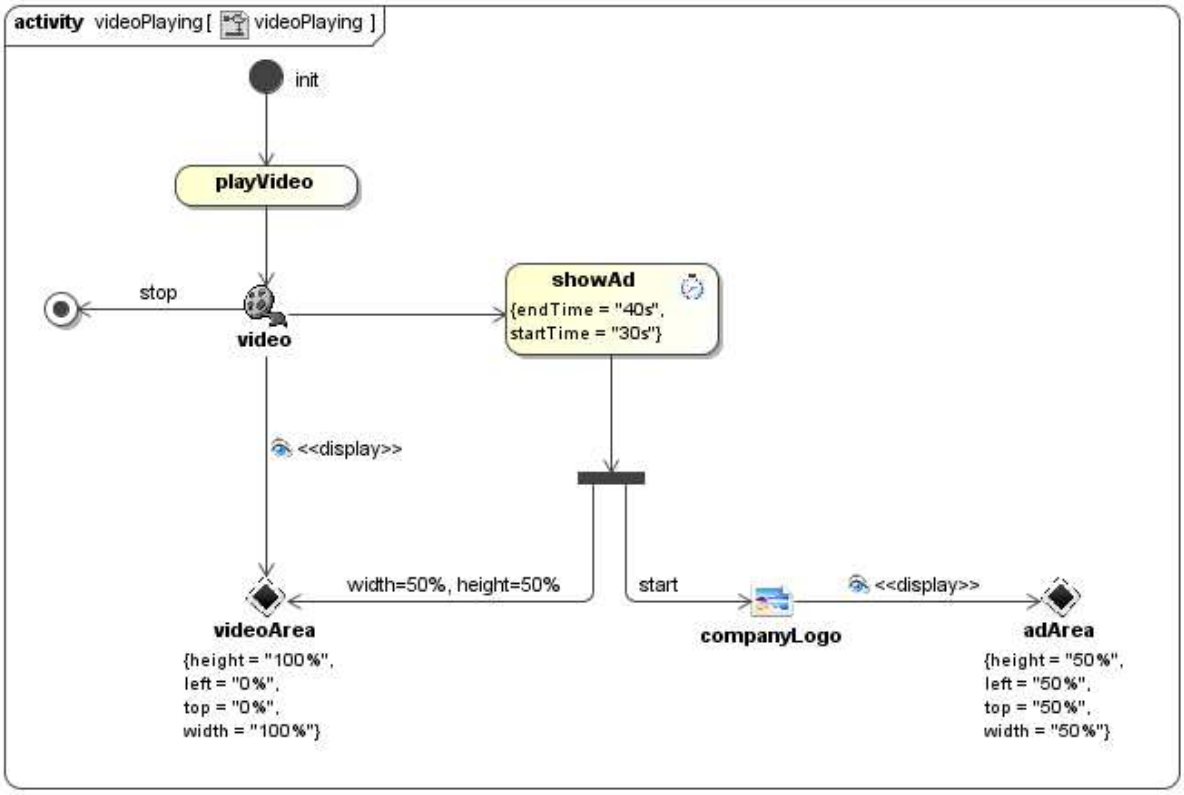


Figura 24 – Diagrama de Atividades da alteração de uma propriedade de uma área de visualização

Como pode ser visto no diagrama, os valores originais das propriedades `height`, `left`, `top` e `width` da área de visualização chamada `videoArea` são 0%, 0%, 100% e 100% respectivamente. Fica explícito que ocorrerá uma ação temporal chamada `showAd` aos trinta segundos do vídeo onde a área de exibição do vídeo sofrerá uma redução de 50% em sua altura e largura tendo seu posicionamento conservado (canto superior esquerdo da tela). Nesse mesmo instante, a imagem chamada `companyLogo` irá ser exibida na área definida por `adArea`, que por sua vez tem como ponto de referência o centro da tela (indicado pelos atributos `top` e `left`) e de tamanho idêntico à outra área de visualização da aplicação.

Ao final da ação temporal, i.e. aos quarenta segundos do vídeo como indicado pelo atributo `endTime` presente na ação `showAd`, a área do vídeo voltará ao seu estado inicial, ou seja, com os valores dos atributos mostrados no diagrama da Figura 24.

4.3.8. Alternando a exibição de dois vídeos

No caso apresentado nesta subseção, é modelada uma aplicação onde o usuário tem a possibilidade de escolher entre dois vídeos qual deles é exibido. Essa escolha é realizada através das teclas verde e vermelha do controle remoto. Uma situação prática dessa funcionalidade pode ser exemplificada como a escolha de um determinado ângulo em que uma cena de um filme foi filmada. Outra aplicação prática que pode ser citada é a escolha de uma entre várias imagens que estão sendo transmitidas durante um evento qualquer, e.g. uma partida de futebol.

Além do vídeo, também é exibida uma imagem que indica para o usuário qual botão deve ser pressionado para que o estado da aplicação seja alterado. Caso o usuário pressione o botão indicado, ocorre a troca do vídeo que esteja sendo exibido como também a mudança da imagem do botão. Assim, durante a execução dessa aplicação, duas visualizações são possíveis: ou são exibidos o primeiro vídeo e a imagem de um botão vermelho, ou o segundo vídeo e a imagem de um botão verde.

O diagrama de Máquina de Estados desse exemplo é mostrado na Figura 25 onde é possível identificar as opções de interação que a aplicação hipermídia proporciona. Durante o estado `playingVideos`, o usuário pode disparar uma transição de estado somente pressionando o botão vermelho, podendo ir para o estado `video2Visible`. Uma vez que a aplicação esteja nesse estado, é possível mudar para o estado `video1Visible` através da tecla verde do controle remoto. Nota-se que o estado inicial não é mais alcançável após a interação do usuário E que, estando no estado `video1Visible`, a aplicação pode voltar ao estado `video2Visible` caso o usuário pressione o botão vermelho novamente.

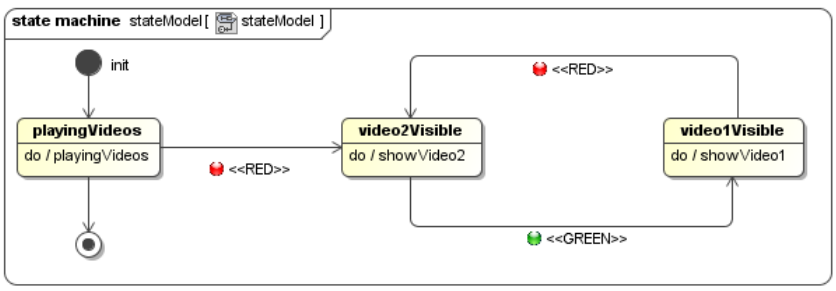


Figura 25 – Diagrama de Máquina de Estados de uma aplicação de alternância entre dois vídeos

Na Figura 26, vemos o detalhamento do que ocorre no primeiro estado da aplicação, que é representado pelo diagrama de Atividades chamado `playingVideos`. Nessa atividade, composta apenas por uma ação, ocorre a inicialização das mídias utilizadas na aplicação hipermídia. Os dois vídeos e as duas imagens são iniciados, mas apenas um vídeo e uma imagem devem ficar visíveis para o usuário. Para isso, os atributos `visible` das mídias que devem ser ocultadas são definidos com o valor `false` enquanto que as mídias que devem ser exibidas ficam inalteradas, assumindo assim seus valores padrões para os atributos. Também nesse diagrama, as áreas da tela da televisão onde serão exibidas as mídias são definidas.

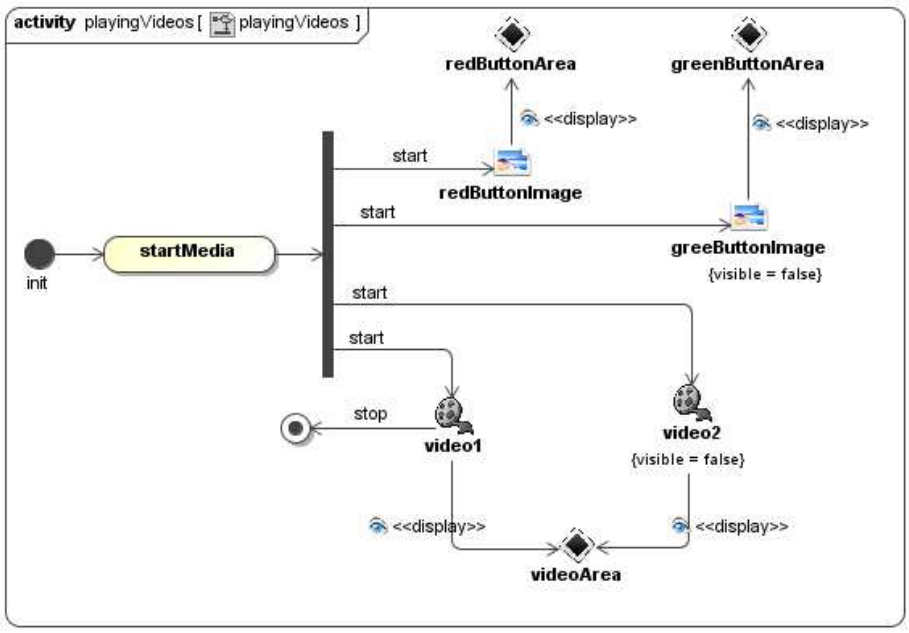


Figura 26 – Diagrama de Atividades onde algumas mídias são iniciadas com valores rotulados diferentes dos valores padrão

É mostrado na Figura 27 como a aplicação se comporta quando o botão vermelho do controle remoto é pressionado. Já que mídias foram iniciadas no primeiro estado da aplicação hipermídia, ocorrem apenas alterações nos atributos das mesmas: a mídia `video2` fica visível e com volume máximo (100%), a mídia `video1` fica invisível e com volume mínimo (mas ainda sendo executada), a imagem `redButtonImage` é ocultada e a imagem `greenButtonImage` é exibida.

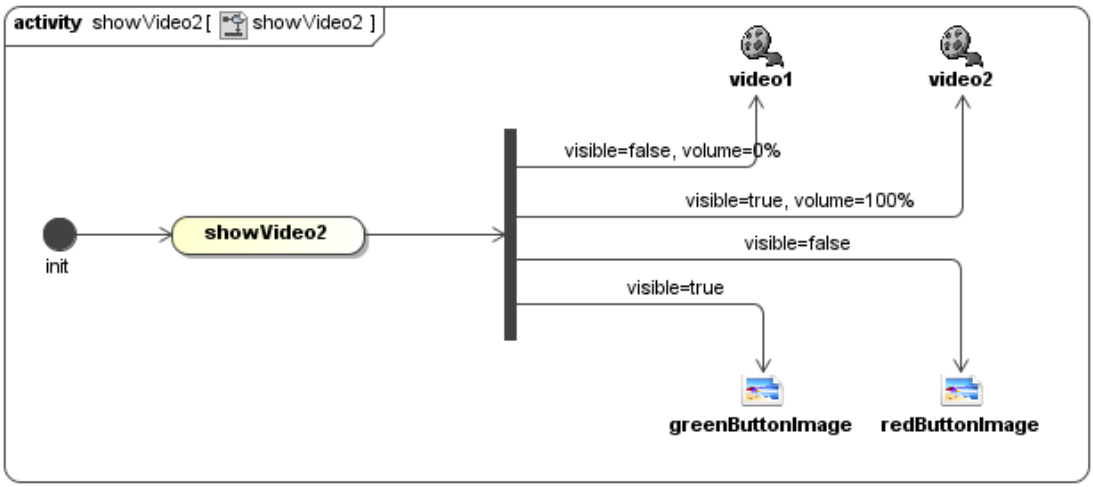


Figura 27 – Diagrama de Atividades referente ao pressionamento do botão vermelho do controle remoto

Vemos na Figura 28 o diagrama de Atividades onde é modelado o comportamento do que ocorre na aplicação quando o usuário pressiona o botão verde do controle remoto, o qual só pode ser alcançado se a aplicação estiver no estado `video2Visible`, como foi mostrado no diagrama de Máquina de Estados. Como ilustrado no diagrama anterior, apenas mudanças de valores rotulados acontecem, sendo essas mudanças opostas àquelas ocorridas caso o botão vermelho seja pressionado.

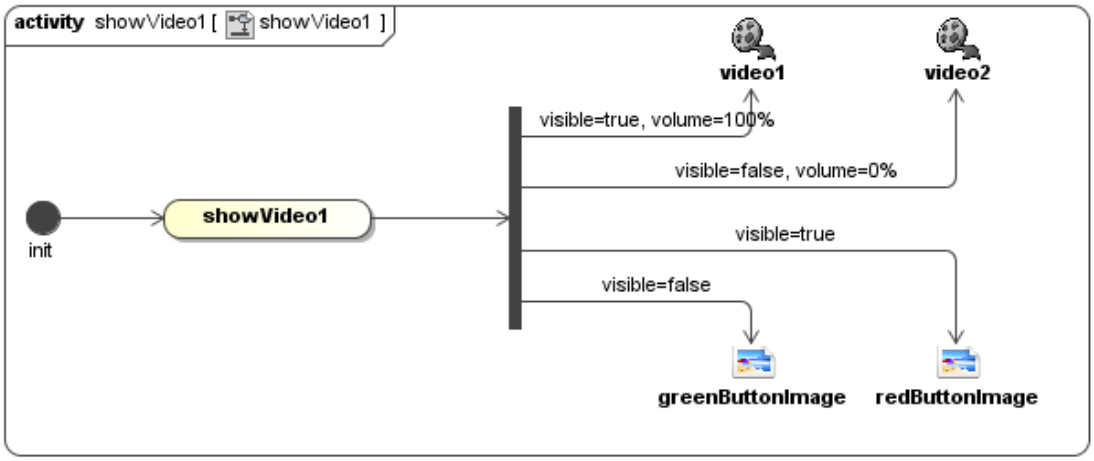


Figura 28 – Diagrama de Atividades referente ao pressionamento do botão verde do controle remoto

4.3.9. Simulação de um menu de DVD

Nesta subseção, é mostrado como uma escolha do usuário pode ser utilizada para definir qual mídia é executada em um momento posterior da aplicação. Um

simples menu de DVD pode ser usado como uma analogia para esse caso. Um vídeo inicial fica sendo executado em *loop* junto a duas imagens que indicam qual idioma será ouvido durante a exibição do vídeo principal. Quando o usuário fizer sua escolha pressionando ou o botão verde ou o botão vermelho do controle remoto, um segundo vídeo intermediário é executado independentemente da escolha do idioma. Após o vídeo intermediário, o vídeo principal é então exibido sincronamente ao áudio escolhido no início da interação.

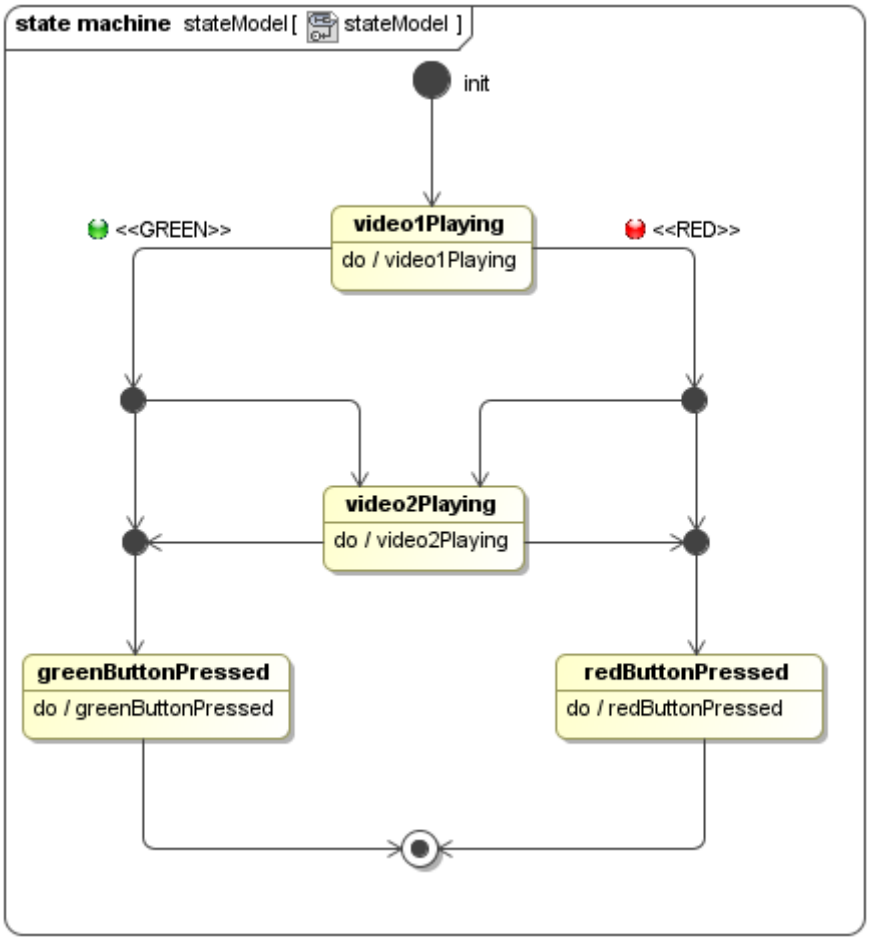


Figura 29 – Diagrama de Máquina de Estados de um menu de DVD

Como pode ser visto na Figura 29, a principal característica deste exemplo é a necessidade de "memorização" da escolha do usuário para uso posterior, pois a escolha ocorre durante o primeiro estado, mas o resultado da decisão de qual áudio executar não é utilizado. no estado imediatamente seguinte. Para permitir essa funcionalidade, foi utilizado o elemento *Junction* (junção) da UML.

De acordo com a especificação da UML (Object Management Group, 2009b), uma junção é um vértice de semântica livre utilizado para encadear múltiplas transições. Por exemplo, uma junção pode ser utilizada para convergir várias transições de entrada em uma única transição de saída, representando assim um caminho compartilhado (também conhecido como *merge*). Por outro lado, junções podem ser usadas para dividir uma transição de entrada em diversas transições de saída.

A função das junções no perfil UML-TV é indicar qual fluxo deve ser seguido quando se faz necessária que uma ação do usuário tenha reflexo em um estado que não seja exatamente posterior ao estado em que a aplicação se encontre. Como pode ser notado na Figura 29, ao sair do estado `video1Playing`, a seta de transição aponta para uma junção com duas saídas que apontam para o estado intermediário `video2Playing` e para outra junção que, por sua vez, tem como entradas a saída da junção anterior e a transição iniciada pelo estado intermediário `video2Playing` que ocorre ao término do referido estado. Para efeitos de modelagem no perfil UML-TV, isso significa que o momento em que os estados `greenButtonPressed` e `redButtonPressed` serão executados depende do término do estado `video2Playing`. Já qual dos dois estados será executado depende do caminho seguido pelas transições, ou seja, por quais junções a aplicação passou de acordo com o botão pressionado no controle remoto.

A Figura 30 mostra o que foi modelado para o estado inicial deste exemplo. Assim como em uma situação previamente ilustrada neste capítulo, nota-se a necessidade de execução de duas ações concorrentemente porque a ação chamada `video1Loop` pode ser reativada caso o vídeo `video1` chegue ao seu fim, tendo que ser reiniciado. A outra ação, `showImages`, simplesmente inicia a exibição das duas imagens que indicarão ao usuário a possibilidade de escolha de idioma do áudio.

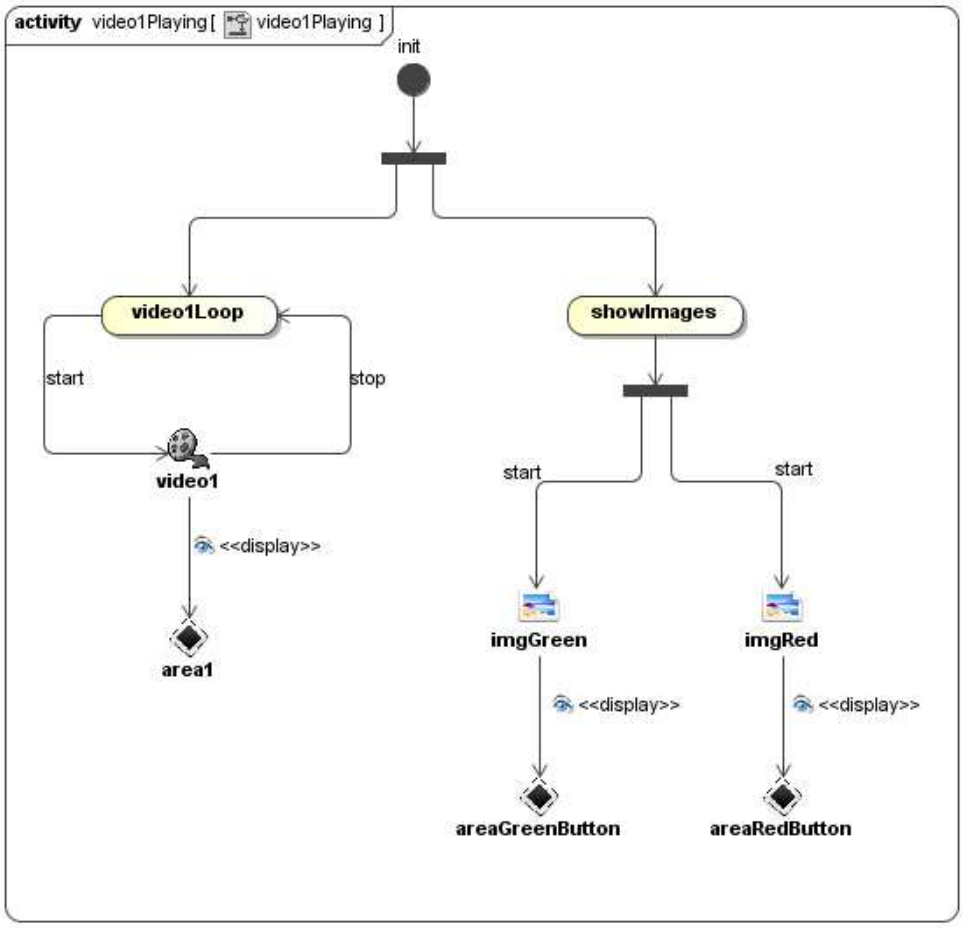


Figura 30 – Diagrama de Atividades inicial de um menu de DVD

A exibição do vídeo intermediário é ilustrada na Figura 31, onde é possível verificar que o vídeo chamado `video2` será exibido na área da tela delimitada pela área chamada `area1`, a mesma área em que a mídia `video1` estava sendo exibida como foi especificado no diagrama da Figura 30.

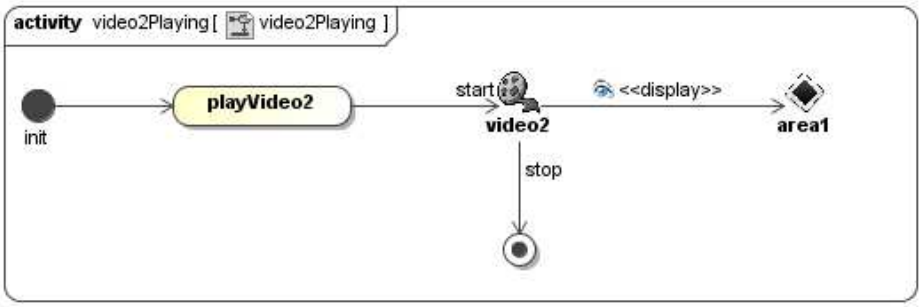


Figura 31 – Diagrama de Atividades da exibição de um vídeo intermediário anterior à execução do vídeo principal

Na Figura 32, estão os dois diagramas que encerram a modelagem deste exemplo: os diagramas de Atividades `greenButtonPressed` e

`redButtonPressed`. É fácil identificar que se o usuário pressionar o botão verde do controle remoto, o idioma escolhido será o Português (representado pela mídia `audioPt`). Caso contrário, o idioma que o usuário terá escolhido o idioma Inglês (representado pela mídia `audioEn`).

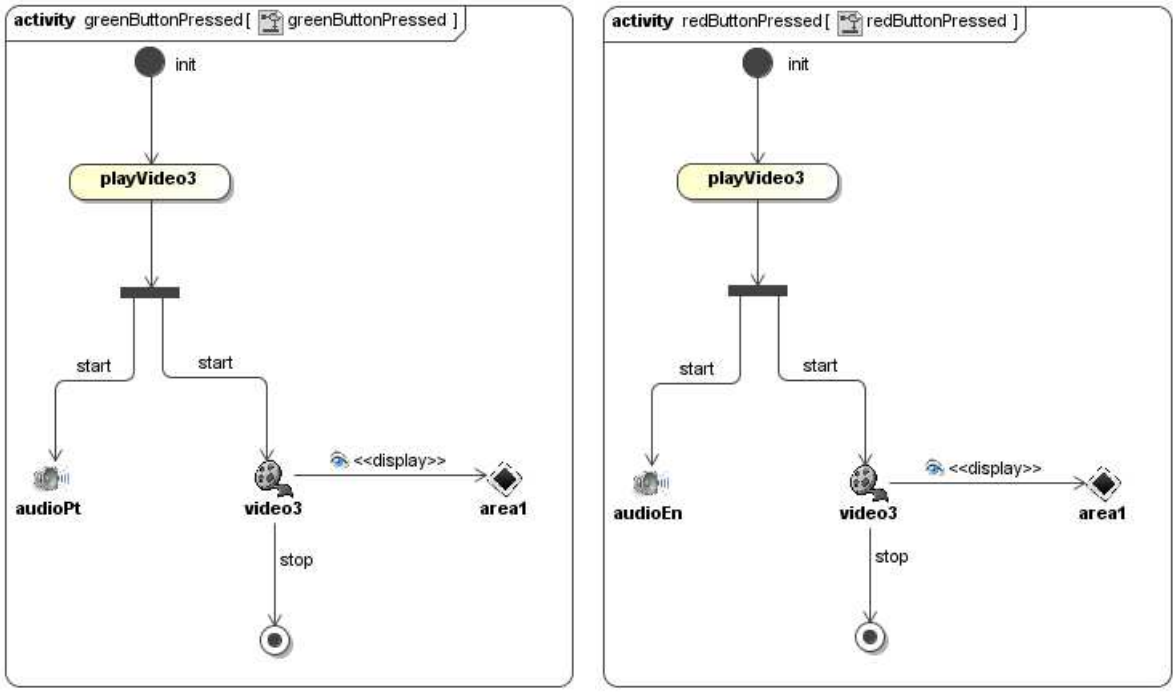


Figura 32 – Diagramas de Atividades da execução do vídeo principal caso o idioma escolhido tenha sido o Português (esquerda) ou o Inglês (direita)

4.3.10. Seleção através das setas do controle remoto

Os diagramas de Máquina de Estados até aqui apresentados foram relativamente simples por apresentarem poucas opções de interação com o usuário. Foram mostradas situações em que o usuário poderia atuar sobre a aplicação através de, no máximo, dois botões do controle remoto. No caso apresentado nesta subseção, é modelado um menu onde cada opção leva a um jogo quando o botão "OK" é pressionado. Qual jogo é iniciado depende do estado no qual se encontra a aplicação hipermídia quando o botão "OK" é pressionado, como mostra a modelagem realizada no diagrama da Figura 33.

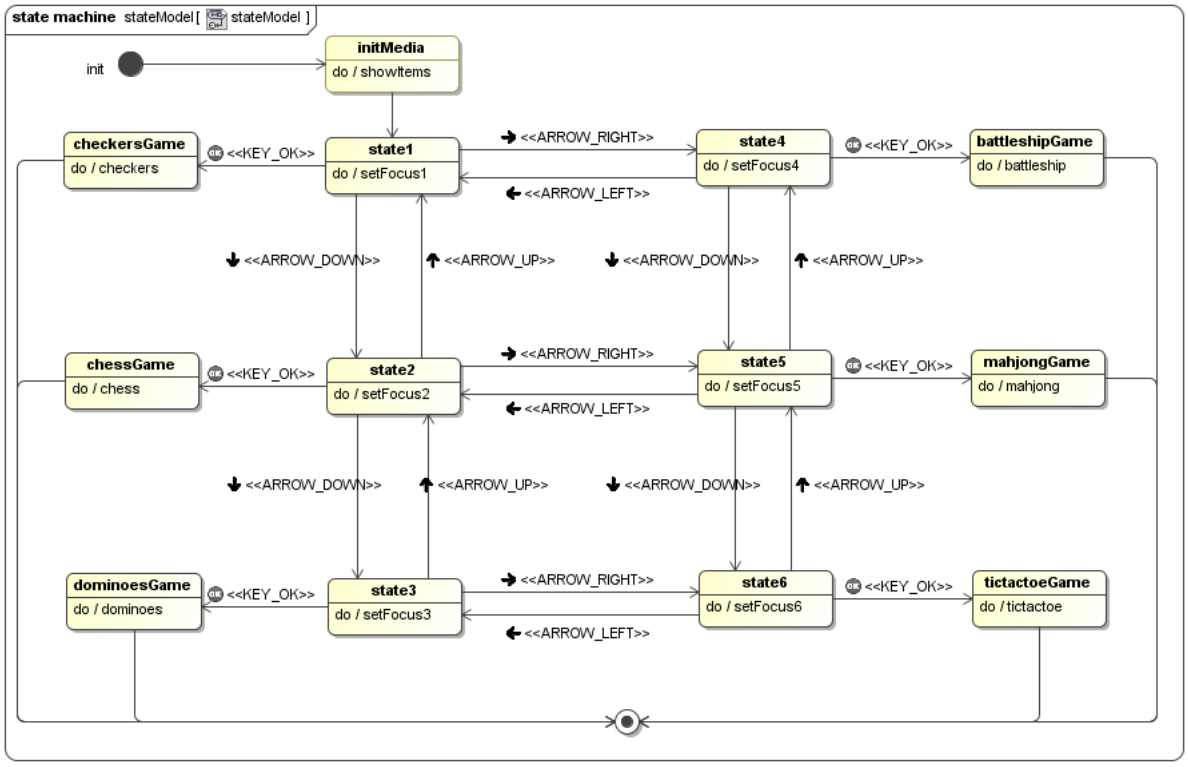


Figura 33 – Diagrama de Máquina de Estados com várias possibilidades de interação

De acordo com o diagrama de Máquina de Estados da Figura 33, o menu exemplificado tem seis opções, cada uma associada a um jogo. Verifica-se que o primeiro estado visitado é o que está representado com o nome `initMedia`, que tem a função de iniciar a exibição das mídias de texto em suas respectivas áreas de visualização. O diagrama de Atividades associado a esse estado é ilustrado na Figura 34. Nele, pode-se verificar que a propriedade `focus` de cada uma das áreas de visualização é alterada de forma que a região chamada `area1` tenha o foco no início da aplicação. Como este diagrama não conta com nenhuma mídia contínua, não há a necessidade de adicionar um nó final ao diagrama. Dessa forma, assume-se que ao final da inicialização das mídias de texto, a atividade é encerrada e a aplicação passa para o próximo estado do diagrama de Máquina de Estados.

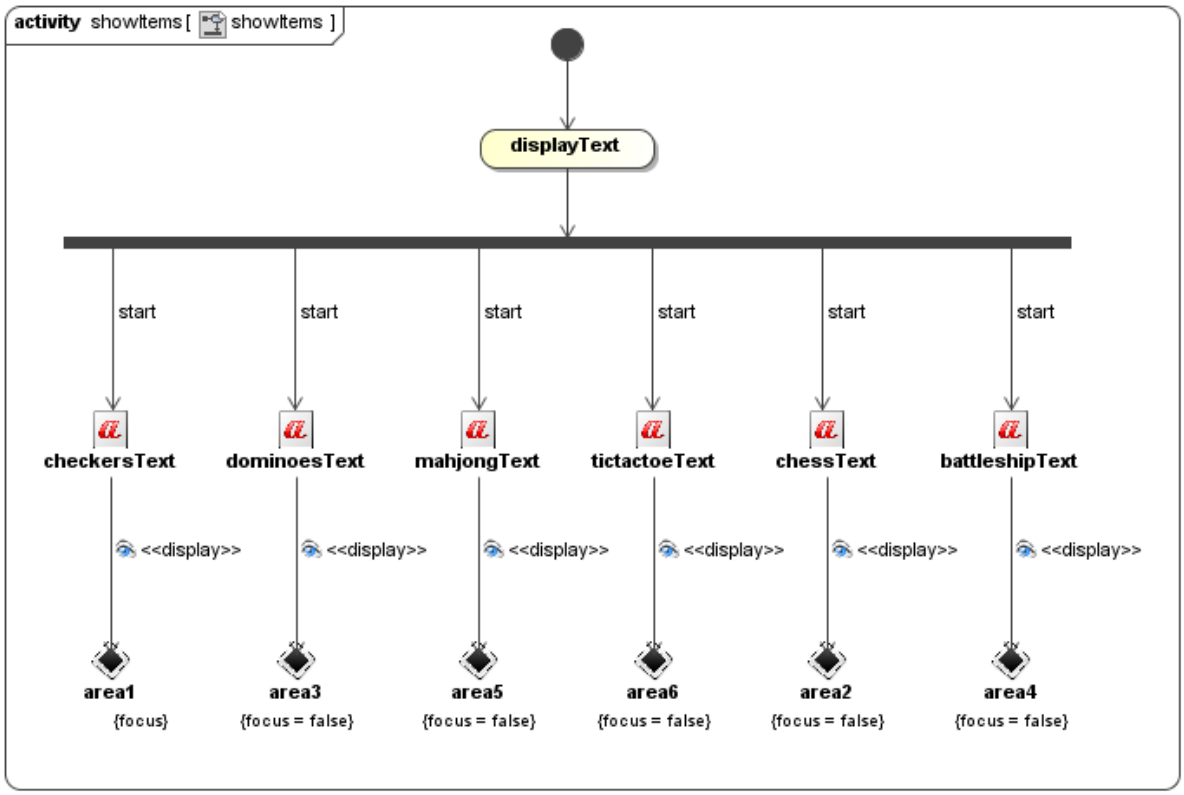


Figura 34 – Diagrama de Atividades que especifica as mídias que são exibidas e suas respectivas localizações na tela

A partir do segundo estado, a aplicação pode sofrer transições para outros estados que alteram qual das seis áreas é destacada através da alteração do valor rotulado `focus` pertencente ao estereótipo `<<DisplayArea>>`. O segundo estado da aplicação chama-se `state1`, cujo diagrama de Atividades associado é o diagrama chamado `setFocus1` (ilustrado na Figura 35) e é executado logo depois que as mídias são exibidas.

Na primeira vez em que esse estado é visitado pela aplicação, as alterações realizadas no valor rotulado `focus` das áreas de visualização `area1`, `area2` e `area4` são a mesma alterações realizadas no estado `initMedia` (o primeiro da aplicação), portanto não surtem nenhum efeito visual para o usuário. Este estado tem importância caso o usuário navegue pelos outros itens do menu e eventualmente volte a selecionar o primeiro item. Nesse caso, o retorno a este estado só pode ocorrer se o foco estiver nas regiões chamadas `area2` e `area4`. Assim, a propriedade `focus` somente dessas duas áreas de visualização são alteradas para `false`.

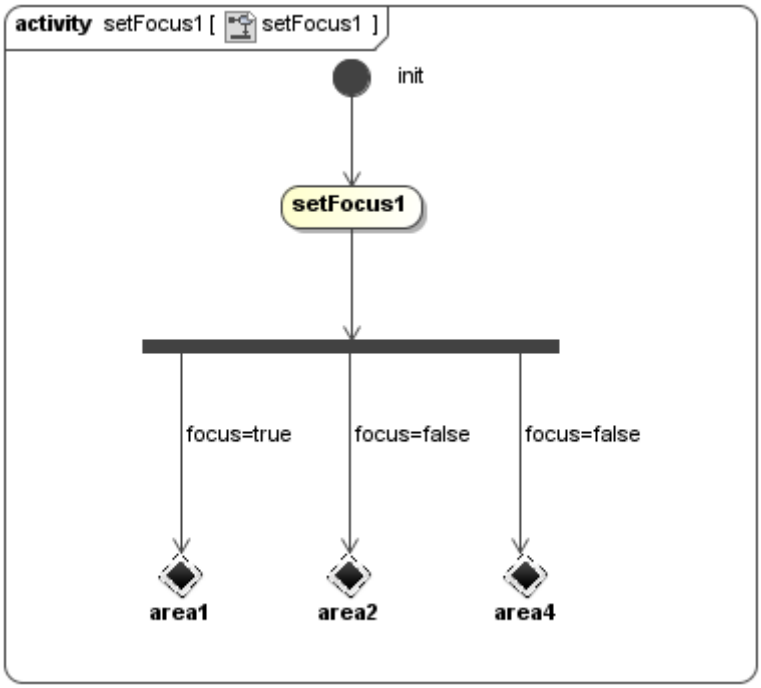


Figura 35 – Diagrama de Atividades representando a seleção do primeiro item de um menu

Os outros diagramas de Atividades simplesmente indicam qual área será destacada, assim como o diagrama de Atividades mostrado anteriormente, portanto, optamos por omiti-los. Os exemplos até aqui apresentados ilustra grande parte das possibilidades de utilização do perfil UML-TV e, apesar de serem simples em sua maioria, constituem-se em uma forma didática de apresentação do perfil. Na Seção 6 é mostrada a modelagem de uma aplicação mais complexa e próxima de uma situação real de uso do perfil UML-TV.

5. TRANSFORMAÇÕES DE MODELOS

Em 2004, a OMG lançou uma requisição de propostas (Object Management Group, 2004) com o objetivo de padronizar a transformação de modelos para representação textual, onde foi solicitada a criação de uma nova linguagem ou a extensão de uma linguagem já existente para transformação de modelos em texto. Estavam entre os principais requisitos dessa linguagem de transformação a compatibilidade com o metamodelo MOF 2.0, funções de manipulação de *strings* e o suporte aos mecanismos de extensão da UML.

A linguagem MOFScript atende a todos os requisitos básicos da OMG e adiciona outros requisitos avaliados como importantes para uma linguagem desse tipo, tais como suporte a mecanismos de controle de fluxo, interação com serviços do sistema operacional (e.g. consultar a data e hora do sistema), e um equilíbrio entre facilidade de uso e poder de expressividade dos domínios. Também é disponibilizado um editor de transformações na forma de um *plug-in* para a IDE Eclipse, chamado *MOFScript tool* (OLDEVIK, 2006), que possui editor léxico com coloração de sintaxe e assistente de digitação de código, visualizador estrutural de regras, gerenciador de configurações e visualizador de problemas. MOFScript já provou ser adequada para a tarefa de transformação de modelos em UML para texto em outras situações como no trabalho de Porres et al. (2007) onde sistemas de apoio à decisão foram gerados a partir de diagramas de Máquina de Estados, e no trabalho de Javed, Strooper e Watson (2007) onde foi utilizada para gerar casos de testes. Um conjunto completo de instruções de uso da ferramenta e da linguagem pode ser encontrado em seu manual do usuário (OLDEVIK, 2009).

Neste trabalho, as regras necessárias para a transformação dos diagramas de Máquinas de Estados e de Atividades que compõem os modelos em conformidade com o perfil UML-TV em documentos NCL foram escritas em MOFScript. Dessa forma, para as aplicações suportadas pelo perfil, i.e. aplicações básicas com sincronismo de mídias e interação do usuário localmente (sem canal de retorno), não foi necessária a criação de PSMs, pois as informações necessárias para a criação dos documentos NCL pode ser extraída diretamente do PIM.

A metodologia utilizada na criação do *script* de transformação foi baseada na estrutura de um documento NCL (criação do cabeçalho, definição das regiões, descritores, etc.), capturando as devidas informações nos modelos e criando as regras de transformação da forma mais geral possível. As subseções seguintes contêm a descrição de algumas dessas regras.

5.1. Geração da base de descritores

A regra de transformação `getAllDescriptors` é responsável por obter e retornar em uma lista as referências de todos os descritores de um diagrama de Atividades. Para isso, dado um diagrama de Atividades, é feita uma consulta que busca os elementos do tipo `ObjectFlow` que possuam o estereótipo `<<display>>` aplicado. O código fonte da transformação é apresentado a seguir.

```
uml.Activity::getAllDescriptors():list{
  var l:list
  self.ownedElement->forEach(objectFlow:uml.ObjectFlow){
    if (objectFlow.hasStereotype("display")) {
      l.add(objectFlow)
    }
  }
  return l

}
```

Ao longo do código de transformação, essa regra é chamada sempre que uma nova atividade é encontrada para que os relacionamentos entre mídias e áreas de visualização sejam identificados. Depois que a lista de elementos do tipo `ObjectFlow` é obtida, uma filtragem sobre ela é realizada para assegurar que um

mesmo descritor não seja adicionado mais do que uma vez no documento NCL resultante. Finalmente, o descritor é criado utilizando o nome da mídia presente na propriedade `source` do elemento `ObjectFlow` e o nome da área de visualização presente na propriedade `target` do elemento `ObjectFlow`.

5.2. Geração da base de regiões

Para a construção da base de regiões do documento NCL, foi criada a regra `getAllDisplayArea` que retorna em uma lista todas as áreas de visualização presentes em uma atividade de acordo com o trecho de código-fonte a seguir. Essa regra é utilizada todas as vezes que uma atividade é encontrada durante a função principal do *script* de transformação.

```
uml.Activity::getAllDisplayArea():list {
    var l:list
    self.ownedElement->forEach(node:uml.CentralBufferNode){
    if (node.hasStereotype("DisplayArea")){
     l.add(node)
    }
  }
  return l

}
```

Uma vez que todas as áreas de visualização são identificadas, a mesma filtragem realizada para os descritores é feita com o objetivo de eliminar as regiões referenciadas mais de uma vez nos modelos da aplicação. Ao fim desse processo, as *tags* do documento NCL referentes às regiões são criadas com os atributos de `DisplayArea` (`top`, `left`, `width` e `height`).

5.3. Geração dos nós de mídia

Em NCL, um nó de mídia descreve, no mínimo, o caminho do arquivo que contém a mídia e o descritor que será responsável pela apresentação da mídia. Para capturar todas as mídias envolvidas na aplicação, foi utilizada a regra `getAllMedias`

mostrada no fragmento de código abaixo, que retorna uma lista com os elementos `ObjectNode` que possuam algum estereótipo de mídia do perfil UML-TV.

```
uml.Activity::getAllMedias():list {
  var l: list
  self.ownedElement->forEach(node:uml.CentralBufferNode){
    if (node.hasStereotype("Video") ||
                node.hasStereotype("Text") ||
                node.hasStereotype("Audio") ||
                node.hasStereotype("Image")) {

      l.add(node)

    }
  }
  return l

}
```

Ao adicionar cada mídia no documento NCL resultante, é realizado um cruzamento de informações com o retorno da regra que cria os descritores da aplicação para que os respectivos descritores sejam referenciados nos nós de cada mídia.

5.3.1. Geração de âncoras

O conceito de âncoras de NCL pode ser mapeado para elementos do tipo `TimeAction` do perfil UML-TV. Dessa forma, a regra auxiliar `getAncoras` foi adicionada ao *script* de transformação. Essa regra, mostrada no quadro abaixo, é chamada para cada atividade do modelo.

```
uml.Activity::getAncoras():list {
  var l: list
  self.ownedElement->forEach(node:uml.Action){
    if (node.hasStereotype("TimeAction")){
      l.add(node)
    }
  }
  return l

}
```

Durante a criação dos nós de mídia, a lista de âncoras é utilizada como fonte das informações necessárias para a criação dos nós `area`, filhos do nó `media` na linguagem NCL. Esses nós possuem as propriedades `name` (mapeado no atributo `name`

da `TimeAction`), `begin` (mapeado no atributo `startTime`), e `end` (mapeado no atributo `endTime`).

As regras apresentadas neste capítulo constituem-se nas principais funções auxiliares que são chamadas no decorrer da função principal (regra `main`) do script de transformação. Existem outras regras secundárias para verificação de outras características das aplicações como a existência de sincronismo, necessidade de conectores, etc. além de funções para recuperação do conteúdo de valores rotulados. Dessa forma, as regras criadas se constituem em um conjunto genérico que foi utilizado para a extração de todos os dados necessários para a transformação dos modelos de alto nível do perfil UML-TV em código-fonte específico para a linguagem NCL.

6. ESTUDO DE CASO

Com o intuito de validar a utilização do perfil UML-TV como uma ferramenta capaz de modelar uma aplicação para televisão digital e validar a corretude das regras de transformação escritas, foi realizado o estudo de caso apresentado neste capítulo. A aplicação escolhida para estudo de caso constitui-se em um programa sobre saúde chamado "Viva Mais" (BECKER; SOARES, 2009), onde o usuário telespectador tem a possibilidade de interagir com o programa escolhendo um entre quatro pratos que contêm diferentes combinações de alimentos. A aplicação interativa consiste em informar ao usuário as características do prato escolhido e se o mesmo pode ser considerado como uma escolha saudável ou não. O diagrama de Máquina de Estados inicial (que tem o estereótipo `<<Container>>` aplicado) é mostrado na Figura 36, o qual contém apenas um estado que por sua vez referencia uma atividade chamada `play`.

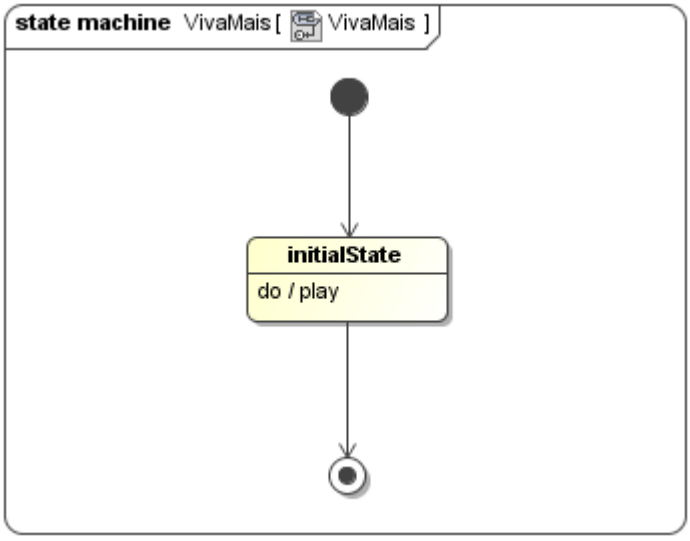


Figura 36 – Diagrama de Máquina de Estados inicial da aplicação

A Figura 37 mostra o diagrama de Atividades da atividade `play`. No diagrama, pode-se notar o vídeo principal sendo exibido em sua respectiva área (`rgVideo`) que ocupa toda a tela da TV. Aos 63 segundos do vídeo, a ação chamada `showIcon` é executada, onde uma imagem chamada `icon` é exibida com o objetivo de indicar para o usuário uma oportunidade de interação com o programa. Essa imagem fica visível por onze segundos, pois é exibida dos 63 segundos até os 74 segundos contados a partir do início do vídeo principal. Durante esse período, o usuário poderá pressionar o botão vermelho do controle remoto ou não. Caso o usuário opte pela interação, o fluxo da aplicação segue pela seta de fluxo contendo o estereótipo `<<RED>>`.

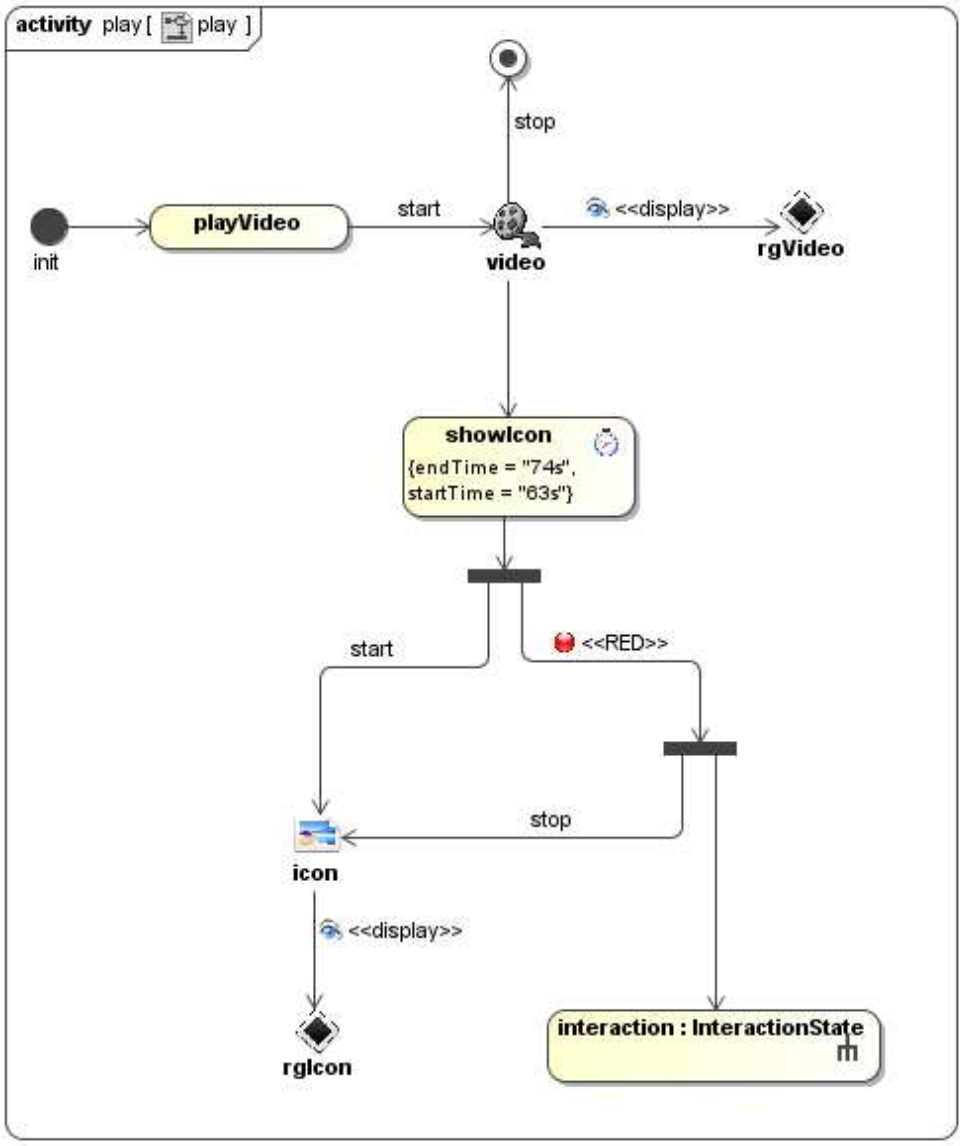


Figura 37 – Diagrama de Atividades do estado inicial da aplicação

Aqui, uma nova característica do perfil UML-TV é utilizada. Como a escolha dos pratos só pode ser alcançada em um determinado momento da aplicação e esse momento é diretamente relacionado a um determinado instante da execução de uma mídia (o vídeo), faz-se necessária a referência a um estado alternativo, que pode ou não ser acessado pelo usuário. Isso é modelado no exemplo deste estudo de caso através da ação de nome `interaction` que referencia um diagrama de Máquina de Estados através de sua propriedade `behavior`. Assim, foi adicionada a possibilidade de interação através do botão vermelho do controle remoto que, se pressionado durante o tempo especificado na ação `showIcon`, irá ativar o estado `InteractionState`, detalhado na Figura 38.

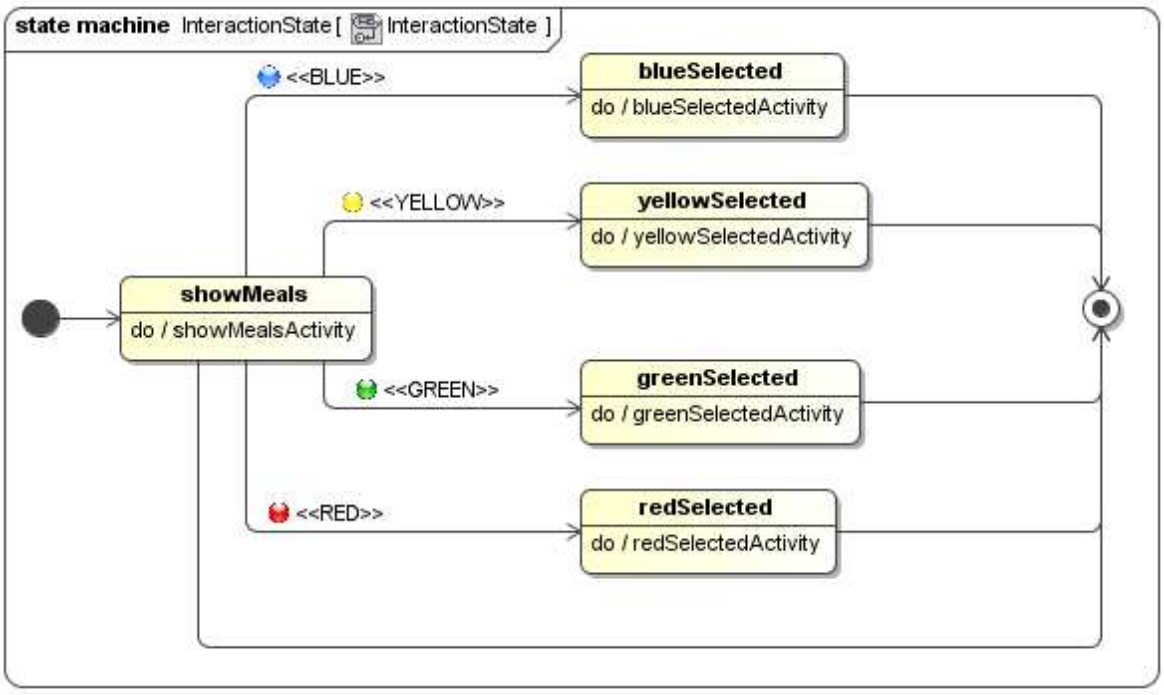


Figura 38 – Diagrama de Máquina de Estados ativado caso o usuário opte por interagir com a aplicação

Uma vez pressionado o botão vermelho durante o tempo especificado na ação `showIcon` (Figura 37), dois dos estados presentes no diagrama de Máquina de Estados `InteractionState` podem ser alcançados. O primeiro deles é aquele que ocorre quando a interação é iniciada, representado pelo estado `showMeals`, detalhado na Figura 39. No diagrama de Atividades `showMealsActivity`, podemos verificar que as imagens dos quatro pratos e uma imagem de fundo são iniciadas juntamente a

um texto informativo (`infoText`). Além disso, a região onde o vídeo era previamente exibido sofre um redimensionamento, resultando na imagem apresentada na Figura 40.

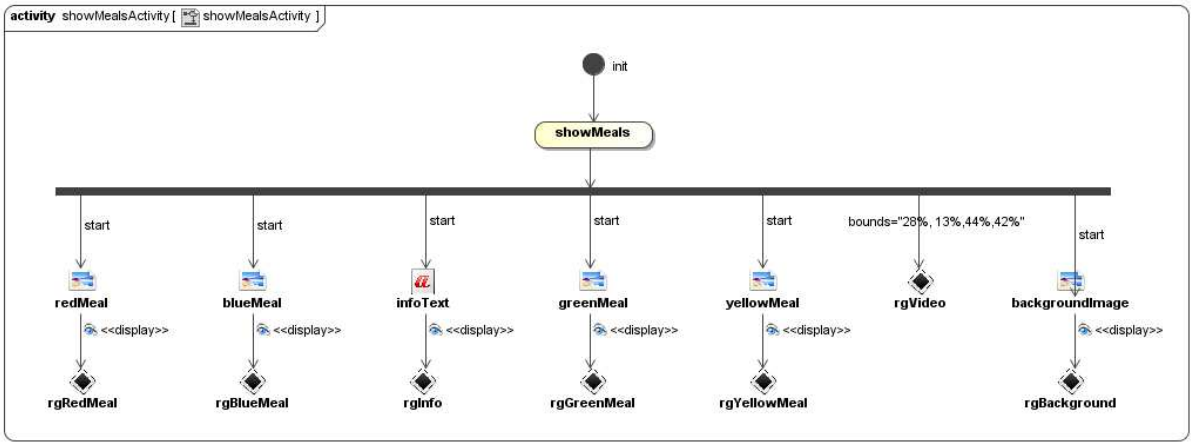


Figura 39 – Diagrama de Atividades detalhando o que ocorre num primeiro momento caso o usuário interaja com a aplicação

Figura 40 – Aplicação interativa em execução

O outro estado que pode ser alcançado constitui-se na exibição do prato escolhido pelo usuário junto a uma descrição do prato, relativa às suas propriedades nutritivas. As situações possíveis nesse caso são representadas na Figura 38 pelos estados `blueSelected`, `yellowSelected`, `greenSelected`, e `redSelected`. A Figura 41 ilustra o caso em que o usuário escolhe o prato representado pela cor verde, através do pressionamento do botão verde do controle remoto (que foi definido com a aplicação do estereótipo `<<GREEN>>` à transição do estado `showMeals` para o estado

`greenSelected`). No diagrama, nota-se que a exibição das imagens dos demais pratos é interrompida, deixando apenas o prato selecionado visível. Além disso, o texto informativo inicial é substituído pela descrição do prato escolhido (representado pela mídia `greenMealText`). O resultado dessa operação na aplicação real pode ser observado na Figura 42.

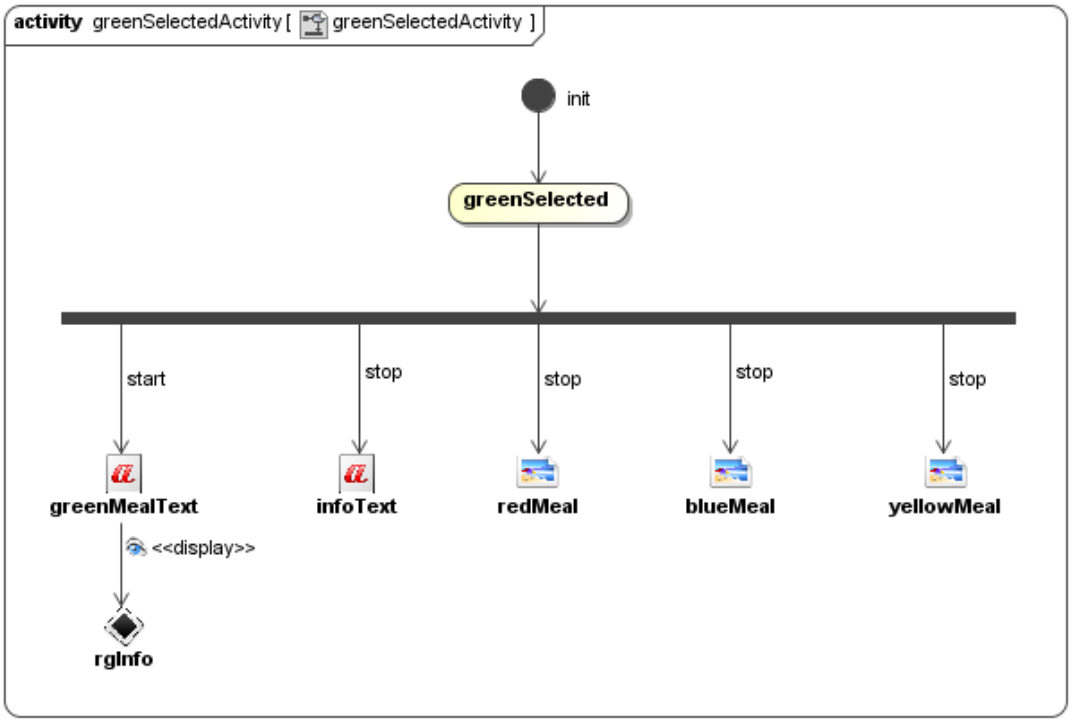


Figura 41 – Diagrama de Atividades que modela a escolha do usuário pelo prato representado pela cor verde

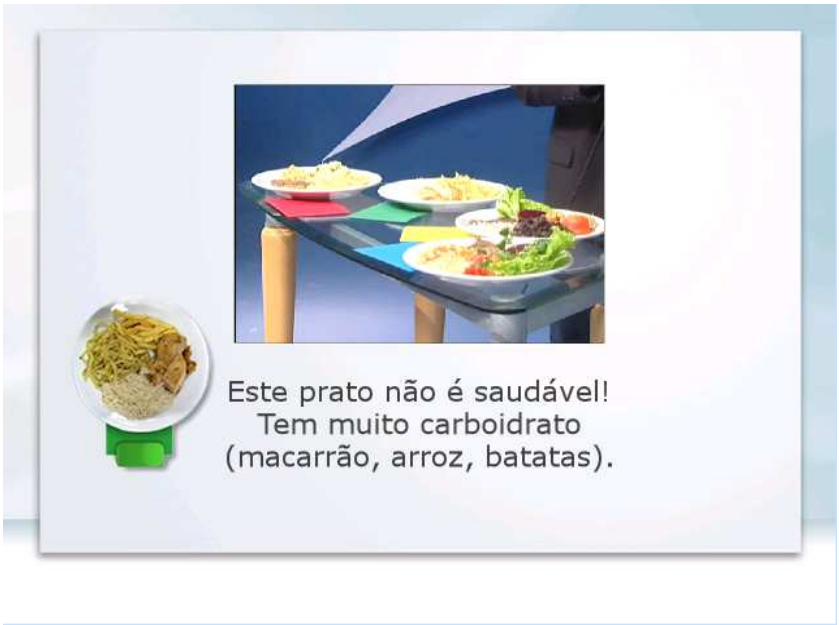


Figura 42 – Imagem da aplicação interativa em execução referente ao diagrama da Figura 41.

Todos os modelos gerados por este exemplo foram submetidos às transformações já descritas no Capítulo 5. O resultado da transformação pode ser executado com sucesso no emulador NCL disponível para testes de aplicações (Telemídia, 2008).

O código gerado para este estudo de caso através do script de transformação (disponível no Apêndice A) não é totalmente igual ao código original de Becker e Soares (2009). Isso se deve ao fato de que nem todos os conceitos de NCL foram mapeados no código de transformação, como, por exemplo, o uso de nós de contexto. Em NCL todo nó de mídia é definido dentro de um contexto, onde o elemento `body` é o contexto que contém todos os nós do documento, sejam eles nós de mídia ou outros contextos. Na transformação produzida neste trabalho, todas as mídias foram inseridas no contexto `body`, enquanto que no código original é utilizado um contexto específico para isso. O código resultante é diferente em comparação com o documento criado pelos autores, mas vale ressaltar que, apesar disso o seu comportamento é totalmente equivalente ao código original da aplicação.

7. CONSIDERAÇÕES FINAIS

Nesta dissertação, foi apresentado o perfil UML-TV, uma extensão da UML criada com o objetivo de facilitar a modelagem de aplicações hipermídia para televisão digital. O perfil foi apresentado através de exemplos de utilização e um estudo de caso onde todos os aspectos das aplicações foram modelados com sucesso. Embora a UML possa não ser uma linguagem familiar para a maioria dos *experts* do domínio de televisão digital, a curva de aprendizado nesse caso pode ser diminuída, pois o perfil UML-TV é uma extensão de uma pequena parte da UML e recursos gráficos como ícones foram utilizados o máximo possível.

Além da contribuição principal, o perfil UML-TV, foram escritas transformações que proporcionaram a geração automatizada de código-fonte para aplicações declarativas na linguagem NCL suportando o sincronismo entre as mídias e a interatividade local com o usuário.

O processo de transformação apoiado pela linguagem MOFScript apresentado neste trabalho constitui-se em um primeiro esforço em busca da automatização da geração de aplicações para televisão digital a partir de diagramas de Máquinas de Estados e de Atividades. Como todos os elementos necessários para a geração de código puderam ser capturados diretamente no modelo independente de plataforma, foi possível contornar uma parte do processo proposto pela MDA (a geração de modelos intermediários) mantendo a aplicação de conceitos de MDSE e provendo uma base para a continuidade do processo de transformação que contemple a modelagem de aplicações mais complexas e que possibilite a sua extensão para outras linguagens de programação específicas para o domínio hipermídia televisivo.

Durante o trabalho de pesquisa, o perfil UML-TV teve uma versão inicial (NOGUEIRA; DE SOUZA; CORTÉS, 2009), a partir da qual vários pontos foram remodelados. Como principal melhoria, pode-se destacar que a versão anterior utilizava diagramas de Comunicação e previa a utilização de apenas um diagrama por aplicação. Essas características limitavam o poder de expressividade do perfil e foram corrigidas no perfil apresentado neste trabalho através da utilização de outros tipos de diagramas UML e da separação da modelagem em vários diagramas, o que proporcionou um melhor entendimento das aplicações modeladas.

O cenário em que todo o trabalho intelectual de um projeto é realizado apenas em modelos ainda pode ser considerado distante, mas como pode ser verificado neste trabalho, a MDSE traz benefícios aos envolvidos na elaboração de sistemas por automatizar parte do processo de concepção, mantendo uma ligação permanente com a documentação das aplicações, na forma de diagramas que estão em conformidade com o perfil UML-TV no caso do trabalho aqui apresentado.

7.1. Trabalhos futuros

Uma série de trabalhos futuros pode ser vislumbrada como continuação do que foi produzido nesta dissertação. Entre esses trabalhos, pretende-se adicionar notações que permitam a modelagem de aplicações com suporte a múltiplos dispositivos, e.g. dispositivos móveis, acesso a bases de dados e persistência de informações no *set-top-box*, fazendo assim uso de funcionalidades possíveis apenas com canal de retorno.

Atualmente, o perfil UML-TV possui uma notação independente de plataforma, o que é essencial para a elevação do nível de abstração dos modelos. No entanto, aspectos específicos de certas linguagens não são suportados pelo perfil, e.g. contextos de NCL. Dessa forma, uma das atividades futuras no desenvolvimento do UML-TV é produzir modelos específicos de plataforma para que seja possível completar o fluxo de transformações sugerido pela MDA.

UML não se mostra muito adequada para a modelagem de interfaces gráficas de usuário. Nesse sentido, modelos independentes de computação (CIM) podem ser desenvolvidos levando em consideração níveis maiores de abstração e aspectos de IHC (interação humano-computacional) podendo ser inseridos em fases iniciais no processo de transformações que levam até a geração de código-fonte.

Outra atividade relacionada à criação de modelos intermediários diz respeito ao suporte para outras linguagens de programação, como Java, SMIL e BML, por exemplo. Ainda no campo das transformações, podem-se realizar trabalhos no sentido de permitir a aplicação de técnicas de engenharia reversa, o que pode ser útil na documentação de sistemas legados sem documentação suficiente.

Outro aspecto importante que pode ser considerado futuramente consiste na transformação dos modelos que utilizam o perfil UML-TV em modelos de Redes de Petri (PETRI, 1962) com o objetivo de identificar problemas nas aplicações em tempo de modelagem, e.g. situações de *deadlock* e mídias visualizáveis que possam estar presentes no diagrama, mas não ligadas a alguma área de visualização.

REFERÊNCIAS BIBLIOGRÁFICAS

AAGEDAL, J. O.; ECKLUND, E. F. Modelling QoS: Towards a UML Profile. In: 5TH INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, 2002, London. **Proceedings...** London: Springer, 2002. p.275-289.

ALBARINO, R. Goldstein's LightWorks at Southampton. **Variety**, Los Angeles, v. 213, n. 12, 10 ago. 1966.

ALENCAR, M. S. **Digital Television Systems**. New York: Cambridge University Press, 2009.

AMBLER, S. **The Object Primer: Agile Model-driven Development with UML 2.0**. 3rd ed. Cambridge: Cambridge Univ. Press, 2007.

ANTONACCI, M. J. **NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia com Sincronização Temporal e Espacial**, 2000. Dissertação (Mestrado), Rio de Janeiro: Departamento de Informática, PUC-Rio.

ANTONACCI, M. J. et al. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia na Web. In: VI SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E HIPERMÍDIA, 2000a, Natal. **Anais...** Natal: SBC, 2000. p.79-95.

_____. Improving the Expressiveness of XML-Based Hypermedia Authoring Languages. In: VII MULTIMEDIA MODELING CONFERENCE, 2000b, Nagano. **Proceedings...** Nagano: IEEE Computer Society, 2000. p.71-88.

ANTONACCI, M. J.; SOARES, L. F. G. **Especificação NCL**, 2000. Relatório técnico, Rio de Janeiro: PUC-Rio.

AZEVEDO, R. **NCL Eclipse**, 2008. Disponível em: <http://laws.deinf.ufma.br/~ncleclipse/>. Acesso em: 24 abr. 2009.

Associação Brasileira de Normas Técnicas. ABNT NBR 15604 - Televisão digital terrestre — Receptores, 2007. [s. l.]: [s. n.].

_____. ABNT NBR 15606-2 - Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital, 2009. [s. l.]: [s. n.].

Association of Radio Industries and Bussinesses. **XML-based Multimedia Coding Scheme. Data Coding and Transmission Specifications for Digital Broadcasting**, 2007. Especificação, Tokyo: [s. n.].

BADII, A. et al. Accessibility-by-Design: A Framework for Delivery-Context-Aware Personalised Media Content Re-purposing In: HOLZINGER, A.; MIESENBERGER, K. (Eds.). **HCI and Usability for e-Inclusion**. Berlin: Springer, 2009. p.209-226.

BANGIA, R. **Dictionary of Information Technology**. New Delhi: Firewall Media, 2007.

BARBOSA, S. D. J.; SOARES, L. F. G. TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade In: KOWALTOWSKI, T.; BREITMAN, K. (Eds.). **Minicursos da XXVII Jornada de Atualização em Informática**. Belém: SBC, 2008.

BARNAGHI, P. M.; KAREEM, S. A. UML Extensions for Hypermedia Navigation and Presentation Modeling. In: ACM SYMPOSIUM ON SOFTWARE VISUALIZATION, 2005, Saint Louis. **Proceedings...** Saint Louis: ACM, 2005.

BAUMEISTER, H.; KOCH, N.; MANDEL, L. Towards a UML extension for hypermedia design. In: 2ND INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE: BEYOND THE STANDARD, 1999, Fort Collins. **Proceedings...** Fort Collins: Springer, 1999. p.614–629.

BECKER, V.; SOARES, L. F. G. **Viva Mais - Alimentação Saudável | Clube NCL**, 2009. Disponível em: <http://clube.ncl.org.br/node/29>. Acesso em: 14 abr. 2010.

BERNERS-LEE, T.; CONNOLLY, D. **Hypertext markup language-2.0**, 1995. Especificação, [s. l.]: [s. n.].

CZARNECKI, K. Overview of Generative Software Development In: BANÂTRE, J.-P. et al. (Eds.). **Unconventional Programming Paradigms**. Berlin: Springer, 2005. p.326-341.

ECMA International. **Standard ECMA-262**, 2009. Disponível em: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>. Acesso em: 1 abr. 2010.

ENGELS, G.; SAUER, S. Object-oriented modeling of multimedia applications In: CHANG, S. K. (Ed.). **Handbook of Software Engineering and Knowledge Engineering**. Singapore: World Scientific, 2002. v. 2, p.21-53.

FONTOURA, M.; PREE, W.; RUMPE, B. **The UML profile for framework architectures**. Boston: Addison-Wesley, 2000.

FUENTES-FERNÁNDEZ, L.; VALLECILLO-MORENO, A. An introduction to UML profiles. **European Journal for the Informatics Professional**, [s.l.]: Novática, 2004, v. 5, n. 2, p. 6-15.

Fórum do Sistema Brasileiro de TV Digital Terrestre. **O que é o ISDB-TB**, 2009. Disponível em: <http://www.forumsbtvd.org.br/materias.asp?id=20>. Acesso em: 15 maio 2009.

GOSLING, J. et al. **Java Language Specification**. 3rd ed. Boston: Addison-Wesley, 2005.

GRÄSSLE, P.; BAUMANN, H.; BAUMANN, P. **UML 2.0 in Action: A Project-based Tutorial**. Birmingham: Packt Publishing Ltd., 2005.

GUEDES, G. T. A. **UML - Uma abordagem prática**. 3rd ed. São Paulo: Novatec, 2008.

GUIMARÃES, R. L.; COSTA, R. M. DE R.; SOARES, L. F. G. Composer: Authoring Tool for iTV Programs. In: 6TH EUROPEAN CONFERENCE ON CHANGING TELEVISION ENVIRONMENTS, 2008, Salzburg. **Proceedings...** Salzburg: Springer, 2008. p.61-71.

HAREL, D. Statecharts: a visual formalism for complex systems. **Science of Computer Programming**, Amsterdam: Elsevier North-Holland, 1987, v. 8, n. 3, p. 231-274.

IERUSALIMSCHY, R. Uma Introdução à Programação em Lua In: CARVALHO, A. P. DE L. F. DE; KOWALTOWSKI, T. (Eds.). **Minicursos da XXVIII Jornadas de Atualização em Informática**. Bento Gonçalves: SBC, 2009.

JAVED, A. Z.; STROOPER, P. A.; WATSON, G. N. Automated Generation of Test Cases Using Model-Driven Architecture. In: SECOND INTERNATIONAL WORKSHOP ON AUTOMATION OF SOFTWARE TEST, 2007, Minneapolis. **Proceedings of the 29th International Conference on Software Engineering**. Minneapolis: IEEE Computer Society, 2007. p.3.

JOUAULT, F. et al. ATL: a QVT-like transformation language. In: 21TH ACM SIGPLAN SYMPOSIUM ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS, 2006, Portland. **Companion to the 21th ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications**. Portland: ACM, 2006. p.719-720.

KIOUSIS, S. Interactivity: a concept explication. **New Media Society**, [s.l.]: [s.n], 2002, v. 4, n. 3, p. 355-383.

KLEPPE, A. G.; WARMER, J.; BAST, W. **MDA explained: the model driven architecture: practice and promise**. Boston: Addison-Wesley, 2003.

LOPES, D. **Étude et applications de l'approche MDA pour des plates-formes de Services Web**, 2005. Tese (Doutorado), Nantes: Université de Nantes.

MARQUES NETO, M. C.; SANTOS, C. A. S. StoryToCode: Um Modelo Baseado em Componentes para Especificação de Aplicações de TV Digital e Interativa

Convergentes. In: XV SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB, 2009, Fortaleza. **Anais...** Fortaleza: SBC, 2009.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, New York: ACM, 2005, v. 37, n. 4, p. 316-344.

Ministério das Comunicações. **Site SBTVD v3**, 2003. Disponível em: <http://sbtvd.cpqd.com.br/?obj=historico&mtd=texto&item=5>. Acesso em: 25 abr. 2009.

NOGUEIRA, T. P.; DE SOUZA, C. T.; CORTÉS, M. I. Geração de Aplicações Hipermídia para Televisão Digital Baseada em Desenvolvimento Orientado a Modelos. In: XXXV CONFERÊNCIA LATINOAMERICANA DE INFORMÁTICA, 2009, Pelotas. **Anais...** Pelotas: SBC, 2009.

OLDEVIK, J. MOFScript Eclipse Plug-In: Metamodel-Based Code Generation. In: ECLIPSE TECHNOLOGY EXCHANGE WORKSHOP, 2006, Nantes. Nantes: [s. n.], 2006.

_____. **MOFScript User Guide**. p.33, 2009. Guia do usuário, [s. l.]: [s. n.].

OLDEVIK, J. et al. Toward Standardised Model to Text Transformations In: HARTMAN, A.; KREISCHE, D. (Eds.). **Model Driven Architecture – Foundations and Applications**. Berlin: Springer, 2005. v. 3748, p.239-253.

OLIVEIRA, C. T.; DE SOUZA, C. T. Especificação de Canal de Retorno em Aplicações para TV Digital Interativa. In: XXII SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2005, Recife. **Anais...** Recife: SBrT, 2005.

ORLIKOWSKI, W. J. CASE tools as organizational change: investigating incremental and radical changes in systems development. **MIS quarterly**, Minessota: Management Information Systems Research Center, 1993, v. 17, n. 3, p. 309-340.

Object Management Group. **MOF Model to Text Transformation Language RFP**, 2004. Disponível em: <http://www.omg.org/cgi-bin/doc?ad/04-04-07>. Acesso em: 14 ago. 2010.

_____. **MOF 2.0**, 2006. Disponível em: <http://www.omg.org/spec/MOF/2.0/>. Acesso em: 24 ago. 2010.

_____. **XMI 2.1.1**, 2007. Disponível em: <http://www.omg.org/spec/XMI/2.1.1/>. Acesso em: 24 ago. 2010.

_____. **QVT 1.0**, 2008. Disponível em: <http://www.omg.org/spec/QVT/1.0/>. Acesso em: 18 mar. 2010.

_____. **Object Constraint Language**, 2009a. Disponível em: <http://www.omg.org/technology/documents/formal/ocl.htm>. Acesso em: 20 maio 2009.

_____. **UML 2.1.2 Superstructure**, 2009b. Disponível em: <http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/>. Acesso em: 25 abr. 2009.

_____. **UML 2.1.2 Infrastructure**, 2009c. Disponível em: <http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/>. Acesso em: 25 abr. 2009.

_____. **MDA**, 2010. Disponível em: <http://www.omg.org/mda/>. Acesso em: 16 abr. 2010.

Oracle. **Java ME - the Most Ubiquitous Application Platform for Mobile Devices**, 2010a. Disponível em: <http://java.sun.com/javame/index.jsp>. Acesso em: 6 abr. 2010.

_____. **Applets**, 2010b. Disponível em: <http://java.sun.com/applets/>. Acesso em: 6 abr. 2010.

PAULINELLI, F.; KULESZA, R. **Middleware Ginga-J - Histórico Ginga-J - GingaCDN**, 2009. Disponível em: <http://ginga.lavid.ufpb.br/projects/ginga-j/wiki/Hist%C3%B3rico_Ginga-J>. Acesso em: 6 abr. 2010.

PETRI, C. A. **Kommunikation mit automaten**, 1962. Tese (Doutorado), Rhein.-Westfäl: Inst. f. Instrumentelle Mathematik an der Univ. Bonn.

POOLEY, R. J.; WILCOX, P. **Applying UML: advanced application**. Oxford: Butterworth-Heinemann, 2004.

PORRES, I. et al. Development of an Ubiquitous Decision Support System for Clinical Guidelines using MDA. In: 19TH INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 2007, Trondheim. **Proceedings...** Trondheim: Springer, 2007.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual (2nd Edition)**. 2nd ed. Massachussets: Addison-Wesley, 2004.

República Federativa do Brasil. **Decreto Nº 4901, de 23 de novembro de 2003**, 2003. Disponível em: <http://sbtvd.cpqd.com.br/downloads/decreto_4901_2003.pdf>. Acesso em: 24 abr. 2009.

SANT'ANNA, F.; CERQUEIRA, R.; SOARES, L. F. G. NCLua: objetos imperativos lua na linguagem declarativa NCL. In: 14TH BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB, 2008, Vila Velha. **Proceedings...** Vila Velha: ACM, 2008. p.83-90.

SAUER, S.; ENGELS, G. Extending UML for modeling of multimedia applications. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1999, Tokyo. **Proceedings...** Tokyo: IEEE Computer Society, 1999. p.80–87.

_____. UML-based behavior specification of interactive multimedia applications. In: IEEE SYMPOSIA ON HUMAN-CENTRIC COMPUTING LANGUAGES AND ENVIRONMENTS, 2001, Stresa. **Proceedings...** Stresa: IEEE Computer Society, 2001. p.248-255.

SEETHARAMAN, K. The CORBA connection. **Communications of the ACM**, New York, v. 41, n. 10, p. 34-36, 1998.

SILVA, H. V. DE O. E et al. NCL 2.0: integrating new concepts to XML modular languages. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING, 2004, Milwaukee. **Proceedings...** Milwaukee: ACM, 2004. p.188-197.

SOARES NETO, C. S. et al. **Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 e a Ferramenta Composer**, 2007. Tutorial, Rio de Janeiro: PUC-Rio.

SOARES, L. F. G. **Nested Context Model version 2.3**, 2000. Relatório técnico, Rio de Janeiro: PUC-Rio.

SOARES, L. F. G.; RODRIGUES, R. F. **Nested Context Language 3.0 Part 8–NCL Digital TV Profiles**, 2006. Relatório técnico, Rio de Janeiro: PUC-Rio.

SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-NCL: The declarative environment of the Brazilian digital TV system. **Journal of the Brazilian Computer Society**, Campinas: SBC, 2007, v. 12, n. 4, p. 37-46.

DE SOUZA FILHO, G. L.; LEITE, L. E. C.; BATISTA, C. Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. **Journal of the Brazilian Computer Society**, Campinas: SBC, 2007, v. 13, n. 4, p. 47-56.

SRIVASTAVA, H. O. **Interactive TV technology and markets**. Boston: Artech House, 2002.

STROUSTRUP, B. **The C++ programming language**. Massachussets: Addison-Wesley, 1986.

Telemídia. **Ginga Digital TV Middleware Specification**, 2008. Disponível em: <http://www.gingancl.org.br/tools.html>. Acesso em: 19 ago. 2010.

_____. **Ginga-NCL - Declarative DTV Middleware**, 2009. Disponível em: <http://www.gingancl.org.br/ferramentas.html>. Acesso em: 14 fev. 2010.

UETANABARA JÚNIOR, J.; CAMARGO, V. V.; CHAVEZ, C. V. F. UML-AOF: a profile for modeling aspect-oriented frameworks. In: 13TH WORKSHOP ON

ASPECT-ORIENTED MODELING, 2009, Charlottesville. **Proceedings...** Charlottesville: ACM, 2009. p.1-6.

WATSON, A. UML vs. DSL: a false dichotomy, 2007. [s. l.]: [s. n.].

World Wide Web Consortium. **XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)**, 2002. Disponível em: <http://www.w3.org/TR/xhtml1/>. Acesso em: 1 abr. 2010.

_____. **Synchronized Multimedia Integration Language (SMIL 2.0) - [Second Edition]**, 2005. Disponível em: <http://www.w3.org/TR/2005/REC-SMIL2-20050107/>. Acesso em: 17 fev. 2010.

_____. **Extensible Markup Language (XML)**, 2010a. Disponível em: <http://www.w3.org/XML/>. Acesso em: 31 mar. 2010.

_____. **W3C XML Schema**, 2010b. Disponível em: <http://www.w3.org/XML/Schema>. Acesso em: 21 fev. 2011.

ZIADI, T.; HELOUET, L.; JEZEQUEL, J. M. Towards a UML profile for software product lines In: LINDEN, F. VAN DER (Ed.). **Software Product-Family Engineering**. Berlin: Springer, 2004. v. 3014, p.129-139.

APÊNDICE A: CÓDIGO-FONTE VIVA MAIS PRATOS

Código-fonte gerado para a aplicação Viva Mais Pratos (BECKER; SOARES, 2009) modelada utilizando o perfil UML-TV:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="vivaMaisCompacto" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.ncl.org.br/NCL3.0/EDTVProfile
      http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd">
<head>

      <regionBase>
            <region id="rgVideo" width="100%" height="100%" />
            <region id="rgIcone" left="2%" top="2%" width="33.59%"
height="7.29%" />
            <region id="rgYellowMeal" left="73.75%" top="7.6667%"
width="19.53%" height="29.58%" />
            <region id="rgBlueMeal" left="73.75%" top="46.6667%"
width="19.53%" height="29.58%" />
            <region id="rgRedMeal" left="7.5%" top="7.6667%" width="19.53%"
height="29.58%" />
            <region id="rgGreenMeal" left="7.5%" top="46.6667%"
width="19.53%" height="29.58%" />
            <region id="rgTexPratos" left="25%" top="60%" width="50%"
height="16.67%" />
            <region id="rgBackground" width="100%" height="100%" />
      </regionBase>

      <descriptorBase>
            <descriptor id="dVideo" region="rgVideo" />
            <descriptor id="dIcon" region="rgIcone" />
            <descriptor id="dYellowMeal" region="rgYellowMeal" />
            <descriptor id="dBlueMeal" region="rgBlueMeal" />
            <descriptor id="dRedMeal" region="rgRedMeal" />
            <descriptor id="dGreenMeal" region="rgGreenMeal" />
            <descriptor id="dInfoText" region="rgTexPratos" />
                  <descriptorParam name="border" value="none"/>
            <descriptor id="dFundoPratos" region="rgBackground" />
      </descriptorBase>

      <connectorBase>
            <importBase alias="connectors" documentURI="connectorBase.ncl"
/>
      </connectorBase>

</head>
```

```
<body>

	<port id="entrada" component="video"/>

		<media descriptor="dVideo" id="video" src="video/vivamais.mp4"
>
			<area begin="63s" end="74s" id="arshowIcon" />
			<property name="bounds"/>
		</media>
		<media descriptor="dIcon" id="icon"
src="imagens/interativo.png" />
		<media descriptor="dYellowMeal" id="yellowMeal"
src="imagens/PratoAmareloFundo.png" />
		<media descriptor="dBlueMeal" id="blueMeal"
src="imagens/PratoAzulFundo.png" />
		<media descriptor="dRedMeal" id="redMeal"
src="imagens/PratoVermelhoFundo.png" />
		<media descriptor="dGreenMeal" id="greenMeal"
src="imagens/PratoVerdeFundo.png" />
		<media descriptor="dInfoText" id="infoText"
src="imagens/InstPratosFundo.png" />
		<media descriptor="dFundoPratos" id="fundoPratos"
src="imagens/fundo_pratos.png" />
		<media descriptor="dInfoText" id="redMealText"
src="imagens/InstPratoVermelhoFundo.png" />
		<media descriptor="dInfoText" id="greenMealText"
src="imagens/InstPratoVerdeFundo.png" />
		<media descriptor="dInfoText" id="yellowMealText"
src="imagens/InstPratoAmareloFundo.png" />
		<media descriptor="dInfoText" id="blueMealText"
src="imagens/InstPratoAzulFundo.png" />


	<link xconnector="connectors#onKeySelectionStartStop">
		<bind component="yellowMeal" role="onSelection">
			<bindParam name="keyCode" value="YELLOW"/>
		</bind>
		<bind component="blueMeal" role="stop"/>
		<bind component="redMeal" role="stop"/>
		<bind component="greenMeal" role="stop"/>
		<bind component="infoText" role="stop"/>
		<bind component="yellowMealText" role="start"/>
	</link>


	<link xconnector="connectors#onKeySelectionStartStop">
		<bind component="blueMeal" role="onSelection">
			<bindParam name="keyCode" value="BLUE"/>
		</bind>
		<bind component="redMeal" role="stop"/>
		<bind component="greenMeal" role="stop"/>
		<bind component="infoText" role="stop"/>
		<bind component="blueMealText" role="start"/>
		<bind component="yellowMeal" role="stop"/>
	</link>


	<link xconnector="connectors#onKeySelectionStartStop">
		<bind component="redMeal" role="onSelection">
			<bindParam name="keyCode" value="RED"/>
```

```
            </bind>
            <bind component="greenMeal" role="stop"/>
            <bind component="infoText" role="stop"/>
            <bind component="redMealText" role="start"/>
            <bind component="yellowMeal" role="stop"/>
            <bind component="blueMeal" role="stop"/>
      </link>


      <link xconnector="connectors#onKeySelectionStartStop">
            <bind component="greenMeal" role="onSelection">
                  <bindParam name="keyCode" value="GREEN"/>
            </bind>
            <bind component="infoText" role="stop"/>
            <bind component="greenMealText" role="start"/>
            <bind component="yellowMeal" role="stop"/>
            <bind component="blueMeal" role="stop"/>
            <bind component="redMeal" role="stop"/>
      </link>

      <link xconnector="connectors#onBegin1StartN">
            <bind component="video" interface="arshowIcon" role="onBegin"/>
            <bind component="icon" role="start"/>
      </link>

      <link xconnector="connectors#onEndStop">
            <bind component="video" interface="arshowIcon" role="onEnd"/>
            <bind component="icon" role="stop"/>
      </link>

      <link id="1FundoPratosRedMealStart"
xconnector="connectors#onBegin1StartN">
            <bind component="fundoPratos" role="onBegin" />
            <bind component="redMeal" role="start" />
            <bind component="blueMeal" role="start" />
            <bind component="yellowMeal" role="start" />
            <bind component="greenMeal" role="start" />
            <bind component="infoText" role="start" />
      </link>

      <link xconnector="connectors#onKeySelectionSetResizeStartStop">
            <bind component="icon" role="onSelection">
                  <bindParam name="keyCode" value="RED"/>
            </bind>
            <bind component="video" interface="bounds" role="set">
                  <bindParam name="var" value="27.97%, 13.33%, 44.22%,
41.87%"/>
            </bind>
            <bind component="fundoPratos" role="start"/>
            <bind component="icon" role="stop"/>
      </link>

</body>

</ncl>
```

APÊNDICE B: DIAGRAMAS DE MÁQUINAS DE ESTADOS E DE ATIVIDADES

Neste apêndice, serão mais detalhados os diagramas de Máquinas de Estados e de Atividades da UML, os quais se constituem em dois importantes diagramas para a modelagem de aspectos dinâmicos de um sistema.

Diagramas de Máquinas de Estados

De acordo com a especificação UML (Object Management Group, 2009b), as máquinas de estados servem para modelar comportamento através de transições entre estados finitos. Além de representar o comportamento de um sistema, máquinas de estados também podem ser utilizadas para determinar o protocolo de utilização de parte de um sistema. Esses dois tipos de máquinas de estados são chamados de máquinas de estados comportamentais e máquinas de estados de protocolo.

Diversos elementos de um sistema podem ter seus comportamentos modelados com máquinas de estados (instâncias de classes, por exemplo), que se constituem em uma variação orientada a objetos dos diagramas de estados de Harel (1987).

Uma máquina de estados é um modelo que engloba todas as possíveis situações de um objeto, parte de um sistema, ou um sistema por completo. Qualquer influência externa é representada como um evento que, ao ser detectado, tem uma resposta dependente do estado atual em que se encontra a máquina de estados onde tal resposta pode ser a execução de uma ação e uma mudança para outro estado. Na Figura 43 consta um diagrama de Máquina de Estados de exemplo contendo os

elementos mais utilizados neste tipo de diagrama: os estados e os pseudoestados[11]: estado inicial, *fork*, *join*, escolha, junção e estado final.

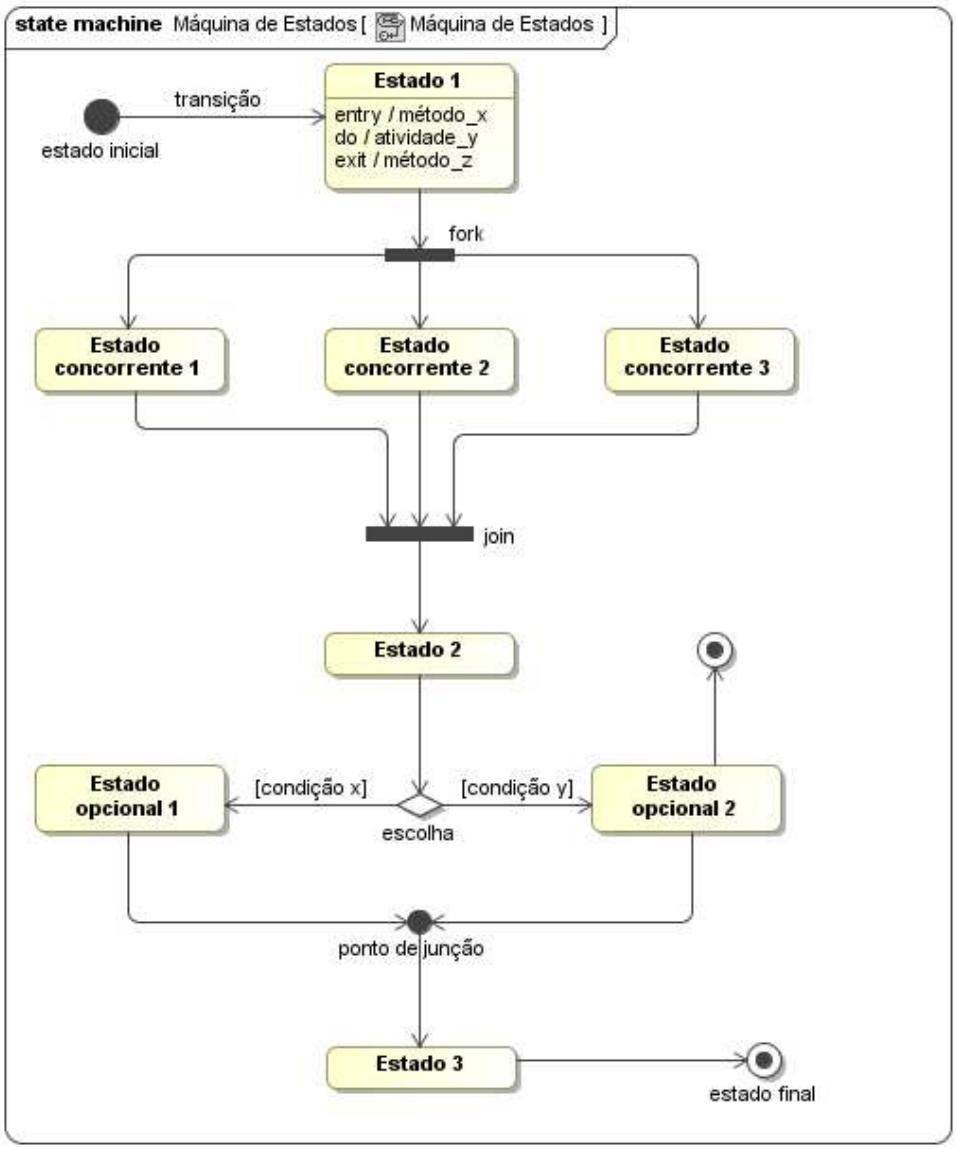


Figura 43 – Exemplo de diagrama de Máquina de Estados

Um estado inicial é um pseudoestado que pode ter somente uma transição de saída, não pode ter condição de guarda (expressões que controlam o disparo de transições) e o alvo de sua transição é estado padrão do diagrama. Uma máquina de estados pode ter um pseudoestado final que indica que a execução da máquina será terminada. Esse estado pode aparecer diversas vezes no diagrama.

[11] Todos os vértices que não se constituem em estados em um diagrama de Máquina de Estados são chamados de pseudoestados

Uma transição é um relacionamento direcionado entre um vértice de origem e um vértice de destino. Opcionalmente, a transição pode estar associada a uma condição de guarda que permite controlar o disparo da transição, tais condições são expressas entre colchetes nos diagramas UML. Quando um evento é lançado, a condição de guarda é avaliada e, caso seja verdadeira, a transição é ativada (ou desativada, caso contrário).

Um estado é uma situação em que se encontra um objeto ou sistema durante a qual uma ou mais condições são satisfeitas ou alguma atividade é realizada (Object Management Group, 2009b). Existem três tipos de estados: estados simples (sem subestados), estados compostos (com subestados), e estados submáquinas (com uma submáquina de estados).

Os estados podem estar ativos ou inativos durante a execução da máquina de estados. Um determinado estado fica ativo quando é uma transição resulta na entrada no referido estado. Quando uma transição resulta na saída do estado, o mesmo se torna inativo. Uma mesma transição também pode resulta na ativação e desativação de um estado.

Sempre que um estado se torna ativo, é executado o que for definido em sua propriedade *entry* antes que qualquer outra ação ocorra. Similarmente, sempre que se sai de um estado, é executado o que está especificado na propriedade *exit* antes da saída. Entre as ações *entry* e *exit* pode ser especificado que ocorrerá algo através da *do-activity*. É importante ressaltar que nenhum desses três comportamentos são obrigatórios.

Vértices do tipo *fork* servem para dividir uma transição de entrada em duas ou mais outras transições que não podem ter condições de guarda. Pseudoestados do tipo *join* são utilizados para unir várias transições de diversos vértices diferentes. Assim como nos *forks*, as transições de entrada não podem ter condições de guarda. Em uma máquina de estados completa, *forks* devem ter exatamente uma transição de entrada e no mínimo duas transições de saída, já *joins* dever ter pelo menos duas transições de entrada e exatamente uma transição de saída.

Junções são vértices de semântica livre usados para ligar múltiplas transições na forma de *merges* (convergindo duas ou mais transições em uma só) e ramificações condicionais estáticas (dividindo uma transição em duas ou mais transições com condições de guarda diferentes). Esse tipo de pseudoestado não possui as limitações com relação às quantidades de transições de entrada e de saída que os pseudoestados *fork* e *join* têm, i.e. uma junção pode ter apenas uma transição de entrada e uma transição de saída.

Pseudoestados de escolha são usados para especificar ramificações condicionais dinâmicas que permitem dividir transições de entrada em duas ou mais transições de saída em função da avaliação de condições de guarda que por sua vez podem ser influenciadas por ações ocorridas previamente na máquina de estados. Se mais de uma das condições for verdadeira, uma transição arbitrária é escolhida entre aquelas que foram avaliadas como verdadeiras. Caso nenhuma condição de guarda seja verdadeira, o modelo está mal formado (isso pode ser evitado adicionando uma condição *else* para cada vértice de escolha). Um pseudoestado de escolha possui as mesmas características das junções no que diz respeito à quantidade de transições de entrada e de saída.

Diagramas de Atividades

O diagrama de Atividades é o que dá maior ênfase no nível de algoritmo na UML, apresentando muitas semelhanças com fluxogramas (GUEDES, 2008). Esse tipo de diagrama era considerado um caso especial dos diagramas de Máquinas de Estados em versões anteriores da UML, portanto muitos dos seus elementos são usados também nos diagramas de Máquina de Estados, como é o caso dos pseudoestados inicial e final, *fork*, *join* e pseudoestado de escolha, portanto não serão novamente apresentados.

A diferença básica entre os dois tipos de diagramas é que o diagrama de Máquinas de Estados é orientado a eventos de estados de um objeto enquanto que o diagrama de Atividades é orientado a fluxos de controle de atividades.

Uma atividade representa execuções de ações utilizando um modelo de fluxo de dados e de controle que podem ser iniciadas por outras ações no modelo, pela disponibilidade de dados e objetos ou pela ocorrência de eventos externos. O fluxo é modelado com nós de atividade que podem ser ações ou elementos de controle de fluxo (e.g. barras de sincronização e escolhas). A Figura 44 mostra um diagrama de Atividades de exemplo contendo vários de seus elementos mais comuns.

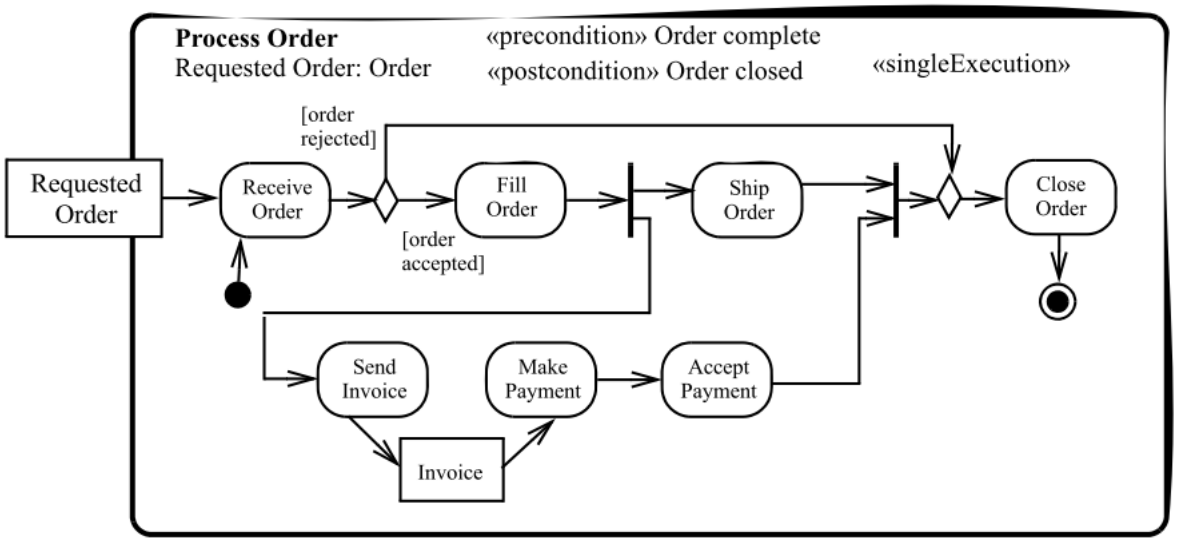


Figura 44 – Exemplo de diagrama de Atividades (Object Management Group, 2009b)

As ações são as unidades principais dos diagramas de Atividades e podem ser de vários tipos, como funções primitivas, chamadas a outras atividades, envio de sinais e manipulações de objetos. A maioria das atividades possui mecanismos de sequenciamento de fluxo de controle e de dados entre as ações, e.g fluxos de objetos para direcionar os dados gerados por ações para outros nós, fluxos de controle para determinar a execução dos nós, nós de controle para estruturar o fluxo de objetos e de controle e nós de objetos para representar objetos e dados ou *tokens*.

APÊNDICE C: NESTED CONTEXT LANGUAGE

Neste apêndice, será descrita uma visão geral sobre a linguagem NCL (*Nested Context Language*), que é uma linguagem declarativa voltada para a composição de documentos hipermídia. Sua primeira versão foi publicada por Antonacci et al. (ANTONACCI et al., 2000b), teve uma segunda versão (SILVA et al., 2004) e depois de algumas versões intermediárias, atualmente se encontra em sua terceira versão (SOARES; RODRIGUES, 2006).

Todas as máquinas de apresentação dos três principais sistemas de televisão digital[12] utilizam uma linguagem baseada em XHTML (Associação Brasileira de Normas Técnicas, 2009). No padrão brasileiro, a principal linguagem declarativa suportada é a NCL, apesar de ser possível inserir objetos XHTML em documentos NCL. Alguns fatores importantes levaram à adoção de NCL em detrimento ao XHTML no ISDB-Tb como o provimento de melhor separação entre conteúdo e estrutura, o suporte a sincronismo espaço-temporal, a adaptabilidade, o suporte a múltiplos dispositivos, entre outros.

Antes de apresentar a estrutura de um documento NCL, é importante conceituar os elementos que compõem uma aplicação desse tipo. Os elementos principais são as regiões, os conectores, os objetos de mídia e os elos. Os objetos de mídia são os elementos centrais da apresentação de um documento hipermídia, são os vídeos, textos, imagens, sons, outros documentos NCL e aplicações externas que são utilizados no decorrer da execução da aplicação. As regiões demarcam as áreas da tela onde os objetos de mídia são mostrados. Os descritores fazem a ligação entre regiões e

[12] No momento em que este trabalho foi escrito, os três principais sistemas de televisão digital eram o americano ATSC, o europeu DVB, e o japonês ISDB.

objetos de mídia indicando de que forma serão executados os objetos. Os elos definem as relações de sincronismo entre os objetos de mídia e regiões. Os conectores servem para especificar o comportamento dos elos ao longo da execução do documento NCL. Neste apêndice, será considerada a existência de um único contexto, contexto principal `body`.

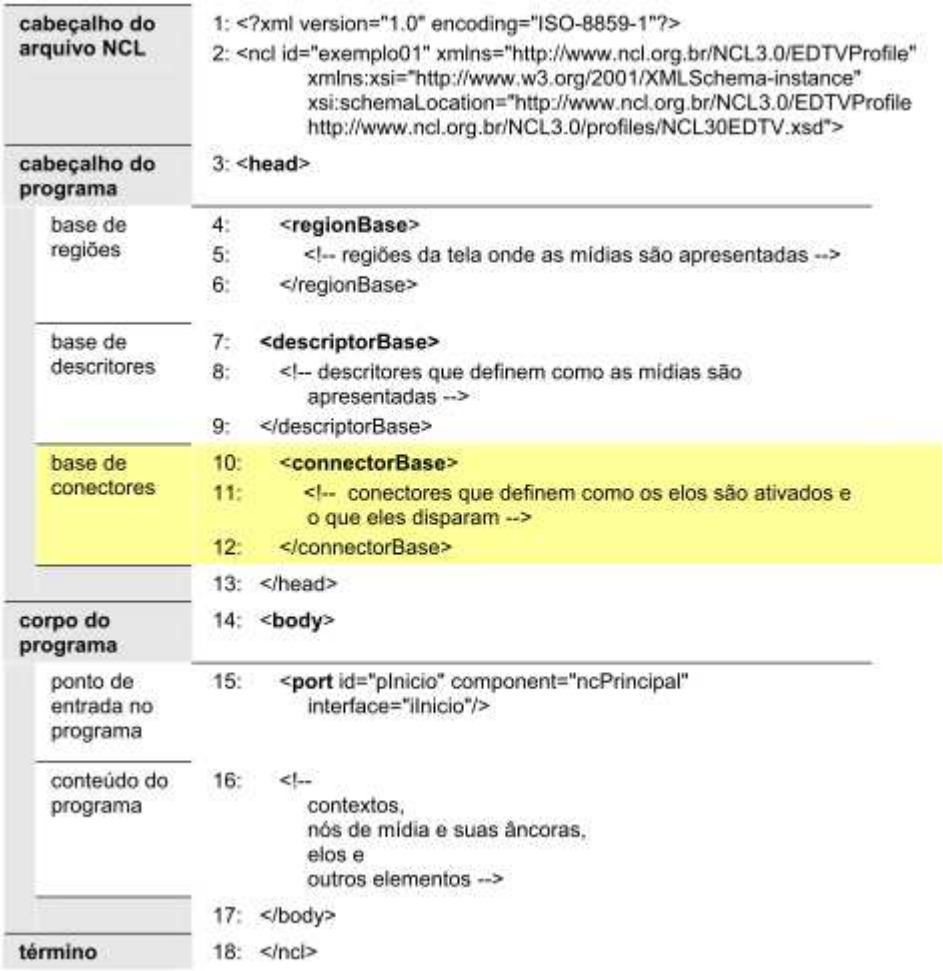


Figura 45 - Estrutura básica de um documento NCL (SOARES NETO et al., 2007)

A Figura 45 mostra a estrutura básica de um documento NCL iniciando pelo cabeçalho do arquivo, passando pelo cabeçalho do programa, corpo do programa e o término do documento. No cabeçalho do arquivo, nota-se a declaração XML e a codificação utilizada no documento. Além disso, é inserida a *tag* raiz `ncl` contendo

como atributos o `id` do documento e os *namespaces*[13] de NCL e XML Schema (World Wide Web Consortium, 2010b).

O cabeçalho do programa pode conter outros três elementos: a base de regiões, a base de descritores e a base de conectores. Na base de regiões, são definidas as áreas da tela dos dispositivos (televisão ou dispositivo móvel) onde serão exibidos os objetos de mídia visualizáveis (textos, imagens e vídeos). Cada região contém um `id` (obrigatório), título, altura, largura, sua distância referente ao topo da tela, sua distância referente ao lado esquerdo da tela, sua posição com relação a outras regiões, além de outras configurações.

A base de descritores contém informações que definem como as mídias serão apresentadas. No caso de mídias visualizáveis, normalmente é feita a associação das mesmas a áreas de visualização definidas na base de regiões. Além da posição onde mídias visualizáveis são exibidas, também é possível definir outros parâmetros inclusive para mídias como áudio, que não podem ser exibidas. Exemplos de parâmetros de execução que podem ser definidos incluem volume, cor da borda da região, transparência, entre outros.

Os conectores definem o comportamento dos elos do documento NCL. Normalmente são especificados em um arquivo a parte para facilitar o reuso e a manutenção dos mesmos. Esse tipo de elemento é responsável pelo sincronismo dos objetos de mídia e regiões, pois nele são definidos os papéis que os nós de origem e destino exercem nos elos que o utilizam (SOARES NETO et al., 2007). O conector causal presente na NCL possui condições (e. g. `onBegin`, `onEnd`, `onAbort`) e ações (e. g. `start`, `stop`, `abort`) de modo que, a partir da combinação de condições e ações, sejam especificadas as relações de sincronismo espaço-temporal. Exemplos de conectores incluem `onEndStop`, `onBeginStop`, `onStarPause`, etc.

O corpo do programa é composto por uma porta, que é o ponto de entrada do contexto `body` onde é especificado qual nó será executado ao se iniciar a aplicação

[13] Namespaces são usados para definir tags únicas com o objetivo de evitar ambiguidades em documentos XML que utilizem vocabulários com tags de nomes idênticos

e é complementado pelos elos, que fazem a junção entre conectores e nós através da atribuição de papéis, condições e ações aos nós.